SOFTWARE ENGINEERING NOTES 19, 4 (OCT. 1994), 64-67.

Part V: Annotated Bibliography of RESOLVE Research

Stephen H. Edwards

This brief annotated bibliography describes a select subset of the RESOLVE literature, focusing on the most accessible references for the interested reader who would like to learn more about RESOLVE. A more complete bibliography of RESOLVE papers is available by anonymous FTP from host `ftp.cis.ohio-state.edu` in the file `pub/rsrg/RESOLVE-refs.txt`.

Most of the papers in this list that are not published in journals or major conference proceedings can be ordered from the National Technical Information Service (NTIS) or from University Microfilms (UMI), as noted, or are available electronically. PostScript versions of all Ohio State University technical reports listed (mostly pre-prints of papers to appear and recent Ph.D. dissertations) are available by anonymous FTP from `ftp.cis.ohio-state.edu` in the directory `pub/tech-report`. Documents available at this location are marked in the bibliography with "[OSU-FTP]". Similarly, many of the papers appearing in the various Workshops on Software Reuse (WISRs) are available by anonymous FTP. These files can be found at the WISR archive maintained at the University of Maine, host `gandalf.umcs.maine.edu`, in the directory `pub/WISR`. Documents available at this location are marked with "[WISR-FTP]".

More information on RESOLVE, including access to these and other documents and a demonstration component archive, is available through World-Wide Web. The URL is:

```
http://www.cis.ohio-state.edu/hypertext/rsrg/RSRG.html
```

References

[Edwards 90] Edwards, S., *An Approach for Constructing Reusable Software Components in Ada*, IDA paper P-2378, Institute for Defense Analyses, Alexandria, VA, Sept. 1990. Available from NTIS (phone: 703-487-4650), access number AD-A233 662.

This report discusses reusable software in Ada. It concentrates on many of the technical problems involved in designing software components, and introduces what amounts to an early version of the RESOLVE/Ada discipline [Hollingsworth 92b]. It also contains a significant number of Ada code examples.

[Edwards 93a] Edwards, S.H., "Inheritance: One Mechanism, Many Conflicting Uses", *Proc. 6th Ann. Workshop on Software Reuse*, L. Latour, ed., Nov. 1993. [WISR-FTP].

This position paper argues that most object-oriented languages inappropriately define inheritance as a (single) programming mechanism, although it is used for many diverse conceptual purposes. Conflicts between different uses of inheritance lead to many of the problems associated with it. Separating out the various uses for inheritance by using distinct language mechanisms is one approach to eliminating these conflicts and the difficulties they bring.

[Edwards 93b] Edwards, S.H., "Common Interface Models for Reusable Software", *Intl. J. of Software Eng. and Knowledge Eng. 3*, 2 (June 1993), 193-206.

This article presents an informal, natural language description of the notion of an abstraction over several families of components, here called a "common interface model" (CIM). CIMs can be used to capture common patterns that recur across many modules, for both programmatic and human uses. This notion is similar to the more formal notion of theories in OBJ, but there are some critical differences.

[Edwards 94] Edwards, S.H., *A Formal Model of Software Subsystems*. Ph.D. diss., Dept. of Comp. and Inf. Sci., Ohio State Univ., Columbus, Dec. 1994, to appear.

This dissertation presents a mathematically formal, language-independent model of software components that captures many intuitive notions about reusable software parts that practitioners have observed through experience. This model is compared with the language-specific models embodied in OBJ, RESOLVE, Eiffel, and Standard ML, as well as the 3C model. The RESOLVE formal semantics is now based on this model.

[Ernst 91] Ernst, G.W., Hookway, R.J., Menegay, J.A., and Ogden, W.F., "Modular Verification of Ada Generics", *Comp. Lang. 16*, 3/4 (1991), 259-280.

This paper describes the foundation that is necessary to verify that a generic component is correct, once and for all, without knowing how it is instantiated. It concentrates on the expressiveness issues that make this a tough technical problem.

[Ernst 94] Ernst, G.W., Hookway, R.J., and Ogden, W.F., "Modular Verification of Data Abstractions with Shared Realizations", *IEEE Trans. on Software Eng. 20*, 4 (Apr. 1994), 288-307.

This paper gives a method for modularly specifying and verifying data abstractions where multiple abstract objects share a common implementation-level data structure. Such implementations allow for very efficient use of memory or other resources, but complicate modular verification. The paper also discusses the model that serves as the basis for RESOLVE's formal semantics.

[Harms 90] Harms, D.E., *The Influence of Software Reuse on Programming Language Design*. Ph.D. diss., Dept. of Comp. and Inf. Sci., Ohio State Univ., Columbus, Aug. 1990. Available from UMI (phone: 800-521-0600).

This work investigates programming language features that encourage and discourage the design of reusable software components. It includes an evaluation of several modern programming languages in this regard, concluding that none has the features necessary to encourage and facilitate the design and implementation of such components. It then describes the (original) RESOLVE programming language and discusses some rationale for core features of the language, including the treatment of math and programming types and constructs that support modular reasoning about behavior.

[Harms 91] Harms, D.E., and Weide, B.W., "Copying and Swapping: Influences on the Design of Reusable Software Components", *IEEE Trans. on Software Eng. 17*, 5 (May 1991), 424-435.

This paper discusses many of the fundamental RESOLVE ideas, including the method of modeling program types by mathematical types; the difficulties with assignment as the built-in data movement operator; and the benefits of basing designs on swapping, especially for generic ADTs.

[Heym 94a] Heym, W.D., Long, T.J., Ogden, W.F., and Weide, B.W., *Mathematical Foundations and Notation of RESOLVE.* OSU-CISRC-8/94-TR45, Dept. of Comp. and Inf. Sci., Ohio State Univ., Columbus, Aug. 1994. [OSU-FTP].

This paper discusses the logical foundations and teminology of RESOLVE, the built-in RESOLVE notation for writing mathematics, and the RESOLVE mechanisms that support description of mathematics that has no built-in notation. It is intended to serve primarily as a reference document.

[Heym 94b] Heym, W.D., *Computer Program Verification: Improvements for Human Reasoning.* Ph.D. diss., Dept. of Comp. and Inf. Sci., Ohio State Univ., Columbus, Dec. 1994, to appear.

This dissertation formalizes, in a direct, natural way, the informal pattern of reasoning generally used with programs written in modular, imperative languages such as RESOLVE. The formal semantics provides a solid basis against which to check the soundness and (relative) completeness of a formal proof system (in this case, the "indexed method"; see [Krone 88] for another method).

[Hollingsworth 92a] Hollingsworth, J.E., and Weide, B.W., "Engineering 'Unbounded' Reusable Ada Generics", *Proc 10th Ann. Natl. Conf. on Ada Tech.,* ANCOST, Inc., Feb. 1992, 82-97.

This paper introduces one of the RESOLVE abstractions that replaces pointers and shows how to use it. It also explains several implementation options for the abstraction and their relative advantages, and provides some Ada code examples.

[Hollingsworth 92b] Hollingsworth, J.E., *Software Component Design-for- Reuse: A Language-Independent Discipline Applied to Ada*. Ph.D. diss., OSU-CISRC-1/93-TR01, Dept. of Comp. and Inf. Sci., Ohio State Univ., Columbus, Aug. 1992. [OSU-FTP].

This work describes the "RESOLVE/Ada discipline": How to write Ada components using RESOLVE specification and design principles, so they are modularly certifiable. It is self-contained in that it includes a terse discussion of the specification approach and language, including all of the constructs used in the examples. It is also the authoritative description of the principles that RESOLVE/Ada programmers (authors of concepts and realizations, and authors of client programs) are expected to follow. Numerous examples are given, including two of the RESOLVE encapsulations of pointers.

[Krone 88] Krone, J., *The Role of Verification in Software Reusability*. Ph.D. diss., Dept. of Comp. and Inf. Sci., Ohio State Univ., Columbus, Aug. 1988. Available from UMI (phone: 800-521-0600).

This work describes the verification rules that form the basis for RESOLVE's original syntax-driven verification procedure. In addition to basic rules for straight-line code and procedure calls, the rules cover conceptual modules, realization modules, and module instantiation. An example of a proof of total correctness for a complete component that is layered on top of another component is given.

[Krone 93] Krone, J., and Sitaraman, M., "On Modularity and Tightness of Real-Time Verification", *Real-Time Newsletter 9*, 1/2 (Spring/Summer 1993), 109-115. Also in *Proc. 10th IEEE Workshop on Real-Time Operating Systems and Software*, IEEE, May 1993.

This paper discusses several problems arising from attempts to modularly verify tight execution-time bounds, and the importance of this objective for software components used in real-time applications.

[Muralidharan 90] Muralidharan, S., and Weide, B.W., "Should Data Abstraction Be Violated to Enhance Software Reuse?", *Proc. 8th Ann. Natl. Conf. on Ada Tech.,* ANCOST, Inc., Mar. 1990, 515-524.

This paper discusses some problems involved in using a hypothetical code inheritance construct of Ada, showing why it does not effectively support reuse. Some Ada code examples are included, which reflect an early RESOLVE/Ada style.

[Parrish 91] Parrish, A., and Zweben, S.H. "Analysis and Refinement of Software Test Data Adequacy Properties", *IEEE Trans. on Software Eng. 17*, 6 (June 1991), 565-581.

This paper discusses component-based software testing from the standpoint of test data adequacy.

[Sitaraman 92a] Sitaraman, M., "A Class of Mechanisms to Facilitate Multiple Implementations of a Specification", *Proc. 1992 Intl. Conf. on Comp. Lang.*, IEEE, Apr. 1992, 182-191.

The paper motivates the need for, and ramifications of, language features supporting multiple implementations of the same abstract specification. A class of mechanisms (not supported in current languages) to independently name, reference, and parameterize specifications and implementations are identified as essential to facilitate development and use of multiple implementations.

[Sitaraman 92b] Sitaraman, M. "Performance-Parameterized Reusable Software Components", *Intl. J. of Software Eng. and Knowledge Eng. 2*, 4 (Oct. 1992), 567-587.

This article describes how components can be parameterized by the lower-level abstractions they depend on, in order to provide greater performance flexibility and escape the "combinatorial explosion" problem that arises when these alternatives are not parameterized. By parameterizing components so that any appropriate lower-level constituent components can be simply "plugged in," inexpensive and flexible performance tuning is provided to a client without requiring any form of component source code modification, recompilation, or reverification.

[Sitaraman 92c] Sitaraman, M., "A Uniform Treatment of Reusability of Software Engineering Assets", *Proc. 5th Ann. Workshop on Software Reuse*, L. Latour, ed., Oct. 1992. [WISR-FTP].

The paper enhances the 3C model with "constraints", which contain non-functional specifications (e.g., performance constraints). It explains the applicability of the 4C model to all software lifecycle artifacts from requirements documents to code components.

[Sitaraman 93] Sitaraman, M., Welch, L.R., and Harms, D.E., "On Specification of Reusable Software Components", *Intl. J. of Software Eng. and Knowledge Eng. 3*, 2 (June 1993), 207-229.

This article provides an overview of RESOLVE from a specification perspective, comparing it to Z and Larch. It explains why component specifications must be formal yet understandable, as well as abstract and implementation-independent. Each specification also must make it possible to demonstrate the correctness of an implementation of the specification and permit formal reasoning about its behavior in a client program.

[Sitaraman 94] Sitaraman, M., "On Tight Performance Specification of Object-Oriented Software Components", *Proc. 1994 Intl. Conf. on Software Reuse*, IEEE, Nov. 1994, to appear.

This paper points out why specification of performance should appear not with an abstract component's functional specification or with a concrete component's implementation, but in an implementation-related module that lies "between" them. One of the reasons for this conclusion is an expressiveness problem: Performance specifications may need to involve "intermediate" mathematical models to permit expression of tight performance specifications while preserving information hiding.

[Weide 91] Weide, B.W., Ogden, W.F., and Zweben, S.H. "Reusable Software Components", in *Advances in Computers, vol. 33*, M.C. Yovits, ed., Academic Press, 1991, 1-65.

This book chapter provides an overview of the software engineering approach used in RESOLVE. It describes the general model of software structure and the related 3C model, several basic concept design principles, and some simple examples; and briefly compares the RESOLVE approach to component engineering with those used frequently in Ada, C++, and Eiffel.

[Weide 92] Weide, B.W., and Hollingsworth, J.E., "Scalability of Reuse Technology to Large Systems Requires Local Certifiability", *Proc. 5th Ann. Workshop on Software Reuse*, L. Latour, ed., Oct. 1992. [WISR-FTP].

This position paper explains why software components must be certified once upon entry into a component library, and not once per use, if the most important benefits of reuse are to be achieved; that it, it argues for the importance of local certifiability to reuse.

[Weide 93] Weide, B.W., Heym, W.D., and Ogden, W.F., "Procedure Calls and Local Certifiability of Component Correctness", *Proc. 6th Ann. Workshop on Software Reuse*, L. Latour, ed., Nov. 1993. [WISR-FTP].

This position paper discusses how common practices involving procedure calls (e.g., in Ada and C++) thwart modular reasoning about component behavior. It also discusses previous approaches to dealing with these problems at a formal level, and argues that call-by-swapping as a parameter-passing mechanism eliminates the difficulties.

[Weide 94a] Weide, B.W., and Hollingsworth, J.E., *On Local Certifiability of Software Components*. OSU-CISRC-1/94-TR04, Dept. of Comp. and Inf. Sci., Ohio State Univ., Columbus, Jan. 1994. [OSU-FTP].

This report discusses probably the most fundamental objective of the RESOLVE approach: to be able to reason about the behavior of component-built programs in a "modular" fashion. It is easy reading for someone not familiar with RESOLVE, and it is a good place to start if you are interested in the RESOLVE approach to verification.

[Weide 94b] Weide, B.W., Edwards, S.H., Harms, D.E., and Lamb, D.A., "Design and Specification of Iterators Using the Swapping Paradigm", *IEEE Trans. on Software Eng. 20*, 8 (Aug. 1994), 631-643.

This paper describes the specification of iterators from a RESOLVE perspective, i.e., based on the principles described in [Harms 91] and [Hollingsworth 92b]. By providing and discussing an evolutionary series of iterator designs, the paper shows how formally-specified iterators can be designed to admit more efficient implementations and support modular reasoning. It gives examples of some clearly non-trivial implications of the RESOLVE style, and examples of more complex specifications than are found in the introductory papers.

[Weide 94c] Weide, B.W., Ogden, W.F., and Sitaraman, M., "Recasting Algorithms to Encourage Reuse", *IEEE Software 11,* 5 (Sept. 1994).

This paper describes a technique for extending object-oriented component design by recasting algorithms as objects (called "machines"), thereby improving both functional and performance flexibility. It is written for an audience unfamiliar with RESOLVE, but the examples are non-trivial: a novel sorting component and a component to find minimum spanning forests in graphs.

[Zweben 92] Zweben, S.H., Heym, W.D., and Kimmich, J., "Systematic Testing of Data Abstractions Based on Software Specifications", *J. of Software Testing, Verification and Reliability 1*, 4 (1992), 39-55.

This paper shows how to adapt conventional white-box strategies to test components specified in a RESOLVE-like manner. The strategy then becomes specification-based, rather than code-based. Theoretical and empirical evaluations of this approach are discussed.

[Zweben 94] Zweben, S.H., Edwards, S.H., Weide, B.W., and Hollingsworth, J.E., *The Effects of Layering and Encapsulation on Software Development Cost and Quality*, OSU-CISRC-4/94-TR21, Dept. of Comp. and Inf. Sci., Ohio State Univ., Columbus, Apr. 1994. [OSU-FTP]. Submitted for publication; currently in revision.

This report discusses three controlled experiments designed to gather empirical evidence supporting the practice of layering newly written code on top of earlier encapsulated components, rather than simply adding code to old modules. The results of the experiments support the contention that layering significantly reduces the effort required to build new components.