AVIONICS SOFTWARE REUSABILITY
OBSERVATIONS AND RECOMMENDATIONS

Brian J. Shelburne
Wittenberg University
Department of Mathematics and Computer Science

Marc J. Pitarys
United States Air Force
Wright Laboratory
Avionics Directorate

ABSTRACT

With weapon systems having to perform more functions, the demand for software is enormous. To address the need, software reuse has been proposed as a solution. The Department of Defense (DoD) has conducted and is continuing to perform research in software reuse. One of the results of this research is the Common Ada Missile Packages (CAMP). This is a large collection of generic real-time embedded missile software that allows users to implement many applications such as navigation routines, Kalman Filters, and mathematical operations. In addition to a library of "reusable" software, a Parts Engineering System (PES) was developed under the CAMP program. The PES assists users in constructing application software from components in the CAMP library. During the summer of 1990 the Avionics Laboratory with the assistance of the Air Force Office of Scientific Research (AFOSR) Summer Faculty Research Program investigated the use of CAMP for avionics software applications. The results of the

Research sponsored by AFOSR 1990 Summer Faculty Research Program contract #F49620-88-C-0053 through Universal Energy Systems Inc.

research was that CAMP in its current state was not suitable for avionics applications. The attempts to create avionics software with CAMP led to an abundance of observations concerning the writing and use of reusable software for avionics applications. We will describe the results of our avionics software reuse research. Finally, we will identify some of the errors found in the CAMP software and list recommendations that need to be adopted if wide scale application of reusable software is to be a success.

INTRODUCTION

A concise explanation of CAMP (Common Ada Missile Packages) is found on page 195 of Developing And Using Ada Parts in Real-Time Embedded Applications [2] which states :

"The main goal of the CAMP program has been to establish the feasibility and value of reusable Ada software within mission-critical real-time domains. This has required a careful evaluation of a particular domain, the development of reusable components, the development of automated

support for software reuse in the software development life cycle, and the application of both the reusable components and the automated tools to a realistic application".

There were three phases to the CAMP project. In 1984 a 12-month feasibility study was undertaken to determine if "sufficient commonality existed within the missile operational flight software to warrant the development of reusable software parts" and if so "to determine the aspects of the parts engineering that could be .. automated and to develop the requirements and top-level design for a ... system to support reuse" [2]. Phase 2 begun in 1985 was a 32 month technology demonstration phase whereby 454 software parts were identified and catalogued from 10 different missile systems. These parts were then used to construct the navigation and guidance systems of an 11th missile which was tested in a MIL-STD 1750A hardware-in-the-loop simulation. The last phase was the transfer of this technology. The number of parts was increased to over 500 and the Phase 2 parts engineering system (PES) was re-engineered and rewritten in Ada.

The objects of our research were simple: learn the CAMP Parts Engineering System (PES), use it to generate sample avionics applications, and evaluate its usefulness.

THE PARTS ENGINEERING SYSTEM (PES)

The CAMP package included a Parts Engineering System (PES) used to access the CAMP parts. A CAMP part is an Ada package, subroutine, or task, often generic. The exact definition is unclear but one criterion for a part is that it must be able to stand alone. Parts

can and frequently do "with" other parts.

PES accesses the parts database by letting the user generate a search list using search criteria (e.g. part name, key word etc.) with a limited capability to generate compound boolean search queries. Once a list of parts is generated, the user can examine either the attributes of a part (eg. abstract, key words, sample usage etc.) or the source code for the part (either specification or body). Final selection of a part generates a small text file containing the specification and body file names plus compilation instructions.

The purpose of a parts engineering system is to facilitate access to the parts database. It ought to be designed so that the user can easily sift through a parts collection and quickly locate potentially useful code at which point the user can evaluate the code to see if it can be used. Reuse is only successful if less effort is required to locate a suitable reusable part instead of building it from scratch.

Three features of the PES made it particularly difficult to use. One was the grouping of many parts into the same file. While PES allowed the user to directly access either the specification or the body file containing a part, no information was given as to where the part was in the file. Secondly, many body files used the Ada "is separate" feature to shift the part code into a different file which was not accessible from PES. The user had to exit PES to find the code. Finally, much of the existing internal documentation was not useful or what could have been useful documentation was non-existent. For example the entire revision history of all parts was

faithfully documented while information and references about the algorithms used for parts or pre- and post-conditions for parts was often missing.

The most telling fact about PES was the way that it was circumvented. Hardcopy of the taxonomy of the parts data base used by PES was made which gave a general overview of all the parts. Hardcopy listings were also made of the directories containing the part specification files. The standardized file naming conventions for CAMP parts made this easy to do. Finally, if needed, hardcopy was made of each specification file. Armed with this small set of papers and using an editor, the parts data base was then easy to access.

It seems significant that the PES was eventually by-passed and a simpler way was used to access the parts data base.

BUILDING APPLICATIONS WITH CAMP PARTS

Since the main goal was to study the suitability of CAMP parts for avionic software, three small programs of increasing complexity and a final avionics application were written.

Since CAMP has a set of mathematical routines, the first application was to write a simple square root program. The specification for the CAMP square root part stated that the exception "Negative_Input" would be raised on a negative argument to the part, so an exception handler was written. Unfortunately, on negative input, a "Constraint_Error" exception was raised instead which crashed the program.

This program uncovered the first serious difficulty with CAMP. As mentioned, CAMP parts are, for the most part, generic packages and procedures. Many of them "with" other parts which results in a complex web of interconnections and dependencies. This coupled with a generous use of "is separate" clauses can make back-tracking an error difficult. As it was, tracing down the origin of the square root constraint error required going back through seven files only to come up empty on why the "Negative_Input" exception was not raised.

The problem lay between the second and third files. The second file, called "General_Purpose_Math", contained the line "package body Square_Root is separate" while the third file contained the line "separate General_Purpose_Math" which "linked" these files. Unfortunately, there happened to be an unknown eighth file which also contained the line "separate General_Purpose_Math" which apparently was compiled later and hence was used instead of the third. There was no indication this eighth file existed and it was uncovered only by accident. Interestingly enough, the documentation for the square root part in the eighth file stated 'The exception "Negative_Input" is raised if "Input" is negative' yet the code failed to do so. The documentation also stated that the "body is not a CAMP part but a work-around required by the CAMP Armonics Benchmark Suite". As an overall lesson to reuse software libraries, some mechanism ought to be used to indicate if several versions of the same routine exist.

The second program was a simple velocity computation which used two CAMP generic procedures contained in a single package. Both

procedures required three type and two operator parameters.

In designing parts, CAMP uses an approach it refers to as the "semi-abstract data type" method [2 pp 205-213]. Many CAMP parts are generics that are tailorable to user defined data types. Using a part requires the overhead of providing type and operator parameters in instantiating the generic to insure strong typing. To ease the burden of providing type and operator parameters, CAMP provides a set of default types and operators in a part called Basic_Data_Types. CAMP parts show examples using these default data types in the sample usage documentation.

The velocity application was written using the default types given in Basic_Data_Types following the example guidelines given in the source code for the part. However, the program would not compile due to an "inconsistency detected during overload resolution" error which occurred while trying to find an actual parameter corresponding to one of the generic formal procedure parameters.

The problem has to do with the instantiation of generics. According to the Ada Language Reference Manual, if a generic package is instantiated and a new type is derived from the instantiation, then subroutines from the instantiation ought to be visible with parameters of the derived type. This does not seem to be the case. The problem was easily circumvented but either the CAMP documented sample usage is misleading, wrong, or there is a Ada compiler problem. (The example was run on two different compilers and the same error occurred.) The overall lesson was that heavy use

of generics by CAMP requires a significant level of expertise in the Ada language.

[illegible]

The final avionics program was a waypoint navigation application which was chosen because CAMP had the parts that would make implementation easy. The only drawback was that the mathematics behind the particular method CAMP used to perform its computations was not clear and had to be extracted from the code. (A reference to an outside source explaining the mathematics would have been very helpful here.) As it turned out, the CAMP part that calculated turn angles contained a fundamental mathematical error. The application did not work properly.

Of the four programs written using CAMP parts, all uncovered problems in the CAMP software.

EVALUATION

CAMP is complex, not well documented, and contains errors.

Complexity is derived from widespread use of "withing" which

generates a complicated web of part dependencies. The heavy use of Ada's "is separate" clause which scatters parts over three or more files makes backtracking difficult and does not help the user to understanding the code. Combining many parts into one file makes locating individual parts difficult.

In terms of documentation, the user is often presented with a mass of comments which obscures important documentation. Other documentation that a user might find helpful is missing. For example, pre and post-conditions for procedures are lacking.

Finally, each of the three programs and the final application uncovered errors ranging from problems in documentation to design flaws.

RECOMMENDATIONS

1. The Parts Engineering System should be re-designed with two goals in mind. First, the selection criteria should allow the user to quickly narrow down the mass of available parts to a few prime candidates that he or she needs. Secondly, the PES should display the salient information about a candidate part quickly so the user can decide whether to use it or not. This includes displaying the code without the user having to search a file for it. A windows-like environment should be considered.

2. Documentation for a software part is crucial. The documentation should state explicitly the meaning of all constraints on input and output for a part. The algorithm or implementation used by the part should be explicitly stated or a reference given for it. Sample usage that has been tested and is known to work should be

included. Dependencies on other parts should be clearly stated. Useless documentation should be avoided. Quality documentation is a necessary precondition for any effective software reuse engineering system.

3. The quality of software is crucial for reuse. It is safe to say that the user and not the author of a software part is responsible for its correct usage. Therefore reusable software parts must be totally visible so that the user may understand a part's function, proper use, and correctness. Reusable software parts must be white boxes, not black boxes. To this end, reusable software parts should not be overly dependent on other parts since this complicates understanding a part's functionality. Dependencies on other parts should not go back more than one or two levels. Every effort must be made to assure the user of the quality of the software part.

4. The use of Ada's "is separate" feature should be avoided since it hides code. The grouping of many software parts into a single file should be avoided to eliminate the need to scan files for code. Nothing should be done that detracts the user from a complete understanding of a part.

5. Reusable software is supported by Ada generics but inhibited by Ada's strong typing. Most CAMP parts are coded as generics but to use a part, a number of generic parameters must be supplied to satisfy the strong typing. CAMP partially solves this problem by providing a large number of defaults for types and operators which the user may or may not find useful. The method seems a reasonable compromise provided part dependencies are minimized and

well documented.

REUSABLE SOFTWARE AND DESIGN DECISIONS

When a software engineering project is undertaken, design choices are made from the highest level of abstraction to the lowest level of implementation. Two designers given the same specification will come up with equally good designs yet at some point their designs will diverge. This was observed when using CAMP parts to write simple avionics applications. CAMP software embodies a set of design decisions and a user of CAMP software must subordinate his or her own design tendencies to those of CAMP's. For example, existing avionics documentation examined for possible use for avionics applications used a "North, East, Down" inertial frame of reference while CAMP used an "East, North, Up" inertial frame. Should the user abandon his or her better understood design tendencies for - CAMP's? Is CAMP's software quicker to implement and more certain of working correctly or will the user's own design be better?

The issue of design divergence has wider implications for reusable software since reusable software parts come at the end of the design process and thus exhibit the greatest design divergence.

CONCLUSION

CAMP in its current state is complex and not well documented. It is difficult to locate and verify the suitability of a software part for a given application. CAMP software contains errors which are difficult to track down because of CAMP's complexity. CAMP was an excellent research effort in large scale software reuse but it needs considerable improvement and/or

development if it is to be used for operational avionics applications.

REFERENCES

- [1] Biggerstaff, T.J. and Perlis, A.J. ed, Software Reusability - Vol I Concepts and Models; ACM Press, New York, 1989.
- [2] Developing and Using Ada Parts in Real-Time Embedded Applications (CAMP-3); McDonnell Douglas Missile System Company, 1990.
- [3] Parts Engineering System Catalog User's Guide Version 1.1; McDonnell Douglas Missile System Company, 1990

Postscript: After this paper was completed the authors met with the CAMP program managers and developers. The following information emerged.

The square root routine that caused problems in the first and third applications was a part delivered with the CAMP benchmarks. This part had the same name as the CAMP square root part and resided in the same directory, however it was less precise, which resulted in the errors.

The difficulty with the generic instantiation "inconsistency" in our second application was caused by the fact that derived subprograms do not work with certain combinations of derived types (Ada Language Reference Manual, Section 3.4 Paragraph 11). This was not a CAMP related error.

The mathematical error in calculating turn angles in the fourth application was due to an undocumented "flat-earth" assumption. A flat-earth model is adequate for missiles but not for avionics systems.