Towards Formal Specification and Generation of Autonomic Policies

Roy Sterritt[1], Michael G. Hinchey[2], James L. Rash[3], Walt Truszkowski[3], Christopher A. Rouff[4], and Denis Gracanin[5]

[1] University of Ulster, Faculty of Engineering, Northern Ireland
r.sterritt@ulster.ac.uk
[2] NASA Software Engineering Laboratory,
NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA
Michael.G.Hinchey@nasa.gov
[3] Advanced Architectures and Automation Branch.
NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA
{james.l.rash, Walter.F.Truszkowski}@nasa.gov
[4] Advanced Concepts Business Unit,
Science Applications International Corp., McLean, VA 22102, USA
rouffc@saic.com
[5] Virginia Tech, Department of Computer Science,
Blacksburg, Virginia, USA
gracanin@vt.edu

Abstract. Autonomic Computing (AC), self-management based on high level guidance from humans, is increasingly gaining momentum as the way forward in designing reliable systems to hide complexity and conquer IT management costs. Effectively, AC may be viewed as Policy-Based Self-Management. In this paper we look at the motivation for utilizing NASA requirements-based programming 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, with the goal of self-generation of provable autonomic policies.

1 Introduction and Motivation

As a rapidly growing field[1], Autonomic Systems (Autonomic Computing and Autonomic Communications) is a promising new approach for developing large-scale complex distributed computer-based systems. In introducing the concept of Autonomic Computing, IBM's Paul Horn likened the needs of large scale systems management to that of the human Autonomic Nervous System (ANS). The ANS, through the self-regulation, is able to effectively monitor, control and regulate the human body without the need for conscious thought [12]. This self-regulation and separation of concerns provides human beings with the ability to concentrate on high level objectives without having to micro-manage the specific details involved.

[1] Consider, e.g., the IEEE Task Force on Autonomous and Autonomic Systems (TFAAS) as of June 2005. See http://www.computer.org/tab.

T. Enokido et al. (Eds.): EUC Workshops 2005, LNCS 3823, pp. 1245–1254, 2005.

The vision and metaphor of Autonomic Computing is to apply the same principles of self-regulation and complexity-hiding to the design of computer-based systems, in the hope that eventually computer systems can achieve the same level of self-regulation as the human ANS [12][21]. In his talk, Horn highlighted that the Autonomic Computing system must "find and generate rules for how best to interact with neighboring systems" [12]. The majority of current efforts are on the 'how' of autonomic systems, such as defining autonomic managers that together with the component that is to be managed make up an autonomic element to exist in a collaborative autonomic environment to provide self-management of the system. Much less is being done on generating the rules and policies that will drive autonomic systems.

The initial 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. It has been argued that for this vision to become a reality would require AI completeness, Software Engineering completeness and so on [2]. What is clear in this vision is the importance of policies to empower the system at all levels to self-manage.

2 Policy Based Management

Policies have been described as a set of considerations designed to guide decisions of courses of action [17] 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 information and communications technology (ICT) resources[2], 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 [16] for example [13]:

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 multiple levels 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. With one definition of Autonomic Computing being Self-Management based on high level guidance from humans [15] and considering IBM's high-level set of self-properties (self-CHOP, configuration, healing, optimisation and protection) against the types of typical

[2] See, e.g., Whatis.com, Online computer and internet dictionary and encyclopedia.

policies mentioned previously (optimization, configuration and protection), the importance and relevance of polices for achieving autonomicity becomes clear.

Policy-based management (PBM) has been the subject of extensive research in its own right. The Internet Engineering Task Force (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 behaviour of a network or distributed system through high-level user-oriented abstractions [18]. 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 [5].

3 Formal Requirements Based Programming

The need for ultra-high dependability systems increases continually, along with a correspondingly increasing need to ensure correctness in system development. By "correctness", we mean that the implemented system is equivalent to the requirements, and that this equivalence can be proved mathematically. Today there is no automated means of producing a system or a procedure that is a provably correct implementation of the customer's requirements. Further, requirements engineering as a discipline has yet to produce an automated, mathematics-based process for requirements validation.

Development of a system that will have a high level of reliability requires the developer to represent the system as a formal model that can be proven to be correct. Through the use of currently-available tools, the model can then be automatically transformed into code with minimal or no human intervention. This serves to reduce the risk of inadvertent insertion of errors by developers. Automatically producing the formal model from customer requirements would further reduce the chance of insertion of errors by developers.

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. Model-Based Development holds that emphasis should be placed on building a model of the system with such high quality that automatic code generation is viable. While this has worked well, and made automatic code generation feasible, there is still the large analysis-specification gap that remains unaddressed. Requirements-Based Programming addresses that issue and 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 essence, Requirements-Based Programming takes Model-Based Development and adds a front end [9][20].

There have been calls for the community to address Requirements-Based Programming, as it offers perhaps the most promising approach to achieving correct systems [16][3]. Although the use of Requirements-Based Programming does not specifically pre-

[3] 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.

suppose the existence of an underlying formalism, the realization that proof of correctness is not possible without formalism [3] certainly implies that Requirements-Based Programming should be formal. 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 [6].

The remainder of this paper describes a method for mechanically transforming system requirements into a provably equivalent formal model that can be used as the basis for code generation and other transformations. The method is applicable to development of policy-based management systems, which, as stated above, is an important part of autonomic systems. In addition, due to the complexity of many policies, development of this part of an autonomic system is crucial to the correct operation of the system and can be very labor intensive. Developing and verifying the policies in an autonomic system in a cost effective manner will be critical for the correct operation of these systems.

4 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 [8][19], 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 [10][11], 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.

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

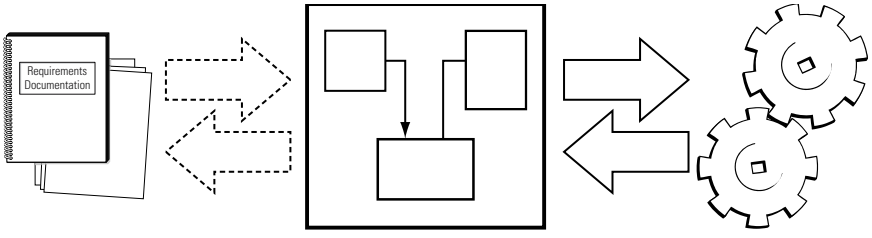

Fig. 1. The R2D2C approach, generating a formal model from requirements and producing code from the formal model, with automatic reverse engineering

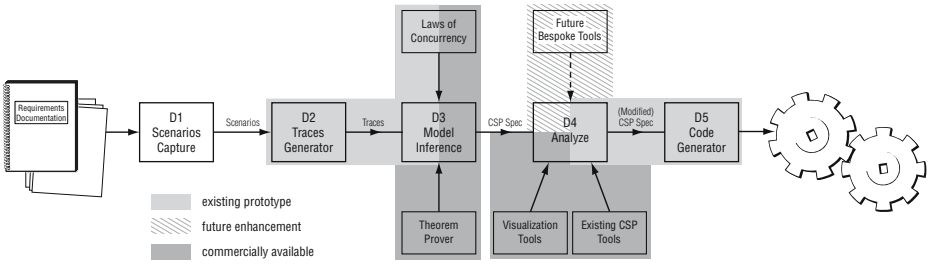

Fig. 2. The entire process with D1 thru D5 illustrating the development approach

This approach is not limited to generating high-level 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 policies from requirements.

4.1 R2D2C Technical Approach

The R2D2C approach involves a number of phases, which are reflected in the system architecture described in Figure 2. 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 [14], using the traces derived in phase D2. A deep[4] embedding of the laws of concurrency [7] 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 [4]) and then using the code generating capabilities of the B Toolkit.

4.2 A Simple Example

The Lights-Out Ground Operating System (LOGOS) is a proof-of-concept NASA system for automatic control of ground stations when satellites pass overhead and under their control. The system exhibits both autonomous and autonomic properties [23] [22], and operates by having a community of distributed autonomous software modules work cooperatively based on policies to perform the functions previously undertaken by human operators using traditional software tools, such as orbit generators and command sequence planners. We will not consider the entire LOGOS/ANTS related system here. Although a relatively small system, it is too extensive to illustrate in its entirety in this paper. We will take an example agent, the Pager agent, and illustrate its mapping from natural language descriptions through to the CSP model that can be used to generate code.

Based on defined policies for the operation of the system, the Pager agent sends pages to engineers and controllers when there is a spacecraft anomaly. For example, the Pager agent receives requests from the user interface agent that no analyst is logged on, so it gets paging information from the Database agent and pages an appropriate analyst, and, when instructed by the user interface agent stops paging the analyst. These policies can be stated as follows:

- When the Pager agent receives a request from the User Interface agent, the Pager agent sends a request to the Database agent for an analyst's pager information and puts the message in a list of requests to the Database agent

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

– When the Pager agent receives a pager number from the Database agent, then the Pager agent removes the message from the paging queue and sends a message to the analyst's pager and adds the analyst to the list of paged people

– When the Pager agent receives a message from the user interface agent to stop paging a particular analyst, the Pager agent sends a stop-paging command to the analyst's pager and removes the analyst from the paged list

– When the Pager agent receives another kind of message, reply to the sender that the message was not recognized

The above policies for handling anomalies would then be translated into CSP. The following is a partial CSP description of the Pager agent:

$$
\begin{aligned}
&PAGER\_BUS_{db\_waiting,paged} = pager.Iin?msg \rightarrow \\
&\quad case \\
&\qquad GET\_USER\_INFO_{db\_waiting,paged,pagee,text} \\
&\qquad\quad if\ msg = (START\_PAGING, specialist, text) \\
&\qquad BEGIN\_PAGING_{db\_waiting,paged,in\_reply\_to\_id(msg),pager\_num} \\
&\qquad\quad if\ msg = (RETURN\_DATA.pager\_num) \\
&\qquad STOP\_CONTACT_{db\_waiting,paged,pagee} \\
&\qquad\quad if\ msg = (STOP\_PAGING, pagee) \\
&\qquad pager.Iout!(head(msg), UNRECOGNIZED) \\
&\qquad\qquad \rightarrow PAGER\_BUS_{db\_waiting,paged} \\
&\quad otherwise
\end{aligned}
$$

This specification states that the process *PAGER_BUS* receives a message on its "*Iin*" channel and stores it in a variable called "*msg*". Depending on the contents of the message, one of four different processes is executed based on the policies. If the message is of type *START_PAGING*, then the *GET_USER_INFO* process is called with parameters of the specialist to page (pagee) and the text to send. If the message is of type *RETURN_DATA* with a pagee's pager number, then the database has returned a pager number and the *BEGIN_PAGING* process is executed with a parameter containing the original message id (used as a key to the *db_waiting set*) and the passed pager number. The third type of message that the Pager agent might receive is one of type *STOP_PAGING*. This message contains a request to stop paging a particular specialist (stored in the pagee parameter). When this message is received, the *STOP_PAGING* process is executed with the parameter of the specialist type. If the Pager agent receives any other message than the above three messages, an error message is returned to the sender of the message (which is the first item of the list) stating that the message is "*UNRECOGNIZED*". After this, the *PAGER_BUS* process is again executed.

The formal model derived (in CSP) now embodies the policy for anomaly resolution that was specified in the scenarios.

4.3 Advantages of the R2D2C Approach

We have not yet had an opportunity to apply R2D2C to policy generation, although that is certainly our plan. In addition to applying it to the HRSM procedures [19], we have applied R2D2C to LOGOS, a NASA prototype Lights-Out Ground Operating System, that exhibits both autonomous and autonomic properties [22][23]. We illustrate the use of a prototype tool to apply R2D2C to LOGOS in [20], and describe our success with the approach.

Here, we summarize some benefits of using R2D2C, and hence of using Formal Requirements-Based Programming in system development. It is our contention that R2D2C, and other approaches that similarly provide mathematical soundness throughout the development lifecycle, will:

- Dramatically increase assurance of system success by ensuring
  - completeness and consistency of requirements
  - that implementations are true to the requirements
  - that automatically coded systems are bug-free; and that
  - that implementation behavior is as expected
- Decrease costs and schedule impacts of ultra-high dependability systems through automated development
- Decrease re-engineering costs and delays

5 Conclusions

Autonomic Computing, Self-Management based on high level guidance from humans, has been gaining ground as a significant new paradigm to facilitate the creation of self-managing systems to deal with the ever increasing complexity and costs inherent in today's (and tomorrow's) systems. Policies and policy based management is a key enabling technology for achieving autonomicity. This paper described a method that can produce fully (mathematically) tractable development of policies for autonomic systems from requirements through to code generation. The use of this method was illustrated through an example showing how user formulated policies can be translated into a formal model which can then be converted to code. The requirements-based programming method described will allow faster, higher quality development and maintenance of autonomic systems based on user formulation of policies.

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 Goddard Space Flight Center, Software Engineering Laboratory (Code 581). At the University of Ulster by the Centre for Software Process Technologies (CSPT), funded by Invest NI through the Centres of Excellence Programme, under the EU Peace II initiative.

References

1. *Failures-Divergences Refinement: User Manual and Tutorial*. Formal Systems (Europe), Ltd., 1999.
2. O. Babaoglu, A. Couch, G. Ganger, P. Stone, M. Yousif, and J. Kephart. Panel: Grand challenges of autonomic computing. In *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC-05)*, Seattle, WA, June 2005.
3. F. L. Bauer. A trend for the next ten years of software engineering. In H. Freeman and P. M. Lewis, editors, *Software Engineering*, pages 1–23. Academic Press, 1980.
4. 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.
5. A. G. Ganek. Autonomic computing: implementing the vision. Keynote presentation, Autonomic Computing Workshop, AMS 2003, 25 June 2003.
6. D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.
7. M. G. Hinchey and S. A. Jarvis. *Concurrent Systems: Formal Development in CSP*. International Series in Software Engineering. McGraw-Hill International, London, UK, 1995.
8. 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.
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. C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.
11. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall International, Englewood Cliffs, NJ, 1985.
12. P. Horn. Autonomic computing: IBM's perspective on the state of information technology. Technical report, IBM T. J. Watson Laboratory, October 15, 2001.
13. D. Kaminsky. An introduction to policy for autonomic computing. white paper, March 2005.
14. M. Kaufmann and Panagiotis Manolios and J Strother Moore. *Computer-Aided Reasoning: An Approach*. Advances in Formal Methods Series. Kluwer Academic Publishers, Boston, 2000.
15. J. O. Kephart and W. E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Proc. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, pages 3–12, 7–9 June 2004.
16. L. Lymberopoulos, E. Lupu, and M. Sloman. An adaptive policy-based framework for network services management. *Journal of Network and Systems Management*, 11(3), 2003.
17. M. J. Masullo and S. B. Calo. Policy management: An architecture and approach. In *Proc. IEEE First International Workshop on Systems Management*, Los Angeles, California, USA, 14–16 April 1993.
18. A. Meissner, S. B. Musunoori, and L. Wolf. MGMS/GML—towards a new policy specification framework for multicast group integrity. In *Proc. 2004 International Symposium on Applications and the Internet (SAINT2004)*, Tokyo, Japan, 2004.
19. 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.

20. J. L. Rash, M. G. Hinchey, C. A. Rouff, D. Gračanin, and J. D. Erickson. A tool for requirements-based programming. In *Proc. International Conference on Integrated Design and Process Technology (IDPT 2005)*, Beijing, China, 13–17 June 2005. The Society for Design and Process Science.
21. R. Sterritt. Towards autonomic computing: Effective event management. In *Proc. 27th Annual IEEE/NASA Software Engineering Workshop (SEW)*, pages 40–47, Greenbelt, Maryland, USA, 3–5 December 2002. IEEE Computer Society Press, Los Alamitos, Calif.
22. 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).
23. W. F. Truszkowski, J. L. Rash, C. A. Rouff, and M. G. Hinchey. Some autonomic properties of two legacy multi-agent systems — LOGOS and ACT. In *Proc. 11th IEEE International Conference on Engineering Computer-Based Systems (ECBS), Workshop on Engineering Autonomic Systems (EASe)*, pages 490–498, Brno, Czech Republic, May 2004. IEEE Computer Society Press, Los Alamitos, Calif.