Towards Modeling, Specifying and Deploying Policies in Autonomous and Autonomic Systems Using an AOSE Methodology

Joaquin Peña
University of Seville
Spain
joaquinp@us.es

Michael G. Hinchey
NASA Goddard Space Flight Center
USA
Michael.G.Hinchey@nasa.gov

Roy Sterritt
University of Ulster
Northern Ireland
r.sterritt@ulster.ac.uk

Abstract

Autonomic Computing (AC), self-management based on high level guidance from humans, is increasingly gaining momentum as the way forward in designing reliable systems that hide complexity and conquer IT management costs. Effectively, AC may be viewed as Policy-Based Self-Management. We look at ways to achieve this, and in particular focus on Agent-Oriented Software Engineering. We propose utilizing an AOSE methodology for specifying autonomic and autonomous properties of the system independently, and later, by means of composition of these specifications, to construct a specification for the policy and its subsequent deployment.

1 Introduction and Motivation

Autonomic Systems (encompassing both Autonomic Computing and Autonomic Communications) is an emerging field [1] for the development of large-scale, self-managing, complex distributed computer-based systems.

As in all emerging fields, there are many fruitful areas for concern, that are worthwhile targets for research and development. Many issues are yet to be addressed, such as, for example, how should autonomic managers, which together with the component being managed make up an autonomic element, be defined such that it can exist in a collaborative autonomic environment, and ultimately provide self-management of the system.

The long term strategic vision of AC highlighted an overarching self-managing vision where the system would have such a level of "self" capability that a senior (human) manager in an organization could specify business policies—such as profit margin on a specific product range, or system quality of service for a band of customers—and the computing systems would do the rest themselves.

It has been argued that for this vision to become a reality, we would require AI completeness, Software Engineering completeness, and so on [2]. What is clear in this vision is the importance of some form of policy that is then translated to all levels in the system in order to achieve self-direction and self-management.

In introducing the concept of Autonomic Computing, IBM's Paul Horn likened the needs of large scale systems management to that of the human Autonomic Nervous System (ANS). The ANS, through self-regulation, is able to effectively monitor, control and regulate the human body without the need for conscious thought [7]. This self-regulation and separation of concerns provides human beings with the ability to concentrate on high level objectives without having to micro-manage the specific details involved. The vision and metaphor of Autonomic Computing is to apply the same principles of self-regulation and complexity-hiding to the design of computer-based systems, in the hope that one day computer systems can achieve the same level of self-regulation as the human ANS [7],[25]. In his talk, Horn highlighted that the Autonomic Computing system must "find and generate rules for how best to interact with neighboring systems" [7].

We propose to use a methodology called MaCMAS (Methodology Fragment for Analyzing Complex Multi-Agent Systems) which provides the models and techniques for adding policies at runtime. We propose to create isolated definitions of the features that we want to use in policies using MaCMAS models. Later, when we specify a policy, we deploy these models over the running system using MaCMAS model composition.

In addition, to illustrate our approach, we use an example from the NASA ANTS concept mission (described in Section 5). This mission involves the use of a swarm of pico-class spacecraft to explore and collect data from the asteroid belt, and exhibits both autonomous and autonomic properties.

Proceedings of the Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE'06)
0-7695-2544-X/06 $20.00 © 2006 IEEE

2 Policy-Based Management

Policies have been described as a set of considerations designed to guide decisions of courses of action [14], and Policy-Based management (PBM) may be viewed as an administrative approach to systems management that *a priori* establishes rules for dealing with situations that are likely to occur.

From this perspective, PBM works by controlling access to and setting priorities for the use of ICT resources[1], for instance, where a (human) manager may simply specify the business objectives and the system will achieve these in terms of the needed ICT [13]. For example: (1) "The customer database must be backed up nightly between 1 a.m. and 4 a.m."; (2) "Platinum customers are to receive no worse than 1-second average response time on all purchase transactions."; (3) "Only management and the HR senior staff can access personnel records."; and (4) "The number of connections requested by the Web application server cannot exceed the number of connections supported by the associated database." [9]. These examples highlight the wide range and multiple levels of policies, the first concerned with system protection through backup, the second with system optimization to achieve and maintain a level of quality-of-service for key customers; while the third and fourth are concerned with system configuration and protection.

Policy-Based Management has been the subject of extensive research in its own right. The Internet Engineering Task Force (IETF) has investigated Policy-Based Networking as a means for managing IP-based multi-service networks with quality-of-service guarantees. More recently, PBM has become extremely popular within the telecommunications industry, for next generation networking, with many vendors announcing plans and introducing PBM-based products. This is driven by the fact that policy has been recognized as a solution for managing complexity, and for guiding the behavior of a network or distributed system through high-level user-oriented abstractions [15]. A PBM tool may also reduce the complexity of product and system management by providing a uniform cross-product policy definition and management infrastructure [4].

With one definition of Autonomic Computing being Self-Management based on high level guidance from humans [11] and considering IBM's high-level set of self-properties (self-CHOP: configuration, healing, optimization and protection) against the types of typical policies mentioned previously (optimization, configuration and protection), the importance and relevance of polices for achieving autonomicity becomes clear [26].

[1] Whatis.com, Online computer and internet dictionary and encyclopedia, 2005.

3 Using AOSE for policy modelling

The field of Agent-Oriented Software Engineering (AOSE) has arisen to address methodological aspects and other issues related to the development of complex multi-agent systems. AOSE is a new software engineering paradigm that augurs much promise in enabling the successful development of more complex systems than is achievable with current Object-Oriented approaches which use agents and organizations of agents as their main abstractions [8].

The organizational metaphor has been proven to be one of the most appropriate tools for engineering Multi-Agent Systems (hereafter, MAS). The metaphor is used by many researchers to guide the analysis and design of MASs, e.g., [18, 20, 28].

A MAS organization can be observed from two different point of view [28]:

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, if we first determine the acquaintance organization, and we define the constraints required for the structural organization, a natural map is formed between the acquaintance organization and the corresponding structural organization. This is the process of assigning roles to agents [28]. Thus, we can conclude that any acquaintance organization can be modeled orthogonally to its structural organization [10].

We use this separation to specify policies at the acquaintance organization level, and deploy them over the structural organizational of the running system. The scope of policies usually implies features of several acquaintance sub-organizations. In such cases, we must first compose the acquaintance sub-organizations, this process being guided by the policy specification, to deploy it later.

4 Overview of MaCMAS/UML

MaCMAS is the AOSE methodology that we use to specify and deploy policies [21]. It is specially tailored to model

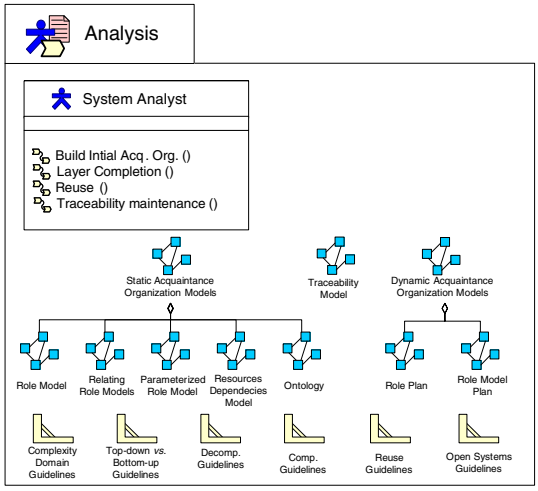

Figure 1. Acquaintance analysis discipline

complex acquaintance organizations [24]. Its main advantages can be observed from three aspects: in the modeling aspect, the main advantage consists in providing an interaction abstraction to enable the modeling of unpredictable behaviors, and providing a notation which, to the best of our knowledge, is the unique UML 2.0-based approach dedicated to modeling the acquaintance organization abstractly; in the techniques aspect, we provide semi-automatic techniques for decomposing and composing models basing on goal-oriented requirements and on dependencies, which is unique in the field; and in the software process aspect, we provide a software process that covers top-down and bottom-up development approaches providing criteria for deciding between them. To the best of our knowledge, our approach is the first to address such criteria.

We use this approach for several reasons. First, it provides UML-based models which are the de-facto standard in modeling, and which will decrease the learning-curve for engineers. Second, it allows modeling at different levels of abstraction, which allows us to specify policies at whichever level of abstraction we need. Third, it provides techniques to compose acquaintance models, which is needed for policies that imply several system-goals and for deploying an acquaintance model that specifies a policy over a structural organization; that is to say, composition of roles.

In Figure 1, we summarize the main Software Process Engineering Metamodel (SPEM) work definitions and models of the methodology. In the following, we detail the most important features for our purposes in this paper.

The MaCMAS/UML modeling process is focused on interactions/acquaintance organization since they are the main source of complexity. In order to represent interactions abstractly we use *multi-Role Interactions* (mRI) [22, 23]. mRIs are first class modeling elements in our models and are used as the minimum building block for modeling. Their use is crucial for performing an incremental layered modeling approach since mRIs can be described internally by means of finer-grain mRIs, or several of them can be abstracted by a coarser-grain one.

An mRI is an *institutionalized pattern of interaction* that abstractly represents the fulfillment of a system goal without detailing how this is achieved. Thus, using mRI as the minimum modeling element we do not have to take into account all of the details required to fulfill a complex system goal nor the messages that are exchanged at stages where these details have not been identified clearly, are not known, or are not even necessary. This allows us to have abstract models where intelligent behavior is carried out by means of neural networks, fuzzy logic, etc., (as, for example, is required in ANTS, cf. Section 5), without the necessity of dealing with all the details. In addition, the direct correlation between system goals and mRIs allows us to establish a clear traceability between goal-oriented requirement documents and analysis models. This is also important for our goal in this paper, since policies usually verse about system goals. Having this kind of model helps in simplifying the way in which policies are specified, and deployed in the system at runtime.

mRIs are represented with UML 2.0 collaborations [19, p. 132] as are all the models we use. We use three views of the acquaintance organization: two for representing the static and dynamic aspects of the organization, and a third for representing the relation between models in different abstraction layers. We use the following models:

a) Static Acquaintance Organization View: This shows the static interaction relationships between roles in the system and the knowledge processed by them. It comprises the following UML models:

- **Role Models:** shows an acquaintance sub-organization as a set of roles collaborating by means of several mRIs. As mRIs allow abstract representation of interactions, we can use these models at whatever level of abstraction we desire. We use role models to represent autonomous and autonomic properties of the system at the level of abstraction we need.
- **Ontology:** shows the ontology shared by roles in a role model. It is used to add semantics to the knowledge owned and exchanged by roles. We do not show it in this paper, but, as we show later, they are also important for deploying policies.

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 [19, 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 [19, 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.

5 ANTS Case Study and some of its models

In this section, we briefly introduce ANTS, a NASA concept mission, that illustrates properties of several potential exploration missions. We show two models of an autonomous and autonomic property of the system.

5.1 ANTS Mission Overview

The Autonomous Nano-Technology Swarm (ANTS) mission [3, 27] is a concept mission that involves the use of swarms of autonomous pico-class (approximately 1kg) spacecraft that will search the asteroid belt for asteroids that have specific characteristics. The mission is envisioned to consist of approximately 1,000 spacecraft launched from a factory ship. As shown in Figure 2, the swarm is envisioned to consist of several types of spacecraft. Many of these spacecraft (called specialists) will have a specialized single instrument for collecting particular types of data. To examine an asteroid, several spacecraft will have to form a sub-swarm, under the control of a ruler, and collaborate to collect data from asteroids of interest, based on the properties of that asteroid. This will be achieved using an insect analogy of hierarchical social behavior with some spacecraft directing others.

5.2 Autonomic Properties of ANTS

The ANTS system may be viewed as an Autonomic System as it meets four key requirements: self-configuration,

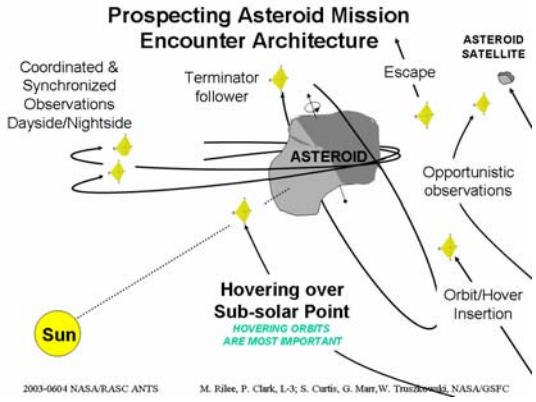

Figure 2. ANTS encounter with an asteroid

self-healing, self-optimization and self-protection, as illustrated in [27]. Here we focus on self-configuration properties as these are illustrated in our case study.

ANTS is self-protecting: The self protecting behavior of the team will be interrelated with the self-protecting behavior of the individual members. The anticipated sources of threats to ANTS individuals (and consequently to the team itself) will be collisions and solar storms.

Collision avoidance through maneuvering will be limited because ANTS individuals will have limited ability to adjust their orbits and trajectories, due to thrust for maneuvering powered by solar sails. Individuals will have the capability of coordinating their orbits and trajectories with other individuals to avoid collisions with them. Given the chaotic environment of the asteroid belt and the highly dynamic trajectories of the objects in it, occasional near approaches of interloping asteroidal bodies (even small ones) to the ANTS team may present threats of collisions with its individuals. Collision-avoidance maneuvering for this type of spacecraft presents a great challenge and is currently under consideration. The main self-protection mechanism for collision avoidance is achieved through the process of planning. The plans involve constraints that will result in acceptable risks of collisions between individuals when they carry out their observational goals. In this way, ANTS exhibits a kind of self-protection behavior against collisions.

Another possible ANTS self-protection mechanism could protect against the effects of solar storms, which is the basis of the case study we use later in this paper. Charged particles from solar storms could subject individuals to degradation of sensors and electronic components. The increased solar wind from solar storms could also affect the orbits and trajectories of the ANTS individuals and thereby could jeopardize the mission. Specific mechanisms to protect ANTS spacecraft against the effects of solar storms have not yet been determined. A potential mech-

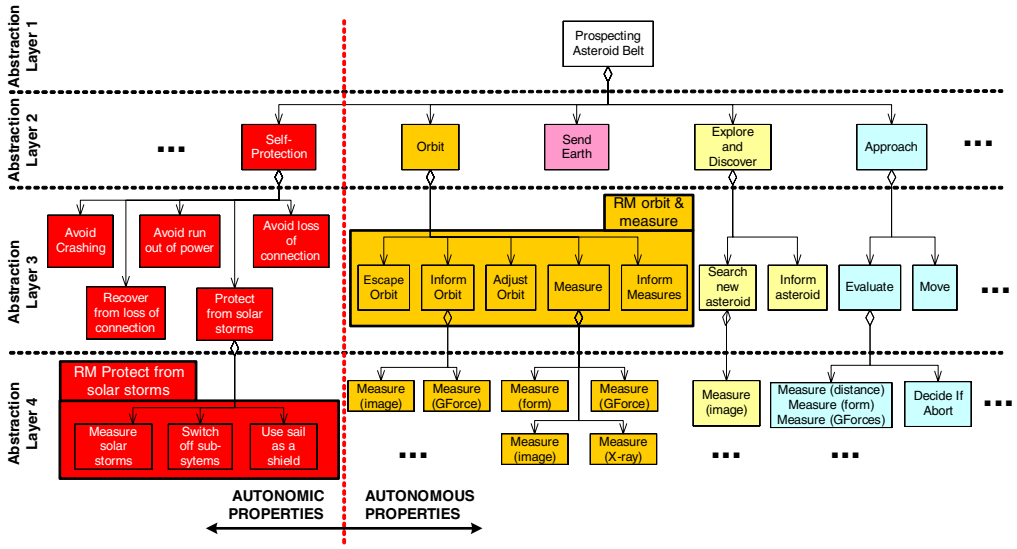

Figure 3. Traceability model of ANTS

anism might, for example, provide spacecrafts with a solar storm sensing capability through on-board, direct observation of the solar disk. When the spacecraft recognize that a solar storm threat exists, they would invoke their goal of protecting themselves from the harmful effects of a solar storm. Part of the protective response might be to orient solar panels and sails to minimize the impact of the solar wind. An additional response might be to power down unnecessary subsystems to minimize disruptions and damage from charged particles.

5.3 Example of Models of Autonomous and Autonomic Properties of ANTS

After applying MaCMAS to the ANTS system, we obtain the traceability diagram of Figure 3. This diagram summarizes the mRIs in the system structured by layers of abstraction. In this diagram, the top layer is the most abstract. As each node represents a system-goal also, we can see here the division of tasks necessarily undertaken to develop the system. As each mRI is inside a role model, we can also see which roles we have determined to carry out by observing the role models. In the model shown, we have depicted several sub-regions. Horizontal subdivisions depict layers of abstraction, while the vertical line denotes the distinction between the parts of the system that represent autonomic and the parts of the system that represent autonomous behaviors. In addition to mRIs, MaCMAS also uses UML packages to represent role models that contain several mRIs. In Figure 3 we identify two of these packages, which group the mRIs used in the example that follows.

To foster reuse, to model an autonomous or an autonomic property in a sufficiently generic and generalized way, and to enable a policy to be deployed at runtime, properties must be independent of the concrete agents over which they will be deployed. As we have shown, the features required to have an appropriate description correlates with the features of an acquaintance sub-organization. As we have also shown, to represent this kind of organization, MaCMAS proposes two kind of models—one for showing the relationships between roles, that is, role models, and another to show how these relationships evolve over time, that is to say, plan models.

For example, showing the autonomous process of orbiting an asteroid to take a measurement requires at least two models–its role model and its plan model. Figure 4b shows the role model for this case. We show here the models from the third layer of abstraction of Figure 3. In this model there are two kinds of elements: roles, which are represented using interface-like icons, and mRIs, which are represented as collaboration-like icons. In this model, roles show which is their general goal and their particular goals when participating in a certain interaction with other roles or with some part of the environment (represented using interfaces with the <<environment>> stereotype). Roles also represent the knowledge they manage (middle compartment) and the services they offer (bottom compartment). For example, the goal of the *Orbiter* role is "maintain the orbit and measure [the asteroid]", while its goal when participating in the *Report Orbit* interaction is to get a model of the orbit it must follow. In addition to roles, mRIs also show us some im-

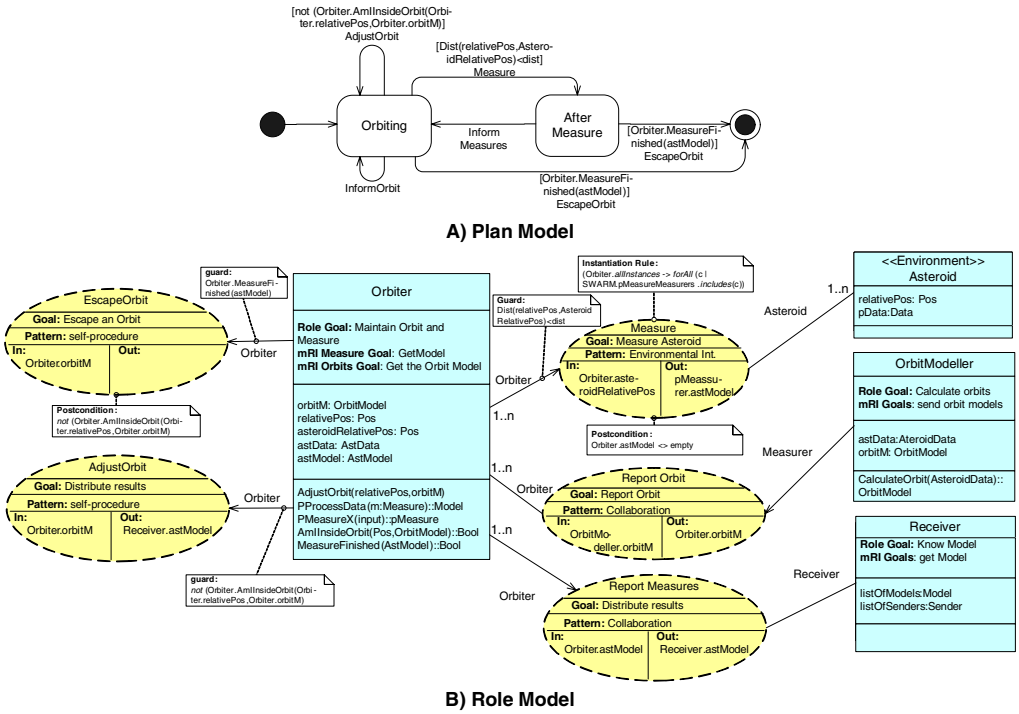

Figure 4. Orbiting and measuring an asteroid autonomous property

portant information. They must also show the system-goal they achieve when executed, the kind of coordination that is carried out when executed, the knowledge used as input to achieve the goal, and the knowledge produced. For example, the goal of the mRI *Report Orbit* is to "Report the Orbit". It is done by taking as input the knowledge of the *OrbitModeler* regarding the orbit and producing as output the model for the orbit (orbitM) in the *Orbiter* role.

Continuing with the example, in Figure 4a, we show the plan model of this role model where the order of execution of all its mRIs is shown. As can be seen, the *Orbiter*, while it is in orbit, is adjusting its orbit and measuring and reporting measures. And when it has completed constructing a model of the asteroid, it escapes the orbit using its knowledge of the orbit model (*orbitM*).

Autonomic properties can be also modeled in this way. As role models can be used at any level of abstraction, we can use them for specifying autonomic properties that concern a single agent, or even a group of agents when dealing with autonomic properties at the swarm level. Thus, as shown in the traceability model, we have a role model at abstraction layer 2 that shows the swarm autonomic behavior, while at layer 4, we show autonomic properties at the level of individual spacecraft.

Here we illustrate a model at abstraction layer 4 for a self-protection autonomic property: protecting from solar storms. The role model for this property is shown in Figure 5b, and, as can be seen, as it is a property at the individual level, a single role is shown (*SelfProtectSpaceCraft*). Its plan model is shown in Figure 5a. As all the spacecraft can be affected by solar storms, this role is applied to all the spacecraft in the swarm, thus adding this autonomic property to all of them.

6 Adding policies to the system

As shown previously, for building and structural organization, used at runtime, we have to compose role models. Since the MaCMAS methodology proposes several methods for composition, we can use them to modify the policies taken into account in the system at runtime or at design-time.

The process for that follows the following steps:

1. Specify the policy using a sub-set of the natural language;
2. Analyze it to find out which role models or interactions, and consequently which autonomic and autonomous properties, are involved in it;
3. Compose these role models, both static and dynamic aspects;

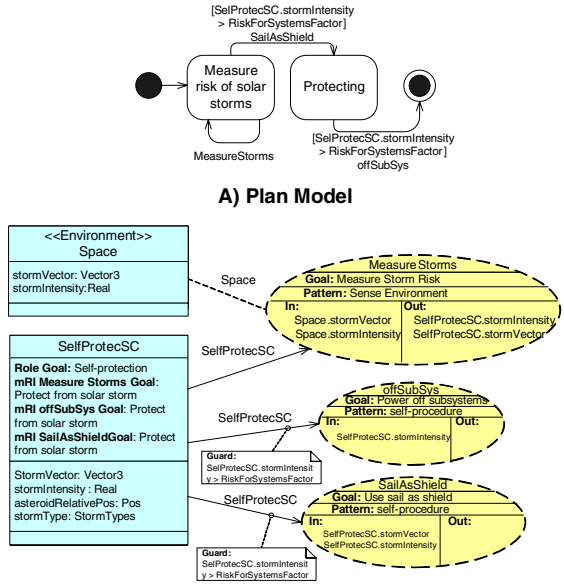

Figure 5. Self-protection from solar storms autonomic property model

4. Deploy the changes in the system using role model composition. That is to say, the running system has a set of role models mapped over its structural organization; thus, adding a new policy consists of composing the current role models with the one that describes the new policy.

We have to take into account that when composing several role models, we can find

emergent roles : roles that appear in the composition yet they do not belong to any of the initial role models;

emergent mRIs : those that are not present in any of the initial role models;

composed roles : the roles in the resultant models that represent several initial roles as a single element;

composed mRIs : mRIs in the resultant model that represents several initial mRIs as a single element;

unchanged roles : those that are left unchanged and imported directly from the initial role models;

unchanged mRIs : those left unchanged and imported directly.

Once relationships between elements have been established by analyzing the policy, we must complete the composite role model. Importing an mRI or a role requires only adding it to the composite role model. The following shows how to compose plans and role models.

6.1 Composing roles

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

1. **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.

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

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

4. **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, as we show below.

5. **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.

6. **Ontologies of an mRI:** The new ontology must cover all the terms described in all the ontologies of roles to be composed (cf. [5, 16, 17]). This procedure also shows how to deal with repeated knowledge in the interface of roles to be composed. That is to say, if as a result of ontology composition, a knowledge entity that is repeated in several roles is shown as the same element in the composed ontology, we can include it once; if it results in different elements in the composed ontology, we must rename them.

6.2 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 this task: extraction of a role plan from the role model

plan and vice versa, and aggregation of several role plans; see [22] 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 superposed 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. [19, 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 [12].

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.

7 Example of applying a new policy to the ANTS case study

We use the following fictitious scenario to document our example: It has been discovered that several spacecraft have collided with an asteroid as a result of self-protection from a solar storm. As a result, it has been decided to avoid protection from solar storms while orbiting, sending the following policy to the system, which is shown graphically in Figure 8.

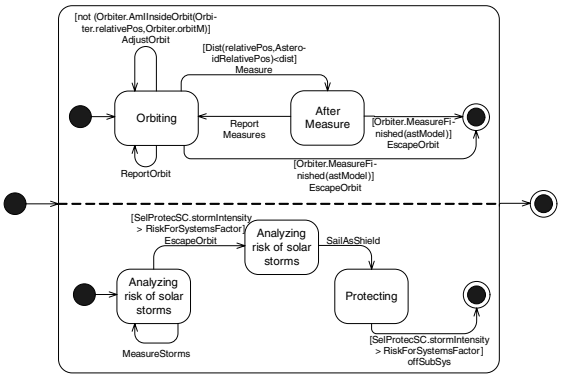

Figure 7. Composed plan

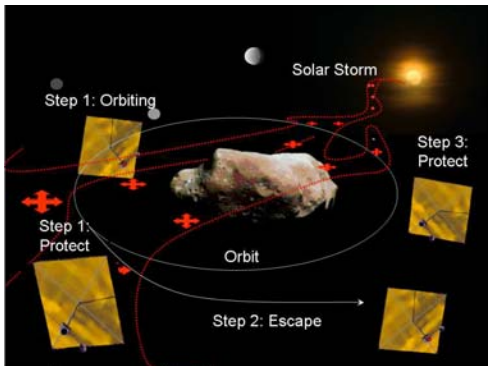

Figure 8. Policy for protecting from solar storms when orbiting

If a spacecraft is orbiting and measuring an asteroid and it determines that there exists risk of a solar storm, the spacecraft must first escape the orbit and later power down subsystems and use its sail as a shield.

Notice that we have limited the policy to two role models to simplify the example, but in the real world we must also take into account the rest of the autonomic properties and associated role models involved in orbiting an asteroid.

The first part of the policy shows the context where it is applied, determining the role models that should be taken into account. Notice that although the second element denotes an interaction, in the traceability diagram we can find out easily the role model it belongs to, namely *Protect from Solar Storms*. The second part shows a modification of the plans where a new order for the interaction is specified.

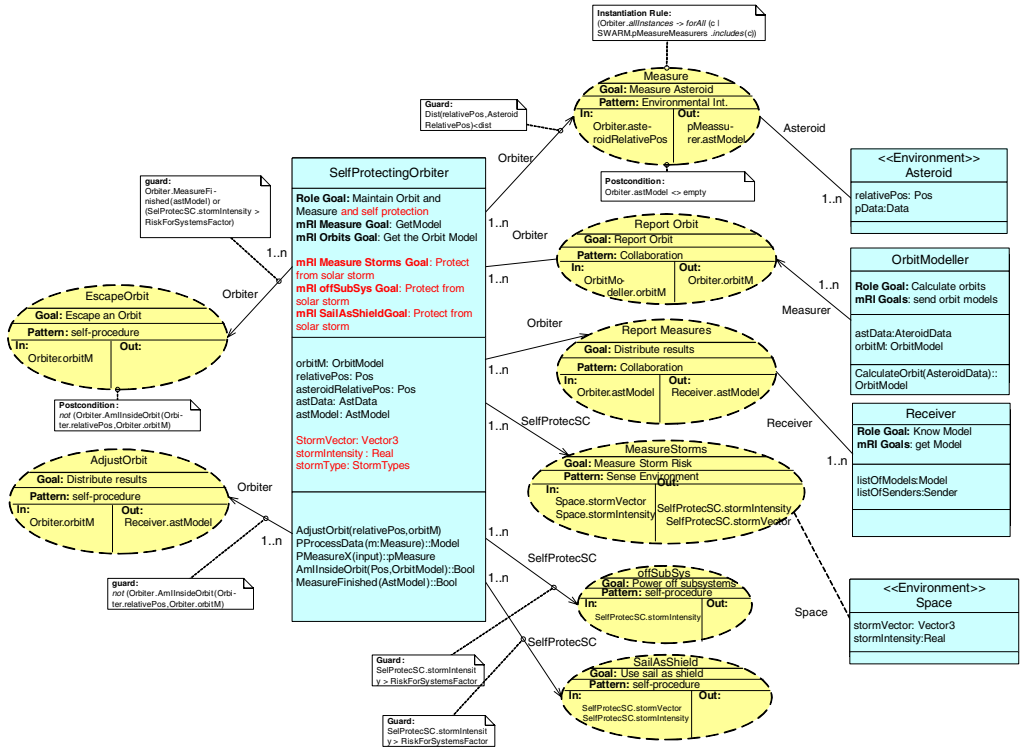

Figure 6. Composed Role Model

If a spacecraft is orbiting and measuring an
Role Model
asteroid and it measures that there exists risk of a solar storm,
Interaction
the spacecraft must *first* escape the orbit and *later*
Interaction
power down subsystems *and* use its sail as a shield
Interaction **Interaction**

As a result, we must compose both models and their plans following the constraints imposed by the policy. The composition of both role models is shown in Figure 6. As we can see, the roles *Orbiter* and *SelfProtectSC* have been composed into a single role called *SelfProtectingOrbiter* following the prescription showm in Section 6.1. 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, as it is shown in Figure 7. Notice also, that the policy tells us the order of mRIs we must follow for self-protection, adding the *Escape Orbit* mRI before protection, which results in the new state machine shown.

8 Conclusions

We have presented an AOSE-based approach for modeling autonomous and autonomic properties of the system. The approach supports models at different levels of abstraction. We also presented a technique for composing these models in order to obtain a particular structural organization. We used this technique to compose those models involved in a new policy and to deploy the resultant model onto the running system.

The main advantage is that, as models are developed at different levels of abstraction, we can specify policies for autonomous and autonomic systems at different levels of abstraction. As these models allow for the abstraction of "intelligent behaviors" since the procedures carried out inside an interaction can be described internally by means of neural networks, fuzzy logic, etc., this allows us to specify policies over these kinds of implementations. Finally, although this approach has not been implemented yet, it seems quite promising and we plan to integrate it into our R2D2C [6], which has been successfully used to specify policies for autonomic systems using constrained natural language [26]. Future work will include the use of ontologies and traceability diagrams, that illustrate constraints on the subset of natural language that we can use.

References

[1] IEEE Task Force on Autonomous and Autonomic Systems, (TFAAS), June 2005. Available at http://www.computer.org/tab.

[2] O. Babaoglu, A. Couch, G. Ganger, P. Stone, M. Yousif, and J. Kephart. Panel: Grand challenges of autonomic computing. In *2nd IEEE International Conference on Autonomic Computing (ICAC'05)*, Seattle, WA, 13-16 June 2005.

[3] S. A. Curtis, W. F. Truszkowski, M. L. Rilee, and P. E. Clark. ANTS for the human exploration and development of space. In *Proc. IEEE Aerospace Conference*, Big Sky, Montana, USA, 9–16 March 2003.

[4] A. Ganek. "autonomic computing: implementing the vision". Keynote presentation at the Autonomic Computing Workshop, AMS'03, seattle, June 2003.

[5] J. Heflin and J. Hendler. Dynamic ontologies on the web. In *AAAI/IAAI*, pages 443–449, 2000.

[6] M. G. Hinchey, J. L. Rash, and C. A. Rouff. Requirements to design to code: Towards a fully formal approach to automatic code generation. Technical Report TM-2005-212774, NASA Goddard Space Flight Center, Greenbelt, MD, USA, 2004.

[7] P. Horn. Autonomic computing: IBM perspective on the state of information technology. In *AGENDA'01*, Scottsdale, AR, 2001, (available at http://www.research.ibm.com/autonomic/).

[8] N. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[9] D. Kaminsky. An introduction to policy for autonomic computing. IBM white paper, March 2005.

[10] E. A. Kendall. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency*, 8(2):34–41, Apr./June 2000.

[11] J. O. Kephart and W. E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *POLICY*, pages 3–12. IEEE Computer Society, 2004.

[12] 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.

[13] L. Lymberopoulos, E. Lupu, and M. Sloman. An adaptive policy-based framework for network services management. *J. Network Syst. Manage.*, 11(3), 2003.

[14] M. Masullo and S. Calo. Policy management: An architecture and approach. In *IEEE First International Workshop on Systems Management*, Los Angeles, CA, April 14-16, 1993.

[15] A. Meissner, S. Musunoori, and L. Wolf. MGMS/GML - towards a new policy specification framework for multicast group integrity. In *SAINT*, pages 233–242. IEEE Computer Society, 2004.

[16] P. Mitra and G. Wiederhold. An ontology-composition algebra. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 93–116. Springer-Verlag, 2004.

[17] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proc. of the 2nd Int. Conf. On Information FUSION'99*, 1999.

[18] 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.

[19] O. M. G. (OMG). Unified modeling language: Superstructure. version 2.0. Final adopted specification ptc/03–08–02, OMG, August 2003. www.omg.org.

[20] 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.

[21] J. Peña. *On Improving The Modelling Of Complex Acquaintance Organisations Of Agents. A Method Fragment For The Analysis Phase*. PhD thesis, University of Seville, 2005.

[22] J. Peña, R. Corchuelo, and J. L. Arjona. Towards Interaction Protocol Operations for Large Multi-agent Systems. In *Proceedings of the Second International Workshop on Formal Approaches to Agent-Based Systems (FAABS 2002)*, volume 2699 of *LNAI*, pages 79–91, NASA-Goddard Space Flight Center, Greenbelt, MD, USA, 2002. Springer–Verlag.

[23] 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.

[24] 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.

[25] R. Sterritt. Towards autonomic computing: Effective event management. In *27th Annual IEEE/NASA Software Engineering Workshop (SEW), IEEE Computer Society Press*, pages 40–47, Maryland, USA, December 3-5 2002.

[26] R. Sterritt, M. Hinchey, J. Rash, W. Truszkowski, C. Rouff, and D. Gracanin. "Towards formal specification and generation of autonomic policies". In *First IFIP Workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES 2005)*. LNCS 3823, Dec 2005.

[27] R. Sterritt, C. A. Rouff, J. L. Rash, W. F. Truszkowski, and M. G. Hinchey. Self-* properties in NASA missions. In *4th International Workshop on System/Software Architectures (IWSSA'05) in Proc. 2005 International Conference on Software Engineering Research and Practice (SERP'05)*, pages 66–72, Las Vegas, Nevada, USA, June 27 2005. CSREA Press.

[28] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, July 2003.