Using Formal Methods and Agent-Oriented Software Engineering for Modeling NASA Swarm-Based Systems

Christopher A. Rouff
SAIC
Advanced Technologies and Solutions Business Unit
McLean, VA 22102
rouffc@saic.com

Michael G. Hinchey
Loyola College in Maryland
Computer Science Department
Baltimore, MD, USA
mike.hinchey@usa.net

Joaquin Peña and Antonio Ruiz-Cortés
University of Seville
Seville, Spain
{joaquinp,aruiz}@us.es

Abstract

NASA is conducting research on advanced technologies for future exploration 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 an engineering point of view, the complexity and emergent behavior of this kind of system is one of the main challenges that has to be overcome, 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, called agent-oriented software engineering, and formal methods. In this paper, we report on the main advances we have made towards modeling, 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 [31]. Traditional missions, reliant on one large spacecraft, are being replaced by missions with smaller collaborating spacecraft, analogous to swarms in nature [7]. This approach offers several advantages: the ability to send spacecraft to explore where traditional craft simply would be impractical, greater redundancy (and greater protection of assets), and reduced costs and risk, to name a few. 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.

Swarms [1] 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; 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. [2], who studied self-organization in social insects, state that "complex collective behaviors may emerge from interactions among individuals that exhibit simple behaviors" and describes 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 macro-level behavior, and the individual behaviors and local interactions as the micro-level behavior. Though swarm behaviors are the combination of often simple individual behaviors, when aggregated, they can form complex and unexpected behaviors.

Although this kind of system presents many advantages, since complex behaviors emerge from the definition of simple individual behaviors, reducing the effort in design, its unpredictability represents a problem since unexpected behaviors may cause the failure of a mission. We survey current NASA efforts towards exploiting intelligent swarm-based systems in the development of new paradigms of exploration missions, with assurances that they are operating safely. For this purpose, the main approaches that are being explored are: Agent-Oriented Software Engineering (AOSE) and Formal Methods.

2. Problems modeling swarm-based systems

Complexity is one of the main problems of engineering software systems, and is especially relevant when tackling the development of swarm-based missions. Several authors agree that the complexity is a consequence of interactions [13, 17]: *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 because of emergent behavior. The reasons for the emergence can be traced to two features present in these systems: self-adaptation, and self-organization [10]. 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.

Jennings [13] adapts the three main principles to manage complexity proposed by Booch in the OO context [3] to AOSE: Abstraction, Decomposition and Organization/Hierarchy[1]:

Abstraction: defines simplified models of the systems that emphasize some details while avoiding others.

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

Composition: consists in identifying and managing the inter-relationships between the various subsystems in the problem.

In addition, automation and reuse have been presented as two important principles to overcome complexity [9, 14]:

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

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

2.1. AOSE and complexity

From the engineering point of view, these principles imply a set of requirements for AOSE methodologies: modeling artifacts, software process, and techniques to manage models [26].

1 Hereafter we call it *Composition* in order to differentiate it from other uses of the term organization in agent-based systems

In modeling artifacts, the first elements we must consider are roles, since this is the concept that allows us to focus on interactions, the main source of complexity. In addition, roles also allow us to decompose an agent by its responsibilities, which represents the decomposition principle. In addition to roles, artifacts for abstracting interactions and organizations are needed, as well as enabling 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 of models in several layers, models devoted to maintaining traceability across layers are also needed.

From the software process point of view, bottom-up and top-down approaches are needed to enable us to go through layers completing models. Using both bottom-up and top-down perspectives permits us to obtain intermediate layers which link micro-level, that is to say lower layers that represent agent behavior, with macro-level, that is to say higher layers that represent the organization/emergent behavior. In addition, the bottom-up software process provides a means for reusing models, an important principle for dealing 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 development, and composition of models into higher-level models (abstraction) helps us to perform a bottom-up development.

2.2. Formal Methods and complexity

Formal methods are proven approaches for assuring the correct operation of complex interacting systems [11]. 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.

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

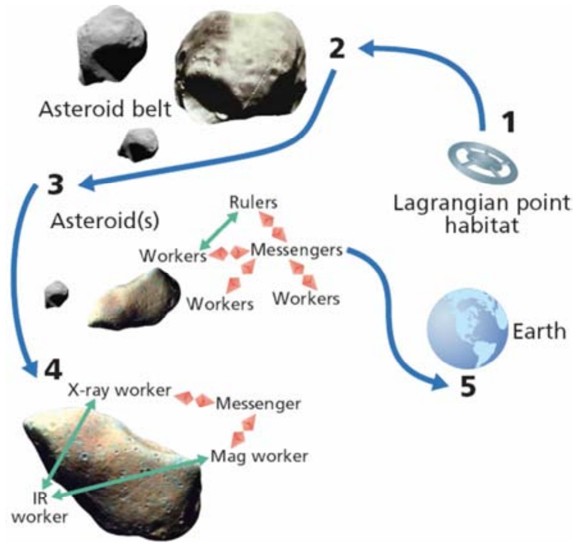

Figure 1. ANTS Mission Concept

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.

Verifying emergent behavior is an area that has been addressed very little by formal methods, although there has been some work done in this area by computer scientists analyzing biological [29] and robotic [32] systems. 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 [5] will launch a swarm of autonomous pico-class (approximately 1kg) spacecraft to explore the asteroid belt for asteroids with certain scientific properties. Figure 1 gives an overview of the ANTS mission [30]. In this mission, a transport ship will travel to 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.

To implement this mission, a heuristic approach is being considered that provides for a social structure to the spacecraft that uses a hierarchical behavior in some respects analogous to colonies or swarms of insects, with some spacecraft directing others. Artificial intelligence technologies such as genetic algorithms, neural nets, fuzzy logic, and onboard 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. 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. 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 mission control on Earth.

The swarm will form sub-swarms, each under the control of a ruler, which contains models of the types of science to be pursued. The ruler will coordinate workers, which use their individual instruments to collect data on specific asteroids and feed 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.

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 [27]. 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 would be suitable for specifying and verifying swarm-based missions and their emergent behavior [27]. 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, and algorithm support.

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 [12] and WSCCS [29], X-Machines [16], and Unity Logic [4]. CSP was chosen as a baseline specification method because the team has had significant experience and success [28] 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.

5. Modeling using MaCMAS

MaCMAS is the AOSE methodology that we use for modeling swarm-based systems and is based on previously developed concepts [22][2]. It is specially tailored to model complex Multiagent Systems covering all the requirements for tackling complexity shown previously and to the best of our knowledge is the only one that covers 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 an appropriate tool for engineering these kinds of systems, and has been successfully applied by other methodologies, e.g., [18, 21]. It shows that a MAS/swarm organization can be observed from two viewpoints [33]:

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.

2 See http://james.eii.us.es/MaCMAS/ for Further details and case studies using this methodology.

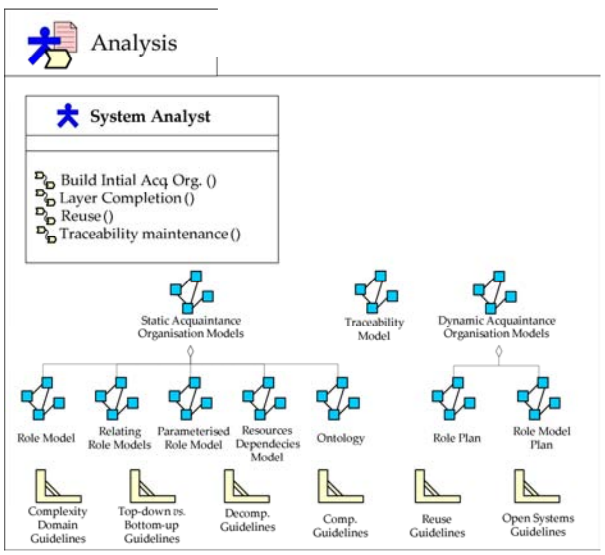

Figure 2. Acquaintance analysis discipline

Both views are intimately related, but 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 [33]. Then, we can conclude that any acquaintance organization can be modeled orthogonally to its structural organization [15].

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, but complementary since they provide a graphical representation of the system that improves the understanding of the formal specifications.

Figure 3, 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 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 meth-

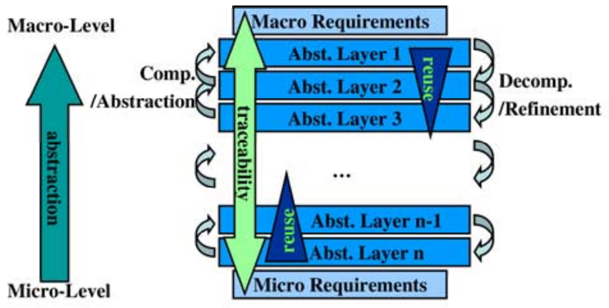

Figure 3. Overview of the structure of MaCMAS models

ods. 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 2, we summarize all the models, activities and guidelines included in MaCMAS. From all the models proposed in this methodology, the most important are:

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) [23]. mRIs are used to abstract the acquaintance relationships amongst roles in the system.

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].

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].

c) Traceability view: 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*.

The software process of MaCMAS starts with requirements documented using a goal-directed approach [8]. After applying MaCMAS we obtain the acquaintance organization of the MAS thus enabling us 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, e.g. NASA swarm-based missions.

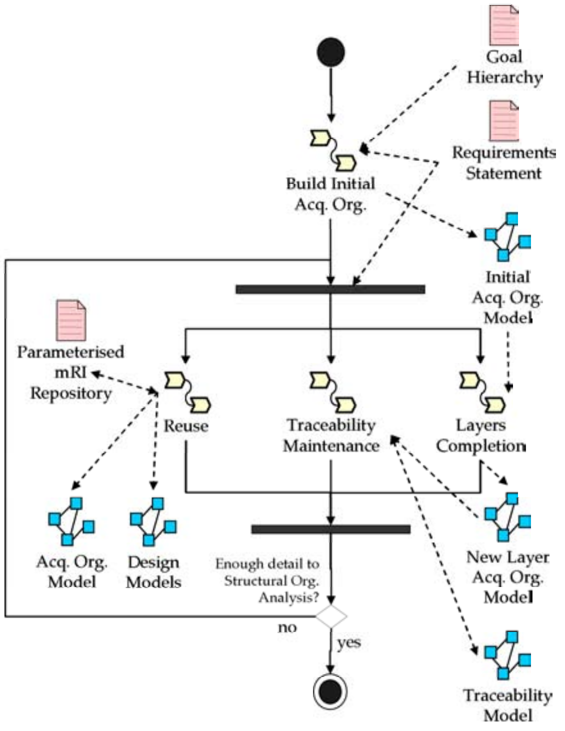

Figure 4. Acquaintance analysis process

The software process of MaCMAS is described as a set of abstract Software Process Engineering Metamodel (SPEM) work definitions which can be instantiated by any work definition that produces the same work products (cf. [19] the SPEM specification). In Figure 4, we show our process by means of several general process components in the 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: an initial set of role models which provide an initial understanding of the system to be built.

Layer Completion: a new acquaintance organization model due to the composition or the decomposition of a model(s) developed in a previous iteration or in the *build initial acquaintance organization work definition*.

Traceability Maintenance: 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 behaviors with micro-level behaviors, and helps us to control the emergent behavior.

Reuse: 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.

5.1. Modeling autonomous and autonomic systems

MaCMAS can be used to model autonomous and autonomic properties of swarm-based systems as required by NASA missions. To exemplify the models of MaCMAS and how they are applied to model both kinds of properties, we use the case study described in Section 3.

After applying all the software process of MaCMAS to the case study, we obtain the traceability diagram of Figure 5. 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 subregions. 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 5 we identify two of these packages, which group the mRIs used in the example that follows.

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.

5.2. Modelling Multiagent Systems Product Lines

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

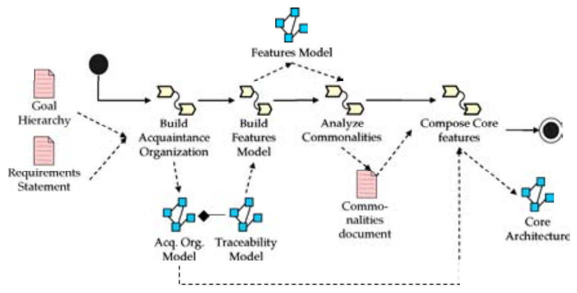

Figure 6. Overview of the process for building the core architecture of a MAS-PL

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. [6], 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 [25].

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

For enabling a product line, one of the important activities to be performed is to identify a core architecture. In Figure 6, we show the SPEM definition of the software process of our approach to build the core architecture: build acquaintance organization, build features model, analyze commonalities, and compose code features.

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

5.3. Modeling evolving systems using MaCMAS

MaCMAS is also able to model evolving systems as required by swarm-based NASA missions [24]. MaCMAS is based 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

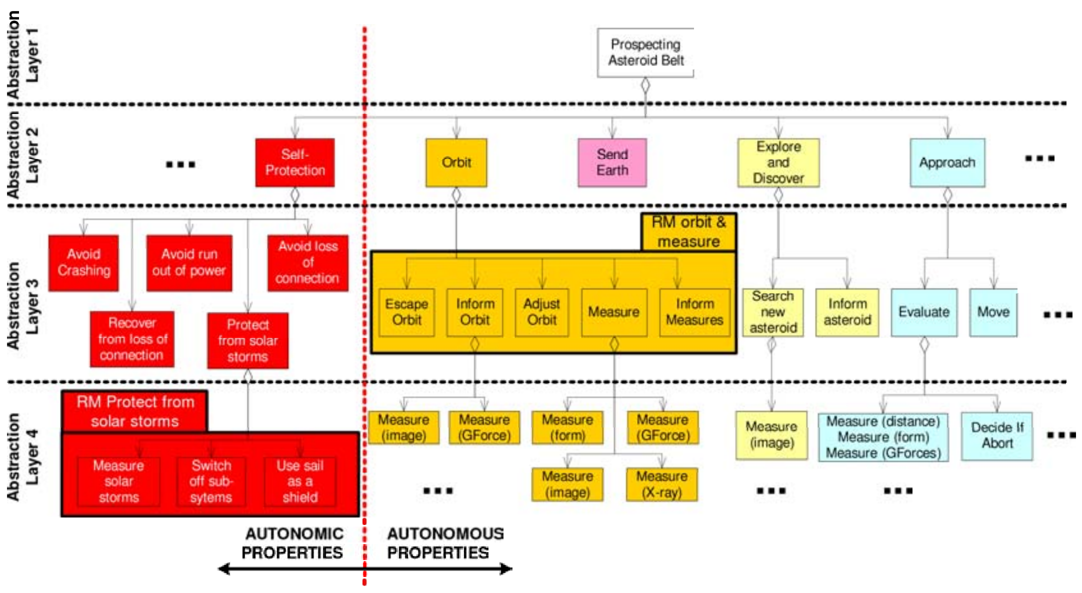

Figure 5. Traceability model of ANTS

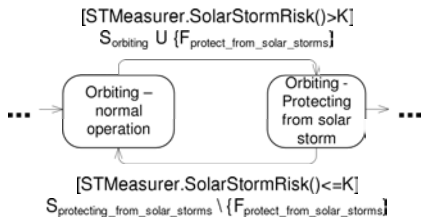
[STMeasurer.SolarStormRisk()>K]
S_orbiting U {F_protect_from_solar_storms}
Orbiting – normal operation
Orbiting - Protecting from solar storm
[STMeasurer.SolarStormRisk()<=K]
S_protecting_from_solar_storms \ {F_protect_from_solar_storms}

Figure 7. Evolution plan of our case study

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

In Figure 7, 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 storms” depending on whether or not the swarm is at risk from a solar storm.

6. Conclusions and Future work

Complexity is a big challenge for 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 system is not feasible using only UML-based tailored models, or only formal methods, but requires a new set of formal techniques along with a tailored set of models and software processes. As shown, we have had to base our approach on techniques from many fields ranging from, but not limited to, the use of several formal methods, autonomic computing, swarm-based systems, and new applications of the software product line approach.

Acknowledgements

This work 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. This work described in this paper was performed while Mike Hinchey was with NASA Goddard Space Flight Center.

References

[1] G. Beni and J. Want. Swarm intelligence. In *Proc. Seventh Annual Meeting of the Robotics Society of Japan*, pages 425–428, Tokyo, Japan, 1989. RSJ Press.

[2] E. Bonabeau, G. Théraulaz, J. Deneubourg, S. Aron, and S. Camazine. Self-organization in social insects. *Trends in Ecology and Evolution*, 12:188–193, 1997.

[3] G. Booch. *Object-Oriented Design with Applications*. Benjamin/Cummings, Redwood City, CA, 1990.

[4] K. Mani Chandy and Misra J. *Parallel Program Design: A Foundation*. Addison-Wesley Publishing Company, 1988.

[5] P.E. Clark, S.A. Curtis, and M.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.

[6] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, August 2001.

[7] S.A. Curtis, J. Mica, J. Nuth, G. Marr, M.L. Rilee, and M.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.

[8] A. Dardenne, A. van Lamsweerde, and S.Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[9] D.F. D'Souza and A.C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison–Wesley, Reading, Mass., 1999.

[10] J. Fromm. *The Emergence of Complexity*. Kassel university press, 2004.

[11] 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, Nov. 2001. NASA Goddard Space Flight Center, IEEE Computer Society Press.

[12] C.A.R. Hoare. *Communicating sequential processess*. Prentice Hall, 1985.

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

[14] 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 Int'l Workshop Engineering Societies in the Agents World*, volume 3071 of *Lecture Notes in Computer Science*, pages 258–274. Springer, 2004.

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

[16] W. Michael and L. Holcombe. X-Machines as a basis for system specification. *Software Engineering*, 3(2):69–76, 1988.

[17] J. Odell. Agents and complex systems. *Journal of Object Technology*, 1(2):35–45, July-August 2002.

[18] J. Odell, H. Parunak, and M. Fleischer. The role of roles in designing effective agent organisations. In A. Garcia, C. Lucenaand, F. Zambonelliand, A. Omiciniand, and J. Castro, editors, *Software Engineering for Large-Scale Multi-Agent Systems*, number 2603 in LNCS, pages 27–28. Springer–Verlag, 2003.

[19] OMG. Software process engineering metamodel, version 1.1, 2005. Available at `http://www.omg.org/technology/documents/formal/spem.htm`.

[20] Object Management Group (OMG). Unified modeling language: Superstructure. version 2.0. Final adopted specification ptc/03–08–02, OMG, August 2003. www.omg.org.

[21] 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 Int'l Conference on Autonomous Agents*, pages 100–101, Montreal, Canada, 2001. ACM Press.

[22] 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.

[23] J. Peña, R. Corchuelo, and J. L. Arjona. A top down approach for mas protocol descriptions. In *ACM Symposium on Applied Computing SAC'03*, pages 45–49, Melbourne, Florida, USA, 2003. ACM Press.

[24] J. Peña, 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 Int'l Workshop on System/Software Architectures (IWSSA'06)*, page to be published, Nevada, USA, 2006. CSREA Press.

[25] J. Peña, M. G. Hinchey, and Antonio Ruíz-Cortes. Multiagent system product lines: Challenges and benefits. *Communications of the ACM*, December 2006.

[26] J. Peña, R. Levy, and R. Corchuelo. Towards clarifying the importance of interactions in agent-oriented software engineering. *Int'l Iberoamerican Jrnl of AI*, 8(25):19–28, 2005.

[27] C. Rouff, M. Hinchey, W. Truszkowski, and J. Rash. Experiences applying formal approaches in the development of swarm-based space exploration systems. *Int'l Journal of on software Tools for Technology Transfer. Special Issue on Formal Methods in Industry. (submitted)*, 2006.

[28] C. A. Rouff, W. F. Truszkowski, M. G. Hinchey, and J. L. Rash. Verification of NASA emergent systems. In *Proc. 9th IEEE Int'l Conference on Engineering of Complex Computer Systems*, Florence, Italy, April 2004.

[29] C. Tofts. Describing social insect behavior using process algebra. *Transactions on Social Computing Simulation*, pages 227–283, 1991.

[30] W. Truszkowski, M. Hinchey, J. Rash, and C. Rouff. NASA's swarm missions: The challenge of building autonomous software. *IEEE IT Professional*, 6(5):47–52, Sept./Oct. 2004.

[31] 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).

[32] Alan F.T. Winfield, Jin Sa, Mari-Carmen Fernandez-Gago, Clare Dixon, and Michael Fisher. On formal specification of emergent behaviours in swarm robotic systems. *Int'l Journal of Advanced Robotic Systems*, 2(4):363–370, Dec. 2005.

[33] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. *ACM Trans. on Software Engineering and Methodology*, 12(3), Sept. 2003.