Toward a Model of Reusable Software Subsystems*

Stephen H. Edwards

Department of Computer and Information Science
The Ohio State University
2036 Neil Avenue Mall
Columbus, OH 43210
Tel: (614) 293–0437
Email: edwards@cis.ohio-state.edu

Abstract

No one has yet developed a generally acceptable model of reusable software that fully captures the nature of software parts and how they should be used to construct new systems. This paper discusses developing a model of reusable subsystems, where a subsystem can vary in grain size from large scale generic architectures to single module components. It draws on previous work from WISR'91 to describe the need for an analytical model of reusable subsystems, the purposes for such a model, the broad properties such a model should have, and the critical issues that have already been identified which must be addressed. It points out the strong incentive researchers and practitioners have to drive toward a model for software building blocks and how they are used to construct systems.

Keywords: Model of reusable software components, large-grain components, generic software architectures, 3C model.

Workshop Goals: Cross-fertilization of ideas with other researchers; building on the results of WISR'91; keeping abreast of other ongoing reuse work; advancing the theoretical foundations of software reuse.

Working Groups: Design guidelines for reuse, reuse and formal methods, reuse and OO methods.

*This work is supported in part by the National Science Foundation, CCR-9111892.

1 Background

Within the past several years, increasing efforts have been made to define exactly what software reuse is, and to describe the nature of the "building blocks" practitioners should use. While this has led to some progress in our understanding of software reuse and how to best support it, no one has yet developed a generally acceptable model that fully captures the nature of software parts and how they should be used to construct new systems. To this end, this position paper describes some of the previous work towards modeling software components, and lays out some of the basic groundwork for the creation of a generally acceptable model of reusable software.

One of the more recent attempts to characterize reusable software components is the "3C model." The 3C model of reusable software components was originally developed at the "Reuse in Practice" workshop, held July 11-13, 1989 in Pittsburgh, Pennsylvania [Tra90], and is an enhancement of the Concept/Context model initially proposed by Tracz in his dissertation work at Stanford [Tra92]. The 3Cs stand for:

Concept The abstraction captured in a component.

Content The implementation of that abstraction.

Context The environment in which the component is designed to operate.

A more detailed description of the 3C model can be found in [Edw90], which uses this model to develop a set of programming guidelines for creating reusable Ada software.

While the 3C model evolved out of module-level concerns about creating reusable components, additional work presented at WISR'91 was aimed at modeling larger grained reusable parts. Such approaches are typified by [LM91], [LvK91], and [Bat91]. These three works are all aimed at getting a better handle on the reuse of larger scale building blocks such as subsystems.

The WISR'91 "Theoretical Foundations" working group centered its activities on the problems of developing an integrated model of reusable software. Ten individuals, including all of the above-cited researchers, brought a very broad base of experience to the table, and the working group compared and debated the features of various proposed models. In the end, the working group gathered a consensus opinion on the purposes of such a model, the key points that a generally acceptable model should address, and the need for formal aspects within the model [LBE+92].

The ideas presented in this paper are primarily derived from the results of the WISR'91 Theoretical Foundations working group. Further, the subject of this paper is developing a model of reusable subsystems, where a subsystem can vary in grain size from large scale generic architectures to single module components.

2 Position

Within the software reuse community, it is a widely held view that the most critical obstacles to software reuse are institutional and rooted in personal attitudes, management practices, and economic concerns. Some individuals may even contend that the technical details of creating and managing reusable components are well under control—this view is particularly popular in the object oriented community. While the nontechnical problems facing software reuse are certainly

important, one might ask whether there are any significant technical problems remaining. These concerns should lead one to raise the following questions about developing a model of reusable subsystems:

- Why do we need one?
- What will it be used for?
- What properties should it have?
- What issues should it address?

The aim of this paper is to propose answers to these basic questions that demonstrate the need for a model of reusable subsystems, and that can form the basis for synthesizing such a model.

2.1 Why Do We Need A Model of Reusable Subsystems?

The development of analytical models is one of the milestones in the evolutionary process of engineering disciplines. There are several critical reasons for this, many of which are based on replacing *ad hoc* practices with more rigorous approaches. The key is the fact that even an incorrect model can push progress forward. For example, Newtonian mechanics is a flawed model of physics, but it is nevertheless extremely useful and was a critical step forward in the evolution of physics.

A model of reusable subsystems will provide a structured framework that can be used to analyze and critique proposed tools, techniques, and environments. Further, evaluation of these analyses and critiques can reflect feedback on the model itself, allowing it to be refined. Such a model would also serve as a structuring mechanism for encoding practitioners' knowledge about constructing reusable software for the education and training of future software engineers.

In addition, a general model of reusable subsystems would highlight the aspects of a software subsystem that are important for reuse. This would allow researchers to identify subsystem properties that either enhance or inhibit reuse. Once enhancing or inhibiting properties are identified, it would be possible to identify how current methods, tools, languages, etc., support or confound creating subsystems with these properties. Similarly, the model could be used to point out limits in current capabilities for reasoning about how subsystems interact and behave when combined. Finally, such a model would allow us to develop criteria for an effective representation for software subsystems.

Taken together, these reasons form a strong argument for pursuing development of a model of reusable software. Most importantly, one will never know whether the critical technical problems are solved unless one can be convinced that the critical technical problems have been identified. The best way to gain this confidence is to ensure that a systematic and rigorous approach has been used to explore and identify those critical issues. It seems that an analytical model of software building blocks is a major milestone along that path.

2.2 What Will Such a Model Be Used For?

If one accepts the need for a model of reusable subsystems, the next step is to identify the purposes that model will serve. Some of the broad goals that can be achieved by such a model have been

mentioned in the previous section. In addition, a model of reusable software can be used for the following purposes[1]:

- Developing a representation of generic software architectures that facilitates understanding, makes manipulation easier, and simplifies instantiation.
- Reasoning about the functional behavior, nonfunctional behavior, resource requirements, and other characteristics of combinations of subsystem components.
- Cleanly dealing with the composition and analysis of highly parameterized subsystems.
- Formalizing an approach to software development (and reuse).
- Developing an effective approach to storing, categorizing, and retrieving subsystem components based on useful properties identified by the model.
- Form a basis for extending experience with developing reusable software to other life cycle objects.

2.3 What Properties Should a Model of Reusable Subsystems Have?

From current experience with previous models of software components, it is critical that a model of reusable subsystems must provide mechanisms for taking into account the following[LBE+92]:

- The abstraction embodied in the subsystem,
- The requirements and needs of a potential client considering using a subsystem,
- The subsystem implementer's guarantees, and
- The subsystem's implementation constraints.

This forms a short list of properties that a comprehensive model should possess, but hints at broader requirements. In fact, one can generalize these ideas into the individual perspectives and needs of many parties involved with a large-grained component:

- The client or user of a subsystem component,
- The designer, implementer, and developer of the subsystem,
- Designer(s) of other subsystems who intend for their components to interact with or work along side the given subsystem,
- Maintainer(s) of system(s) built using the given subsystem as a building block, and
- Maintainer(s) of the subsystem itself.

Clearly, an effective model of reusable subsystems will be useful to individuals acting in each of the above roles. As such, the model should be formulated with careful attention to the divergent needs of these roles to assure that relevant subsystem properties are highlighted.

[1] The bulk of this list is distilled from [LBE+92].

In addition, the WISR'91 Theoretical Foundations working group recognized the need for a formal basis for any model of reusable software. In particular, there is a very strong need for formal notions of:

- A software subsystem.
- The equivalence of two or more software subsystems, particularly when interface conventions vary.
- The definition of one subsystem component in terms of another (i.e., enhancement, refinement, inheritance, etc.).
- Component composition methods, with an emphasis on reasoning about the properties of the resulting composite system.

Corresponding proof methods that support the modeled processes are also of central theoretical importance. Such proof methods would include methods for proving subsystems match their specifications, determining the validity of subsystem compositions, and reasoning about nonfunctional properties of composite subsystems.

Further, it is of paramount importance that the model stress *modular* subsystem properties [Hol92] [WOZ91]. Component properties that cannot be established outside the context of a complete system are of little use to reusable component clients, maintainers, librarians, or anyone else. Crafting a model that allows critical subsystem properties to be established for all uses of a subsystem, and that highlights why other desirable properties may not be certified in a modular way, can go a long way toward formalizing many notions of "quality" reusable software.

2.4 What Issues Should Such a Model Address?

In addition to the broad properties presented in the previous section, reuse researchers and practitioners have identified many critical issues that must be addressed by a reuse model [LBE+92]. These issues, which are refinements of the broader requirements presented in previous sections, indicate that a mode of reusable software should do the following:

- It should focus on reuse-in-the-large.
- It should be flexible enough to effectively characterize generic software architectures.
- It should account for "building blocks" based on generative approaches[2] to reuse.
- It should reflect the complexities of integrated component libraries as well as stand-alone components [LM91].
- It should allow for the capability of formally specifying subsystems and verifying the correctness of their implementation.
- It should provide a formalism for defining the relationship between subsystem component specifications and their corresponding implementations. It should also facilitate human understanding of this mapping.

[2]The distinction between generative and compositional approaches to reuse dates back to [BR87]. The generative approach is also a central theme in [Bat91].

- It should capture modeling the definition of one abstraction in terms of (one or more) others.
- It should capture modeling the definition of one subsystem implementation in terms of (one or more) others.

2.5 Discussion

Together, Sections 2.1–2.4 provide tentative answers to the questions raised in Section 2. These sections describe the need for an analytical model of reusable subsystems, the purposes for such a model, the broad properties such a model should have, and the critical issues that have already been identified which must be addressed. It seems there is a strong incentive for researchers and practitioners to drive toward a model for software building blocks and how they are used to construct systems. Further, such a model can clearly be put to good use. The next step is to evolve an understandable and generally acceptable model of reusable subsystem components that can serve as a seed crystal for a discipline of software reuse.

3 Comparison

There have been many proposed approaches, both formal and informal, to modeling software components. Three separate efforts that WISR'91 participants are likely to be familiar with give a broad flavor of past approaches: the 3C model [Tra90, Edw90], Latour's layered generic architectures [LM91], and Batory's Genesis [Bat91].

The 3C model was the initial focus of the WISR'91 Theoretical Foundations working group [LBE+92]. It evolved out of a distillation of experience with small scale, module level components, and it is centered around a separation of concerns between the client and the implementer of a component. However, the model has often been criticized for concentrating on reuse-in-the-small, and for failing to scale well to larger grained components. Further, it currently lacks a sufficiently formal definition, an an easily understandable description.

Nevertheless, others have continued to build on this promising start. Latour and Meadows have been working on scaling up the 3C model by developing a schema for thinking about generic software architectures and integrated component libraries [LM91]. This *layered generic architecture* (LGA) schema is an attempt to refocus on reuse-in-the-large with coarse grained components. This work addresses some of the current shortcomings in the 3C model, but still falls short of the goals and issues set forth in Sections 2.3 and 2.4. In particular, it still lacks a formal basis, it does not address the perspectives of some individual roles involved in the reuse process, it does not completely encompass generative approaches to subsystem implementations, and so on.

The Genesis work of Batory [Bat91, Bat88] presents a complementary model of reuse aimed at addressing the gap between *very large scale reuse* and *large scale reuse*. This work, which was demonstrated at WISR'91, is based heavily on generative approaches to reuse using coarse grained, plug-compatible components to create working database systems. It addresses several critical issues that are missing in both the 3C model and the LGA schema. Unfortunately, this work has been criticized as being domain specific and difficult to generalize. It also provides inadequate mechanisms for describing the abstractions embodied by subsystems, describing a subsystem implementer's guarantees, or capturing a subsystem's implementation and environmental constraints. Finally, like the other two approaches, it lacks a sufficient formal basis.

There are many additional research projects that are relevant to modeling software subsystems for reuse. However, like the three presented in this section, all seem to lack properties listed in Section 2.3 or fail to address critical issues listed in Section 2.4. In some sense then, this paper presents the beginnings of a "metamodel" that describes the properties that a successful model of reusable components should have, within the limits of current research experience. This metamodel points out what is currently known about the limitations of existing approaches, and can possibly be used to hint at the path toward a more realistic model of reusable subsystem components.

References

[Bat88] D. S. Batory. Concepts for a database system synthesizer. *ACM PODS*, pages 184–192, 1988.

[Bat91] Don Batory. On the difference between very large scale reuse and large scale reuse. In Larry Latour, Steve Philbrick, and Chandu Bhavsar, editors, *Proceedings of the Fourth Annual Workshop on Software Reuse*, November 1991.

[BR87] Ted Biggerstaff and Charles Richter. Reusability framework, assessment, and directions. *IEEE Software*, 4(2):41–49, March 1987.

[Edw90] Stephen H. Edwards. An approach for constructing reusable software components in Ada. IDA Paper P-2378, Institute for Defense Analyses, Alexandria, VA, September 1990.

[Hol92] Joseph Hollingsworth. *Software Component Design-for-Reuse: A Language Independent Discipline Applied to Ada*. PhD thesis, Dept. of Computer and Information Science, The Ohio State University, Columbus, OH, to appear 1992.

[LBE+92] Larry Latour, Don Batory, Stephen Edwards, Haim Kilov, Trudy Levine, Haikuan Li, Luqi, Ben Sijtsma, Will Tracz, and Martin Wirsing. Theoretical foundations working group report. Available by Anonymous FTP from the WISR Software Repository at gandalf.umcs.maine.edu, 1992.

[LM91] Larry Latour and Curtis Meadow. Scaling up the 3cs model—a schema for extensible generic architectures. In Larry Latour, Steve Philbrick, and Chandu Bhavsar, editors, *Proceedings of the Fourth Annual Workshop on Software Reuse*, November 1991.

[LvK91] Haikuan Li and Jan van Katwijk. A model for reuse-in-the-large. In Larry Latour, Steve Philbrick, and Chandu Bhavsar, editors, *Proceedings of the Fourth Annual Workshop on Software Reuse*, November 1991.

[Tra90] Will Tracz. Implementation working group summary. In Jr. James Baldo, editor, *Reuse in Practice Workshop Summary*, pages 10–19, Alexandria, VA, April 1990. IDA Document D-754, Institute for Defense Analyses.

[Tra92] William Tracz. *Formal Specification of Parameterized Programs in LILEANNA*. PhD thesis, Dept. of Electrical Engineering, Stanford University, Stanford, CA, to appear 1992.

[WOZ91] Bruce W. Weide, William F. Ogden, and Stuart H. Zweben. Reusable software components. In M. C. Yovits, editor, *Advances in Computers*. Academic Press, 1991.

4 Biography

Stephen H. Edwards is a doctoral student in the Department of Computer and Information Science at the Ohio State University. His research interests are in software engineering, the use of formal methods in programming languages, and information retrieval technology. The current focus of his work is on developing and refining a formal model of software components which explicitly addresses reuse concerns. Prior to entering the Ohio State University, Mr. Edwards was a research staff member at the Institute for Defense Analyses, where he worked on software reuse activities, simulation frameworks, active databases, and Ada programming issues. Mr. Edwards received his MS in computer science from Ohio State in 1992, and his BS in electrical engineering from Caltech in 1988.