Some Verification Issues at NASA Goddard Space Flight Center

Michael G. Hinchey[1] James L. Rash[2] and Christopher A. Rouff[3]

[1] Loyola College in Maryland
Department of Computer Science
Baltimore, MD 21210, USA
mhinchey@loyola.edu
[2] Advanced Architectures and Automation Branch
NASA Goddard Space Flight Center
Greenbelt, MD 20771, USA
james.l.rash@nasa.gov
[3] Advanced Technology Laboratories
Lockheed Martin Corporation
Arlington, VA 22203, USA
crouff@atl.lmco.com

Abstract. NASA is developing increasingly complex missions to conduct new science and exploration. Missions are increasingly turning to multi-spacecraft to provide multiple simultaneous views of phenomena, and to search more of the solar system in less time. Swarms of intelligent autonomous spacecraft, involving complex behaviors and interactions, are being proposed to accomplish the goals of these new missions. The emergent properties of swarms make these missions powerful, but simultaneously far more difficult to design, and to verify that the proper behaviors will emerge. In verifying the desired behavior of swarms of intelligent interacting agents, the two significant sources of difficulty are the exponential growth of interactions and the emergent behaviors of the swarm. NASA Goddard Space Flight Center (GSFC) is currently involved in two projects that aim to address these sources of difficulty. We describe the work being conducted by NASA GSFC to develop a formal method specifically for swarm technologies. We also describe the use of requirements-based programming in the development of these missions, which, it is believed, will greatly reduce development lead-times and avoid many of the problems associated with such complex systems.

1 Introduction

It is planned that future NASA missions will exploit exciting new paradigms for space exploration [22, 23]. To perform new science and exploration, traditional missions, reliant upon the use of a single large spacecraft, are being replaced with missions that will involve several smaller spacecraft. These new missions will behave as a "system of systems," operating in collaboration, analogous to swarms in nature [11].

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 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 approximately 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; constellations of satellites flying in formation; and, the use of miniaturized pico-class spacecraft to explore the asteroid belt, where heretofore it was impossible to send exploration craft without high likelihood of loss [18].

However, these new approaches to exploration missions simultaneously pose many challenges. The missions will be unmanned and necessarily highly autonomous. They will also exhibit all of the properties of *autonomic* systems, being self-protecting, self-healing, self-configuring, and self-optimizing [12]. 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. Furthermore, learning and continual improvements in performance will mean that emergent behavior patterns simply cannot be fully predicted [19].

2 Formal Approaches to Swarm Technologies

These missions are orders of magnitudes more complex than the traditional missions and verifying these new types of missions will be impossible using current techniques. New verification methods will be needed to address the added complexity resulting from the nondeterminate nature of these systems as well as the emergent behavior of swarms. To support the level of assurance that NASA missions require, formal specification techniques and formal verification will play vital roles in the future development of NASA space exploration missions. The role of formal methods will be in the specification and analysis of forthcoming missions, enabling software assurance and proof of correctness of the behavior of the 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 to further reduce the probability of adding new errors during coding.

To address the challenge in verifying the above missions a NASA project, *Formal Approaches to Swarm Technology* or *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 a future NASA swarm-based mission.

2.1 FAST

As part of the FAST project, the planned ANTS (Autonomous Nano Technology Swarm) mission is being used as an example swarm-based mission. The ANTS sub-mission PAM (Prospecting Asteroid Mission) will involve the launch of 1000 picoclass satellites into the asteroid belt [11, 21]. Many of these will be lost on first launch, others

through collisions with asteroids and with other ANTS spacecraft. The surviving spacecraft (approximately 30% to 40%) will form subswarms under the control of a leader or *ruler*. *Worker* spacecraft will carry individual instruments (e.g., a magnetometer, x-ray, gamma-ray, visible/IR, neutral mass spectrometer, etc.) which will be used to collect various types of data. Based on these data, the ruler will determine which asteroids are worthy of further investigation. Because of distances, low bandwidth, and roundtrip delays in communication with Earth, the mission will be required to operate more or less autonomously. It must also be able to recover from collisions, loss of instruments, loss of rulers or *messengers* (used to facilitate communication between spacecraft and/or with ground control), in addition to solar storms, whereby charged particles from the Sun can damage spacecraft and/or the solar sails (panels) that they use to obtain power from the sun.

2.2 Properties of an Appropriate Formal Method for Intelligent Swarms

An effective formal method must be able to predict the emergent behavior of 1000 agents operating as a swarm, as well as the behavior of the individual agent. Crucial to the mission will be the ability to modify operations autonomously to reflect the changing nature of the mission. For this, the 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 determination of which instruments will be needed, at what location, with what goals, etc.

Most importantly, the formal specification must allow for verification, and for the analysis of various properties and conditions within the evolving system. The ANTS mission details are still being determined and are constantly changing as more research is conducted. The formal specification technique employed must be sufficiently flexible to allow for efficient changes and re-prediction of emergent behavior.

Bearing all of this in mind, the following list summarizes the properties necessary for effective specification and emergent behavior prediction of the ANTS swarm and other swarm-based missions, and looks to existing formal methods to provide some of the desired properties [10].

Process representation: Processes can be specified using the various manifestations of transition functions.

Reasoning: Other forms of possibly non-standard logics may need to be employed to allow for intelligent reasoning with uncertain and possibly conflicting information.

Choosing Action Alternatives: A means of expressing probabilities and frequencies of events (as in WSCCS) is most beneficial in choosing between different enabled actions. A modified version of the WSCCS ability may be used to supply an algebra for choosing between possible actions.

Asynchronous messaging: Asynchronous messaging will need to be supported, as this is the most common type of messaging in swarm applications. This is not a significant problem as most synchronous messaging is implemented via asynchronous "handshakes". There are variants of CSP and other process algebras that support asynchronous messaging, either by having all processes be receptive (as in Receptive Process Theory), or through infinite buffering as in ACSP.

Message buffering: Message buffering may be needed due to the possibly asynchronous nature of messaging between members of the swarm. Several asynchronous variants of CSP achieve this through infinite buffering.

Concurrent agent states for each spacecraft: This requirement is well supported by available process algebras.

Communication protocols between agents: Available process algebras are highly effective in this area.

Adaptability to programming: Any formal specification languages that are developed will need to keep in mind the ease of converting the formal specification to program code and as input to model checkers.

Determining whether goals have been met: The goals of each spacecraft are constantly under review. We will need to be able to specify a method by which the spacecraft will know when the goals have been met. A modification to X-Machines may be able to solve this since the goals could be tracked using X-Machines (effectively finite state machines with memory).

Method for determining new goals: Once goals are met, new goals must be formed. We need to be able to specify a method for forming these goals.

Model checking: Model checking will help to avoid semantic inconsistencies in the specifications. Notations employed will need to be suitable for use as input to a model checker.

Tracking Models: X-Machines have the ability to track the universe model in memory but we need a more robust way to detail what the model is, how it is created, and how it is modified.

Associating agent actions with priorities and frequencies: A suitable formal method requires a means of expressing the probability of certain actions being enabled, and the frequency with which this will occur.

Predicting emergent behavior: Current approaches are not robust enough for the purpose of predicting individual and swarm emergent behavior and will need to be enhanced by greater use of Probability, Markov Chains, and/or Chaos Theory.

3 An Integrated Formal Method

The requirements detailed above point to the need to employ multiple formal methods in order to provide both a sufficiently expressive specification notation (that can deal with concurrency, real-time constraints, data manipulation, goal-oriented operation, etc.) *and* to facilitate verification.

The FAST project has surveyed various formal methods examining them for application to swarm technologies, and more generally "systems of systems" [20]. Various formal specification notations have been applied to parts of the ANTS mission (such as it currently stands, realizing that much of it will change before its launch) to provide feedback on the appropriateness of various approaches.

The project has concluded, unsurprisingly, that no single formal method will be appropriate for dealing with such a complex mission, and has consequently been concentrating on blending together various notations to provide a sufficiently expressive notation [10]. Future work on the project will include developing support tools for the integrated notation, and developing verification techniques for swarm-based missions.

4 Formal Requirements-Based Programming

Requirements-Based Programming (RBP) has been advocated [4, 5] 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 [8]. 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 [2]. 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 high [7, 9]. 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.

4.1 R2D2C

In the R2D2C approach, 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 [13, 14], 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 (Figure 2). 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) [16]. We are also

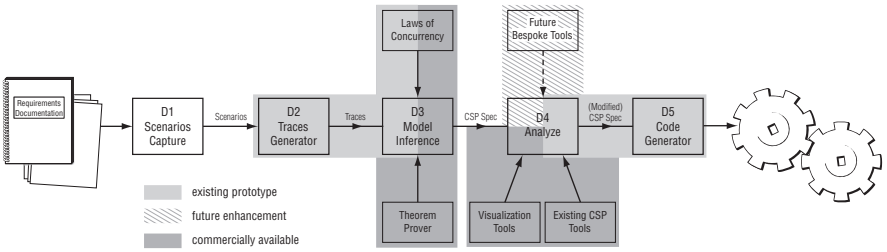

Fig. 1. The R2D2C approach and current status of the prototype.

experimenting with using it as a basis for an expert system verification tool, and as a means of capturing expert knowledge for expert systems [17].

4.2 Technical Approach

The R2D2C approach involves a number of phases, which are reflected in the system architecture described in Figure 1. The following describes each of these phases.

D1 Scenarios Capture: Engineers, end users, and others write scenarios describing intended system operation. The input scenarios may be represented in a constrained natural language using a syntax-directed editor, or may be represented in other textual or graphical forms.

D2 Traces Generation: Traces and sequences of atomic events are derived from the scenarios defined in D1.

D3 Model Inference: A formal model, or formal specification, expressed in CSP is inferred by an automatic theorem prover – in this case, ACL2 [15] – using the traces derived in phase 2. A deep[4] embedding of the laws of concurrency [6] in the theorem prover gives it sufficient knowledge of concurrency and of CSP to perform the inference. The embedding will be the topic of a future paper.

D4 Analysis: Based on the formal model, various analyses can be performed, using currently available commercial or public domain tools, and specialized tools that are planned for development. Because of the nature of CSP, the model may be analyzed at different levels of abstraction using a variety of possible implementation environments. This will be the subject of a future paper.

D5 Code Generation: The techniques of automatic code generation from a suitable model are reasonably well understood. The present modeling approach is suitable for the application of existing code generation techniques, whether using a tool specifically developed for the purpose, or existing tools such as FDR [1], or converting to other notations suitable for code generation (e.g., converting CSP to B [3] and then using the code generating capabilities of the B Toolkit).

[4] "Deep" in the sense that the embedding is semantic rather than merely syntactic.

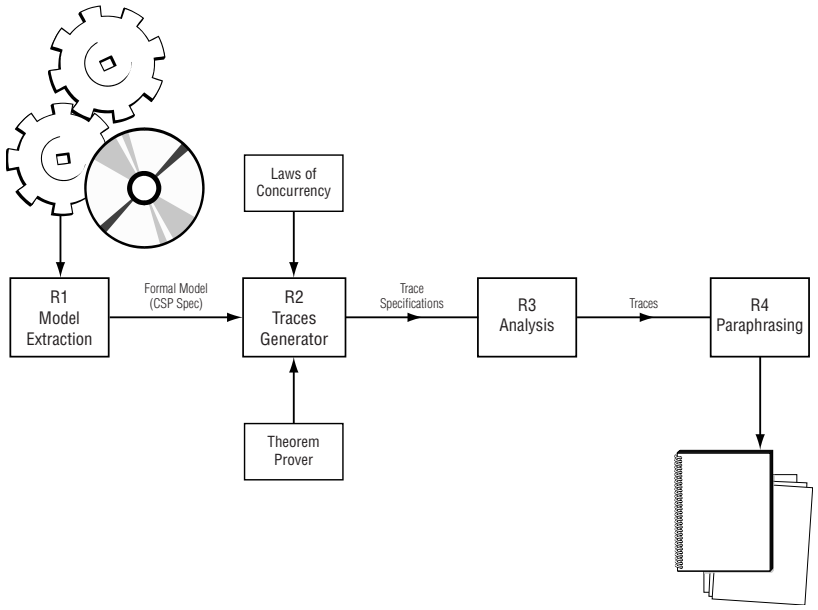

Fig. 2. Reverse engineering a system using R2D2C.

It should be re-emphasized that the "code" that is generated may be code in a high-level programming language, low-level instructions for (electro-) mechanical devices, natural-language business procedures and instructions, or the like. As Figure 2 illustrates, the above process may also be run in reverse:

R1 *Model Extraction*: Using various reverse engineering techniques [24], a formal model expressed in CSP may be extracted.

R2 *Traces Generation*: The theorem prover may be used to automatically generate traces based on the laws of concurrency and the embedded knowledge of CSP.

R3 *Analysis*: Traces may be analyzed and used to check for various conditions, undesirable situations arising, etc.

R4 *Paraphrasing*: A description of the system (or system components) may be retrieved in the desired format (natural language scenarios, UML use cases, etc.).

Paraphrasing, whereby more understandable descriptions (above and beyond existing documentation) of existing systems or system components are extracted, is likely to have useful application in future system maintenance for systems whose original design documents have been lost or systems that have been modified so much that the original design and requirements document do not reflect the current system.

5 Conclusion

NASA scientists and engineers are setting goals for future NASA exploration missions that will greatly challenge all of us. Future missions will exhibit levels of complexity that have never been seen before. They will be autonomous, pervasive, autonomic, surviving in harsh environments and with strict constraints on their behavior.

Swarm technologies will be widely used in such missions, exploiting the fact that more complex behaviors can emerge from the combination of several (in many cases hundreds, or even thousands) more simple individual behaviors. Swarms augur great potential, but pose a great problem for verification. Such systems simply cannot be adequately tested, both because of their inherent complexity, and the evolutionary nature of the systems due to learning.

The use of an appropriate formal specification notation is essential to facilitating formal verification. We have described the FAST project, which aims at addressing this issue.

In addition, NASA GSFC is exploring the use of Requirements-Based Programming to enable engineers, and others, to be fully involved in the development process, to ensure that we build the system we intended, and to appropriately exploit automatic programming, with the prospect of reducing costs and lead-times.

Acknowledgements

The *Formal Approaches to Swarm Technologies (FAST)* project is funded by the NASA Office of Systems and Mission Assurance (OSMA) through its Software Assurance Research Program (SARP), administered by the NASA IV&V Facility, Fairmont, WV.

The Requirements-Based Programming work described in this paper is supported in part by the Information Systems Division and the Technology Transfer Office at NASA Goddard Space Flight Center.

Research described in this paper was performed while Mike Hinchey was with NASA Software Engineering Laboratory, NASA Goddard Space Flight Center, and Chris Rouff was with SAIC. The work benefited from collaborations with Walt Truszkowski (NASA GSFC), Denis Gračanin (Virginia Tech), John Erickson (University of Texas at Austin), and Roy Sterritt (University of Ulster), amongst others.

References

1. *Failures-Divergences Refinement: User Manual and Tutorial*. Formal Systems (Europe), Ltd., 1999.
2. 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.
3. M. J. Butler. *csp2B : A Practical Approach To Combining CSP and B*. Declarative Systems and Software Engineering Group, Department of Electronics and Computer Science, University of Southampton, February 1999.

4. D. Harel. From play-in scenarios to code: An achievable dream. *IEEE Computer*, 34(1):53–60, 2001.
5. 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.
6. M. G. Hinchey and S. A. Jarvis. *Concurrent Systems: Formal Development in CSP*. International Series in Software Engineering. McGraw-Hill International, London, UK, 1995.
7. 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.
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)*. IEEE Computer Society Press, Los Alamitos, Calif., 3–8 April 2005.
9. 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"*, 29(2):246–256, January 2006.
10. M. G. Hinchey, J. L. Rash, C. A. Rouff, and W. F. Truszkowski. Requirements of an integrated formal method for intelligent swarms. In *Proc. FMICS 2005, 10th International Workshop on Formal Methods for Industrial Critical Systems*, Lisbon, Portugal, September 5–6 2005. ACM Press.
11. M. G. Hinchey, J. L. Rash, W. F. Truszkowski, C. A. Rouff, and R. Sterrit. Autonomous and autonomic swarms. In *Proc. The 2005 International Conference on Software Engineering Research and Practice (SERP'05)*, pages 36–42, Las Vegas, Nevada, USA, 27 June 2005. CSREA Press.
12. 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.
13. C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
14. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall International, Englewood Cliffs, NJ, 1985.
15. M. Kaufmann and Panagiotis Manolios and J Strother Moore. *Computer-Aided Reasoning: An Approach*. Advances in Formal Methods Series. Kluwer Academic Publishers, Boston, 2000.
16. 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.
17. 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.
18. 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.

19. C. Rouff, A. Vanderbilt, M. Hinchey, W. Truszkowski, and J. Rash. Properties of a formal method for prediction of emergent behaviors in swarm-based systems. In *Proc. 2nd IEEE International Conference on Software Engineering and Formal Methods*, Beijing, China, September 2004.
20. 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.
21. R. Sterritt, C. A. Rouff, J. L. Rash, W. F. Truszkowski, and M. G. Hinchey. Self-* properties in NASA missions. In *4th International Workshop on System/Software Architectures (IWSSA'05) in Proc. 2005 International Conference on Software Engineering Research and Practice (SERP'05)*, pages 66–72, Las Vegas, Nevada, USA, June 27 2005. CSREA Press.
22. 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.
23. 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: Applications and Reviews*, 36(3):279–291, May 2006.
24. H. J. van Zuylen. *The REDO Compendium: Reverse Engineering for Software Maintenance*. John Wiley and Sons, London, UK, 1993.