Reuse and Formal Methods for Ada

Maureen Stillman

Odyssey Research Associates
Ithaca, NY 14850
Tel: (607) 277-2020
Email: maureen@oracorp.com

Abstract

We support the application of formal methods to the process of developing software. Reuse is an important piece of this software development process. Our aim is to give software engineers access to reusable designs, components and systems developed using formal methods. This paper describes a variety of formal methods tools and techniques, requiring different levels of training and sophistication from their users. The result of applying these tools and techniques will be high assurance, high quality systems composed of reusable building blocks.

Keywords: Reuse, Formal methods, Ada, Reuse libraries

Workshop Goals: To advance the state of the theory and practice combining formal methods and reuse.

Working Groups: reuse and formal methods, reuse process models, reuse education.

1 Background

We believe that trust and reliability are the central issues in reuse. When a software engineer contemplates reuse of even the smallest fragment of code he will begin the process with uneasy and suspicious feelings. Unanswered questions will haunt him such as:

- What is this software?
- Who wrote this software?
- Will I be sued for using this software?
- Will this software work the way I need it to work?
- Will this software work at all?
- How buggy will this software be?
- Does this software contain viruses?
- How hard will I have to work to modify this software to do what I want it to do?
- Can I really trust one or more software engineers who I don't know to write more reliable software than I can write myself?

In many cases, these nagging questions lead the software engineer to conclude, "I just better do it myself. It's safer. It will also be a lot less work and aggravation to debug my own code than to understand and fix someone else's code."

We believe that the use of formal methods will help mitigate some of these fears and encourage more software engineers to attempt reuse. Software should be labeled in some way with the degree of reliability of the code. One possibility is that the method by which the software was developed could be described and the formal approach used (if any) would be identified.

2 Experience in Reuse

ORA is a member of the Paramax STARS/ARPA team. Paramax has produced a Conceptual Framework for Reuse Processes (CFRP) [1, 2]. ORA is also a DSSA contractor and works in the domain of intelligent control systems [3]. We are well known for our expertise in the area of formal methods. Our experience in the reuse area includes the verification of a number of Booch reusable components using ORA's Penelope verification system [4]. We are currently working with NASA and Honeywell on a Boeing 777 reuse and formal methods project in avionics. ORA is writing a formal specification of the navigation control system.

3 Position

Our approach is to offer a range of formally based tools and techniques that are designed for use by software engineers with different levels of sophistication and training. Each of these methods offer increasing levels of assurance but require increased user training in exchange for greater assurance. We address the issue of reuse at each assurance level. Finally, we discuss libraries of reusable components that have different formal methods techniques and tools applied to them.

3.1 Lightweight Tools

The "lightweight" formal methods tools require little user training and no knowledge of specification languages or proof techniques. ORA has developed a set of these tools called AdaWise based on our formal work in Ada semantics. The AdaWise tools, while easy to use, are rigorous because they are based on existing formal methods technology developed at ORA. With these tools, we have combined the advantages of two approaches: the precision of formal verification and the automation of compilers. Our tools provide the precision of formal verification without requiring the user to write formal specifications or do proofs. The tools automatically check any Ada program for the presence of potential errors that typically are not detected at compile time by a compiler.

Currently under development is an alias checking tool, an elaboration order checking tool, an incorrect order dependence checker, and a definedness checker. These AdaWise tools verify a limited class of specialized properties for large programs. These properties include the absence of run-time anomalies, such as accessing variables before they have been assigned a value, and independence of a program's behavior from compiler choice by checking for erroneous executions and incorrect order dependences. The automatic checks for these properties are conservative , meaning that if no warnings are issued by the tool, then the property holds, but if warnings are issued, the property may or may not be violated. If the tool issues a warning, it also points out a specific region where a possible error may be. The user then has a localized region in which to look for a violation.

These tools can address areas where subtle bugs are introduced particularly when modifying code (for purposes of reuse). For example, changing the properties of variables and violating implicit assumptions can introduce unexpected bugs.

3.2 Middleweight Tools

"Middleweight" formal methods tools are specification languages that allow software engineers to state properties of their design or implementation. The language we have developed is based on Larch and is called Larch/Ada [5]. Formal specification of requirements, design or code can help users identify what the software or system is supposed to do in rigorous terms. This will give the software engineer a better idea of specifically what the software is intended to do. Thus, he can more intelligently select components for reuse and have greater confidence in their ability to work correctly.

These tools require educating programmers in a specification language. They would also have to be taught how to write formal specifications.

3.3 Verification Tools

Finally, we offer a formal verification system that requires the software engineer to write specifications and do proofs. This provides the highest level of assurance, but requires significant training.

The user must be trained in formal specifications and must understand first order logic and be able to guide the theorem proving system through proofs.

Our Ada verification system is called Penelope [6]. Penelope is an interactive verification CASE tool, developed by ORA, that helps programmers develop programs and their correctness proofs from mathematical specifications. Penelope can be used to formally verify the correctness of programs written in a large subset of Ada.

Penelope generates a set of verification conditions (VCs) from an Ada program specified in Larch/Ada. A proof of the VCs implies that the program meets its formal specifications.

3.4 Libraries of Components built using Formal Methods

ORA presented work on verification libraries at Tri-Ada in 1991 [7]. Real world software projects will vary in their use of formal methods and some will use none. We must generalize the idea of verification libraries to include support for software developed with different degrees of assurance.

Background to Ada Libraries

One of the design goals of Ada was to maintain semantic consistency across several separately compiled units. The Ada library was introduced to provide a repository of information about the various units that can be accessed by the compiler. In this way a newly compiled program will be semantically consistent with the units currently in the library.

Program units may be changed, however, and a change may invalidate the consistency of other units. The typical mechanism used to address this problem is to mark certain units as obsolete whenever any of their assumptions have changed. An obsolete unit has to be recompiled. Recompilation constitutes a change, and so this process is transitive.

Changes to an Ada program library may have far-reaching effects, and may require significant amounts of recompilation. In the case of compilation, this is only a minor problem. In a library where the components need to meet a high level of assurance, the traditional obsolescence mechanism becomes unacceptable. The fact that a modified piece of software now compiles does not mean that it meets the previous level of reliability or that it now performs the new functions we presume it will perform.

High Assurance Libraries

Formal methods libraries must deal with reliability problems on a global level: the meeting of previous assurance levels after the code has been modified or the design has been altered.

Superficially, the problems of a formal methods library are the same as those of a conventional Ada library: in addition to static semantic consistency, we now need to be concerned with dynamic semantic consistency, i.e. the correctness proofs. The details, however, are quite different. In addition to program objects, a high assurance library needs to deal with specifications, lemmas and proofs. New dependencies arise between these components. Automatic mechanisms to support a mix of reliability levels is a topic of further research.

We advocate some way of labeling library components in ordinary libraries with some measure of their reliability and the method used to achieve this. This will help the software engineer in making a decision concerning reuse. In formal methods libraries, with a fixed range of methods in use, we can use the above techniques, with a certain amount of automated assistance, to re-establish the original assurance level at a minimum cost.

4 Related work

Several researchers are working on connections between formal methods and reuse and have a variety of approaches [8, 9, 10]. Our approach is more low level than those cited since it deals directly with the reliability and correctness of design and code. Cheng has done work in the area of specifying software using a hierarchical approach.

5 Conclusions

Formal methods can be applied to systems and designs to increase system reliability. A software engineer's level of confidence in the correctness of other systems and components will encourage reuse. The method used to ensure the reliability of a reusable system or component should be kept with the system. The software engineer should restore the system to its previous level of assurance after making modifications for reuse.

6 Bibliography

References

[1] Software Technology for Adaptable Reliable Systems (STARS), "STARS Reuse Concepts Volume I - Conceptual Framework for Reuse Processes," Paramax STARS Technical Report STARS-TC-04040/001/01, US Air Force Systems Command, Electronic Systems Division, Hanscom Air Force Base, MA, September 1992.

[2] Software Technology for Adaptable Reliable Systems (STARS), "STARS Reuse Concepts Volume II - Reuse Process Architecture," Paramax STARS Technical Report STARS-TC-04040/002/00, US Air Force Systems Command, Electronic Systems Division, Hanscom Air Force Base, MA, September 1992.

[3] D. J. H. Taylor and D. R. Platek, "Domain-Specific Software Architectures for Hybrid Control," Special Report CMU/SEI-92-SR-9, Carnegie-Mellon University Software Engineering Institute, June 1992.

[4] C. T. Eichenlaub, C. D. Harper, and G. Hird, "Using Penelope to Assess the Correctness of NASA Ada Software: A Demonstration of Formal Methods as a Couterpart to Testing," NASA Contract Report 4509, Odyssey Research Associates, Inc., Ithaca, NY, May 1993.

[5] O. R. Associates., *Larch/Ada Reference Manual.* Ithaca, NY, September 1989.

[6] D. Guaspari, C. Marceau, and W. Polak, “Formal verification of Ada programs,” *IEEE Software Engineering*, vol. 16, pp. 1058–1075, Sept 1990.

[7] G. R. Hird, “Towards Reuse of Verified Ada Software,” in *Proceedings of Tri-Ada '90*, (Ithaca, NY), pp. 14–21, December 1990.

[8] M. Simos, “Towards an industry-wide consensus reuse process model,” in *Proceedings of the Fifth Annual Workshop on Software Reuse*, IEEE Computer Society and University of Maine, 1992.

[9] J. Knight and D. M. Kienzle, “Reuse of specifications,” in *Proceedings of the Fifth Annual Workshop on Software Reuse*, IEEE Computer Society and University of Maine, 1992.

[10] B. Cheng and J. jang Jeng, “Formal methods applied to reuse,” in *Proceedings of the Fifth Annual Workshop on Software Reuse*, IEEE Computer Society and University of Maine, 1992.

7 Biography

Maureen J. Stillman is VP of Engineering at ORA Corporation. Ms. Stillman has been manager of the Penelope formal methods project for six years. At ORA, she has managed a number of projects in formal methods, security, and network management. From 1985-1987 she worked for Bolt, Beranek and Newman as manager of a network management project. From 1978-1983 she worked for MIT Lincoln Laboratory designing network protocols for distributed mobile packet radio networks. She received an MS in Computer Science from Northwestern University in 1985 and a BS in Mathematics from the University of Illinois in 1977.