Challenges of Developing New Classes of NASA Self-Managing Missions

M. G. Hinchey, J. L. Rash, W. F. Truszkowski
NASA GSFC
Greenbelt, MD, USA
michael.g.hinchey@nasa.gov

C. A. Rouff
SAIC
McLean, VA 22102
rouffc@saic.com

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

Abstract

NASA is proposing increasingly complex missions that will require a high degree of autonomy and autonomicity. These missions pose hereto unforeseen problems and raise issues that have not been well-addressed by the community. Assuring success of such missions will require new software development techniques and tools. This paper discusses some of the challenges that NASA and the rest of the software development community are facing in developing these ever-increasingly complex systems. We give an overview of a proposed NASA mission as well as techniques and tools that are being developed to address autonomic management and the complexity issues inherent in these missions.

1 An Historic Problem

The realization that software development has lagged greatly behind hardware is hardly a new one [2]. Brooks, in a widely-quoted article [3], warns of complacency in software development. He stresses that, unlike hardware development, we cannot expect to achieve great advances in productivity in software development unless we concentrate on more appropriate development methods. Harel, in an equally influential paper, written as a rebuttal to Brooks [6] points to developments in CASE and visual formalisms [5] as potential "bullets" (solutions).

Clearly there have been significant advances in software engineering tools, techniques, and methods, since the time of Brooks' and Harel's papers. In many cases, however, the advantages of these developments have been mitigated by corresponding increases in demand for greater, more complex, functionality, stricter constraints on performance and reaction times, and attempts to increase productivity and reduce costs, while simultaneously pushing systems requirements to their limits. NASA, for example, continues to build more and more complex systems, with impressive functionality, and increasingly autonomous behavior. In the main, this is essential. NASA missions are pursuing scientific discovery in ways that will require automated, autonomous systems. While manned exploration missions are clearly in NASA's future (such as the Exploration Initiative's plans to return to the moon and put man on Mars), for reasons we will explain below, several current and future NASA missions necessitate autonomous behavior by unmanned spacecraft [9].

We will describe some of the challenges for software engineering emerging from new classes of complex systems being developed by NASA and others. We will discuss these with reference to a NASA concept mission that is exemplary of many of these new systems. Then, in Section 3 we will present some techniques that we are addressing, including autonomic management, which may contribute to finding the Silver Bullet.

2 Challenges of Future NASA Missions

Future NASA missions will exploit new paradigms for space exploration, heavily focused on the (still) emerging technologies of autonomous and autonomic systems. Traditional missions, reliant on one large spacecraft, are being replaced with missions that involve several smaller spacecraft, operating in collaboration, analogous to swarms in nature. This 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. Planned missions entail, for example, the use of several unmanned autonomous vehicles (UAVs) flying approximately one meter above the surface of Mars, which will cover as much of the surface of Mars in a few seconds as the now famous Mars rovers did in their entire time on the planet.

These new approaches to exploration missions simultaneously pose many challenges. The missions will be unmanned and necessarily 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 problems and undesirable situations must be made *in situ* rather than from ground control on Earth. The degree of autonomy that such missions will possess would require a prohibitive amount of testing to ensure correct behavior. Furthermore, learning and continual improvements in performance by each individual platform will mean that emergent behavior patterns simply cannot be fully predicted.

2.1 ANTS: A NASA Concept Mission

One of these future NASA missions is the Autonomous Nano-Technology Swarm (ANTS) mission, which will involve the launch of a swarm of autonomous pico-class (approximately 1kg) spacecraft that will explore the asteroid belt for asteroids with certain characteristics. Figure 1 gives an overview of the ANTS mission [17]. A transport ship, launched from Earth, will travel to a point in space where gravitational forces on small objects (such as spacecraft) are all but negligible. From this point, termed a Lagrangian, 1000 spacecraft that have been assembled *en route* from Earth, will be launched into the asteroid belt. Because of the nature of the asteroid belt, spacecraft will experience a significant risk of collision with asteroidal bodies. Further, since the individual spacecraft have no onboard propulsion, and can maneuver only by using solar sails, collisions between spacecraft are possible during exploration operations around asteroids, so that 60% to 70% of them may 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. As a result, spacecraft must cooperate and coordinate using a hierarchical social behavior analogous to colonies or swarms of insects, with some spacecraft directing others. To implement this mission, a heuristic approach is being considered that provides for a social structure based on the notion of a hierarchy among the spacecraft. 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.

2.2 Problematic Issues

2.2.1 Size and Complexity

While the use of a swarm of miniature spacecraft is essential for the success of ANTS (by enabling many points of simultaneous observation and data collection), it also

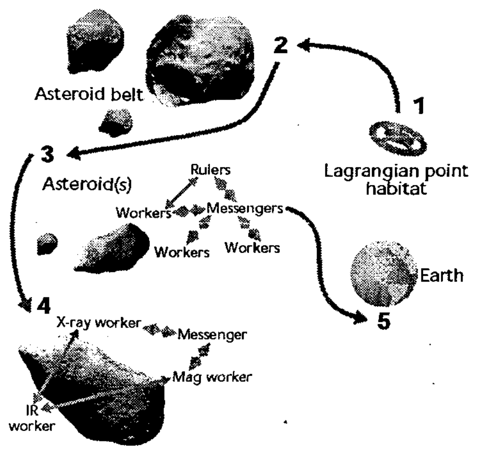

Figure 1. NASA's Autonomous Nano Technology Swarm (ANTS) mission scenario.

poses several problems in terms of adding significantly to the complexity of the mission. The mission will launch 1000 pico-class spacecraft. Even with a possible loss rate of 60% to 70%, we expect to have several hundred surviving spacecraft, all of which must be kept organized in effective groups that will collect science data and make decisions as to which asteroids warrant further investigation.

2.2.2 Emergent Behavior

In swarm-based systems, a group of interacting agents (often homogeneous or near homogeneous) are developed to take advantage of their emergent behavior. In these systems, each of the agents is given certain parameters that it tries to maximize. Intelligent swarms [1] involve the use of swarms of simple intelligent agents. Swarms have no central controller: they are self-organizing based on the emergent behaviors of the simple interactions. There is no external force directing their behavior and no one agent has a global view of the intended macroscopic behavior. Though current NASA swarm missions are not true swarms as described above, they do have many of the same attributes and may exhibit emergent behavior. In addition, there are a number of government projects that are looking at true swarms to accomplish complex missions.

2.2.3 Autonomy

Autonomous operation is essential for the success of the ANTS mission. Round trip communications delays of up to 40 minutes, and limited bandwidth on communications links with Earth, mean that control from the ground is impossible. The data concerning a swarm emergency situation (e.g., a projected collision between a spacecraft and an asteroid or between two spacecraft in the swarm) would already be stale and effectively unusable when finally received by ground control personnel. Furthermore, the actual swarm situation would likely have changed so much after the additional signal propagation delay on any instructions transmitted back to the swarm that the attempt by ground control to handle the emergency would be invalid and ineffective.

Autonomy implies an absence of centralized control. Individual ANTS spacecraft will operate autonomously under the control of that subgroup's *ruler*. That ruler will itself autonomously make decisions regarding asteroids of interest, and formulate plans for continuing the mission of collecting science data. The success of the mission is predicated on the validity of the plans generated by the rulers, and requires that the rulers generate sensible plans that will collect valid science data, and then make valid informed decisions.

That autonomy is possible is not in doubt. What is in doubt is that autonomous systems can be relied upon to operate correctly, in particular in the absence of a full and complete specification of what is required of the system. Our goal is to address this crucial issue.

2.2.4 Testing and Verification

One of the most challenging aspects of using swarms is how to verify that the emergent behavior of such systems will be proper and that no undesirable behaviors will occur. In addition to emergent behavior in swarms, a large number of concurrent interactions occur between the agents that make up the swarms. These interactions can also contain errors, such as race conditions, that are very difficult to detect until they occur. Once they do occur, it can also be very difficult to recreate the errors, since they are usually data and time dependent.

3 Some Potentially Useful Techniquess

3.1 Autonomicity

Autonomy may be considered as having the properties of self-governance and self-driven-ness, i.e., control over one's goals. Autonomicity is having the ability to self-manage through properties such as self-configuring, self-healing, self-optimizing, and self-protecting [4, 10, 14]. These are achieved through other self-properties such as self-awareness (including environment awareness), self-monitoring, and self-adjusting [15].

Increasingly, self-management is seen as the only viable way forward to cope with the ever increasing complexity of systems. From one perspective, self-management may be considered a specialization of self-governance, i.e., autonomy where the goals/tasks are specific to management roles [16]. Yet from the wider context, an autonomic element (AE), consisting of an autonomic manager and managed component, may still have its own specific goals, but also additional responsibility of management tasks, in particular to the wider system environment.

It is envisaged that in an autonomic environment, the AEs communicate to ensure a managed environment that is reliable and fault tolerant and meets high level specified policies (with an overarching vision of system-wide policy-based self-management). This may result in AEs monitoring or "watching out for" other AEs. In terms of autonomy and the concern of undesirable emergent behavior, an environment that dynamically and continuously monitors can assist in detecting race conditions and reconfiguring to avoid damage (self-protecting, self-healing, self-configuring, etc.). As such, Autonomicity becoming mainstream in the industry can only assist to improve techniques, tools, and processes for autonomy [14].

3.2 Hybrid Formal Methods

To overcome the complexity and many other issues in developing future NASA missions, formal specification techniques and formal verification will need to play vital roles. NASA is currently investigating the use of formal methods and formal techniques for verification and validation of these classes of mission. The primary role of formal methods will be in the specification and analysis of forthcoming missions, with a further role in software assurance and proof of correctness of the behavior of a swarm, whether or not this behavior is emergent (as a result of composing a number of interacting entities, producing behavior that was not foreseen). Formal models derived may also be used as the basis for automating the generation of much of the code for the mission [8]. A current project, *Formal Approaches to Swarm Technologies (FAST)*, is investigating the requirements of appropriate formal methods for use in such missions, and is beginning to apply these techniques to specifying and verifying parts of the ANTS mission.

Hybrid, or integrated, formal approaches have been very popular in specifying concurrent and agent-based systems. No doubt this is due to the monolithic systems that most formal methods were developed to specify and verify. Hybrid approaches often combine a process algebra or logic-based approach with a model-based approach. The process algebra or logic-based approach allows for easy specification of concurrent systems, while the model-based approach provides strength in specifying the algorithmic part of a system.

As part of the FAST project, new hybrid formal methods are being investigated to address complex NASA missions including swarms.

3.3 Automatic Programming

For many years, automatic programming has referred, primarily, to the use of very high-level languages to describe solutions to problems, which could then be translated down and expressed as code in more familiar programming languages. Parnas [11] implies that the term is glamorous, rather than having any real meaning, precisely because it is the solution that is being specified,rather than the problem that must be solved.

Autonomous and autonomic systems, exhibiting complex emergent behavior cannot, in general, be fully specified at the outset. The roles and behaviors of the system will vary greatly over time. While we may try to write specifications in such a manner that constrain the system, it is clear that not all behavior can be specified in advance. This is particularly true of systems exhibiting self-management The classes of system we are discussing will often require code to be generated, or modified, during execution. Consequently, automatic code generation will be *required.*

Several tools already exist that successfully generate code from a given model. Unfortunately, many of these tools have been demonstrated to generate code, portions of which are never executed, or portions of which cannot be justified from either the requirements or the model. Moreover, existing tools do not and cannot overcome the fundamental inadequacy of all currently available automated development approaches, which is that they include no means to establish a provable equivalence between the requirements stated at the outset and either the model or the code they generate. That is why, we believe, future approaches to automatic code generation, in particular for autonomic systems, must be based on Formal Requirements-Based Programming.

3.4 Formal Requirements Based Programming

Requirements-Based Programming refers to the development of complex software (and other) systems, where each stage of the development is fully traceable back to the requirements given at the outset.

Requirements-Based Programming ensures that there is a direct mapping from requirements to design, and that this design (model) may then be used as the basis for automatic code generation. In fact, Formal Requirements-Based Programming, coupled with a graphical representation for system requirements (e.g., UML use cases) possesses the features and advantages of a visual formalism described by Harel [5].

R2D2C, or Requirements-to-Design-to-Code [7, 12], is a NASA patent-pending approach to Requirements-Based Programming. In R2D2C, engineers (or others) may write specifications 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.

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.

3.5 Tool support

John Rushby [13] argues that tools are not the *most* important thing about formal methods, they are the *only* important thing about formal methods. Although we can sympathize, we do not support such an extreme viewpoint. Formal methods would not be practical without suitable representation notations, proof systems (whether automated and supported by tools, or not), a user community, and evidence of successful application.

We do agree, however, that tool support is vital, and not just for formal methods. Structural design methods "took off" when they were "standardized", in the guise of UML. But it was only with the advent of tool support for UML that it became widely used. The situation is analogous to high level programming languages: while the community was well convinced of their benefits, it was only with the availability of commercial compilers that they became widely used.

Tools are emerging for the development of complex agent-based systems such as Java-based Aglets and tools for autonomic systems. For automatic code generation and formal Requirements-Based Programming to be practical, the development community will need commercial-quality tools. Similarly, the autonomic management of complex systems will require adequate tool support.

4 Conclusion

We have re-iterated several problems facing the software development community. Unfortunately, while well known for many decades, these issues still prevail. More importantly, new classes of systems—namely complex, highly-distributed autonomous systems and their autonomic management—will pose many other challenges, which yet remain unaddressed.

We have described one concept system that exemplifies forthcoming classes of complex autonomous systems that NASA, and others, are developing. These pose hereto unforeseen problems and raise issues that have not been well-addressed by the community. We have mentioned some techniques under development by NASA that may be fruitful in addressing these problems.

Acknowledgements

This work is funded in part by the NASA Office of Safety and Mission Assurance, under its Software Assurance Research Program project *Formal Approaches to Swarm Technologies* (FAST), administered by the NASA IV&V Facility; by the Office of Technology Transfer, NASA Goddard Space Flight Center; by the NASA Software Engineering Laboratory, NASA Goddard Space Flight Center; and the University of Ulster Centre for Software Process Technologies (CSPT), funded by Invest NI through the Centres of Excellence Programme under the European Union Peace II initiative.

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] J. P. Bowen and M. G. Hinchey. *High-Integrity System Specification and Design*. FACIT Series. Springer-Verlag, London, UK, 1999.

[3] F. P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, April 1987.

[4] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

[5] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

[6] D. Harel. Biting the silver bullet: Toward a brighter future for system development. *IEEE Computer*, 25(1):8–20, January 1992.

[7] M. G. Hinchey, J. L. Rash, and C. A. Rouff. Enabling requirements-based programming for highly dependable complex parallel and distributed systems. In *Proc. 1st International Workshop on Distributed, Parallel and Network Applications (DPNA 2005)*, Fukuoka, Japan, 20–22 July 2005. IEEE Computer Society Press.

[8] 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*, Greenbelt, Maryland, USA, 4–5 April 2005. IEEE Computer Society.

[9] M. G. Hinchey, J. L. Rash, W. F. Truszkowski, C. A. Rouff, and R. Sterritt. You can't get there from here! problems and potential solutions in developing new classes of complex systems. In *Proc. Eighth International Conference on Integrated Design and Process Technology (IDPT)*, Beijing, China, 13–17 June 2005. The Society for Design and Process Science.

[10] P. Horn. Autonomic computing: IBM's perspective on the state of information technology. Technical report, IBM Corporation, October 15, 2001.

[11] D. L. Parnas. Software aspects for strategic defense systems. *American Scientist*, November 1985.

[12] 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.

[13] J. Rushby. Remarks, panel session on the future of formal methods in industry. In J. P. Bowen and M. G. Hinchey, editors, *Proc. 9th International Conference of Z Users, LNCS 967*, pages 239–241, Limerick, Ireland, September 1995. Springer-Verlag.

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

[15] R. Sterritt and D. W. Bustard. Autonomic computing—a means of achieving dependability? In *Proc. IEEE International Conference on the Engineering of Computer Based Systems (ECBS-03)*, pages 247–251, Huntsville, Alabama, USA, April 2003. IEEE Computer Society Press.

[16] R. Sterritt and M. G. Hinchey. Why computer based systems *Should* be autonomic. In *Proc. 12th IEEE International Conference on Engineering of Computer Based Systems (ECBS 2005)*, pages 406–414, Greenbelt, MD, April 2005.

[17] 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, September/October 2004.