Enabling Requirements-Based Programming for Highly-Dependable Complex Parallel and Distributed Systems

Michael G. Hinchey, James L. Rash
NASA Goddard Space Flight Center
Information Systems Division
Greenbelt, Maryland 20771, USA
michael.g.hinchey@nasa.gov

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

Abstract

The manual application of formal methods in system specification has produced successes, but in the end, despite any claims and assertions by practitioners, there is no provable relationship between a manually derived system specification or formal model and the customer's original requirements. Complex parallel and distributed systems present the worst case implications for today's dearth of viable approaches for achieving system dependability. No avenue other than formal methods constitutes a serious contender for resolving the problem, and so recognition of requirements-based programming has come at a critical juncture. We describe a new, NASA-developed automated requirements-based programming method that can be applied to certain classes of systems, including complex parallel and distributed systems, to achieve a high degree of dependability.

Key Words: Distributed Systems Validation, Verification, Formal Methods, Automatic Code Generation, CSP

1. Introduction

The inherent complexity in building highly-dependable parallel and distributed systems is well known. As we build more and more sophisticated applications with more stringent timing constraints, the difficulties continue to rise in measure with the complexity, despite the best efforts of the research and development community to address means of tackling the problem. Expectations for dependability, efficiency, performance, maintainability and cost-effectiveness over a system's entire life cycle cannot be lowered, and indeed are seen to rise from any given level achieved in the past. Consequently, the combined effect of increasing system complexity and rising expectations suggests that a critical need has arisen for a new way to resolve this issue.

Requirements-Based Programming (RPB) has been advocated as a means, perhaps the *only* means, of developing highly dependable complex systems that can be guaranteed to be correct [2, 3]. We present *Requirements-to-Design-to-Code*, a NASA-developed approach to Requirements-Based Programming, which has the distinct advantage of offering a fully formal underpinning. The approach has been demonstrated to have applicability in a variety of domains.

2. Requirements-Based Programming

Requirements-Based Programming effectively extends Model-Based Development (MBD) by adding a "front-end". While MBD holds that code-generation can be efficient and practical from an appropriate model, RBP addresses the adequate development of that model, by ensuring a direct mapping from requirements through to specification and modeling.

RBP, therefore, affords the systematic and mechanical transformation of requirements into executable code. While this is an obvious goal of system development, other approaches (including the manual application of formal specification to derive a system model) have failed to address the entire lifecycle and ensure that generated code does in fact match the requirements [5, 6].

As we will describe in Section 3, we have found it to have a number of interesting applications in verification and validation, as well as in system development *ab initio*.

2.1. R2D2C

Requirements-to-Design-to-Code (R2D2C) is the provisional name for a NASA technology that provides a mathematically tractable round-trip engineering approach to system development.

In this approach, 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 mechanically 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 [8], 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.

2.2. Formal Requirements-Based Programming

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 the requirements capture stage to the automatic code generation phase [11]. Applicable to the class of system whose behavior can be described as a set of scenarios, the approach may also be used for reverse engineering, that is, in retrieving models and formal specifications from existing code. The approach can also be used to "paraphrase" (in natural language, etc.) formal descriptions of existing systems.

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 [9]—using the traces derived in phase 2. A deep[1] embedding of the laws of concurrency [4] in the theorem prover gives it sufficient knowledge of concurrency and of CSP to perform the inferencing. 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, or converting to other notations suitable for code generation (e.g., converting CSP to B and then using the code generating capabilities of the B Toolkit).

It should be re-emphasized that the "code" 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.

2.3. Short-cut R2D2C

The approach described in Section 2.2 is the way that R2D2C is intended to be applied, from requirements specification through code generation. The approach, however, requires significant computing power in the form of an automated theorem prover performing significant inferences based on traces input and its "knowledge" of the laws of concurrency. While this is well warranted for certain applications, it is likely to be beyond the resources of many developers and organizations. As a practical concession, we also define a reduced version of R2D2C called the "short-cut version" (Figure 2), whereby the use of a theorem prover is avoided, yet without sacrificing high confidence in the validity of the approach. The following describes each of the phases for the shortcut R2D2C:

S1 *Scenarios Capture*: As before, intended system behavior is described by scenarios input in natural language, or an appropriate graphical or semi-formal notation.

S2 *Translation to Intermediate Notation*: Scenarios are translated to an intermediate notation, termed EzyCSP, which is a simple natural language-like subset of CSP that can be used to describe a large number of situations and scenarios (recall that scenarios are domain specific).

S3 *Analysis*: While far more simple than CSP, EzyCSP allows some simple analyses to be performed.

S4 *Implementation in Java*: EzyCSP is sufficiently simple that it may easily be translated to Java and executed.

This simplified or shortcut approach clearly has significant disadvantages when compared to our full approach. First, the correctness of the development process is contingent on the correctness of both the translation of scenarios to the intermediate (EzyCSP) notation and the translation of

1 "Deep" in the sense that the embedding is semantic rather than merely syntactic.

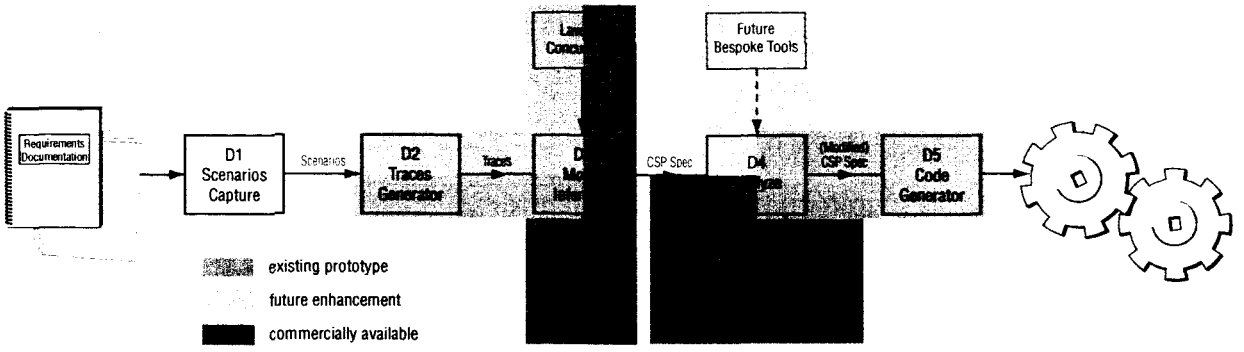

Figure 1. The entire process with D1 through D5 illustrating the development approach and R1 through R4 the reverse engineering.

EzyCSP to Java. However, the correctness of the translators for these is assured via a proof of correctness undertaken with the ACL2 theorem prover. Second, we do not have a reverse process, suitable to support reverse and (ultimately) re-engineering, for free; however, a Java-to-EzyCSP translator would certainly be possible for highly constrained subsets of Java.

The significant advantage of this simplified approach, however, is that although a proof of correctness involving a theorem prover is still required, this is required exactly once and would be performed by the support system developers (presumably expert in the art). This is significantly less expensive computationally than using a theorem prover in the development of each individual application.

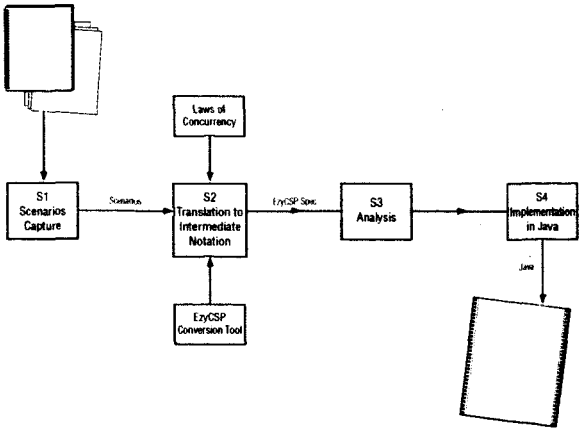

Figure 2. Short cut R2D2C.

3. Application Areas

The motivation for this work was the need for automatic code generation for ultra-high dependability systems, but the method described in this paper has a broad range of applicability.

We have developed a prototype tool to support the R2D2C approach [10, 12] and have successfully applied it in a range of areas:

***MultiAgent Systems*:** We have successfully undertaken the redevelopment of a prototype NASA system to automate the "lights out" (untended) operation of a ground control system. The system had previously been formally specified and checked by hand. Our R2D2C tool successfully found all the errors that we detected by hand, and additionally found a number of undetected errors, all in a matter of seconds [10].

***Wireless Sensor Networks*:** An application of the approach to the development of highly dependable wireless sensor networks is detailed in [7].

***Robotic Operations*:** We have been experimenting with generating code to control robotic devices. Perhaps more interesting is the use of this approach to investigate the validity and correctness of procedures for complex robotic assembly or repair tasks involving multiple robots and interacting parallel processes. We have begun exploratory work in this direction, to validate procedures from the Hubble Robotic Servicing Mission (HRSM)—for example, verification of the procedures for replacement of a wide-field camera on the Hubble Space Telescope (HST) is described in [11].

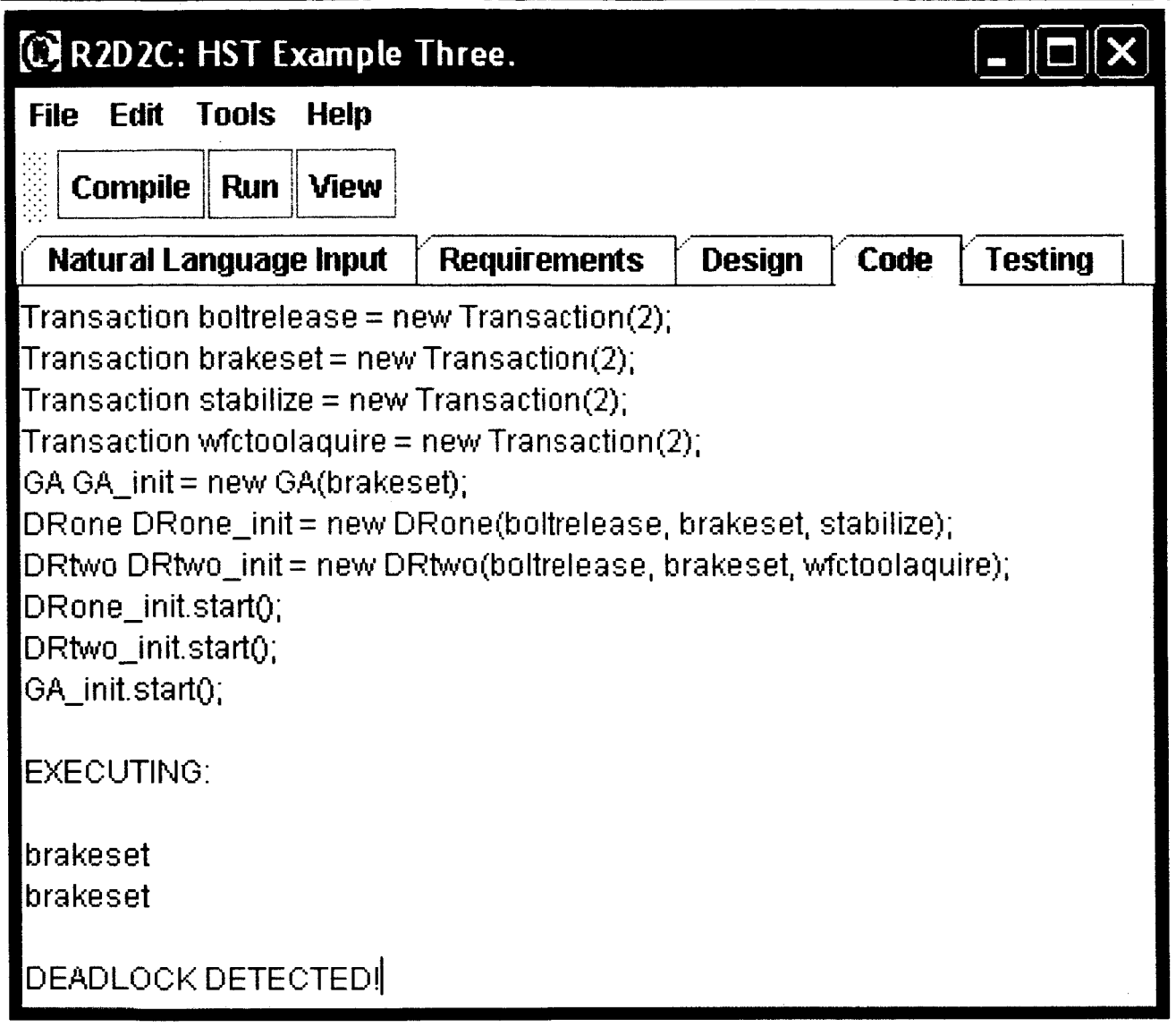

Figure 3. Application of R2D2C prototype tool to example robotic repair procedure, showing detection of deadlock.

4. An Example

To illustrate the capability of the R2D2C method using the prototype tool, we have selected an example procedure from HRSM for the replacement of a wide-field camera on the Hubble Space Telescope (HST).

This example procedure, one of many possible variations for replacing the camera, was modified to intentionally include an error for purposes of demonstrating the error-detection capability of our prototype tool.

Figure 3 shows an execution fragment from a run of the tool, in which the tool has produced a message indicating the example procedure leads to a deadlock condition. While this error was detected by running the Java program produced automatically by the tool, the formal model generated automatically by the tool can be subjected to automated theorem proving techniques to detect the same error, as well as many others.

The importance of this procedure verification capability looms large when it is recognized that no other tool or method can mechanically and automatically transform natural language requirements into a mathematically equivalent formal model and analyze that model for errors.

While procedure verification is an important area of ap-

plication of the R2D2C method, the more general area of software verification and requirements validation is targeted for application of an expanded prototype tool. Additional future work will include expanding the error-detection capability, incorporating theorem-proving relative to the formal model generated by the tool from the user's requirements expressed in natural language.

5. Conclusions

The development of complex parallel and distributed systems continues to pose difficulties and challenges. These will continue to grow unless we develop better means of controlling the complexity inherent in such systems and/or develop new techniques that will allow us to ensure that implementations truly meet requirements *and* allow for formal analysis and formal proof. Without a formal underpinning, such analysis and proof is impossible [1].

Requirements-Based Programming has been recommended as an approach that will lead to improved system quality, particularly in complex applications involving significant amounts of concurrent processing. R2D2C is the provisional name of a NASA approach to Requirements-Based Programming, which not only has automated (prototype) tool support, but also is fully formal, giving greater levels of confidence in the correctness of parallel and distributed systems.

Acknowledgments

This work was funded in part by the NASA Goddard Space Flight Center Technology Transfer Office. Denis Gračanin (Virginia Tech) and John Erickson (University of Texas at Austin) worked with us on the intermediate approach described in Section 2.3, and undertook the implementation of the prototype tool. The approach described in this paper is protected under United States and international Patent Applications assigned to the United States government.

References

[1] 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.

[2] D. Harel. From play-in scenarios to code: An achievable dream. *IEEE Computer*, 34(1):53–60, 2001.

[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.

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

[5] 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, 2004.

[6] 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.

[7] M. G. Hinchey, J. L. Rash, and C. A. Rouff. Towards an automated development methodology for dependable systems with application to sensor networks. In *Workshop on Information Assurance in Wireless Sensor Networks (WSNIA2005), Proc. 24th IEEE International Performance Computing and Communications Conference (IPCCC 2005)*, Phoenix, AZ, 7–9 April 2005. IEEE Computer Society Press.

[8] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall International, Englewood Cliffs, NJ, 1985.

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

[10] J. L. Rash, M. G. Hinchey, C. A. Rouff, D. Gracanin, and J. D. Erickson. Experiences with a requirements-based programming approach to the development of a NASA autonomous ground control system. In *EASe, 2nd IEEE Workshop on Engineering of Autonomic Systems, Proc. ECBS 2005, 12th IEEE International Conference on Engineering of Computer-Based Systems*, Greenbelt, MD, 4–7 April 2005.

[11] 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.

[12] 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.