Sustainable and Autonomic Space Exploration Missions

Mike Hinchey,
James Rash and Walt Truszkowski
NASA
michael.g.hinchey@nasa.gov, james.l.rash@nasa.gov, walter.f.truszkowski@nasa.gov

Roy Sterritt,
University of Ulster,
r.sterritt@ulster.ac.uk

Christopher Rouff,
SAIC ACBU ,
rouffc@saic.com

Abstract

Visions for future space exploration have long term science missions in sight, resulting in the need for the sustainable missions. Survivability is a critical property of sustainable systems and may be addressed through autonomicity, an emerging paradigm for self-management of future computer-based systems. This paper examines some of the ongoing research efforts to realize these visions, with specific emphasis on Autonomic Policies.

1. Introduction

The vision for future Space Exploration Missions (SEMs) are "not looking at planting flags," and "not being able to go back for 100 years." [1] Systems that would take humans, via the ISS and the Moon, to Mars or to the asteroids, would be reusable systems (that might be nuclear in nature), with mission durations lasting upwards of 10 years. At this stage we are only beginning to "build the rail roads." [2] This vision requires *sustainable space capabilities* [1], in particular since it will mean the establishment of bases on the Moon for the eventual trip to Mars. [2]

Sustainable SEMs will have many dependant properties, not least of which is survivability. Survivable Systems are systems that are able to complete their mission in a timely manner, even if significant portions are compromised by attack or accident [3][4].

The case has been well presented in the literature for the need to create self-managing systems due to the complexity problem that causes the ever increasing total cost of ownership, or to provide the way forward to enable future pervasive and ubiquitous computation and communications [5][6][7][8]. Another aspect for self-management is to facilitate survivable systems [9][10]. To enable self-management (autonomicity), a system requires many self properties (self-* or selfware), such as self-awareness.

This paper looks at some ongoing research in the autonomic and autonomous systems area that will contribute to the creation of these survivable and sustainable exploration missions.

2. Requirements for sustainable systems

Computer-based systems are expected to be effective. This means that they serve a useful purpose when they are first introduced and continue to be useful as conditions change. From this perspective, they should also be survivable. Decisions and directions taken by the system automatically without real-time human intervention are autonomous decisions. Responses taken automatically by a system without real-time human intervention are autonomic responses [11]. The NASA view of autonomic and autonomous is slightly different from this general systems view. NASA views Autonomy as indicating without assistance from ground control, and as such this could have the astronaut in the loop.

Many branches of computer science research and development will contribute to the progress in this area. Research on dependable systems should be especially influential, as dependability covers many relevant system properties such as reliability, availability, safety, security, survivability and maintainability [13],[14].

Figure 1 highlights the fact that when the mission requirements are being established there will be intrinsic survivability requirements underpinning the mission.

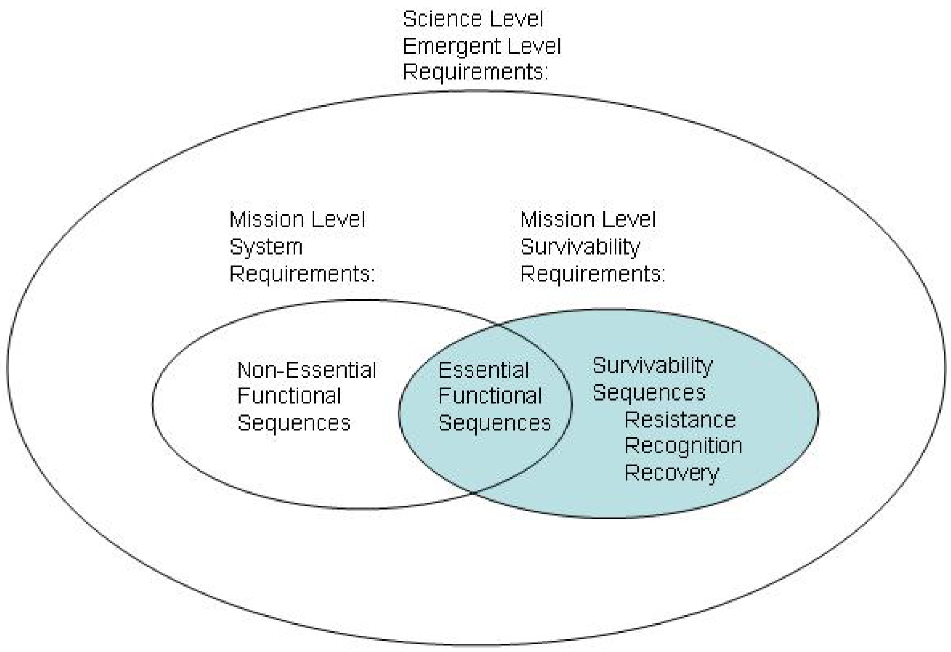

Figure 1. Integrating survivability requirements with system requirements (adapted from [4])

3. Survivability through Autonomic Systems

The autonomic concept is inspired by the human body's autonomic nervous system. By analogy, humans have good mechanisms for adapting to changing environments and repairing minor physical damage. The autonomic nervous system monitors heartbeat, checks blood sugar levels and keeps the body temperature normal without any conscious effort from the human. This biological autonomicity is influencing a new paradigm for computing to create self-management within computer-based systems (Autonomic Computing, Autonomic Communications and Autonomic Systems). There is an important distinction between autonomic activity in the human body and autonomic responses in computer-based systems. Many of the decisions made by autonomic elements in the body are involuntary, whereas autonomic elements in computer-based systems make decisions based on tasks chosen to delegate to the technology [12].

In the late 1990s DARPA/ISO's Autonomic Information Assurance (AIA) program studied defense mechanisms for information systems against malicious advance cyber-adversaries and (2) coordinated responses [11]. These hypotheses may provide general guidance for creating autonomic survivable systems responses are more effective than local reactive. adversaries. The AIA program resulted in two hypotheses; (1) fast responses are necessary to counter

'Autonomic' became mainstream within Computing in 2001 when IBM launched their perspective on the state of information technology [5]. IBM defined four key self properties: self-configuring, self-healing, self-optimizing and self-protecting [12]. In the few years since, the self-x list has grown as research expands, bringing about the general term selfware or self-*, yet these four initial self-managing properties along with the four enabling properties; self-aware (of internal capabilities and state of the managed component), self-situated (environment and context awareness), self-monitor and self-adjust (through sensors, effectors and control loops), cover the general goal of self management [14].

The premise is that the sustainable need for survivable properties such as resistance, recognition and recovery (Figure 1) can be provided through autonomic techniques.

Autonomic Systems work through creating a cooperative environment where elements, nodes and components are each assigned an autonomic manager (Figure 3).

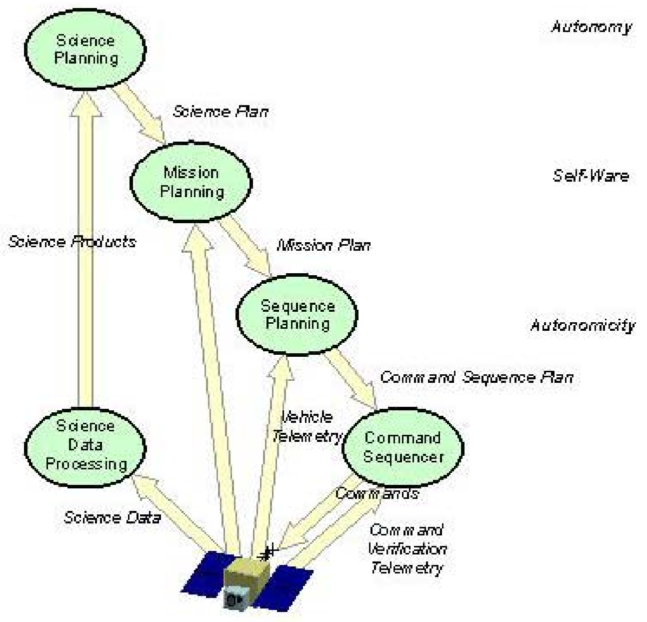

Figure 2. Progressive autonomy and autonomicity [15]

These autonomic managers provide the self-awareness (self-monitoring and self-adjusting of the managed component) and environment-awareness (monitoring and reacting to the dynamic conditions of the environment). The autonomic manager to autonomic communications (AM⇔AM in Figure 3) includes several dynamic loops of control, for instance a fast loop provide reflex reactions and a slower loop providing coordinated event telemetry. These loops will not only trigger autonomic/self-management activity but also feed up into higher planes.

Figure 2 depicts the layers in a mission. The top layer contains the goal of the mission – the science. This has been classified as autonomous layer due to the fact it contains the self-governance high level goals and policies that the mission must meet including the emergent constraints for discovering and planning new opportune science.

The middle (self-ware) layer (Figure 2) depicts the day to day autonomous and autonomic activity to meet the mission plans.

The bottom (autonomic) layer depicts the instant/reflex reaction activity that will need to occur to arising situations to ensure correct and survivable activity takes place.

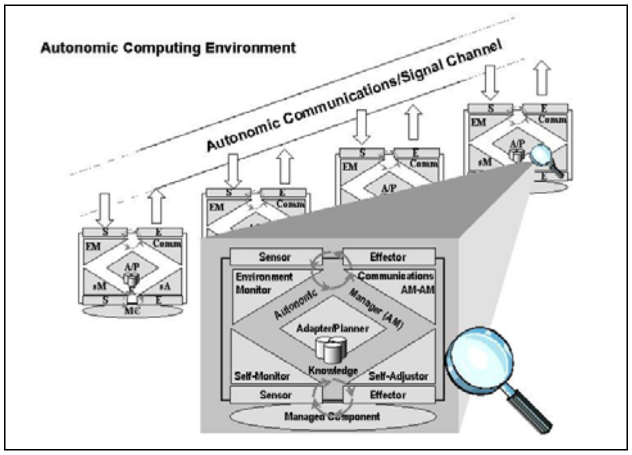

Figure 3. Autonomic elements (autonomic manager + managed component)

4. Challenge in developing sustainable systems

The required complexity in these systems is evident. By their very nature the systems are critical systems due to the remoteness of the missions and the human life and costs involved. Components in the system will fail but the system must be flexible and dynamic in nature to self-configure and self-heal to avoid system failure and provide an effective work-around. This requirement to adapt encourages the facilitating of self-adaptation and emergence in the system, while at the same time raises concerned with non-desirable emergent behavior that may endanger the mission. Ongoing efforts to address this are discussed in section *6. Development and verification of autonomic policy-based systems*.

Another challenge is the orchestration between the different planes (autonomic ⇔ selfware ⇔ autonomous planes). As in communications and more recently in IT (through Autonomic Computing) the research area of policy based management has been identified as a potential means to specify top level policies that are then implemented and self-managed by the system. Ongoing research efforts in this area are discussed in section *5. Policies for autonomic systems*.

5. Policies for autonomic systems

Policies have been described as a set of considerations designed to guide decisions of courses of action [16] and Policy-based management may be viewed as an administrative approach to systems management that establishes rules in advance to deal with situations that are likely to occur. From this perspective policy-based management works by controlling access to and setting priorities for the use of ICT resources [17], for instance, where a (human) manager may simply specify the business objectives and the system will make it so in terms of the needed ICT [18] for example [19]: (1) "The customer database must be backed up nightly between 1 a.m. and 4 a.m.," (2) "Platinum customers are to receive no worse than 1-second average response time on all purchase transactions," (3) "Only management and the HR senior staff can access personnel records," and (4) "The number of connections requested by the Web application server cannot exceed the number of connections supported by the associated database." These examples highlight the wide range and multi-level of policies available, the first concerned with system protection through backup, the second with system optimization to achieve and maintain a level of quality of service for key customers; while the third and forth are concerned with system configuration and protection.

Policy-based Management has been the subject of extensive research in its own right. The IETF has investigated Policy-based Networking as a means for managing IP-based multi-service networks with quality of service guarantees. More recently, PBM has become extremely popular within the telecom industry, for next generation networking, with many vendors announcing plans and introducing products. This is driven by the fact that policy has been recognized as a solution to manage complexity, and to guide the behavior of a network or distributed system through high-level user-oriented abstractions [20].

A policy-based management tool may also reduce the complexity of product and system management by providing uniform cross-product policy definition and management infrastructure [21].

One perspective of Autonomic Computing is Policy-Based Self-Management.

The long term strategic vision highlighted an overarching self-managing vision where the system would have such a level of 'self' capability that a senior (human) manager in an organization could specify business policies—such as profit margin on a specific product range or system quality of service for a band of customers— and the computing systems would do the rest [22].

It has been argued, and counter argued, that for this vision to become a reality would require the computer science grand challenges to be solved before hand: AI completeness, Software Engineering completeness and so on [23]. What is clear in this vision is the importance of policies to empower the system at all levels to self-manage.

With one definition of Autonomic Computing being *Self-Management based on high level guidance from humans* [24] and considering the IBM's high-level set of self-properties (self-CHOP, configuration, healing, optimization and protection) against the types of typical policies mentioned previously (optimization, configuration and protection), the importance and relevance of polices for achieving autonomicity becomes clear [25].

The field of Agent-Oriented Software Engineering (AOSE) has arisen to address methodological aspects and other issues related to the development of complex multi-agent systems. AOSE is a new software engineering paradigm that augurs much promise in enabling the successful development of more complex systems than is achievable with current Object-Oriented approaches which use agents and organizations of agents as their main abstractions [26].

The organizational metaphor has been proven to be one of the most appropriate tools for engineering Multi-Agent Systems (MAS). The metaphor is used by many researchers to guide the analysis and design of MASs, e.g., [27][28][29]. A MAS organization can be observed from two different point of view [29]:

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

Structural point of view: shows agents as artifacts that belong to sub-organizations, groups, teams. In this view agents are also structured into hierarchical structures showing the social structure of the system.

Both views are intimately related, but they show the organization from radically different viewpoints. Since any structural organization must include interactions between their agents in order to function, it is safe to say that the acquaintance organization is always contained in the structural organization. Therefore, if we first determine the acquaintance organization, and we define the constraints required for the structural organization, a natural map is formed between the acquaintance organization and the corresponding structural organization. This is the process of assigning roles to agents [29]. Thus, we can conclude that any acquaintance organization can be modeled orthogonally to its structural organization [30]. We use this separation to specify policies at the acquaintance organization level, and deploy them over the structural organizational of the running system. The scope of policies usually implies features of several acquaintance sub-organizations. In such cases, we must first compose the acquaintance sub-organizations, this process being guided by the policy specification, to deploy it later. For more information on this work please refer to [31].

6. Development and verification of autonomic policy-based systems

As autonomic systems are essentially concerned with bring self-management to highly complex systems, all the existing issues with developing and maintaining complex systems are still present from developing effective systems and software in the first place, and then managing their evolution through reverse and re-engineering the of systems.

In this section we briefly discuss our work on formal requirements based programming extending it as a means to provide provably correct code generated from policies for autonomic systems. Specifically, we are developing NASA's R2D2C technologies for mechanically transforming policies (expressed in restricted natural language, or appropriate graphical notations) into a provably equivalent formal model that can be used as the basis for code generation and other transformations, including reverse engineering the process and working towards the self-generation of provable autonomic policies.

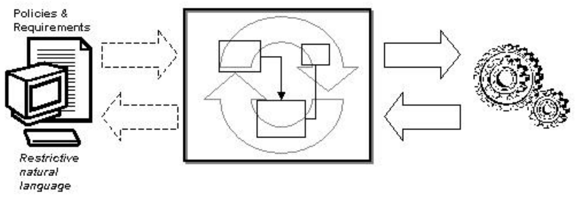

Figure 4. The R2D2C approach, generating a formal model from requirements and producing code

6.1. R2D2C

Our experience at NASA Goddard Space Flight Center (GSFC) has been that while engineers are happy to write descriptions as natural language scenarios, or even using semi-formal notations such as UML use cases, they are loath to undertake formal specification. Absent a formal specification of the system under consideration, there is no possibility of determining any level of confidence in the correctness of an implementation. More importantly, we must ensure that this formal specification fully, completely, and consistently captures the requirements set forth at the outset. Clearly, we cannot expect requirements to be perfect, complete, and consistent from the outset, which is why it is even more important to have a formal specification, which can highlight errors, omissions, and conflicts. The formal specification must also reflect changes and updates from system maintenance as well as changes and compromises in requirements, so that it remains an accurate representation of the system.

R2D2C, or Requirements-to-Design-to-Code [32][33], is a NASA patent-pending approach to Requirements- Based Programming that provides a mathematically tractable round-trip engineering approach to system development. 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 (Figure 1) 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 methods. Currently we are using CSP, Hoare's language of Communicating Sequential Processes [34][35], which is suitable for various types of analysis and investigation, and as the basis for fully formal implementations as well as for use in automated test case generation, etc.

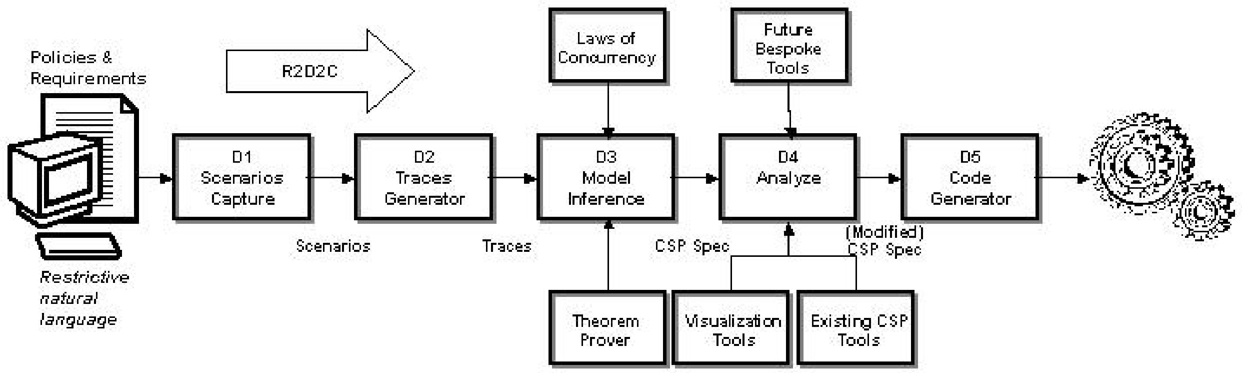

Figure 5. The entire forward process with D1 thru D5 illustrating the development approach

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, as shown in Figure 4. The approach can also be used to "paraphrase" (in natural language, etc.) formal descriptions of existing systems.

This approach is not limited to generating code. It may also be used to generate business processes and procedures, and we have been experimenting with using it to generate instructions for robotic devices that were to be used on the Hubble Robotic Servicing Mission (HRSM), which, at the time of writing, has not received a final go-ahead. We are also experimenting with using it as a basis for an expert system verification tool, and as a means of capturing domain knowledge for expert systems, and most recently for generating code from policies.

6.2. R2D2C technical approach

The R2D2C approach involves a number of phases, which are reflected in the system architecture described in Figure 5. The following describes each of these phases.

D1 Scenarios Capture: Engineers, end users, and others write scenarios describing intended policies. 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. Scenarios effectively describe policies that must be adhered to. They describe who various situations and events are to be handled. At the lower (micro) level, these may describe policies of an individual autonomic element. At the overall (macro) level, they may describe policies for a complete system. Policies may be viewed as being analogous to requirements, but are likely to be expressed at differing levels, and to express a mixture of both functional and non-functional requirements that must be implemented in order to satisfy the policies.

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

D3 Model Inference: A formal model, or formal specification, expressed in CSP is inferred by an automatic theorem prover, in this case, ACL2 [36], using the traces derived in phase D2. A deep4 embedding of the laws of concurrency [37] 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.

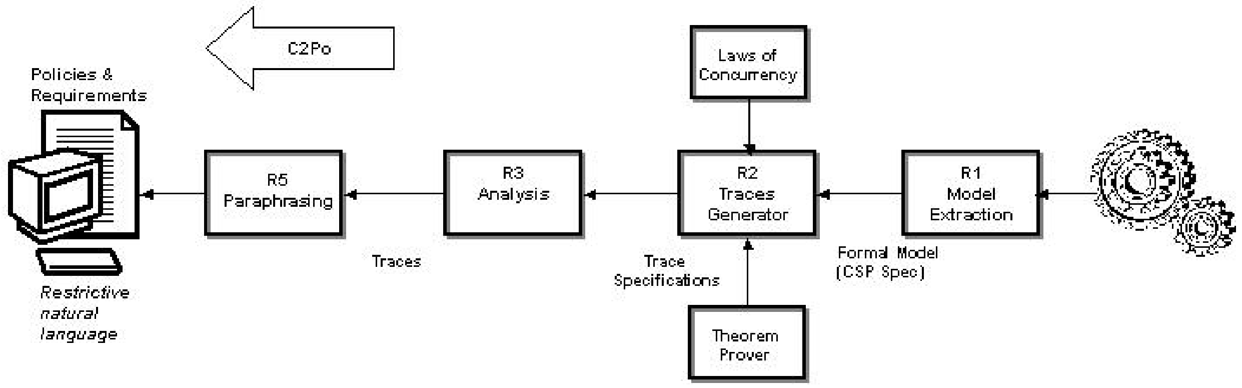

Figure 6. The reverse process with R1 thru R4 illustrating the code to policies approach

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 [38], or converting to other notations suitable for code generation (e.g., converting CSP to B [40]) and then using the code generating capabilities of the B Toolkit.

6.3. Reverse engineer: Code-to-Policies (C2Po)

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, low level autonomic element policies, or the like. As Figure 6 illustrates, the above process may also be run in reverse:

R1 Model Extraction: Using various reverse engineering techniques [39], 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, 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.).

7. Conclusion

Sustainable and Survivable Systems are essential to realize future space exploration missions. Autonomic Systems – self-managing computer-based systems inspired by the self-managing activity of the biological autonomic nervous system – may contribute to achieving sustainable systems.

One vision of Autonomic Computing is Self-Management based on high level guidance from humans. As such policies and policy based management are a key enabling technology for achieving autonomicity. Their importance to SEM lies in realizing and orchestrating high level science goals (or policies) with the low-level dynamic day to day survivability requirements.

This paper has briefly described some of our research in this area including a method that can produce fully (mathematically) tractable development of policies for autonomic systems from requirements through to code generation.

8. Acknowledgements

Part of this work has been supported by the NASA Office of Systems and Mission Assurance (OSMA) through its Software Assurance Research Program (SARP) project, Formal Approaches to Swarm Technologies (FAST), and by NASA Software Engineering Laboratory, Goddard Space Flight Center (Code 581).

This research is partly supported at University of Ulster by the Computer Science Research Institute (CSRI) and the Centre for Software Process Technologies (CSPT) which is funded by Invest NI through the Centres of Excellence Programme, under the EU Peace II initiative.

9. References

[1] G. Martin, “NASA Exploration Team (NExT) program,” World Space Congress, Houston, Texas, Oct. 10-19, 2002.

[2] D.J. Atkinson, "*Constellation Program Return to the Moon: Software Systems Challenges*," Keynote presentation, 3rd IEEE International Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2006), Columbia, MD, USA, April 2006.

[3] R.C. Linger, N.R. Mead, H.F. Lipson, “Requirements Definition for Survivable Systems,” 3rd IEEE Int. Conf. Requirements Engineering. Colorado Springs, CO, April 6-10, 1998, pp 14-23.

[4] N. Mead, “Requirements Engineering for Survivable Systems,” Technical Report CMU/SEI-2003-TN-013, 2003.

[5] P. Horn, “Autonomic computing: IBM perspective on the state of information technology,” IBM T.J. Watson Labs, NY, 15th October 2001.

[6] R. Sterritt, “Towards Autonomic Computing: Effective Event Management,” Proc. IEEE/NASA SEW, Greenbelt, MD, Dec. 2002.

[7] J.O. Kephart, D.M. Chess. “The Vision of Autonomic Computing,” Computer, 36(1):41–52, 2003.

[8] R. Sterritt, "Autonomic Computing," Innovations in Systems and Software Engineering, Vol. 1, No. 1, Springer, pp 79-88, 2005.

[9] R. Sterritt, G. Garrity, E. Hanna, P. O’Hagan, "Survivable Security Systems through Autonomicity," Proc. 2nd NASA/IEEE Workshop on Radical Agent Concepts (WRAC 2005), NASA GSFC, Maryland, USA, 20-22 September 2005, in "LNCS", Springer.

[10] R. Sterritt, G. Garrity, E. Hanna, P. O’Hagan, "Autonomic Agents for Survivable Security Systems," Proc. 1st IFIP Workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES 2005) at EUC'05, Nagasaki, Japan, 6-9th December 2005, in "LNCS 3823," Springer, pp 1235-1244.

[11] SM Lewandowski, DJ Van Hook, GC O'Leary, JW Haines, LM Rossey, “SARA: Survivable Autonomic Response Architecture,” DARPA Information Survivability Conference and Exposition II Proceedings, Vol. 1, pp. 77-88, June 2001.

[12] IBM, “An architectural blueprint for autonomic computing,” 2003.

[13] B. Randell, “Turing Memorial Lecture – Facing Up to Faults,” Comp. J. 43(2), pp 95-106, 2000.

[14] R Sterritt, DW Bustard, "Autonomic Computing: a Means of Achieving Dependability?" Proc IEEE Int. Conf. on the Engineering of Computer Based Systems (ECBS'03), Huntsville, Alabama, USA, April 7-11 2003, pp 247-251.

[15] W. Truszkowski, C.A. Rouff, H.L. Hallock, J. Karlin, J.L. Rash, M.G. Hinchey and R. Sterritt, “Autonomous and Autonomic Systems: With Applications to NASA Intelligent Spacecraft Operations and Exploration Systems, NASA Monographs in Systems and Software Engineering,” Springer Verlag, London, 2006.

[16] MJ Masullo, SB Calo, “Policy Management: An Architecture and Approach,” Proc. IEEE 1st Int. Workshop on Systems Management, Los Angeles, CA, April 14-16, 1993.

[17] Whatis?com, *Online computer and internet dictionary and encyclopaedia*, 2005.

[18] Lymberopoulos L, Lupu E, Sloman M, “An adaptive policybased framework for network services management,” J Netw Syst Manage 11(3), 2003.

[19] D. Kaminsky, “An Introduction to Policy for Autonomic Computing,” IBM white paper, March 2005.

[20] A. Meissner, S. B. Musunoori, L. Wolf, “MGMS/GML - Towards a new Policy Specification Framework for Multicast Group Integrity,” Proceedings 2004 Int. Symposium Applications and the Internet (SAINT2004), Tokyo, Japan, 2004.

[21] A.G. Ganek, “Autonomic computing: implementing the vision.” Keynote talk at the autonomic computing workshop (AMS 2003), Seattle, 25th June 2003.

[22] Horn P, “Autonomic computing: IBM perspective on the state of information technology,” Presented at AGENDA 2001, Scottsdale, AR, 2001.

[23] Babaoglu O., Couch A., Ganger G., Stone P., Yousif M., Kephart J., “Panel: Grand Challenges of Autonomic Computing,” ICAC’05, Seattle, WA, June 2005.

[24] Kephart JO, Walsh WE, "An Artificial Intelligence Perspective on Autonomic Computing Policies," Policy, Fifth, pp 3-12, 2004.

[25] Sterritt R, Hinchey MG, Rash JL, Truszkowski W, Rouff CA, Gracanin D, (Dec 2005) "Towards Formal Specification and Generation of Autonomic Policies," Proceedings of 1st IFIP Workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES 2005) at EUC'05, Nagasaki, Japan, 6-9th December, in "LNCS 3823", Springer, Pages 1245-1254.

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

[27] J. Odell, H. Parunak, and M. Fleischer. “The role of roles in designing effective agent organizations,” in LNCS 2603, pages 27–28, Berlin, 2003. Springer–Verlag.

[28] H. V. D. Parunak and J. Odell. Representing social structures in UML. Proceedings of the Fifth International Conference on Autonomous Agents, pages 100–101, Montreal, Canada, 2001. ACM Press.

[29] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA methodology. ACM Transactions on Software Engineering and Methodology, 12 (3) pp317–370, July 2003.

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

[31] J. Pena, M.G. Hinchey, R. Sterritt, “Towards Modeling, Specifying and Deploying Policies in Autonomous and Autonomic Systems Using an AOSE Methodology,” Proceedings of the Third IEEE International Workshop on the Engineering of Autonomic and Autonomous Systems (EASe 2006), March 2006.

[32] M. G. Hinchey, J. L. Rash, and C. A. Rouff. Requirements to design to code: Towards a fully formal approach to automatic code generation. Tech. Rep. TM-2005-212774, NASA Goddard Space Flight Center, Greenbelt, MD, 2004.

[33] J. L. Rash, M. G. Hinchey, C. A. Rouff, and D. Graˇcanin. Formal requirements-based programming for complex systems. In Proc. Int. Conference on Engineering of Complex Computer Systems, Shanghai, China, 16–20 June 2005. IEEE Computer Society Press.

[34] C.A.R. Hoare, “Communicating sequential processes,” Comms of the ACM, 21(8):666–677, 1978.

[35] C.A.R. Hoare, “Communicating Sequential Processes,” Prentice Hall International Series in Computer Science. Prentice Hall Int., Englewood Cliffs, NJ, and Hemel Hempstead, UK, 1985.

[36] M. Kaufmann and Panagiotis Manolios and J Strother Moore. Computer-Aided Reasoning: An Approach. Advances in Formal Methods Series. Kluwer Academic Publishers, Boston, 2000.

[37] M. G. Hinchey and S. A. Jarvis. Concurrent Systems: Formal Development in CSP. International Series in Software Engineering. McGraw-Hill Int., 1995.

[38] Failures-Divergences Refinement: User Manual and Tutorial. Formal Systems (Europe), Ltd., 1999.

[39] H.J. van Zuylen, “The REDO Compendium: Reverse Engineering for Software Maintenance,” Wiley, 1993.

[40] M.J. Butler, csp2B : A Practical Approach To Combining CSP and B, Dept. Electronics and Computer Science, University of Southampton, February 1999.