Identification and Tailoring of Reusable Software Components

Constance Palmer
McDonnell Douglas Missile Systems Company
Dept. EBE2, MS 3064025
P.O. Box 516
St. Louis, Missouri 63166-0516
(314) 232-0278

palmer%cstc.decnet@mdcgwy.mdc.com

Approved for Public Release
Export Authority: 22CFR 125.4(b)(13)

Abstract

Two significant technical challenges facing application developers are early identification of potentially applicable software components, and adaptation and incorporation of reusable software components in new applications. Software reuse can be facilitated by tools that support these reuse-specific tasks and that are integrated with the user's other development tools.

The CAMP program developed prototype tools to support both early component identification and tailoring of components. The early identification is based on domain- or application-specific search, i.e., the user's view of the search is in terms of the domain or application area rather than in terms of keywords or other component attributes. A domain analysis generally forms the basis for providing this type of search mechanism.

Tailoring support is provided by "constructors". These constructors automate much of the tailoring process, making reuse easier. They can lower the cost of reuse in terms of both time and resources. The CAMP project investigated ways of producing cost-effective, flexible constructors.

Keywords: Software reuse, software tailoring, component identification

1 Background

The Common Ada Missile Packages (CAMP) program is a U.S. Air Force sponsored contracted research and development effort. The program began in 1984, with the initial goal of determining the feasibility of software reuse in a highly constrained real-time embedded domain, specifically, the missile operational flight software domain. Once commonality within this domain was established, attention was turned to investigating optimal methods for component design and incorporation of software reuse into the software development lifecycle. There were three main thrusts: the development of reusable resources; the development of a reuse support environment; and technology transfer. The research efforts associated with commonality identification and component design and development have been well-documented in [McNi 86, McNi 88, Palm 90]. The initial tool investigation and development efforts have also been documented in these same sources.

Recognizing that most software developers would not adopt software reuse practices based on studies alone, the CAMP program combined software reuse research and application. For example, after the initial set of 450 reusable Ada components and the prototype reuse support tools were developed, a testbed program was put in place to demonstrate the feasibility of incorporating software reuse into the development process for real-time embedded applications. Specifically, the goal of the testbed effort was to show that missile operational flight software applications incorporating significant levels of reuse could be effectively developed.

2 Problem Statement

It seems to be a commonly held view today that the real barriers to software reuse are not technical in nature, but rather are managerial, legal, social, and psychological. While there are barriers in these areas, there are still significant technical challenges associated with software reuse.

Two problems that have persisted in the reuse arena are those of easily identifying and incorporating reusable components into new applications. Significant resources have been expended in developing cataloging and library schemes, but effective software reuse requires more than a library of available components. One problem with the traditional approach to software reuse catalogs and libraries is that the user needs to have a fairly good idea of the types of components that he is looking for in order to construct appropriate queries. This poses a barrier to reuse because early in the lifecycle, when software reuse must first be considered (e.g., during system concept development and software requirements analysis), the developer may not have sufficient information about the application to get meaningful information out of a catalog or library.

The application developer faces another barrier to reuse when, after identifying candidate reusable components, he tries to adapt or tailor them to fit his requirements. Obviously, this is not a problem with low-level, black box components, but can become apparent when the application developer is working with higher level families of components. These higher level components may be at the subsystem or system level, and thus, have the potential for significantly higher payback than low-level, black box components. The developer faces a further challenge when trying to compose lower level components into higher level entities. Despite high quality documentation, it may still be difficult for the application developer to easily and correctly tailor and compose existing components into new applications.

3 Current CAMP Research Inititatives

Two of the most recent initiatives on the CAMP program were concerned with early identification of reusable software components and with tailoring and composition of reusable components for new applications. The first initiative was embodied in the development of a reuse support tool called "parts exploration", and the latter initiative was embodied in the development of a set of facilities and procedures referred to as a "meta-constructor".

3.1 Parts Exploration

The parts exploration (PE) facility allows the user to couch his queries about available software components in terms of his application or domain rather than in terms of specific catalog attributes. This permits more meaningful interaction between the application developer and the component library early in the application development lifecycle, thus enhancing the prospect that significant levels of reuse will be achieved.

A prototype parts exploration system that can accommodate multiple domains simultaneously was developed. This system was used for the CAMP missile operational flight software domain, as well as for a smaller, test domain.

The PE system prompts the user for information about his domain or application, and based on the information obtained, PE will identify potentially applicable reusable components. This information can be presented to the user in either list form or in the form of a system hierarchy tree that the user can traverse until he gets to the lowest level reusable components that comprise the leaf nodes in the tree. Once the user has this list of components, he can obtain detailed information about them from the catalog.

The parts exploration system must be initialized for a given domain before that domain can be supported by the tool. Initialization consists of entering domain-specific information that would generally be obtained during a domain analysis.

Several issues were not addressed in our development of the parts exploration system. For example, the definition of domains, and the treatment of overlapping domains and related domains were not addressed. Although the definition of domains impacts the treatment of overlapping and related domains, this definition was left to the user. In the case of overlapping domains, components may be developed independently for these overlapping domains, and thus the components in the domain intersection are not really in both domains; they are only in the domain for which they were developed. This results in the user not being informed of potentially relevant components that were developed for other domains. A similar situation arises with related domains. For example, missile and aircraft autopilot applications are, at some level, similar. If the user were in the missile operational flight software domain and the system identified the need for an autopilot in his application, he would only be informed of the availability of reusable components for the development of missile autopilots, despite the fact that he might be able to tailor an aircraft autopilot for his application. Another issue that bears further exploration is that of the granularity of the domain information that is needed in order for the PE system to effectively identify relevant components for the user.

Earlier in the CAMP program, a version of PE was prototyped for the missile operational flight software domain using an expert system shell. The current system was developed in Ada running

under VAX VMS. The interface is very simple and can accommodate even a VT100-type of terminal. The system itself makes use of reusable components.

3.2 Meta-Constructor

Early in the investigation of the feasibility of software reuse, we recognized the importance of providing support for component tailoring and composition, as well as providing reusable components. Thus, the concept of component constructors was developed. A component constructor assists the user in tailoring and composing reusable software components for new applications. A constructor may perform limited code generation in order to meet these requirements, but this is code generation in a very narrow sense as opposed to generalized code generation.

A number of prototype constructors were developed to support adaptation of different types of CAMP components, e.g., there were autopilot constructors for producing application-specific lateral directional and pitch autopilots. Other constructors were developed for Kalman filters, navigation subsystems, finite state machines, etc. These constructors demonstrated the feasibility and usefulness of the concept. We had domain experts use the Kalman filter constructor. They were able to generate 18 and 20 state filters in approximately an hour. The constructor provided the required data types, operators, and tailored components based on the user's specifications. If this development were done from scratch, it could easily have taken several weeks to produce and debug the code.

Constructors provide the ability to easily perform "what ifs" by providing the capability to modify the specifications and regenerate code. This is particularly important in Kalman filter development where implementors frequently change the number of states in the filter in an attempt to obtain optimal performance. These changes often go on fairly late into the development cycle, thus increasing the value of constructor-type support.

The most significant limitations of the early CAMP prototype constructors were (1) the linkage between the constructors and the underlying components that they were designed to tailor, and (2) the fact that, although there was a common paradigm for the constructors, each was custom built at a non-trivial cost with respect to effort. The linkage between the constructors and the components resulted in significant changes to the constructor if the underlying components changed, increasing the cost of providing constructors. The "custom built" aspect added to their cost. Thus, the value of constructors as productivity enhancers was demonstrated, but their viability as a long term solution was questionable. This conclusion led us to further research in this area during later phases of the CAMP program. We wanted to investigate more efficient and effective ways of producing constructors.

A number of alternatives were possible. We considered embedding directives in the reusable components and using a preprocessor to generate the appropriate user queries for tailoring information. Although this idea had merit, we ultimately decided that alteration of the reusable component itself by embedding these directives was not desirable. The approach that we prototyped was that of a "meta-constructor". The meta-constructor consists of a set of facilities and procedures for constructing constructors. The final constructor that is produced is implemented in Ada, and thus, is able to take advantage of the Ada compiler for much of the error checking.

The meta-constructor facilities consist of utilities of different types. For example, there are user interface utilities that handle forms and menus, as well as utilities for querying the user about specific

types of information, for error checking, and for code generation. Routines have been developed for obtaining different types of Ada data types, for obtaining information about procedures and functions, and for instantiating Ada generic units.

The meta-constructor procedures guide the constructor developer in the development of new constructors. Much of the implementation of a constructor consists of calls to existing meta-constructor utilities and to utilities that are either specific to that particular constructor or that are general, but that were not previously needed for existing constructors.

The meta-constructor is designed for extensibility. That is, it consists of a basic set of facilities for implementing constructors together with a paradigm for their construction. The initial set of facilities was identified by performing an informal domain analysis on the 12 previously developed CAMP component constructors. It is anticipated that, as constructors are implemented using the meta-constructor facilities, additional utilities will be added, thus facilitating implementation of future constructors.

In order to demonstrate the capabilities of the meta-constructor, it was used to develop 2 constructors that are representative of the types of constructors that can be developed. These two constructors are the Kalman filter (KF) constructor and the finite state machine (FSM) constructor. The FSM constructor supports tailoring of a schematic part, i.e., a design component rather than a code component. The KF constructor combines aspects of both generic constructors and schematic constructors.

These types of constructors are based on the types of components that were identified during the initial CAMP domain analysis. At that time, 2 different types of components were identified: simple components and meta-parts. Simple components are components that can be used "as is" or, in the case of Ada, they can also be generic units that have little or no interaction with other units and can be instantiated with the provision of a relatively small number of types and operators. Meta-parts are either complex Ada generics or schematic parts. Complex Ada generic units may require a fairly large number of types and operators for instantiation, and may have fairly significant interactions with other generic units, e.g., there may be embedded instantiations or a number of dependencies between the units. Schematic components are, in essence, design components rather than code components. The design may be relatively straightforward and the code production may be algorithmic, but the variability that is possible in the family of implementations embodied by this component is not easily or possibly captured in the implementation language.

The two representative constructors have been implemented. Although the meta-constructor does not negate the need for the constructor developer to have a thorough understanding of the component family he is going to provide, it does facilitate the development process. A comparison of the previous, custom-crafting approach with the new meta-constructor approach will be conducted in order to assess the gain.

Although all of the constructor work performed under the CAMP program was directed at the production of tailored, executable code, this approach could also be applied to the production of documents and test procedures and code. Further work is also needed to determine the flexibility and scalability of our approach. For example, we applied the meta-constructor approach to the tailoring of components on the scale of Kalman filters and autopilots, but we have not attempted its use on much larger scale entities (e.g., command and control center applications).

References

[McNi 86] McNicholl, D.G., Palmer, C., Cohen, S.G., et al, "Common Ada Missile Packages (CAMP)", Tech. Report AFATL-TR-85-93, Volumes 1-3, U.S. Air Force, Wright Laboratory, Armament Directorate, Eglin Air Force Base, Florida, 32542, May 1986, Distribution limited to DoD and DoD contractors only.

[McNi 88] McNicholl, D.G., Cohen, S.G., Palmer, C., et al, Common Ada Missile Packages - Phase 2 (CAMP-2), Tech. Report AFATL-TR-88-62, Volumes 1-3, U.S. Air Force, Wright Laboratory, Armament Directorate, Eglin Air Force Base, Florida, 32542, November 1988, Distribution limited to DoD and DoD contractors only.

[Palm 90] Palmer, Constance, "Developing and Using Ada Parts in Real-Time Embedded Applications", Tech. Report AFATL-TR-90-67, U.S. Air Force, Wright Laboratory, Armament Directorate, Eglin Air Force Base, Florida, 32542, April 1990.

4 About the Author

Connie Palmer is the CAMP program manager at McDonnell Douglas Missile Systems Company (MDMSC). The Common Ada Missile Packages (CAMP) program is a USAF contracted research and development effort that has been investigating the potential of software reuse in mission-critical real-time embedded DOD domains; the specific domain addressed is missile operational flight software. The CAMP program has developed over 500 reusable Ada software parts, as well as explored issues and developed tools to support reuse efforts.

Ms. Palmer has been at McDonnell Douglas for almost 8 years, and has been involved in the CAMP program since its inception in 1984. She also leads a group that is working to improve the software engineering practices and processes at MDMSC.

Ms. Palmer has an M.S. in Computer Science from Washington University in St. Louis, and a B.A. in Mathematics from George Washington University in Washington, D.C..

She is a member of IEEE, ACM, AAAI, and AIAA.