Journal of Software Systems and Modeling manuscript No.
(will be inserted by the editor)

Modeling NASA Swarm-Based Systems

Using Formal Methods and Agent-Oriented Software Engineering

Christopher A. Rouff[1], **Joaquin Peña**[2], **Michael G. Hinchey**[3], **Antonio Ruiz-Cortés**[2*]

[1] CHRISTOPHER.A.ROUFF@saic.com
Science Applications International Corporation (SAIC)
McLean, VA, USA
[2] {joaquinp,aruiz}@us.es
University of Seville
Seville, Spain
[3] Michael.G.Hinchey@nasa.gov
NASA Goddard Space Flight Center
Greenbelt, MD, USA

Received: date / Revised version: date

Abstract NASA is researching advanced technologies for future exploration missions using intelligent swarms of robotic vehicles. One of these missions is the Autonomous Nano Technology Swarm (ANTS) mission that will explore the asteroid belt using 1,000 cooperative autonomous spacecraft.

From the engineering point of view, the complexity of this kind of systems is one of the main challenges that has to be overcame, since it makes the behavior of the swarm unpredictable. In NASA, many approaches are being explored towards this goal, mainly, a tailored software engineering approach for this kind of systems, called agent-oriented software engineering, and formal methods. In this paper, we report on the main advances we have done towards modelling, implementing, and testing NASA swarms-based concept missions.

1 Introduction

NASA is investigating new paradigms for future space exploration, heavily focused on the (still) emerging technologies of autonomous and autonomic systems [66, 67]. Traditional missions, reliant on one large spacecraft, are being replaced with missions that involve smaller collaborating spacecraft, analogous to swarms in nature [15]. This approach offers several advantages: the ability to send spacecraft to explore regions of space where traditional craft simply would be impractical, greater redundancy (and, consequently, greater protection of assets), and reduced costs and risk, to name but a few. Concept swarm missions entail the use of unmanned autonomous vehicles (UAVs) flying approximately one meter above the surface of Mars, which will cover as much of the surface of Mars in three seconds as the now famous Mars rovers did in their entire time on the planet; the use of armies of tetrahedral walkers to explore the Mars and Lunar surface [13]; constellations of satellites flying in formation; and, the use of miniaturized spacecraft to explore the asteroid belt, where heretofore it has been impossible to send exploration craft without the high likelihood of loss [15].

These new approaches to exploration simultaneously pose many challenges. The missions will be unmanned and highly autonomous. They will also exhibit the properties of autonomic systems, being self-protecting, self-healing, self-configuring, and self-optimizing. Many of these missions will be sent to parts of the solar system where manned missions are simply not possible, and to where the round-trip delay for communications to spacecraft exceeds 40 minutes, meaning that the decisions on responses to exploration opportunities as well as problems and undesirable situations must be made *in situ* rather than from ground control on Earth.

Swarms [3, 4] consist of a large number of simple agents that have local interactions (between each other and the environment). There is no central controller directing the swarm and no one agent has a global view;

* The work reported in this article was supported by the Spanish Ministry of Science and Technology under grants TIC2003-02737-C02-01 and TIN2006-00472, by the NASA Software Engineering Laboratory, NASA Goddard Space Flight Center, Greenbelt, MD, USA, and by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) and managed by the NASA Independent Verification and Validation (IV&V) Facility.

they are self-organizing based on the emergent behaviors of the simple interactions. This type of behavior is observed in insects and flocks of birds. Bonabeau et al. [7], who studied self-organization in social insects, state that "complex collective behaviors may emerge from interactions among individuals that exhibit simple behaviors" and describe emergent behavior as "a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions among its lower-level components." The emergent behavior is sometimes referred to as the macroscopic behavior, also called macro-level behavior, and the individual behavior and local interactions as the microscopic behavior, also called micro-level behavior. Though swarm behaviors are the combination of often simple individual behaviors, when aggregated, they can form complex and often unexpected behaviors.

Agent swarms are often used as a modeling technique and as a tool to study complex systems [22]. In swarm simulations, a group of interacting agents [68] (often homogeneous or near homogeneous agents) is studied for their emergent behavior. Examples of the use of swarm simulations includes studying flocks of birds [11, 53], business and economics [35], and ecological systems [62]. In swarm simulations, each of the agents is given certain parameters that it tries to maximize. In terms of bird swarms, each bird tries to find another bird to fly with, and then fly off to one side and slightly higher to reduce its drag. Eventually the birds form flocks. Swarms are also being investigated for use in applications such as telephone switching, network routing, data categorizing, and shortest path optimizations [6].

Intelligent swarm technology is where the individual members of a swarm also exhibit intelligence [5,6]. With intelligent swarms, members may be heterogeneous or homogeneous. Even if members start out as homogeneous, due to their differing environments they may learn different things, develop different goals, and thereby become a heterogeneous swarm. Intelligent swarms may also from the beginning be made up of heterogeneous elements, such as the NASA concept mission described below, reflecting different capabilities as well as a possible social structure.

However, although this kind of systems present many advantages since complex behavior emerge from the definition of simple individuals behaviors, reducing the effort on design, its unpredictability represents a problem since unexpected behaviors may cause the failure of a mission that cost million of dollars.

In this paper, we survey on the current NASA efforts towards benefiting from intelligent swarm-based systems at the time that the macro-level behavior of the missions is controlled and ensured to be running into a safe operational space. For this purpose, the main approaches that are being explored can be summarized in two: Agent-Oriented Software Engineering (AOSE) and Formal Methods.

2 Problems of modeling swarm-based systems

Complexity is one of the main problems of engineering software systems, that is specially relevant when tackling the development of swarm-based missions.

Several authors agree that the complexity is a consequence of interactions [32,39]: *Complexity is caused by the collective behavior of many basic interacting agents.* In fact, many authors point out that the complexity is the consequence of those interactions among agents, and that these interactions can vary at execution time, and cannot be predicted thoroughly at design time, namely, emergent behavior. The reasons for the emergence can be traced to two features present in these systems: self-adaptation, and self-organization [19, page 20–21][32, 39]. It is important to observe that this capability of demonstrating emergent behavior is the key factor that drove us to implement swarm-based solutions in the first place, since this key capability is essential to address solutions to the targeted domains.

In Ref. [32],Jennings adapts to agency the three main principles to manage complexity proposed by Booch in the OO context [8]: Abstraction, Decomposition and Organization/Hierarchy[1]:

Abstraction: is based on defining simplified models of the systems that emphasize some details avoiding others. It is interesting since it limits the designer scope of interest and the attention can be focused on the most important details at a given time.

Decomposition: is based on the principle "divide and conquer". It helps to limit the designer scope to a portion of the problem.

Composition: consists in identifying and managing the inter-relationships between the various subsystems in the problem. It makes it possible to group together various basic agents or organizations and treat them as higher-level units of analysis. It also provides means of describing the high-level relationships between several units.

In addition, automation and reuse have been presented as two important principles to overcome complexity [18,33]:

Automation: Automating the modeling process results in lower complexity of models and reduces effort and errors. Some procedures must definitely be carried out based on the judgment of the human modeler. However, some steps can be performed using automatic techniques to transform models which can be carried out by a software tool.

Reuse: Reuse is based on using previous knowledge in designing systems. It saves modelers from redesigning some parts of the system and avoids errors, thus achieving lower complexity of models. Reuse involves

[1] Notice that hereafter we call it *Composition* in order to differentiate it from the organization term in AOSE

processes, modeling artifacts, techniques, guidelines, management processes, models of previous projects, etc.

2.1 AOSE and complexity

From the engineering point of view, these principles imply a set of requirements for AOSE methodologies at the three dimensions of a methodology: modeling artifacts, software process, and techniques to manage models. The main requirements are [52]:

In the modelling artifacts dimension, the first element we must considers are roles, since this is the concept that allows us to focus on interactions, the main source of complexity. In addition, roles also allows us to decompose an agent by its responsibilities, which represent the decomposition principle.

In addition to roles, artifacts for abstracting interactions and organizations are needed, as well as enable the possibility of modeling a Multiagent System (hereafter MAS) using several abstraction layers, which is also crucial to cover the abstraction principle. Because of structuring models in several layers, it is also needed models devoted to maintain traceability across layers.

From the software process point of view, bottom-up and top-down approaches are needed to finally allow us to go through layers completing models. Using both perspectives permits to obtain intermediate layers which link micro-level, that is to say, bottom layers that represents the agents behavior, with macro-level, that is to say, top layers that represent the organization/emergent behavior. In addition, the bottom-up software process provides means for reusing models, an important principle to deal with complexity.

Regarding automatic techniques to transform and analyze models, AOSE methodologies must provide techniques to decompose and to compose models, techniques to refine and abstract them, and techniques to determine where to draw the limits for composition/decomposition. These techniques must be automated as much as possible and can be used to support top-down and bottom-up software processes since decomposition of models into finer grain descriptions (refinement) helps top-down, and composition of models into higher level models (abstraction) helps us to perform a bottom-up approach.

2.2 Formal Methods and complexity

Formal methods are proven approaches for assuring the correct operation of complex interacting systems [23,24,38,57]. Formal methods are mathematically-based tools and techniques for specifying and verifying systems. They are particularly useful for specifying complex parallel and distributed systems where the entire system is difficult for a single person to fully understand and when more than one person was involved in the development. This is done thanks to applying the decomposition principle.

With formal methods, we may propose that certain properties hold, and prove that they hold automatically or semi-automatically, thus applying the automation principle. In particular this is invaluable for properties that we cannot test on Earth. By its nature, a good formal specification can guide us to propose and verify certain behaviors (or lack of certain behaviors) that we would often not think of when using regular testing techniques. Moreover, if properly applied, and properly used in the development process, a good formal specification can guarantee the presence or absence of particular properties in the overall system well in advance of mission launch, or even implementation. Indeed, various formal methods offer the additional advantage of support for simulation, model checking and automatic code generation, making the initial investment well worth while.

It has been stated that formal analysis is not feasible for emergent systems due to the complexity and intractability of these systems, and that simulation is the only viable approach for analyzing emergence of a systems [10]. For NASA missions, relying on simulations and testing alone are not sufficient even for systems that are much simpler than the ANTS mission, as noted above. The use of formal analysis would complement the simulation and testing of these complex systems and would give additional assurance of their correct operation. Given that one mistake can be catastrophic to a system and result in the loss of hundreds of millions of dollars and years of work, development of a formal analysis tool, even at a great cost, could have huge returns even if only one mission is kept from failing.

Verifying emergent behavior is an area that has been addressed very little by formal methods, though there has been some work done in this area by computer scientist analyzing biological systems [36,64,65]. However, formal methods may provide guidance in determining possible emergent behaviors that must be considered. Formal methods have been widely used for test case generation to develop effective test cases. Similar techniques may be used with formal methods, not to generate a test plan, but to propose certain properties that might or might not hold, or certain emergent behaviors that might arise.

3 Case study

The Autonomous Nano-Technology Swarm (ANTS) concept mission [13,15,16] will involve the launch of a swarm of autonomous pico-class (approximately 1kg) spacecraft that will explore the asteroid belt for asteroids with certain scientific characteristics. Figure 1 gives an overview of the ANTS mission [67]. In this mission, a transport ship, launched from Earth, will travel to a point in space where net gravitational forces on small objects (such as

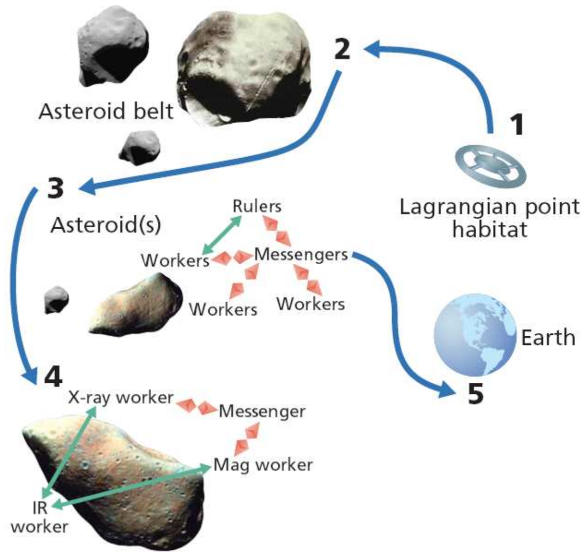

Figure 1 ANTS Mission Concept

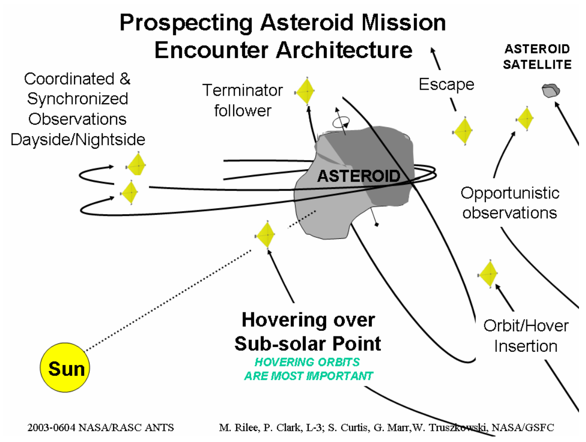
Prospecting Asteroid Mission
Encounter Architecture
Coordinated & Synchronized Observations Dayside/Nightside
Terminator follower
Escape
ASTEROID SATELLITE
ASTEROID
Opportunistic observations
Hovering over Sub-solar Point
HOVERING ORBITS ARE MOST IMPORTANT
Orbit/Hover Insertion
Sun
2003-0604 NASA/RASC ANTS
M. Rilee, P. Clark, L-3; S. Curtis, G. Marr, W. Truszkowski, NASA/GSFC

Figure 2 ANTS encounter with an asteroid

pico-class spacecraft) are negligible (a Lagrangian point). From this point, 1000 spacecraft, that have been manufactured en route from Earth, will be launched into the asteroid belt. The asteroid belt presents a large risk of destruction for large (traditional) spacecraft. Even with pico-class spacecraft, 60 to 70 percent of them are expected to be lost. Because of their small size, each spacecraft will carry just one specialized instrument for collecting a specific type of data from asteroids in the belt.

To implement this mission, a heuristic approach is being considered that provides for a social structure to the spacecraft that uses a hierarchical behavior analogous to colonies or swarms of insects, with some spacecraft directing others. Artificial intelligence technologies such as genetic algorithms, neural nets, fuzzy logic, and on-board planners are being investigated to assist the mission to maintain a high level of autonomy. Crucial to the mission will be the ability to modify its operations autonomously to reflect the changing nature of the mission and the distance and low bandwidth communications back to Earth. As shown in Figure 2, the swarm is envisioned to consist of several types of spacecraft. Approximately 80 percent of the spacecraft will be workers that will carry the specialized instruments (e.g., a magnetometer, x-ray, gamma-ray, visible/IR, neutral mass spectrometer) and will obtain specific types of data. Some will be coordinators (called rulers or leaders) that have rules that decided the types of asteroids and data the mission is interested in and that will coordinate the efforts of the workers. The third type of spacecraft are messengers that will coordinate communication between the rulers and workers, and communications with the mission control center on Earth.

The swarm will form sub-swarms, each under the control of a ruler, which contains models of the types of science that are to be pursued. The ruler will coordinate workers, each of which uses its individual instrument to collect data on specific asteroids and feeds this information back to the ruler, who will determine which asteroids are worth examining further. If the data matches the profile of a type of asteroid that is of interest, an imaging spacecraft will be sent to the asteroid to ascertain the exact location and to create a rough model to be used by other spacecraft for maneuvering around the asteroid. Other teams of spacecraft will then coordinate to finish mapping the asteroid to form a complete model.

3.1 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 [63]. 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 mechanism 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.

4 The FAST project

A NASA project, Formal Approaches to Swarm Technology (FAST), is investigating appropriate formal methods for use in swarm-based missions, and is beginning to apply these techniques to specifying and verifying parts of the NASA ANTS mission as a test-bed [58,56]. To verify NASA swarm-based missions an effective formal method must be able to predict the emergent behavior of 1000 agents as a swarm as well as the behavior of the individual agent. Crucial to the mission will be autonomic properties and the ability to modify operations autonomously to reflect the changing nature of the mission. For this, a formal specification will need to be able to track the goals of the mission as they change and to modify the model of the universe as new data comes in. The formal specification will also need to allow for specification of the decision-making process to aid in the decision as to which instruments will be needed, at what location, with what goals, etc.

The FAST project identified several important attributes needed in a formal approach for verifying swarm-based systems and surveyed a wide range of formal methods and formal techniques to determine whether existing formal methods, or a combination of existing methods, could be suitable for specifying and verifying swarm-based missions and their emergent behavior [58,61,56]. Various methods were surveyed based on a small number of criteria that were determined to be important in their application to intelligent swarms. These included:

- support for concurrency and real-time constraints;
- formal basis;
- (existing) tool support;
- past experience in application to agent-based and/or swarm-based systems;
- algorithm support.

A large number of formal methods that support the specification of one of, but not both, concurrent behavior and algorithmic behavior were identified. In addition, there were a large number of integrated or combination formal methods that have been developed over recent years with the goal of supporting the specification of both concurrency and algorithms.

Based on the results of the survey, four formal methods were selected to be used for a sample specification of part of the ANTS mission. These methods were: the process algebras CSP [25,31] and WSCCS [64,65], X-Machines [37], and Unity Logic [12]. CSP was chosen as a baseline specification method because the team has had significant experience and success [59,60] in specifying agent-based systems with CSP. WSCCS and X-Machines were chosen because they have already been used for specifying emergent behavior by others, apparently with some success. Unity Logic was also chosen because it had been successfully used for specifying concurrent systems and was a logic-based specification, which offered a contrast to the other methods.

The project is currently integrating these methods to develop a new formal method for swarm-based systems and will test this new formal method by developing a formal specification of the NASA ANTS mission.

5 Modeling using MaCMAS

MaCMAS is the AOSE methodology that we use for modeling swarm-based systems and is based on previously developed concepts [44][2]. It is specially tailored to model complex Multiagent Systems covering all the requirements for tackling complexity shown previously and by the best of our knowledge is the unique that cover all of these principles.

MaCMAS can be categorized in the AOSE methodologies that follows the organizational metaphor that is based on engineer MASs mimicking human organizations which can be also applied to swarm-based structures.

The organizational metaphor has been proven to be one of the most appropriate tools for engineering this kind of systems, and has been successfully applied by other methodologies, e.g., [40,43,69]. It shows that a MAS/swarm organization can be observed from two viewpoints [69]:

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

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

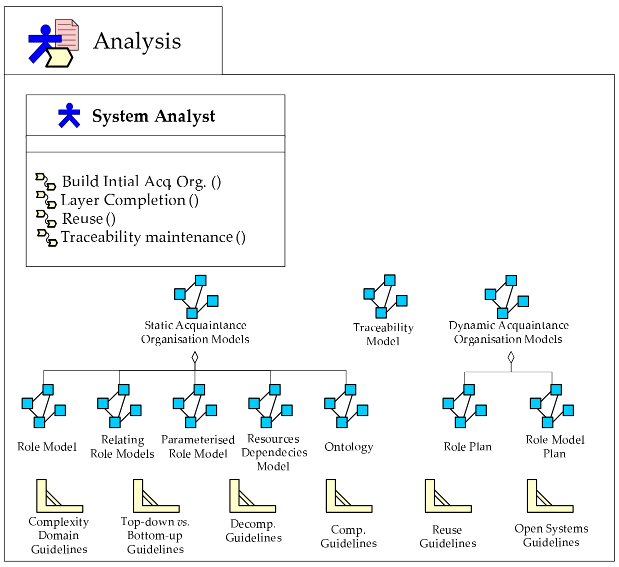

Figure 3 Acquaintance analysis discipline

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 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 [69]. Then, we can conclude that any acquaintance organization can be modeled orthogonally to its structural organization [34].

This concepts has been applied to the models we have performed of NASA ANTS where each spacecraft is assigned with a set of roles that change over time depending on the environment.

Entering into details, MaCMAS focuses on the acquaintance organization view providing a set of UML2.0-based models, a software process to build them, and a set of techniques to compose, decompose, refine and abstract models, as required by the principles to deal with complexity. These models are not orthogonal to the formal specifications performed using formal methods, but complementary since they provide a graphical representation of the system that improves the understanding of the specifications at the time they have a strong mathematical support thanks to formal methods.

Figure 4, summarize the structure of models produced by MaCMAS and how they are obtained. Roughly speaking, MaCMAS produces a set of models of the system at different levels of abstraction in order to tackle the complexity of a system iteratively. Thus, a model of

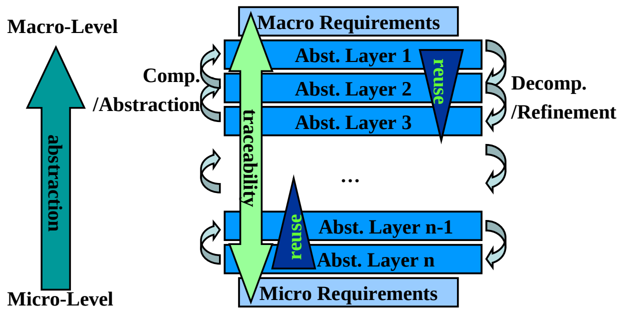

Figure 4 Overview of the structure of MaCMAS models

the system at the micro-level, where all details are modeled, can be linked with a model of the system at the macro-level where only relevant properties are modeled using abstraction. This allows us to ensure properties at the micro, macro, and intermediate levels of the system by means of formal methods. In addition to this structure of models, as shown in the following, MaCMAS also provides techniques to refine and abstract these models to complete layers.

In Figure 3, we summarize all the models, activities and guidelines included in MaCMAS. From all the models proposed in this methodology, the most important are the following:

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. The most important are the role models:
   Role Models: show an acquaintance sub-organization as a set of roles collaborating by means of several *multi-Role Interactions* (mRI) [46]. 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. This allows to represent abstractedly complex joint process that implies IA techniques such as negotiation, learning or self-* techniques, allowing modeling emergent features by means of abstraction.
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 [42]. 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

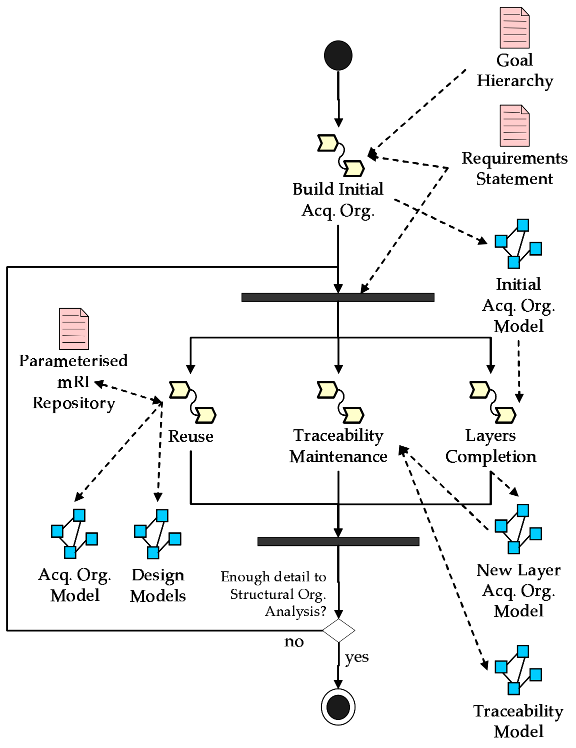

Figure 5 Acquaintance analysis process

represented using UML 2.0 StateMachines [42]. It is used to facilitate the 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 at whichever level of abstraction, from micro to macro-level.

The software process of MaCMAS starts with requirements documented using a goal-directed approach [17]. After applying MaCMAS we obtain the acquaintance organization of the MAS thus enabling to assign roles to run-time agents using a middleware that supports this feature. This allows us to build the structural organization and to change it at run-time, thus easing the modeling and management of self-organizing systems, as need by NASA swarm-based missions.

The software process of MaCMAS is described as a set of abstract SPEM work definitions which can be instantiated by any work definition that produces the same work products (c.f. Ref. [41] for consulting the SPEM specification). In Figure 5, we show our process by means of several general process components in form of SPEM work definitions. These work definitions are strictly related to the main principles dealing with complexity and are responsible for applying them:

Build Initial Acquaintance organization model: It consists in producing an initial set of role models. These models provide an initial understanding of the system to be built which is augmented by means of the rest of work definitions. This work definition applies the abstraction principle to provide a first approach to the model of the system.

Layer Completion: It consists in producing a new acquaintance organization model because of the composition or the decomposition of a model(s) developed in a previous iteration or in the *build initial acquaintance organization work definition*. The model produced is used to fill a bottom/top layer when using decomposition/composition of mRIs and is used to reduce/augment a model maintaining the level of abstraction when using decomposition/composition of role models, thus performing a top-down or bottom-up designs[45,46].
These processes are supported by formal methods that help us to perform these operations and to prove properties over models. Thus, we can ensure that the emergent behavior that we obtain is enclosed into a safe operational space.
Thus, this work definition is in charge of applying abstraction, decomposition and composition principles, and the automation principle to composition and decomposition when possible.

Traceability Maintenance: It updates the *traceability model* which documents the relations between models in different layers and requirements system goals.
This model helps us to link macro-level behavior of the system with micro-level behavior, what finally helps us to controll the emergent behavior we obtain, for example, giving information on which macro-level properties of the system will be affected by changes in certain properties of single spacecrafts, or adding formal specifications of these models that allow to check properties.

Reuse: It instantiates parameterized role models stored in a repository when appropriate. It is also responsible for analyzing models produced in the *layer completion work definition* to add them to the repository. This work definition is responsible for enforcing the application of the reuse principle.

5.1 Modeling autonomous and autonomic systems using MaCMAS

MaCMAS can be used to model autonomous and autonomic properties of swarm-based systems as needed by NASA missions [50]. To exemplify the models of MaC-

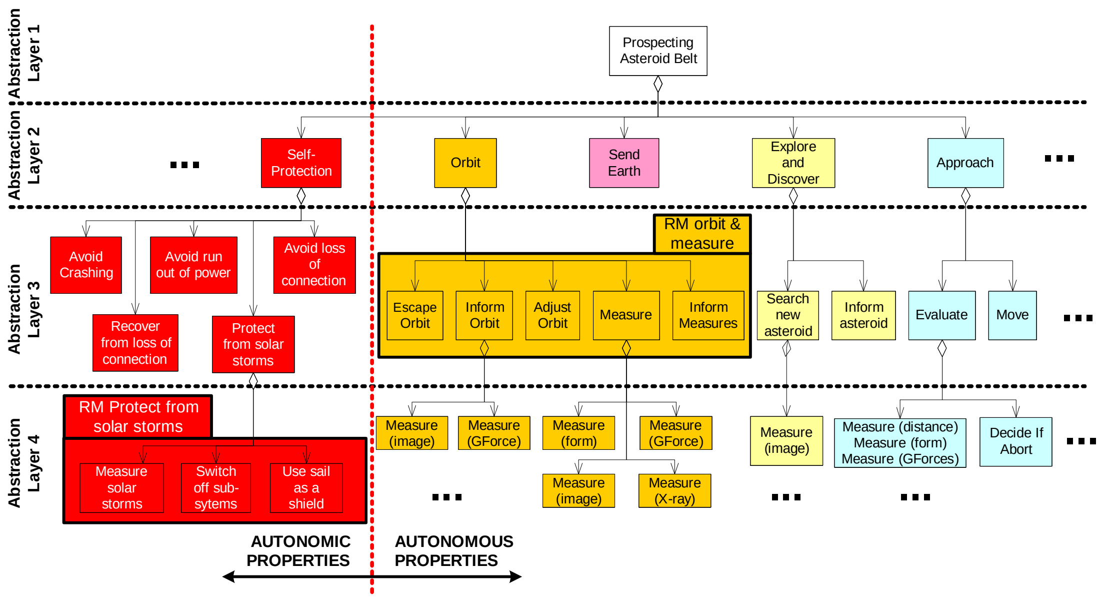

Figure 6 Traceability model of ANTS

MAS and how the are applied to model both kind of properties, we use the case study presented in Section 3.

After applying all the software process of MaCMAS to the case study, we obtain the traceability diagram of Figure 6. This diagram summarizes the mRIs in the system structured by layers of abstraction. In this diagram, the top layer is the most abstract, i.e. the macro-level. 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 6 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 the deploying of these features 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.

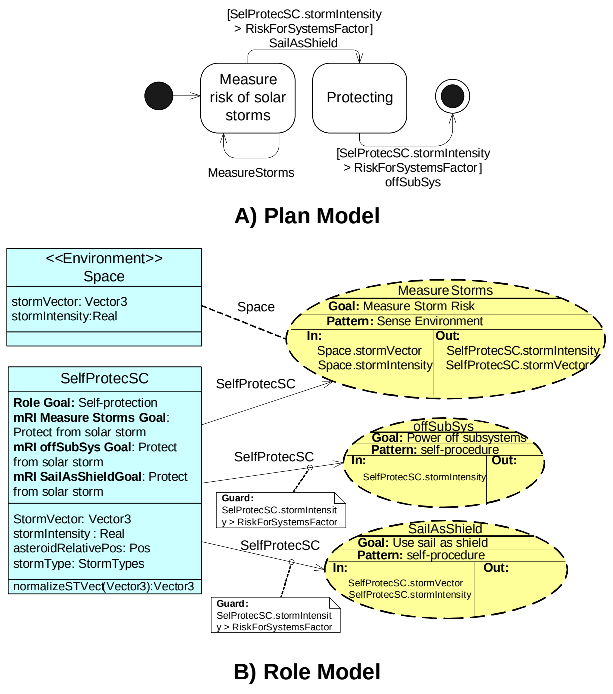
[SelProtecSC.stormIntensity > RiskForSystemsFactor] SailAsShield
Measure risk of solar storms
Protecting
MeasureStorms
[SelProtecSC.stormIntensity > RiskForSystemsFactor] offSubSys
A) Plan Model
<<Environment>> Space
stormVector: Vector3
stormIntensity:Real
Space
MeasureStorms
Goal: Measure Storm Risk
Pattern: Sense Environment
In:
Space.stormVector
Space.stormIntensity
Out:
SelfProtecSC.stormIntensity
SelfProtecSC.stormVector
SelfProtecSC
Role Goal: Self-protection
mRI Measure Storms Goal: Protect from solar storm
mRI offSubSys Goal: Protect from solar storm
mRI SailAsShieldGoal: Protect from solar storm
StormVector: Vector3
stormIntensity : Real
asteroidRelativePos: Pos
stormType: StormTypes
normalizeSTVec(Vector3):Vector3
SelfProtecSC
offSubSys
Goal: Power off subsystems
Pattern: self-procedure
In:
Out:
SelfProtecSC.stormIntensity
Guard: SelProtecSC.stormIntensity > RiskForSystemsFactor
SelfProtecSC
SailAsShield
Goal: Use sail as shield
Pattern: self-procedure
In:
Out:
SelfProtecSC.stormVector
SelfProtecSC.stormIntensity
Guard: SelProtecSC.stormIntensity > RiskForSystemsFactor
B) Role Model

Figure 8 Self-protection from solar storms autonomic property model

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. Notice that the plan model can be formaly specified using CSP thus being able to check properties such as deadlocks, etc. Figure 7b shows the role model for this case. We show here the models from the third layer of abstraction of Figure 6. In this model there are two kinds of el-

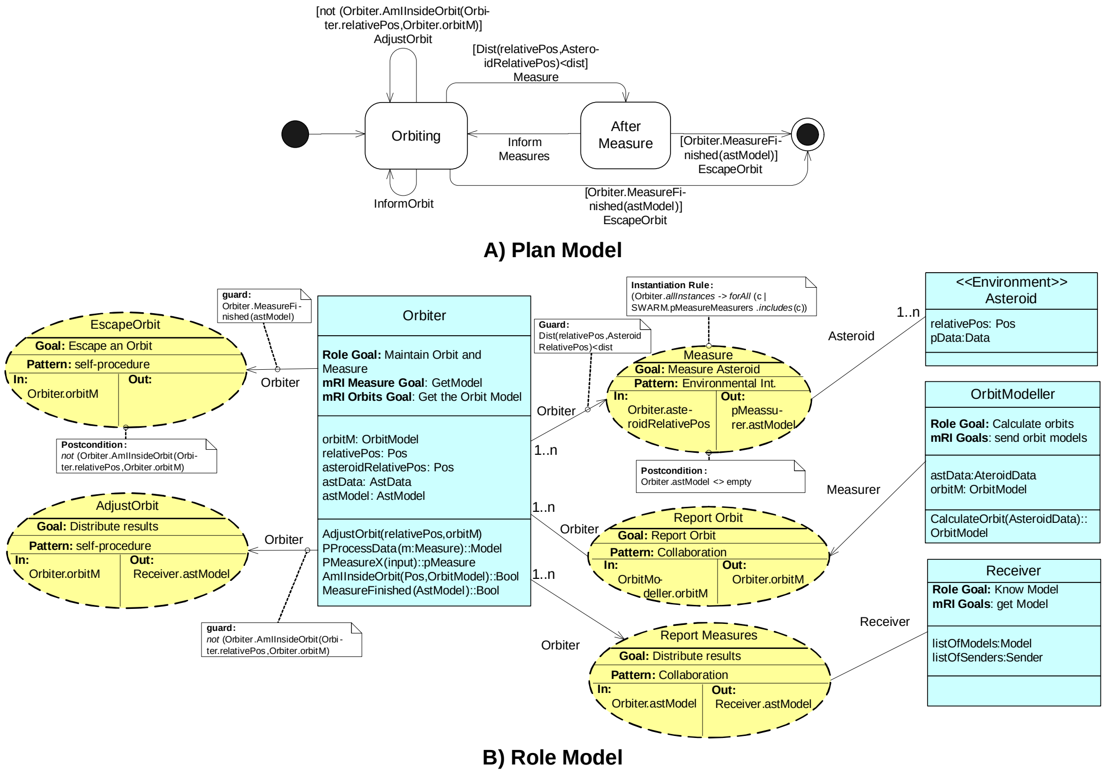
[not (Orbiter.AmIInsideOrbit(Orbiter.relativePos,Orbiter.orbitM)] AdjustOrbit
[Dist(relativePos,AsteroidRelativePos)<dist] Measure
Orbiting
After Measure
Inform Measures
[Orbiter.MeasureFinished(astModel)] EscapeOrbit
InformOrbit
[Orbiter.MeasureFinished(astModel)] EscapeOrbit
A) Plan Model
guard: Orbiter.MeasureFinished(astModel)
EscapeOrbit
Goal: Escape an Orbit
Pattern: self-procedure
In: Orbiter.orbitM
Out:
Postcondition: not (Orbiter.AmIInsideOrbit(Orbiter.relativePos,Orbiter.orbitM)
Orbiter
Role Goal: Maintain Orbit and Measure
mRI Measure Goal: GetModel
mRI Orbits Goal: Get the Orbit Model
orbitM: OrbitModel
relativePos: Pos
asteroidRelativePos: Pos
astData: AstData
astModel: AstModel
AdjustOrbit(relativePos,orbitM)
PProcessData(m:Measure)::Model
PMeasureX(input)::pMeasure
AmIInsideOrbit(Pos,OrbitModel)::Bool
MeasureFinished(AstModel)::Bool
AdjustOrbit
Goal: Distribute results
Pattern: self-procedure
In: Orbiter.orbitM
Out: Receiver.astModel
guard: not (Orbiter.AmIInsideOrbit(Orbiter.relativePos,Orbiter.orbitM)
Instantiation Rule: (Orbiter.allInstances -> forAll (c | SWARM.pMeasureMeasurers .includes(c))
Guard: Dist(relativePos,AsteroidRelativePos)<dist
Measure
Goal: Measure Asteroid
Pattern: Environmental Int.
In: Orbiter.asteroidRelativePos
Out: pMeassurer.astModel
Postcondition: Orbiter.astModel <> empty
Asteroid
1..n
<<Environment>> Asteroid
relativePos: Pos
pData:Data
OrbitModeller
Role Goal: Calculate orbits
mRI Goals: send orbit models
astData:AteroidData
orbitM: OrbitModel
CalculateOrbit(AsteroidData):: OrbitModel
Measurer
Report Orbit
Goal: Report Orbit
Pattern: Collaboration
In: OrbitModeller.orbitM
Out: Orbiter.orbitM
Receiver
Role Goal: Know Model
mRI Goals: get Model
listOfModels:Model
listOfSenders:Sender
Report Measures
Goal: Distribute results
Pattern: Collaboration
In: Orbiter.astModel
Out: Receiver.astModel
B) Role Model

Figure 7 Orbiting and measuring an asteroid autonomous property

ements: 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 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 Figure 7a, 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 8b, 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 8a. 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.

5.2 MaCMAS for modelling Multiagent Systems Product Lines (MAS-PL)

Many organizations, and software companies in particular, develop a range of products over periods of time that exhibit many of the same properties and features. The multiagent systems community exhibits similar trends.

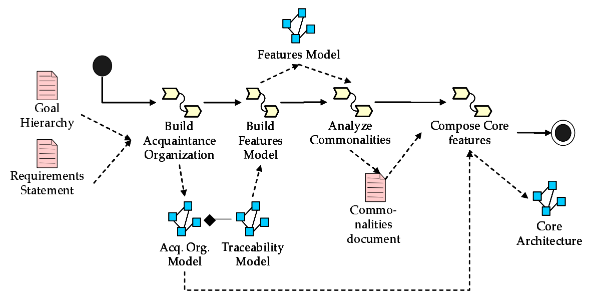

Figure 9 Overview of the process for building the core architecture of a MAS-PL

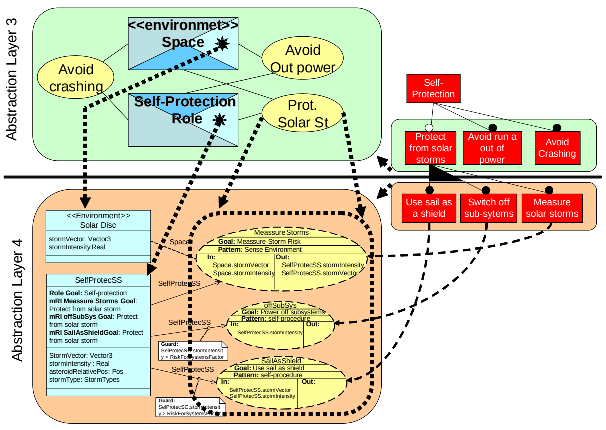
Abstraction Layer 3
<<environmet>> Space
Avoid crashing
Avoid Out power
Self-Protection Role
Prot. Solar St
Self-Protection
Protect from solar storms
Avoid run a out of power
Avoid Crashing
Abstraction Layer 4
Use sail as a shield
Switch off sub-sytems
Measure solar storms
<<Environment>> Solar Disc
stormVector: Vector3
stormIntensity:Real
SelfProtecSS
Role Goal: Self-protection
mRI Meassure Storms Goal: Protect from solar storm
mRI offSubSys Goal: Protect from solar storm
mRI SailAsShieldGoal: Protect from solar storm
StormVector: Vector3
stormIntensity : Real
asteroidRelativePos: Pos
stormType: StormTypes
Space
SelfProtecSS
MeassureStorms
Goal: Meassure Storm Risk
Pattern: Sense Environment
In: Space.stormVector Space.stormIntensity
Out: SelfProtecSS.stormIntensity SelfProtecSS.stormVector
offSubSys
Goal: Power off subsystems
Pattern: self-procedure
In: SelfProtecSS.stormIntensity
Out:
Guard: SelProtecSC.stormIntensity > RiskForSystemsFactor
SailAsShield
Goal: Use sail as shield
Pattern: self-procedure
In: SelfProtecSS.stormVector SelfProtecSS.stormIntensity
Out:

Figure 10 Role model/features relationship

However, the community has not as yet developed the infrastructure to develop a core multiagent system (hereafter, MAS) from which concrete (substantially similar) products can be derived.

The software product line paradigm (hereafter, SPL) augurs the potential of developing a set of core assets for a family of products from which customized products can be rapidly generated, reducing time-to-market, costs, etc. [14], 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 [49,48].

All NASA swarm-based missions present many common features, thus it is feasible of applying a MAS-PL approach improving the application of the reuse principle, and thus improving our capabilities to deal with its complexity. This may also dramatically reduce the costs in money and time of these related missions.

For enabling a product line, one of the important activities to be performed is to identify a core architecture for the family of software products.

In Figure 9, we show the SPEM definition of the software process of our approach to build the core architecture.

Build Acquaintance Organziation. The first stage to be performed consists of developing a set of models in different layers of abstraction where we obtain a traceability model and a set of role models showing how each goal is materialized. This is achieved by applying the MaCMAS software process.

Build Features Model. The second activity shown is responsible for adding commonalities and variabilities to the traceability model. This is done modifying the traceability diagram to add information on variability and commonalities, as shown in Figure 11, to obtain a feature model of the family. We do not detail this process since it relies on taking each node of the traceability diagram and determining if it is mandatory, optional, alternative, or-exclusive, or if it depends on other(s), as shown in the figure.

There exists a direct traceability between features and role models. When a system goal is complex enough to require more than one agent in order to be fulfilled, a group of agents are required to work together. Hence, a role model shows a set of agents, represented by the role they play, that join to achieve a certain system goal (whether by contention or cooperation). MaCMAS uses mRIs to represent all of the joint processes that are required and are carried out amongst roles in order to fulfill the system goal of the role model. These also pursue system sub-goals as shown in Figure 10, 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 8.

Analize Commonalities. Later, we perform a commonality analysis to find out which features, called core features, and thus which role models, are more used across products.

To build the core architecture of the system we must include those features that appear in all the products and those whose probability of appearing in a product is high. In [1,2] the authors define the commonality of a feature as the percentage of products defined within a feature model that contains the feature. A calculation method for this and many other operations related to feature models analysis is proposed using Constraint Satisfaction Problems (CSP). To perform this calculation, we have extended the prototype[3] presented in [1] to automatically calculate the commonality of all the features of our case study. The results obtained with the prototype are shown in Figure 12. As shown in this figure, these features are ordered by their commonality. The figure also show the threshold that we have selected, set up at the 60%, for considering a feature to be core or not. As shown in the Figure, the commonality for the features *self-protection from a solar storm* and *orbiting* is 100%. Thus we have to add them to the

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

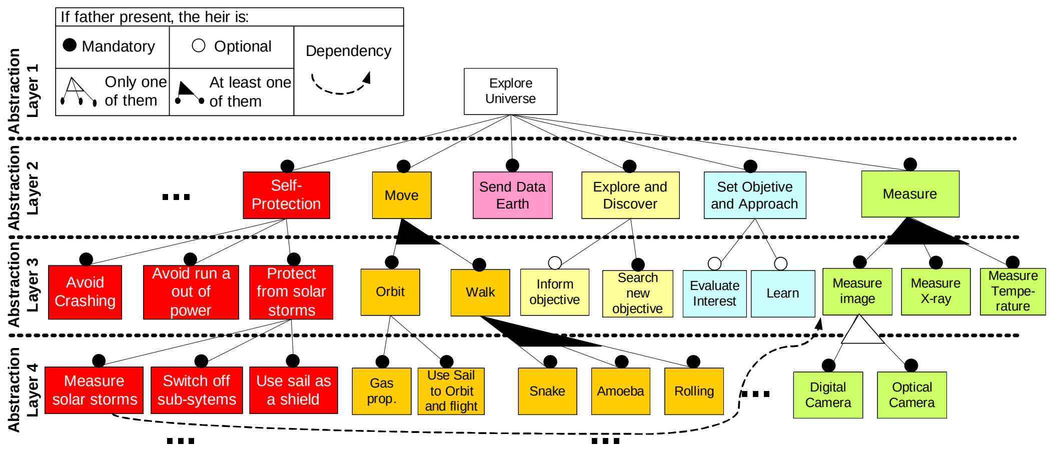

Figure 11 Features model of our case study

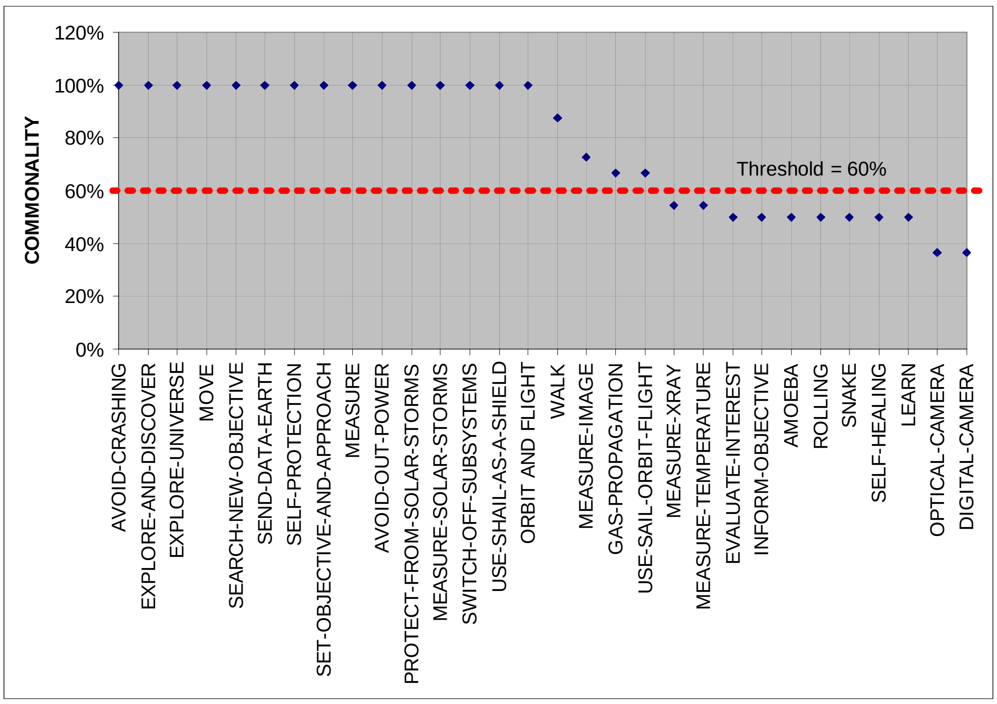

Figure 12 Commonalities of the features in our example

core architecture, since they appear in all the possible products.

Compose Core Features. Then, given that traceability diagrams and thus features models present a direct traceability between system goals and role models, we can use the composition operation of MaCMAS to compose the role models corresponding to the core features to obtain the core architecture.

5.3 Modeling evolving systems using MaCMAS

MaCMAS is also able to model evolving systems as required by swarm-based NASA missions [47]. MaCMAS bases on viewing different instances of a system as it evolves as different "products´´ in a Software Product Line. That Software Product Line is in turn developed with an agent-oriented software engineering approach and views the system as a Multiagent System Product Line. The use of such an approach is particularly appropriate as it allows us to scale our view to address enterprise architectures where various entities in the enterprise are modeled as software agents. This, improves the application of the decomposition principle allowing to factorize the models and the formal specifications making them simpler and more understandable.

Each product in a MAS-PL is defined as a set of features. Given that all the products present a set of

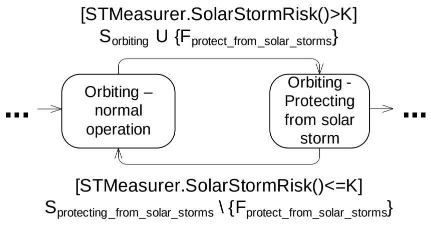
[STMeasurer.SolarStormRisk()>K]
S_orbiting ∪ {F_protect_from_solar_storms}
Orbiting – normal operation
Orbiting - Protecting from solar storm
...
...
[STMeasurer.SolarStormRisk()<=K]
S_protecting_from_solar_storms \ {F_protect_from_solar_storms}

Figure 13 Evolution plan of our case study

features that remain unchanged, the core architecture is defined as the part of all of the products that implement these common features[48]. Thus, a system can evolve by changing, or evolving, the set of non-core features.

A product or a state in our evolutionary system can be defined as a set of features. Let $F = \{f_1..f_n\}$ be the set of all features of a MAS-PL. Let $cF \subset F$ be the set of core features and $ncF = F \setminus CF$ be the set of non-core features. We define a valid state of the system as the set of core features and a set of non-core features, that is to say, $S = cF \cup sF$, where $sF \subset ncF$ is a subset of non-core features.

Given that, the evolution from one state S_{i-1} to another S_i is defined as:

$$S_i = S_{i-1} \cup nF_{i,i-1} \setminus dF_{i,i-1}$$

where $nF_{i,i-1} \subset ncF$ is the set of new features and $dF_{i,i-1} \subset ncF$ is the set of deleted features.

Finally, $\Delta_{i,i-1}$ describes the variation between the product of the state $i-1$ and the product of the state i, that is to say, $nF_{i,i-1} \setminus dF_{i,i-1}$.

In [48], we show that a feature correlates with a role model, also shown in previous section. Thus, for a system to evolve from one state to another, we must compose or decompose the role models in nF and dF. Specifically, we must compose the role models corresponding to the features in nF with the role models corresponding to the features that remain unchanged from the initial state S_{i-1}, that is to say $S_i \setminus dF_{i,i-1}$. Decomposition is used for role models that must be eliminated.

We represent the evolution plan using a UML state machine where each state represents a product, and each transition represents the addition or elimination of a set of features, that is to say, Δ. In addition, the conditions in the transitions represent the properties that must hold in the environment and in the system in order to evolve to the new product.

In Figure 13, we show part of the evolution plan of our case study. There we represent two products, one representing the swarm when orbiting an asteroid, and another representing the swarm when orbiting and protecting from a solar storm. As can be seen, we add or delete the feature corresponding to protect from solar storm depending on whether or not the swarm is under risk of solar storm.

The main advantage of this approach resides in the fact that it allows us to derive a formal model of the system and of each state that it may reach. This allows us to clearly specify the differences from one state of the architecture and any subsequent states of that evolving system. This significantly improves our capabilities to understand, analyze and test evolving systems. Additionally, thanks to the use of MaCMAS which allows for the description of the same feature at different levels of abstraction, we can also specify and test the architectural changes at different levels of abstraction.

Finally, such an approach provides support at run time for the addition and deletion of roles in the architecture. It provides reflection mechanisms that enable understanding of the features, roles, and agents at different levels of abstraction, providing capabilities for ensuring quality of service by means of self-organization, self-protection, and other self-* properties identified by the Autonomic Computing initiative.

6 Tool support

All this techniques are not valuable if they have to be performed by hand. Thus, and in order to fulfill the automation principle to deal with complexity, we have implemented several tools to support automatically the techniques exposed in this paper.

6.1 Requirements-to-Design-to-Code (R2D2C)

Requirements-Based Programming (RBP) has been advocated [20, 21] as a viable means of developing complex evolving systems. The idea that it embodies is that requirements can be systematically and mechanically transformed to executable code.

This may seem to be an obvious goal in the engineering of computer-based systems, but requirements-based programming does in fact go a step further than current development methods. System development, typically, assumes the existence of a model of reality, called a design (more correctly, a design specification), from which an implementation will be derived [27]. This model must itself be derived from the system requirements, but there is a large ‘gap’ in going from requirements to design. Requirements-Based Programming seeks to eliminate this ‘gap’ by ensuring that the ultimate implementation can be traced fully back to the actual requirements. NASA’s experience has been that emphasizing sufficient effort at the requirements phase of development can significantly reduce cost overruns later [9]. RBP promises a significant payoff for increasing effort at the requirements phase by reducing the level of effort in subsequent verification.

R2D2C (Requirements-to-Design-to-Code) is a NASA patent pending approach to the engineering of complex computer systems, where the need for correctness of the system, with respect to its requirements, is significantly

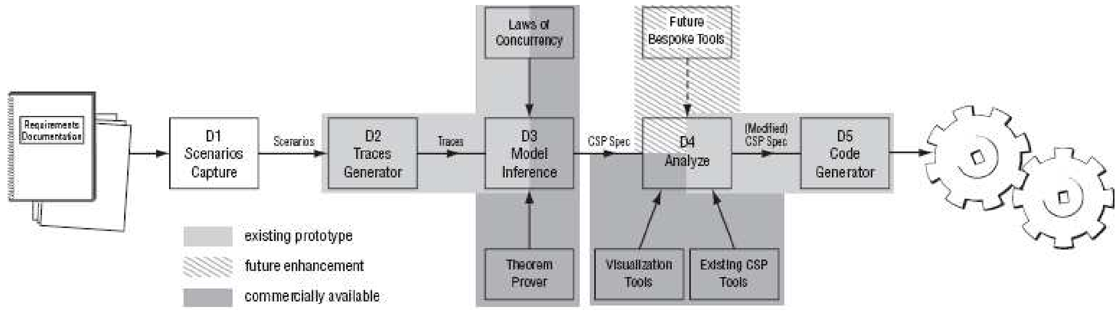

Figure 14 The R2D2C approach and current status of the prototype

high [26,28]. In this category, we include NASA mission software, most of which exhibits both autonomous and autonomic properties, and must continue to do so in order to achieve survivability in harsh environments.

In the R2D2C approach (Figure 14), engineers (or others) may write requirements as scenarios in constrained (domain-specific) natural language, or in a range of other notations (including UML use cases). These will be used to derive a formal model that is guaranteed to be equivalent to the requirements stated at the outset, and which will subsequently be used as a basis for code generation. The formal model can be expressed using a variety of formal notations. Currently we are using CSP, Hoare's language of Communicating Sequential Processes [29,30], which is suitable for various types of analysis and investigation, and as the basis for fully formal implementations as well as automated test case generation, etc.

R2D2C is unique in that it allows for full formal development from the outset, and maintains mathematical soundness through all phases of the development process, from requirements through to automatic code generation. The approach may also be used for reverse engineering, that is, in retrieving models and formal specifications from existing code. The method can also be used to "paraphrase" (in natural language, etc.) formal descriptions of existing systems.

In addition, the approach is not limited to generating executable code. It may also be used to generate business processes and procedures, and we have been experimenting (successfully) with using a rudimentary prototype to generate instructions for robotic devices to be used on the Hubble Robotic Servicing Mission (HRSM) [54]. We are also experimenting with using it as a basis for an expert system verification tool, and as a means of capturing expert knowledge for expert systems [55].

6.2 MaCMAS Case tool

We have also implemented a module plug-in for the open source UML modeling tool ArgoUML[4].

This case tool allows to represent all the MaCMAS models. It provides automatic support for operations for decomposing/composing and for refining/abstracting models using an MDA approach [51]. Furthermore, it provides code generation for the skeletons of roles and its plans in JAVA, which will be extended to conform with R2D2 code generation.

Another important feature is that it maintains traceability between goal-oriented requirements and all the models used in MaCMAS allowing to navigate to the models that materialize a system-goal at whichever level of abstraction.

[4] See `james.eii.us.es/MaCMAS/` for further details

7 Conclusions and Future work

Complexity is a big challenge of current software development. When tackling systems with the properties of NASA ANTS, complexity become unmanageable with current engineering and software tools.

The main conclusion we can draw from our work on these missions is that the modeling of this kind of systems is not feasible using only UML-based tailored models, or only formal methods, but a new set of formal techniques along with a tailored set of models and software processes. As shown, we have had to base on techniques from many fields ranging from, and not only, the use of several formal methods, autonomic computing, swarm-based systems, to new applications of the software product lines approach.

As shown, all these techniques carefully covers the principles to deal with complexity which is crucial for the success of our work.

However, although many promising results have been already reached, a long research path has still to be traveled. The main future research lines will consist on improving the integration of formal methods and AOSE and the improvement and integration of the tools we are developing. Indeed, the test of these approaches with NASA swarm-based case studies.

References

1. D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.

2. D. Benavides, A. Ruiz-Cortés, P. Trinidad, and S. Segura. A survey on the automated analyses of feature models. *XV Jornadas de Ingeniería del Software y Bases de Datos,JISBD 2006*, 2006.
3. Gerardo Beni. The concept of cellular robotics. In *Proc. 1988 IEEE International Symposium on Intelligent Control*, pages 57–62. IEEE Computer Society Press, Los Alamitos, Calif., 1988.
4. Gerardo Beni and Jing Want. Swarm intelligence. In *Proc. Seventh Annual Meeting of the Robotics Society of Japan*, pages 425–428, Tokyo, Japan, 1989. RSJ Press.
5. Eric Bonabeau, Marco Dorigo, and Guy Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, 1999.
6. Eric Bonabeau and Guy Théraulaz. Swarm smarts. *Scientific American*, pages 72–79, March 2000.
7. Eric Bonabeau, Guy Théraulaz, Jean-Louis Deneubourg, Serge Aron, and Scott Camazine. Self-organization in social insects. *Trends in Ecology and Evolution*, 12:188–193, 1997.
8. G. Booch. *Object-Oriented Design with Applications.* Benjamin/Cummings, Redwood City, CA, 1990.
9. J. P. Bowen and M. G. Hinchey. Ten commandments revisited: A ten year perspective on the industrial application of formal methods. In *Proc. FMICS 2005, 10th International Workshop on Formal Methods for Indutrial Critical Systems*, Lisbon, Portugal, 5 – 6 September 2005. ACM Press.
10. S. Brueckner and H. V. Dyke Parunak. Resource-aware exploration of the emergent dynamics of simulated systems. In *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS)*, pages 781–788, 2003.
11. Shawn Carlson. Artificial life: Boids of a feather flock together. *Scientific American*, November 2000.
12. K. Mani Chandy and Misra J. *Parallel Program Design: A Foundation.* Addison-Wesley Publishing Company, 1988.
13. Pamela E. Clark, Steven A. Curtis, and Michael L. Rilee. ANTS: Applying a new paradigm to Lunar and planetary exploration. In *Proc. Solar System Remote Sensing Symposium*, Pittsburgh, Pennsylvania, USA, 20–21 September 2002.
14. P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns.* SEI Series in Software Engineering. Addison–Wesley, August 2001.
15. Steven A. Curtis, J. Mica, J. Nuth, G. Marr, Michael L. Rilee, and Maharaj K. Bhat. ANTS (Autonomous Nano-Technology Swarm): An artificial intelligence approach to Asteroid Belt resource exploration. In *Proc. Int'l Astronautical Federation, 51st Congress*, October 2000.
16. Steven A. Curtis, W. F. Truszkowski, Michael L. Rilee, and Pamela E. Clark. ANTS for the human exploration and development of space. In *Proc. IEEE Aerospace Conference*, Big Sky, Montana, USA, 9–16 March 2003.
17. A. Dardenne, A. van Lamsweerde, and S.Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.
18. D.F. D'Souza and A.C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach.* Addison–Wesley, Reading, Mass., 1999.
19. J. Fromm. *The Emergence of Complexity.* Kassel university press, 2004.
20. D. Harel. From play-in scenarios to code: An achievable dream. *IEEE Computer*, 34(1):53–60, 2001.
21. D. Harel. Comments made during presentation at "Formal Approaches to Complex Software Systems" panel session. *ISoLA-04 First International Conference on Leveraging Applications of Formal Methods*, Paphos, Cyprus. 31 October 2004.
22. David E. Hiebeler. The swarm simulation system and individual-based modeling. In *Proc. Decision Support 2001: Advanced Technology for Natural Resource Management*, Toronto, Canada, September 1994.
23. M. Hinchey, J. Rash, and C. Rouff. Verification and validation of autonomous systems. In *Proc. SEW-26, 26th Annual NASA/IEEE Software Engineering Workshop*, pages 136–144, Greenbelt, MD, November 2001. NASA Goddard Space Flight Center, IEEE Computer Society Press, Los Alamitos, Calif.
24. M. G. Hinchey and J. P. Bowen, editors. *Industrial-Strength Formal Methods in Practice.* FACIT Series. Springer-Verlag, London, UK, 1999.
25. M. G. Hinchey and S. A. Jarvis. *Concurrent Systems: Formal Development in CSP.* International Series in Software Engineering. McGraw-Hill International, London, UK, 1995.
26. 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.
27. M. G. Hinchey, J. L. Rash, and C. A. Rouff. A formal approach to requirements-based programming. In *Proc. IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005).* IEEE Computer Society Press, Los Alamitos, Calif., 3–8 April 2005.
28. M. G. Hinchey, J. L. Rash, C. A. Rouff, and D. Gračanin. Achieving dependability in sensor networks through automated requirements-based programming. *Journal of Computer Communications, Special Issue on"Dependable Wireless Sensor Network"*, (To appear), 2005.
29. C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
30. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall International Series in Computer Science. Prentice Hall International, Englewood Cliffs, NJ, 1985.
31. C.A.R. Hoare. *Communicating sequential processess.* Prentice Hall, 1985.
32. N. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
33. A. Karageorgos and N. Mehandjiev. A design complexity evaluation framework for agent-based system engineering methodologies. In A. Omicini, P. Petta, and J. Pitt, editors, *Fourth International Workshop Engineering Societies in the Agents World*, volume 3071 of *Lecture Notes in Computer Science*, pages 258–274. Springer, 2004.
34. E. A. Kendall. Role modeling for agent system analysis, design, and implementation. *IEEE Concurrency*, 8(2):34–41, April/June 2000.
35. Francesco Luna and Benedikt Stefansson. *Economic Simulations in Swarm: Agent-Based Modelling and Ob-*

ject Oriented Programming. Kluwer Academic Publishers, 2000.
36. W. Michael and L. Holcombe. Mathematical models of cell biochemistry. Technical Report CS-86-4, Sheffield University, UK, 1986.
37. W. Michael and L. Holcombe. X-Machines as a basis for system specification. *Software Engineering*, 3(2):69–76, 1988.
38. P. Pandurang Nayak, D. E. B., G. Dorais, E. B. Gamble Jr., B. Kanefsky, J. Kurien, W. Millar, N. Muscettola, K. Rajan, N. Rouquette, B. D. Smith, W. Taylor, and Yu wen Tung. Validating the DS1 remote agent experiment. In *Proc. of the 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space (iSAIRAS-99)*, 1999.
39. J. Odell. Agents and complex systems. *Journal of Object Technology*, 1(2):35–45, July-August 2002.
40. J. Odell, H. Parunak, and M. Fleischer. The role of roles in designing effective agent organisations. In A. Garcia and C. Lucenaand F. Zambonelliand A. Omiciniand J. Castro, editors, *Software Engineering for Large-Scale Multi-Agent Systems*, number 2603 in LNCS, pages 27–28, Berlin, 2003. Springer–Verlag.
41. OMG. Software process engineering metamodel, version 1.1, 2005. Available at `http://www.omg.org/technology/documents/formal/spem.htm`.
42. Object Management Group (OMG). Unified modeling language: Superstructure. version 2.0. Final adopted specification ptc/03–08–02, OMG, August 2003. www.omg.org.
43. H. Van Dyke Parunak and James Odell. Representing social structures in UML. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 100–101, Montreal, Canada, 2001. ACM Press.
44. 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.
45. 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.
46. 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.
47. J. Peña, M. G. Hinchey, M. Resinas, R. Sterritt, and J. L. Rash. Managing the evolution of an enterprise architecture using a mas-product-line approach. In *5th International Workshop on System/Software Architectures (IWSSA'06)*, page to be published, Nevada, USA, 2006. CSREA Press.
48. J. Peña, M. G. Hinchey, and A. Ruiz-Cortés. Building the core architecture of a nasa multiagent system product line. In *7th International Workshop on Agent Oriented Software Engineering 2006*, page to be published, Hakodate, Japan, May, 2006. LNCS.
49. J. Peña, M. G. Hinchey, and Antonio Ruíz-Cortes. Multiagent system product lines: Challenges and benefits. *Communications of the ACM*, December 2006.
50. J. Peña, M. G. Hinchey, and R. Sterritt. Towards modeling, specifying and deploying policies in autonomous and autonomic systems using an aose methodology. *Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE'06)*, 0:37–46, 2006.
51. J. Peña, M. G. Hinchey, R. Sterritt, A. Ruiz-Cortés, and M. Resinas. A model-driven architecture approach for modeling, specifying and deploying polices in autonomous and autonomic systems. In *The 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC'06)*, page to be published, Indianapolis, USA, 2006. IEEE CS Press.
52. J. Peña, R. Levy, and R. Corchuelo. Towards clarifying the importance of interactions in agent-oriented software engineering. *International Iberoamerican Journal of AI*, 8(25):19–28, 2005.
53. C raig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.
54. J. L. Rash, M. G. Hinchey, C. A. Rouff, and D. Gračanin. Formal requirements-based programming for complex systems. In *Proc. International Conference on Engineering of Complex Computer Systems*, Shanghai, China, 16–20 June 2005. IEEE Computer Society Press, Los Alamitos, Calif.
55. J. L. Rash, M. G. Hinchey, C. A. Rouff, D. Gračanin, and J. D. Erickson. Experiences with a requirements-based programming approach to the development of a NASA autonomous ground control system. In *Proc. IEEE Workshop on Engineering of Autonomic Systems (EASe 2005) held at the IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005).* IEEE Computer Society Press, Los Alamitos, Calif., 3–8 April 2005.
56. C. Rouff, M. Hinchey, W. Truszkowski, and J. Rash. Experiences applying formal approaches in the development of swarm-based space exploration systems. *International Journal of on software Tools for Technology Transfer. Special Issue on Formal Methods in Industry. (submitted)*, 2006.
57. C. Rouff, J. L. Rash, M. G. Hinchey, and W. Truszkowski. Formal methods at NASA Goddard Space Flight Center. In *Agent Technology from a Formal Perspective*, NASA Monographs in Systems and Software Engineering, pages 287–310. Springer-Verlag, London, UK, 2005.
58. C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash. Formal methods for swarm and autonomic systems. In *Proc. 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA)*, Cyprus, Oct 30–Nov 2 2004.
59. C. A. Rouff, J. L. Rash, and M. G. Hinchey. Experience using formal methods for specifying a multi-agent system. In *Proc. Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2000)*, Tokyo, Japan, 2000. IEEE Computer Society Press, Los Alamitos, Calif.
60. C. A. Rouff, W. F. Truszkowski, M. G. Hinchey, and J. L. Rash. Verification of NASA emergent systems. In *Proc. 9th IEEE International Conference on Engineering of Complex Computer Systems*, Florence, Italy, April 2004. IEEE Computer Society Press, Los Alamitos, Calif.

61. C. A. Rouff, W. F. Truszkowski, J. L. Rash, and M. G. Hinchey. A survey of formal methods for intelligent swarms. Technical Report TM-2005-212779, NASA Goddard Space Flight Center, Greenbelt, Maryland, 2005.
62. Melissa Savage and Manor Askenazi. Arborscapes: A swarm-based multi-agent ecological disturbance model. Working paper 98-06-056, Santa Fe Institute, Santa Fe, New Mexico, 1998.
63. Roy 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.
64. D. J. T. Sumpter, G. B. Blanchard, and D. S. Broomhead. Ants and agents: a process algebra approach to modelling ant colony behaviour. *Bulletin of Mathematical Biology*, 63(5):951–980, September 2001.
65. C. Tofts. Describing social insect behavior using process algebra. *Transactions on Social Computing Simulation*, pages 227–283, 1991.
66. W. F. Truszkowski, M. G. Hinchey, J. L. Rash, and C. A. Rouff. Autonomous and autonomic systems: A paradigm for future space exploration missions. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 2006 (to appear).
67. Walt Truszkowski, Mike Hinchey, James Rash, and Christopher Rouff. NASA's swarm missions: The challenge of building autonomous software. *IEEE IT Professional*, 6(5):47–52, September/October 2004.
68. Gerhard Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, Massachusetts, 1999.
69. F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3), September 2003.