Uncontrolled Reference Semantics Thwart Local Certifiability

Joe Hollingsworth

Indiana University Southeast

LF218A-4201 Grant Line Road

New Albany, Indiana 47150

Tel: (812)941-2425

Email: jholly@ius.indiana.edu

Abstract

This paper's position focuses on software component design-for-reuse down in the trenches, similar to the focus of WISR 92's working group "Design-for-Reuse in the Trenches." The technical focus is on components implemented using reference semantics. The position is: components implemented using uncontrolled reference semantics should not be allowed into the reusable component library. This is because they thwart local certifiability. This position is controversial because 1) not everybody agrees on the necessity of locally certifiable components, and 2) most published component libraries rely heavily on uncontrolled reference semantics.

Keywords: design-for-reuse, design principles, local certifiability

Workshop Goals: interact; advance software engineering and reuse technology

Working Groups: design guidelines for reuse, reuse and formal methods, reusable component certification

1 Background

Our recent research has concentrated on design principles that lead to the design of high-quality software components. The design principles focus on the technical details of component design, not on the higher level reuse problems (e.g., library organization, and searching). In [4] we identified 41 explicit principles for the design and implementation of components in Ada. Components designed by these principles are called RESOLVE/Ada components. Since January of 1993, the author's Ph.D. dissertation (which contains these principles and their justification) has been available via anonymous ftp over the Internet. Over 100 researchers and practitioners across world have ftp'ed a copy and notified the author via e-mail. The author has continued this line of research by developing more components in Ada which follow the discipline outlined in [4] and has begun investigating similar design principles for components in C++.

2 Position

In our research we identify local certifiability as fundamental to successful systematic reuse of software components (see [10] and [4]). By reuse, we mean that the component can be reused without alteration of the code that implements it (of course supplying different actual parameters to generic components is allowed). At a minimum, software components need to be locally certifiable for correctness. There are other properties that we want to be locally certifiable (e.g., composability), but correctness is of the utmost importance. Additionally, the Certification Working Group from WISR '92 reported that "all reusable components ... should be certified" and "local certification is necessary for non-trivial components" (see [5]). In our research on component design, we found many component designers frequently used practices which thwart local certifiability. One in particular is the use of uncontrolled reference semantics in the implementation of a software component.

[6] uses *value semantics* for those data types whose representation is "small" and requires little overhead when moving the actual data around. For example, the data type integer is usually implemented using value semantics. Meyer uses *reference semantics* for data types whose representation can be large. For example, a queue of 100 items. The reference semantic representation uses (at least) one level of indirection. Therefore, if one were to "move" the queue found in the previous example, it would require moving or copying one pointer, not all 100 items.

Uncontrolled reference semantics is when the component's design also allows this implementation detail (the use of reference semantics) to leak through the component's abstraction barrier. An example of this is when a client of the component can unknowingly (or knowingly) create pointer aliases, garbage or unwanted side-effects just by using the component's exported operations. Components designed like this have been published in [1] and [7] to name a few, so this is not an uncommon practice.

Components designed in such a manner thwart local certification of correctness. Basically this means certification of that kind of component must be done every time the component is "reused" from the library in some system. That is, if it is not locally certifiable, then it cannot be certified just once when it is put in the library. On the contrary, it has to be certified each time it is used and the certification almost always has to be done with the code that comprises the entire system. In the example given above using the queue, if the client of the queue component can create aliases (either knowingly or unknowingly), then a change to the queue (e.g., a call to Dequeue) will change the value of the queue for all who have access because of aliasing. Thus what seem to be only changes to a local variable end up having non-local effects. This thwarts local certification of correctness [2]. In our opinion, this is one reason why some people believe that correctness proofs will never be developed for any non-trivial system.

Given the above, our position is: components implemented using uncontrolled reference semantics should not be allowed into the reusable component library.

Local certifiability is just too important to be traded away for components designed using uncontrolled reference semantics. This is especially true when it has been shown that a highly useful library of components (RESOLVE/Ada components) can be built without the use of uncontrolled reference semantics [4].

3 Comparison

As a comparison to other published work in the area of component design guidelines/principles, we provide the following table (reproduced from [4]).

The columns numbered from one to 18 represent the first 18 principles for designing RESOLVE/Ada components. We pick these principles because they deal only with the component's interface, and not with its implementation. These principles when followed go a long way toward controlling "uncontrolled reference semantics." In particular, principles 1, 5, 6, 7, 14 and 16 have direct effect on controlling the use of reference semantics. They do not eliminate its use, they only control it so that it does not thwart local certifiability.

The rows represent five different published component libraries and/or guidelines. The authors are listed below the table. An "x" in a particular column indicates that column's principle is followed, at least most of the the time, by that row's component library and/or guidelines. Notice that no row has an "x" for all of the principles listed above that directly effect the control of reference semantics.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A		x		x	x		x						x					x
B		x	x	x	x								x					
C		x		x									x		x		x	x
D													x					
E	x	x	x		x		x						x					x

A	—	Booch [1]	D	—	St. Dennis [8]
B	—	Hibbard [3]	E	—	Wallis [9]
C	—	Musser [7]			

References

[1] [Booch, G., *Software Components with Ada*, Benjamin/Cummings, Menlo Park, California, 1987.

[2] [Ernst, G.W., Hookway, R.J., Menegay, J.A., and Ogden, W.F., "Modular Verification of Ada Generics," *Comp. Lang.* Vol. 16, No. 3/4 (1991), pp. 259-280.

[3] Hibbard, P., Hisgen, A., Rosenberg, J., Shaw, M., and Sherman, M., *Studies in Ada Style*, Springer-Verlag, New York, New York, 1983.

[4] Hollingsworth, J., *Software Component Design-for-Reuse: A Language Independent Discipline Applied to Ada*, Ph.D. thesis, Dept. of Computer and Information Science, The Ohio State University, Columbus, Ohio, 1992. (A PostScript version of the dissertation is available via anonymous ftp from "archive.cis.ohio-state.edu," in the directory "pub/tech-report/TR1-1993." Source code is also available for a small set of RESOLVE/Ada components in "pub/rsrg/tutorial.")

[5] Knight, J., "WISR '92 Certification Working Group Report," in Larry Latour, Steve Philbrick, Mark Stevens, editors, *Proceedings of the WISR '92 Fifth Annual Workshop on Software Reuse*, October 1992.

[6] Meyer, B., *Object-oriented Software Construction*, Prentice-Hall International, Cambridge, U.K., 1988.

[7] Musser, D.R., Stepanov, A.A., *The Ada Generic Library: Linear List Processing Packages*, Springer-Verlag, New York, New York, 1989.

[8] St. Dennis, R., "A Guidebook for Writing Reusable Source Code in Ada," Technical report, Computer Sciences Center, Honeywell, Golden Valley, Minnesota, 1986.

[9] Wallis, P.J.L., Gautier, R.J., eds., *Software Reuse with Ada*, Peter Peregrinus Ltd., London, United Kingdom, 1990.

[10] Weide, B., Hollingsworth, J., "Scalability of Reuse Technology to Large Systems Requires Local Certifiability," in Larry Latour, Steve Philbrick, Mark Stevens, editors, *Proceedings of the WISR '92 Fifth Annual Workshop on Software Reuse*, October 1992.

4 Biography

Hollingsworth is an assistant professor in computer science at Indiana University Southeast. He has a Ph.D. in software engineering from The Ohio State University (1992). His research focuses various aspects of software engineering and reuse including design-for-reuse, formal methods, testing and education, to name a few. He has authored several technical papers on related topics in software engineering. Prior to receiving his Ph.D., he worked as a software engineer at Texas Instruments (Dallas, Texas) and at Battelle Memorial Institute (Columbus, Ohio).