Innovations Syst Softw Eng (2007) 3:17–31
DOI 10.1007/s11334-006-0017-5

ORIGINAL PAPER

Building and implementing policies in autonomous and autonomic systems using MaCMAS

A case study based on a NASA concept mission

Joaquin Peña · Michael G. Hinchey · Roy Sterritt · Antonio Ruiz-Cortés

Received: 15 September 2006 / Accepted: 1 December 2006 / Published online: 1 February 2007

Abstract Autonomic Computing, self-management based on high level guidance from humans, is increasingly being accepted as a means forward in designing reliable systems that both hide complexity from the user and control IT management costs. Effectively, AC may be viewed as policy-based self-management. We look at ways of achieving this, with particular focus on agent-oriented software engineering. We propose utilizing MaCMAS, an AOSE methodology for specifying autonomic and autonomous properties of the system independently. Later, by means of composition of these specifications, guided by a policy specification, we construct a specification for the policy and its subsequent deployment. We illustrate this by means of a case study based on a NASA concept mission and describe future work on a support toolkit.

Keywords Autonomic computing · Policy-based management · Agent-oriented software engineering

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472) and grant TIC2003-02737-C02-01, by NASA Software Engineering Laboratory and NASA office of Safety and Mission Assurance Software Assurance Research Program (SARP), and at University of Ulster by the Computer Science Research Institute (CSRI) and the Centre for Software Process Technologies (CSPT), funded by Invest NI through the Centres of Excellence Programme, under the EU Peace II initiative.

J. Peña · A. Ruiz-Cortés
University of Seville, Seville, Spain
e-mail: joaquinp@us.es

A. Ruiz-Cortés
e-mail: aruiz@us.es

M. G. Hinchey
Loyola College, Baltimore, MD, USA
e-mail: mike.hinchey@usa.net

R. Sterritt (✉)
University of Ulster, Ulster, Northern Ireland
e-mail: r.sterritt@ulster.ac.uk

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, who together with the component being managed make up an autonomic element, be defined such that they can exist in a collaborative autonomic environment and ultimately provide self-management of the system.

The long term strategic vision of Autonomic Computing (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 [8]. 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 [8,28]. In his talk, Horn highlighted that the autonomic computing system must "find and generate rules for how best to interact with neighboring systems" [8].

We propose to use a methodology called MaCMAS (methodology fragment for analyzing complex multi-agent systems,[1]) which provides the models and techniques for adding policies at runtime. We propose creating 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 Sect. 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.

2 Policy-based management

Policies have been described as a set of considerations designed to guide decisions of courses of action [15], 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,[2] for instance, where a (human) manager may simply specify the business objectives and the system will achieve these in terms of the needed ICT [14]. 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-s 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." [10]. 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 [16]. 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 [12] 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 [29].

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 [9].

[1] see http://www.james.eii.us.es/MaCMAS/ for further details.

[2] Whatis.com, online computer and internet dictionary and encyclopedia, 2005.

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., [19,21,31].

A MAS organization can be observed from two different point of view [31]:

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 [31]. Thus, we can conclude that any acquaintance organization can be modeled orthogonally to its structural organization [11].

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 [22]. It is specially tailored to model complex acquaintance organizations [26]. 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 provide 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 are 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 are 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, a composition of roles.

In Fig. 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) [23,24]. 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

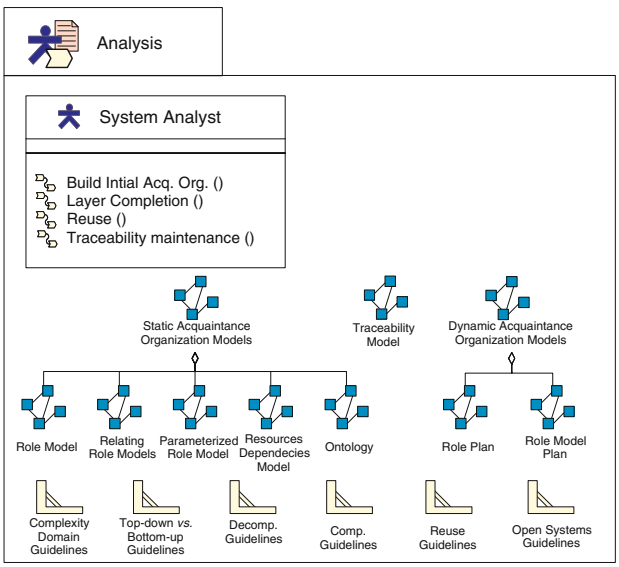

Fig. 1 Acquaintance analysis discipline

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. Sect. 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 [20, 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: These show 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: This 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 [20, 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 [20, 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 relationship 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 [3,30] is a concept mission that involves the

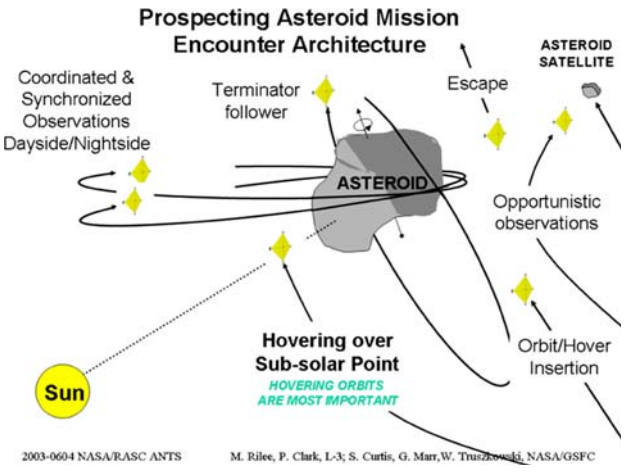

Fig. 2 ANTS encounter with an asteroid

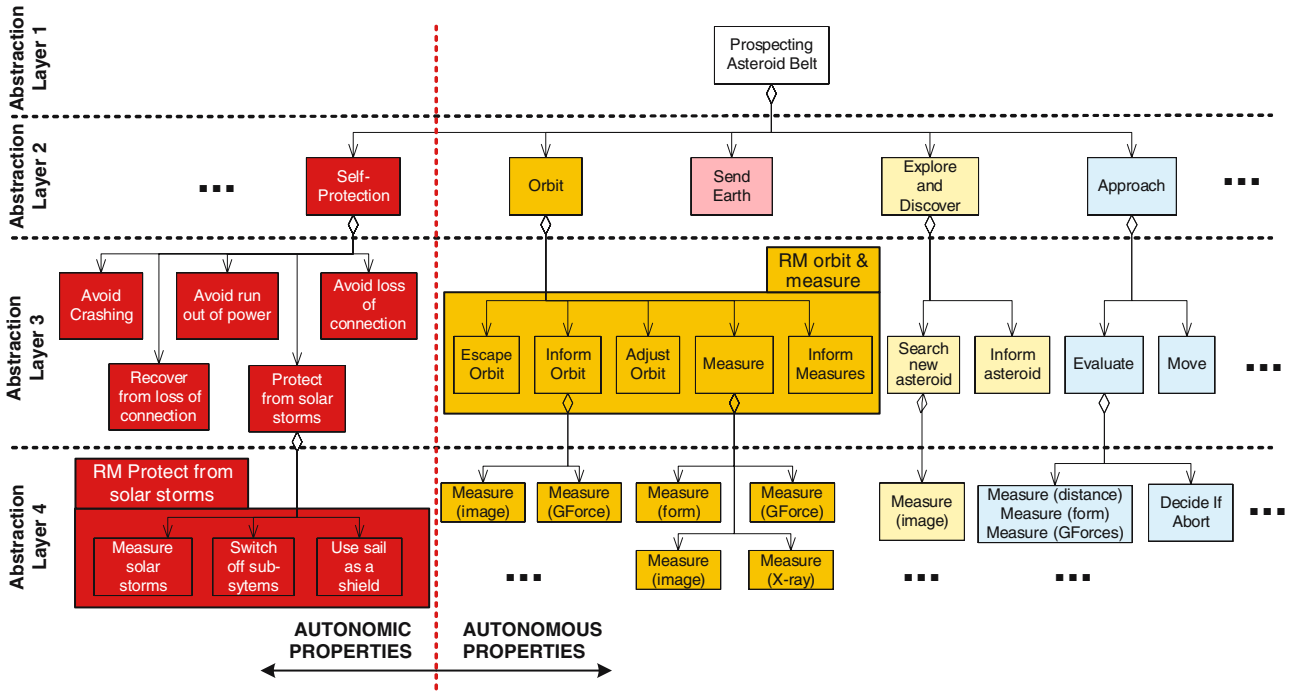

Fig. 3 Traceability model of ANTS

use of swarms of autonomous pico-class (approximately 1 kg) 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 Fig. 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, self-healing, self-optimization and self-protection, as illustrated in [30]. 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

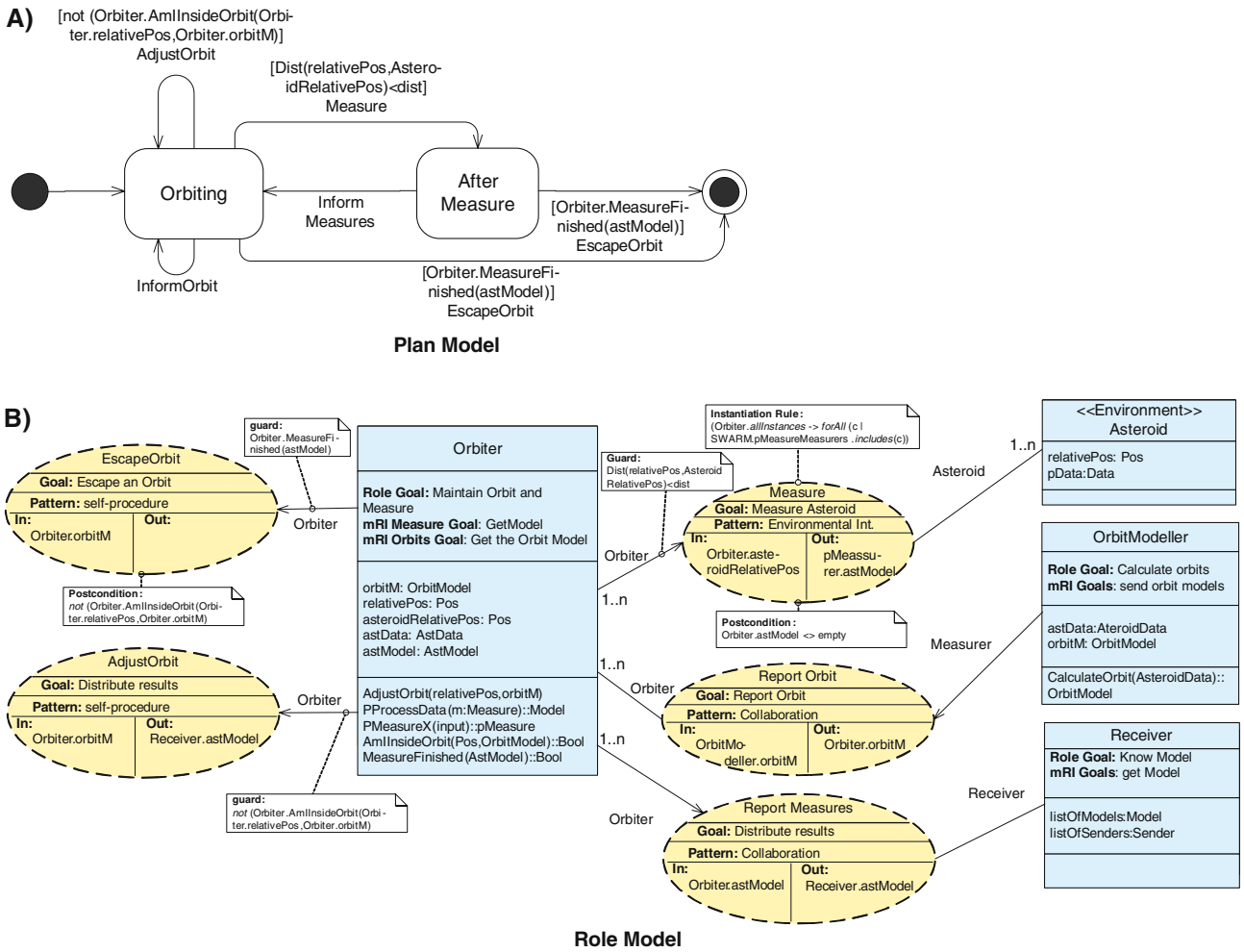

Fig. 4 Orbiting and measuring an asteroid autonomous property

potential mechanism might, for example, provide spacecraft with a solar storm sensing capability through onboard, 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 Fig. 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 Fig. 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-orga-

nization. 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 Fig. 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* is to "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 important 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 Fig. 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

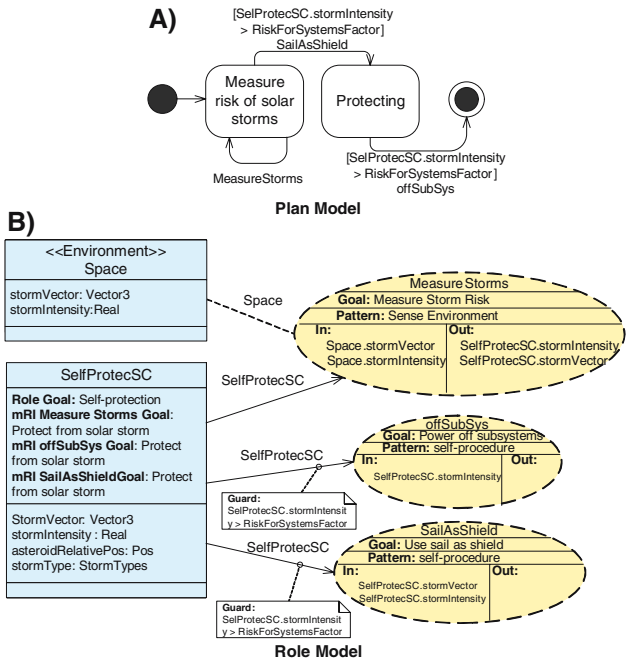

Fig. 5 Self-protection from solar storms autonomic property model

in Fig. 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 Fig. 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 a structural organizations, 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.

As shown in Fig. 6, the process is illustrated in the following steps:

1. **Specify policy:** Specify the policy using a sub-set of the natural language and the acquaintance models available.
2. **Determine involved models:** Analyze it to find out which role models or interactions, and consequently which autonomic and autonomous properties, are involved in it.
3. **Compose roles and plans:** Compose these role models, both static and dynamic aspects (Fig. 7 and Fig. 8).

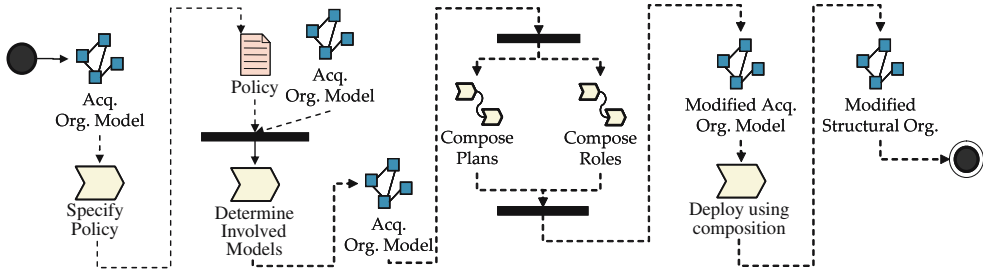

Fig. 6 Software process overview

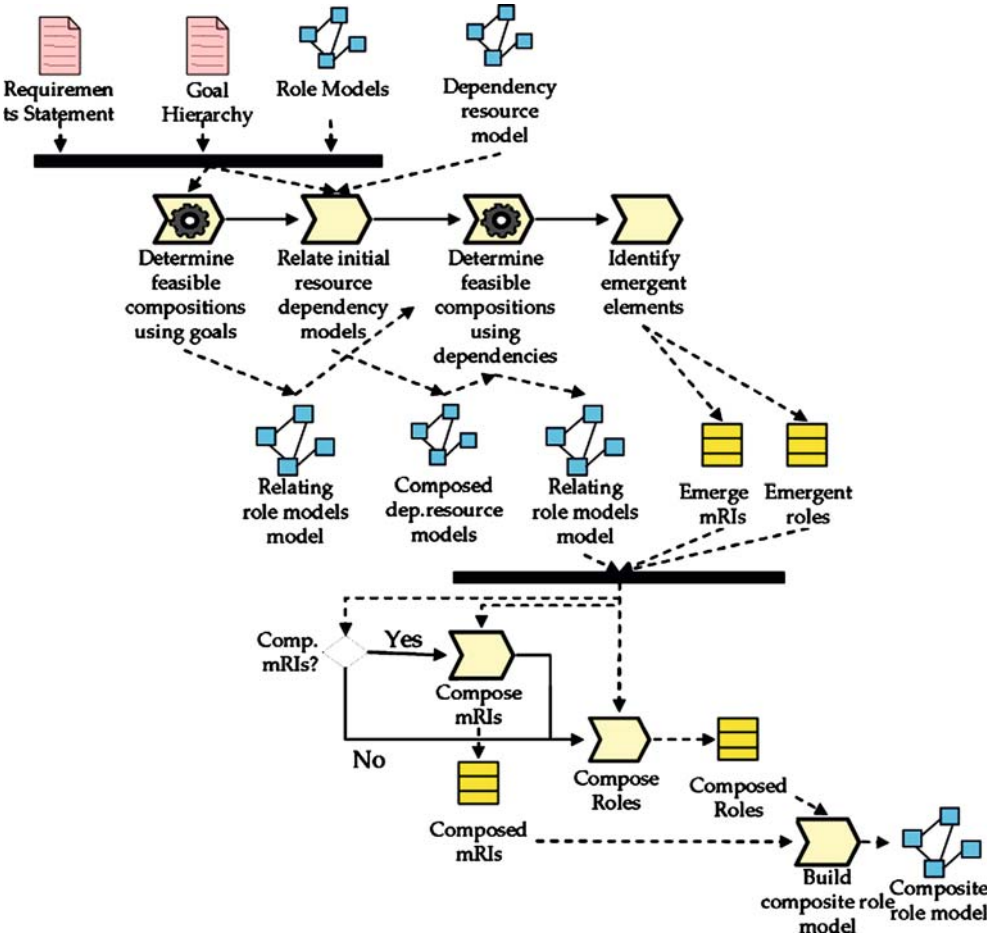

Fig. 7 Software process of role composition

4. **Deploy using composition:** 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 find the following:

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;

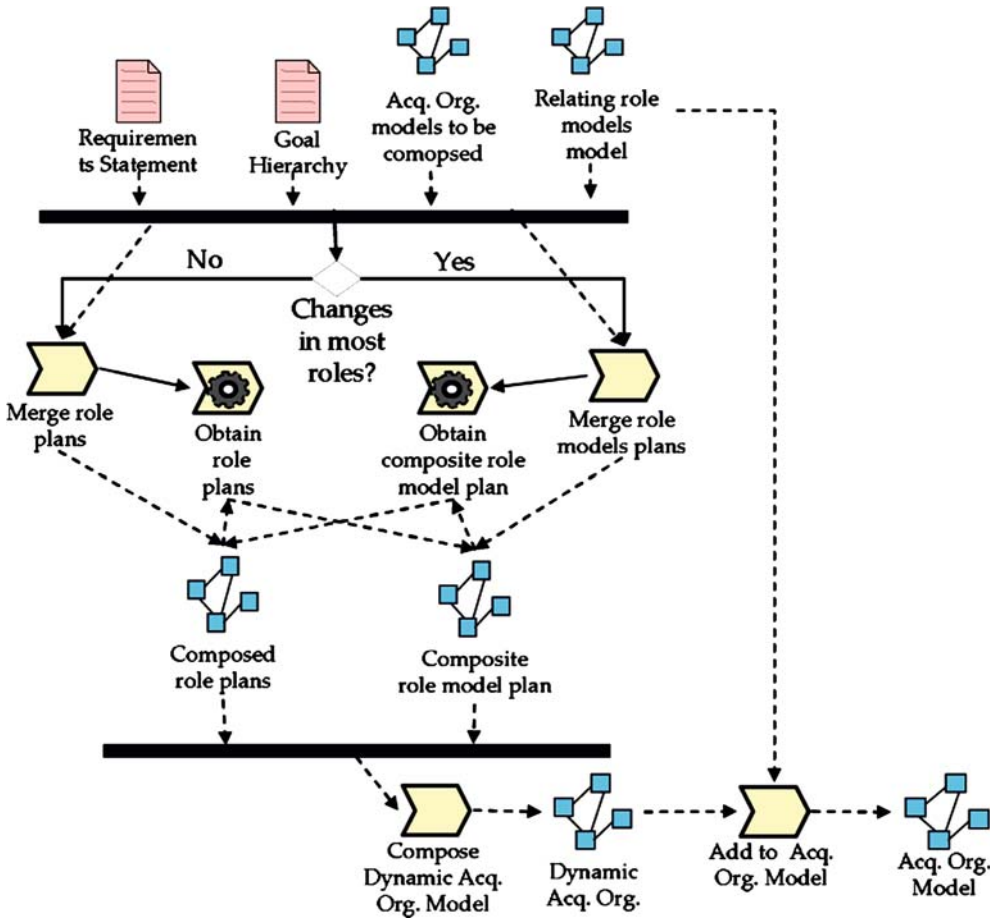

Fig. 8 Software process of plan composition

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 to abstract 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. [5,17,18]). This procedure also

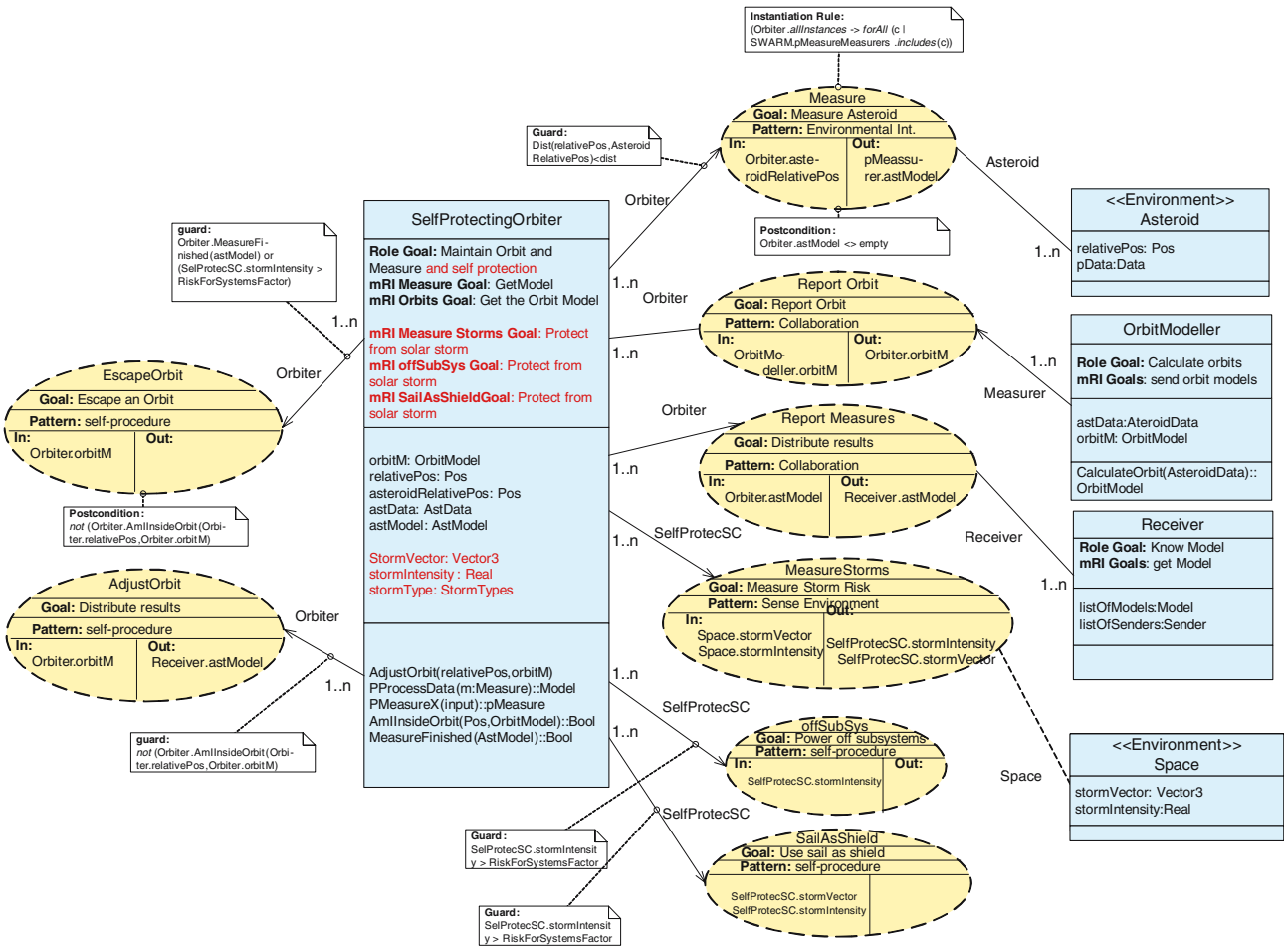

Fig. 9 Composed role model

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 [23] 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. [20, 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 pre-

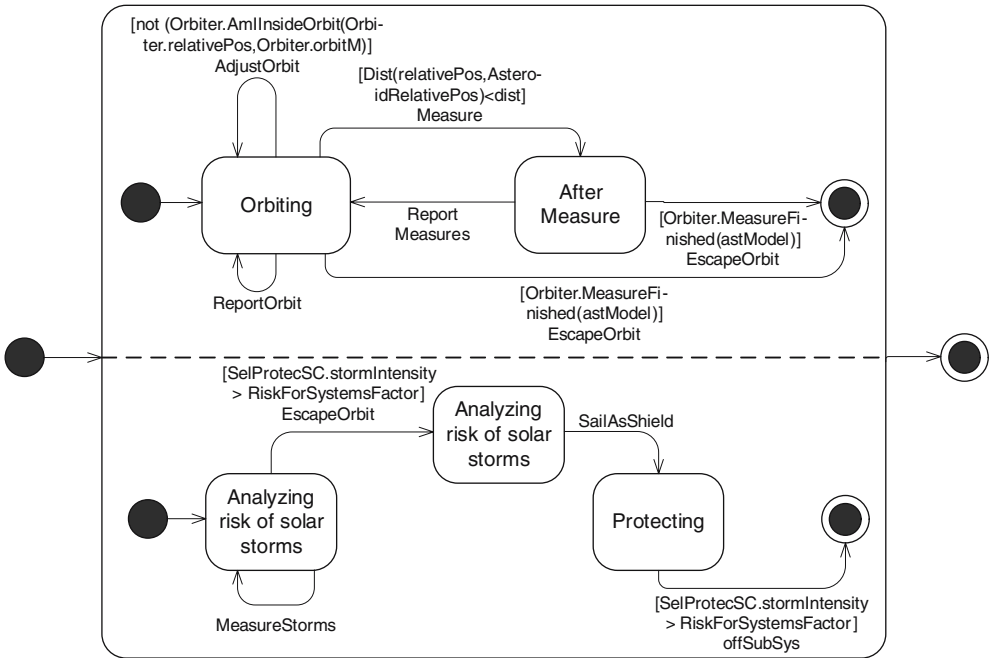

Fig. 10 Composed plan

served, since to ensure this property, the composed plan must inherit from all the initial plans [13].

As depicted in Fig. 8, the composition of role model plans has to be performed following one of the plan composition techniques described previously to later, if 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.

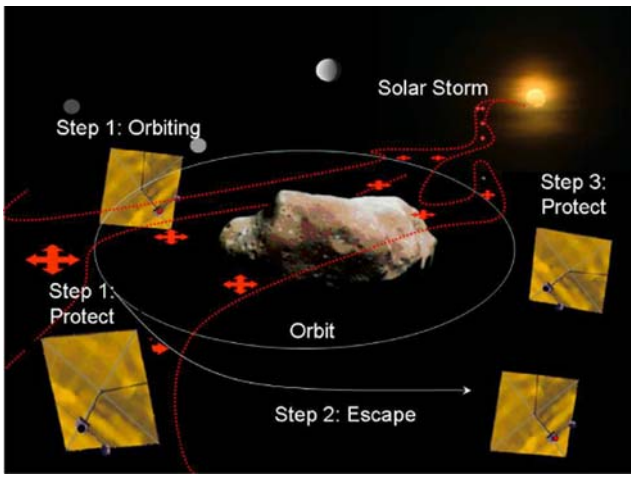

Fig. 11 Policy for protecting from solar storms when orbiting

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 Fig. 11.

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

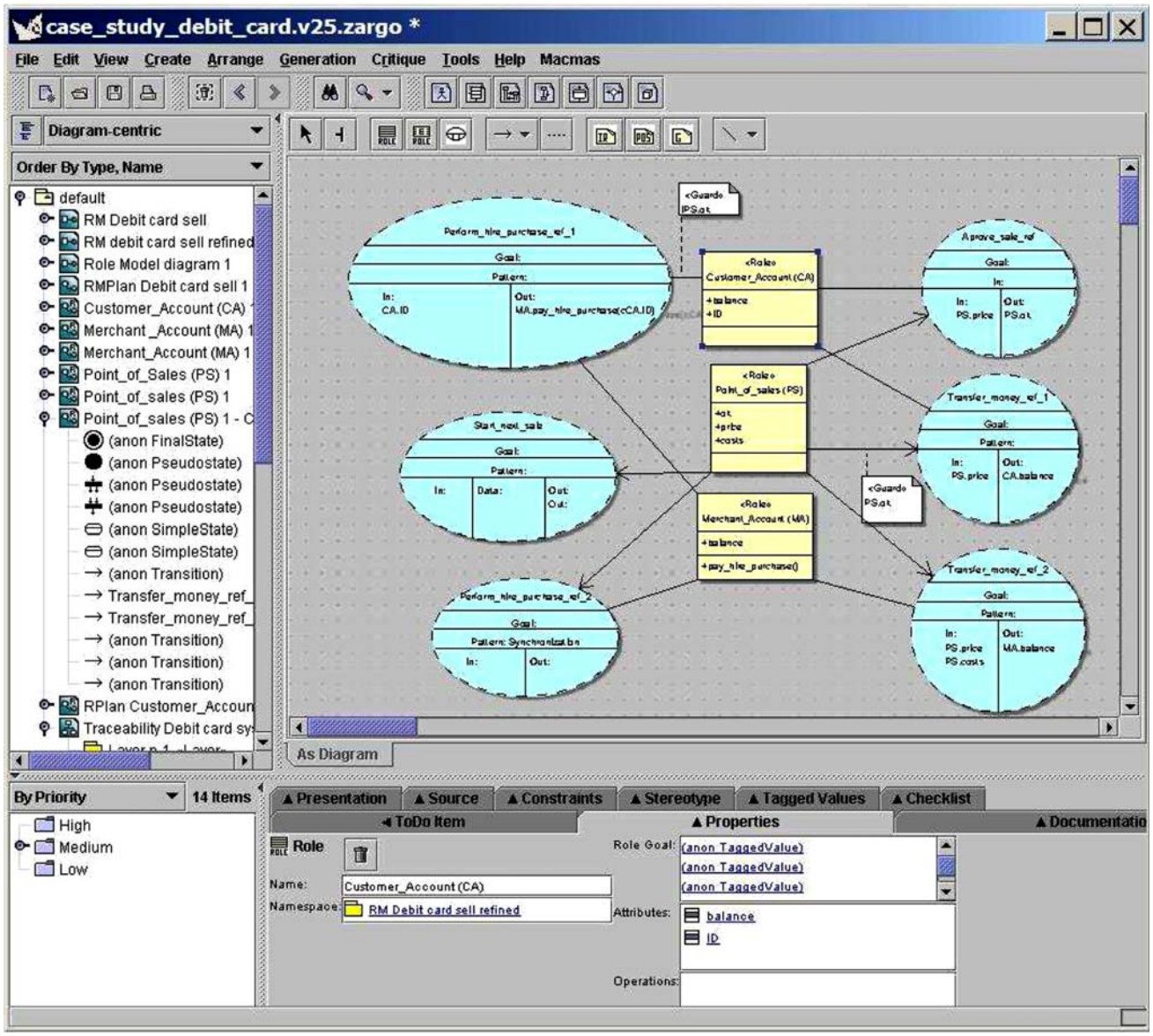

Fig. 12 MaCMAS screenshot

shows a modification of the plans where a new order for the interaction is specified.

If a spacecraft is orbiting and measuring an
Role Model
asteroid and it measures that there exists risk ofa 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 Fig. 9. As we can see, the roles *Orbiter* and *SelfProtectSC* have been composed into a single role called *SelfProtectingOrbiter* following the prescription shown in Sect. 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 Fig. 10. Notice also, that the policy tells us the order of mRIs we must follow for self-protection, adding the *escape orbit*

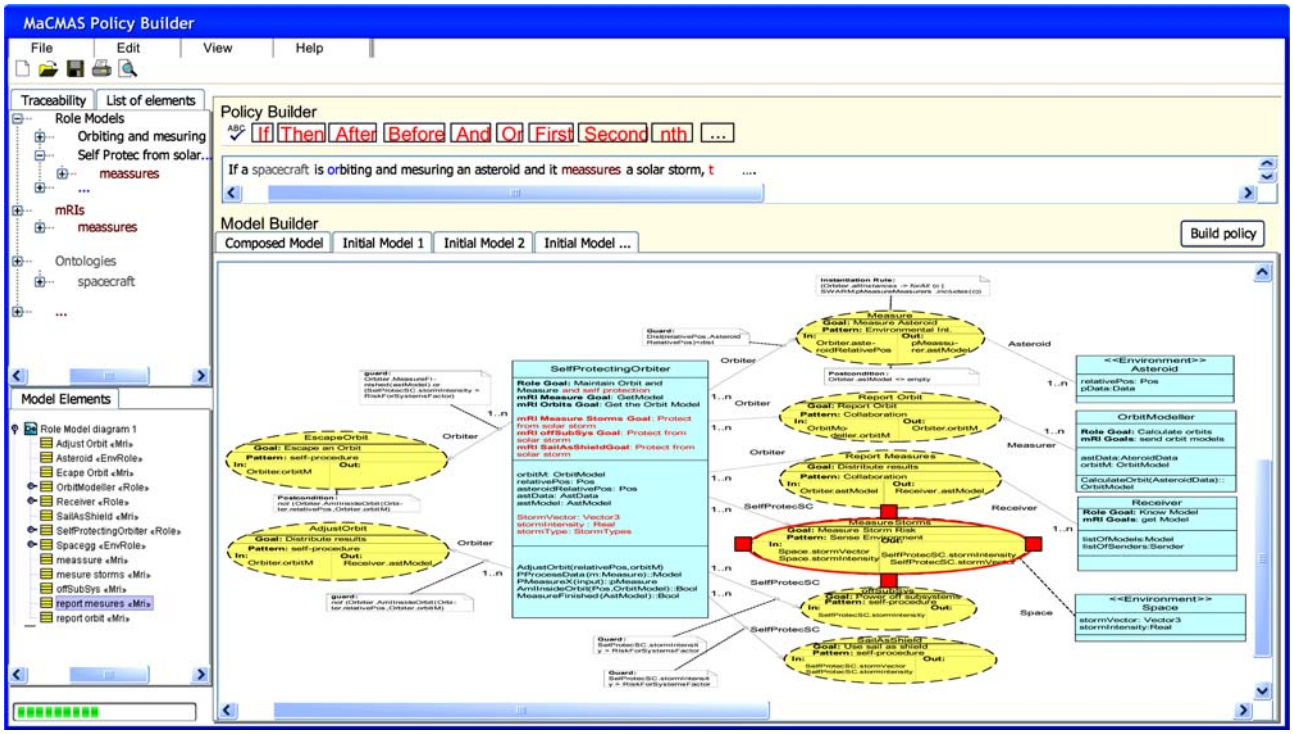

Fig. 13 Policy builder tool schema

mRI before protection, which results in the new state machine shown.

8 Future work: implementing the policy manager

Although this work is still ongoing, since the publication of our previous paper, i.e., [27], we have implemented part of the planned tool. This is based on two existing tools: (i) an ArgoUML extension used to model MaCMAS diagrams, called the MaCMAS CASE tool (MaCMAS-CT), and (ii) a NASA proprietary prototype tool called R2D2C, which allows for policies to be specified in natural language (or a variety of other input notations) and then generates a provably-correct corresponding specification (currently in Hoare's language of communicating sequential processes, or CSP [7], although other formal languages may be used) that can be checked and analyzed.

In Fig. 14, we have sketched how our final tool might look. As can be seen, the tool will provide information about the vocabulary that can be used in the policy. This vocabulary is composed of the role models, mRIs, etc. in the MaCMAS model of the system, along with some temporal, conditional, etc., operators, e.g., *and*, *or*, *if*, etc. It will also provide models of the policy so that the user can see if the results obtained are correct. This also allows the user to change the obtained models as needed and to keep track of these changes in the policy specification.

Both tools have been developed separately: MaCMAS-CT at the University of Seville, and R2D2C in the NASA Software Engineering Laboratory in collaboration with researchers from Virginia Tech and SAIC.

MaCMAS-CT has been implemented over ArgoUML, defining an UML profile that extends UML with a set of stereotypes, e.g., *<<role>>*, and tagged values, e.g., *RoleGoal*. We have also extended this tool to provide the graphical representation of MaCMAS models. In Fig. 12, we show a screenshot of this tool.

R2D2C is a tool that allows for the specification requirements in natural language, use cases, and other parseable notations. From them, it produces a CSP model which can be analyzed, adapted, corrected, etc., reflecting changes back in the use cases and/or natural language, or other input notations. It can be also used to as a basis for sound code generation [6].

In addition, it has been successfully used to specify policies for autonomic systems using constrained natural language [29]. In R2D2C, policies are viewed as scenarios (just as in the example of this paper), and are used to generate a CSP model that is guaranteed to be equivalent, and which can then be checked for various issues of consistency and completeness, along

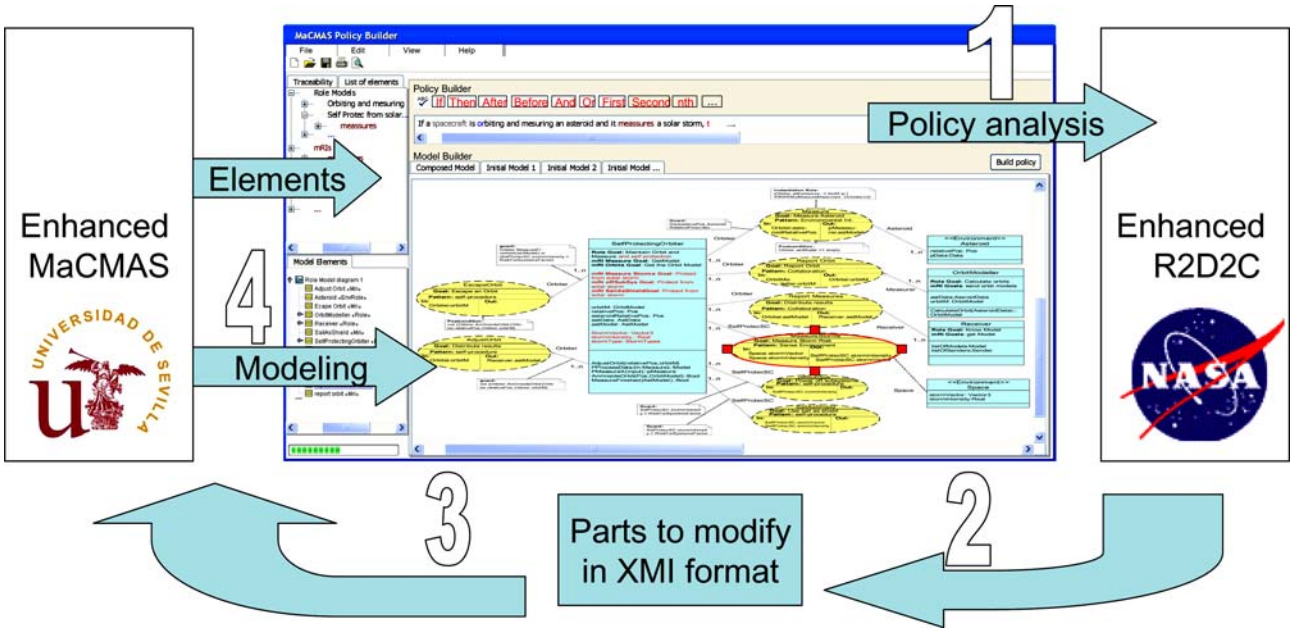

Fig. 14 Policy builder tool working schema

with other problems, which may then be addressed and rectified.

In Fig. 14, we show how we plan to integrate both tools, which will required for future enhancement of both tools. As illustrated in the figure, ArgoUML allows us to store models using XMI, an OMG standard based on XML that is primary used for exchanging UML models, that we plan to use as the exchange format between MaCMAS-CT and R2D2C. Thus, we plan to enhance R2D2C in order to accept input as MaCMAS models represented in XMI. This provides the tool with the capability to check properties and to generate code. Another point of integration between R2D2C and the planned tool will consist of integrating the policy parser provided by R2D2C, limiting the vocabulary to the elements present in the XMI model. Finally, MaCMAS-CT will also need an extension to support the composition of roles and plans automatically. In this sense, we have obtained some results applying MDA [25].

Thus the usual steps performed by the planned tool will be the following:

1. generate the Acquaintance model using MaCMAS-CT;
2. produce an XMI equivalent of this model using MaCMAS-CT that will be accepted by R2D2C in order to obtain the names of role models and mRIs that we allow in the textual specification;
3. express the policy in the graphical interface of the tool;
4. R2D2C will analyze the textual description and converts it to an XMI specification with the models that have to be composed and whose elements will be imported, composed or deleted modifying the XMI of our system;
5. MaCMAS will perform the composition when needed and import/delete the elements prescribed;
6. allow the user to manually perform changes reflecting these changes in the policy;
7. R2D2C will take the last XMI file and the policy in order to generate the required code.

9 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 have also presented a technique for composing these models in order to obtain a particular structural organization. We have used this technique to compose those models involved in a new policy and to deploy the resultant model on the running system. We have also shown our first results regarding the implementation of a CASE tool that implements the techniques described in this paper.

The main advantage of this approach 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. Thus, a human designer, not expert in the details of the implementation, may easily modify the system while maintaining its integrity.

References

1. IEEE Task Force on Autonomous and Autonomic Systems (TFAAS) (2005) Available at http://www.computer.org/tab
2. Babaoglu O, Couch A, Ganger G, Stone P, Yousif M, Kephart J (2005) Panel: grand challenges of autonomic computing. In: 2nd IEEE international conference on autonomic computing (ICAC'05), Seattle, WA, 13–16 June 2005
3. Curtis SA, Truszkowski WF, Rilee ML, Clark PE (2003) ANTS for the human exploration and development of space. In: Proc IEEE aerospace conference, Big Sky, Montana, 9–16 March 2003
4. Ganek AG (2003) Autonomic computing: implementing the vision. Keynote presentation at the Autonomic Computing Workshop, AMS'03, Seattle, WA
5. Heflin J, Hendler J (2000) Dynamic ontologies on the web. In: AAAI/IAAI, pp 443–449
6. Hinchey MG, Rash JL, Rouff CA (2004) 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
7. Hoare CAR (1980) Communicating sequential processes. In: McKeag RM, Macnaghten AM (eds) On the construction of programs — an advanced course. Cambridge University Press, pp 229–254
8. Horn P (2001) Autonomic computing: IBM perspective on the state of information technology. In: Scottsdale AR (eds) AGENDA'01 (available at http://www.research.ibm.com/autonomic/)
9. Jennings N (2001) An agent-based approach for building complex software systems. Commun ACM 44(4):35–41
10. Kaminsky D (2005) An introduction to policy for autonomic computing. white paper, IBM
11. Kendall EA (2000) Role modeling for agent system analysis, design, and implementation. IEEE Concurrency 8(2):34–41
12. Kephart JO, Walsh WE (2004) An artificial intelligence perspective on autonomic computing policies. In: POLICY. IEEE Computer Society, pp 3–12
13. Liskov B, Wing JM (1993) Specifications and their use in defining subtypes. In: Proceedings of the eighth annual conference on object-oriented programming systems, languages, and applications. ACM Press, pp 16–28
14. Lymberopoulos L, Lupu E, Sloman M (2003) An adaptive policy-based framework for network services management. J Network Syst Manage 11(3)
15. Masullo MJ, Calo SB (1993) Policy management: An architecture and approach. In: IEEE first international workshop on systems management, Los Angeles, CA, April 14–16
16. Meissner A, Musunoori SB, Wolf LC (2004) MGMS/GML — towards a new policy specification framework for multicast group integrity. In: SAINT. IEEE Computer Society, pp 233–242
17. Mitra P, Wiederhold G (2004) An ontology-composition algebra. In: Staab S, Studer R (eds) Handbook on ontologies, International Handbooks on Information Systems. Springer, Heidelberg, pp 93–116
18. Mitra P, Wiederhold G, Jannink J (1999) Semi-automatic integration of knowledge sources. In: Proc of the 2nd Int Conf on Information FUSION'99
19. Odell J, Parunak H, Fleischer M (2003) The role of roles in designing effective agent organisations. In: Garcia A, Lucenaand C, Zambonelliand F, Omiciniand A, Castro J (eds) Software engineering for large-scale multi-agent systems. LNCS, vol 2603. Springer, Heidelberg, pp 27–28
20. Object Management Group (OMG) (2003) Unified modeling language: Superstructure. version 2.0. Final adopted specification ptc/03–08–02, OMG http://www.omg.org
21. Van Dyke Parunak H, Odell J (2001) Representing social structures in UML. In: Müller JP, Andre E, Sen S, Frasson C (eds) Proceedings of the fifth international conference on autonomous agents, Montreal. ACM Press, pp 100–101
22. Peña J (2005) On improving the modelling of complex acquaintance organisations of agents. A method fragment for the analysis phase. PhD thesis, University of Seville
23. Peña J, Corchuelo R, Arjona JL (2002) Towards interaction protocol operations for large multi-agent systems. In: Proceedings of the 2nd Int workshop on formal approaches to agent-based systems (FAABS 2002), LNAI, vol 2699, NASA-GSFC, Greenbelt. Springer, Heidelberg, pp 79–91
24. Peña J, Corchuelo R, Arjona JL (2003) A top down approach for MAS protocol descriptions. In: ACM symposium on applied computing SAC'03, Melbourne, ACM Press, pp 45–49
25. Peña J, Hinchey MG, Sterritt R, Ruiz-Cortés A, Resinas M (2006) A model-driven architecture approach for modeling, specifying and deploying policies in autonomous and autonomic systems. In: Second international symposium on dependable autonomic and secure computing (DASC 2006), 29 September – 1 October 2006, Indianapolis. IEEE Computer Society, pp 19–30
26. Peña J, Levy R, Corchuelo R (2005) Towards clarifying the importance of interactions in agent-oriented software engineering. Int Iberoamerican J AI 9(25):19–28
27. Pena J, Hinchey MG, Sterritt R (2006) Towards modeling, specifying and deploying policies in autonomous and autonomic systems using an AOSE methodology. In: EASE '06: proceedings of the third IEEE international workshop on engineering of autonomic and autonomous systems (EASE'06), Washington. IEEE Computer Society, pp 37–46
28. Sterritt R (2002) Towards autonomic computing: Effective event management. In: 27th Annual IEEE/NASA Software Engineering Workshop (SEW), Maryland, December 3–5. IEEE Computer Society Press, pp 40–47
29. Sterritt R, Hinchey MG, Rash J, Truszkowski W, Rouff C, Gracanin D (2005) Towards formal specification and generation of autonomic policies. In: First IFIP workshop on trusted and autonomic ubiquitous and embedded systems (TAUES 2005). LNCS, vol 3823
30. Sterritt R, Rouff CA, Hinchey MG, Rash JL, Truszkowski WF (2006) Next generation system and software architectures: challenges from future NASA space exploration missions. J Sci Comput Program 61(1):48–57
31. Zambonelli F, Jennings N, Wooldridge M (2003) Developing multiagent systems: the GAIA methodology. ACM Trans Software Eng Methodol 12(3):317–370