Some Considerations for Reengineering Navy Legacy Systems

Mark L. Wilson
Naval Surface Warfare Center DD/WO
Silver Spring, Maryland
USA

Abstract
Navy shipboard and aircraft systems and the super systems in which they interoperate are some of the most complex computer based systems in the world. This together with the shear number of platforms, their long service lives, and rolling upgrades schedules make reengineering them a formidable task. In this paper some of the considerations involved are laid out, some of the possible approaches are shown to be undesirable, and current efforts and needed work are discussed.

Overview
Legacy systems abound in the Navy. In many cases it is infeasible or ill advised to develop new systems from scratch. Yet due to new system requirements, increasing maintenance costs, or other reasons it is desirable to somehow replace or upgrade these systems. There is also a desire to capture some of the capabilities inherent in these systems and somehow reuse them. This paper looks at some of the issues involved in reengineering, and in particular those that tend to be unique to or more pronounced in Navy systems. The view here is to tactical systems; those deployed on ships and aircraft which have real-time requirements, and tend to be safety critical.

Introduction and Background
As stated, this paper is concerned primarily with real time, safety critical, mission critical, complex systems. Most of these types of systems involve intensive use of computers. Traditionally these programs run on Navy military standard computers and use Navy military standard displays and peripherals.

Many of these systems were designed with memory or other architectural constraints which no longer apply. Thus thorough reengineering of them requires consideration of the original design rationale which may not be explicit in the existing documentation. Moreover, there may be timing or other relationships which only become apparent during the most thorough system test.

The complexity of these systems means that optimal designs are not feasible and that limits must be set on the design process [3]. Continuous improvement through redesign or reengineering is viewed as a way of limiting future design effort and leveraging off of past work.

The close relationship between the Navy standard computer programming language, known as CMS-2, and the Navy standard hardware has gone from a blessing to a curse. The processors and instruction set architectures (ISA) are specific to these machines. Moreover, with the exception of the 68xxx family, the CMS-2 compilers have never been capable of generating code for commercially available processors.

This leads to a desire to change both the hardware and software within the context of a single reengineering effort. At present the language of choice would appear to be Ada, but almost all of the following discussions apply equally well to C or any other equivalent language.

Motivation
Driving factors for reengineering may include a desire to: avoid parts obsolescence, provide increased computational performance to enable implementation of increased requirements, reduce the cost of enhancements, speed the implementation of new features, reduce software maintenance costs, reduce hardware procurement costs, and/or reduce hardware maintenance costs. From this it is obvious that an econometrics model would be helpful to guide the decisions of
when and how to reengineer. Some work has been done on the software front [2] but much remains to be done from a systems engineering level.

Objectives

One of the short term goals should be the ability to efficiently transform highly hardware dependent legacy systems into moderately, if not fully, hardware independent open systems.

A longer term goal should be the ability to effectively transform complex legacy systems into systems that can evolve gracefully over time. Evolution will typically involve increased requirements, new processor hardware, new display technologies, and integration with additional systems. It may also involve partial reuse within other systems, totally new requirements unanticipated during design, partition into parallel or distributed architectures, combination into fewer more powerful processors, new human computer interfaces, networks to replace point-to-point communication, or translation into new languages.

Some of the techniques which may be useful include: a layered approach to system and software design, automatic or semiautomatic language translation, and graphical aids to software and system understanding.

Options for Reengineering Navy Tactical Systems

Most Navy tactical systems run on Navy standard computers such as the UYK-43, UYK-44, AYK-14, or even older systems using code written in the Navy standard CMS-2 programming language and/or assembly language. In addition to significantly lagging the current state of the art and the state of the practice, this hardware is extremely expensive. It's high cost is largely a function of its low production volume when compared to commercial off the shelf computers. The hardware is also expensive to maintain due to the relatively higher failure rates inherent in the older technologies involved and the high cost of replacement parts, some of which are essentially obsolete.

Similar situations exist with respect to related equipment such as tape drives, mass storage devices, printers, and displays. Newer display technologies may also enable a reduction in crew, which has a significant impact on cost reduction. Thus it is desirable both from a technological and an economic viewpoint to move from these Navy standard processors and peripherals to commercially available equipment. There will of course be requirements to package or isolate this equipment to withstand the rigorous military environment in terms of shock, vibration, etc. but there are reasonable engineering solutions available for that.

Moving to new processors of course implies that all the hardware specific code in these systems will have to be replaced. This includes device drivers, input/output related code, most display related code, and of course the assembler code that is not included in any of the above. Since it is also likely that the present point to point connections between computers will likely be replaced by some sort of local area network (LAN) additional portions of the code may change as well. There are a number of issues involved here but this presentation will not address the implications of changes in topology.

This leaves the Navy with millions of lines of CMS-2 that need to somehow be ported to new instruction set architectures (ISAs). While it is true that MTASS, the CMS-2 development environment can generate code for some of the Motorola 68xxx family of processors, few if any CMS-2 compilers exist to generate code for COTS processors. Two obvious ways exist to solve this shortcoming. Add CMS-2 front ends to existing or new compilers which generate code for new processors or add back end code generation capabilities to the existing CMS-2 compilers for each new COTS processor the Navy wishes to use. Both of these would be expensive.

Adding new front ends to existing compilers is complicated by the fact that there are several versions of CMS-2 with minor variations. Worse still, these new compilers would need to be thoroughly tested since the resulting code is part of real time weapons systems where errors can be catastrophic.

Adding back end code generators to existing CMS-2 compilers would mean restricting the Navy to a very limited range of processors. Moreover, it would probably result in compilers that generated less than optimal machine code, since the expertise inherent in the processor specific compilers would likely not be fully exploited.

There is another disadvantage to propagating CMS-2 onto new processors. Most of the new design, development, analysis, and maintenance tools for software are available initially or exclusively to support languages with large user bases. Few of them are available with support for CMS-2 and many of those that do support CMS-2 only do so because funding was supplied by DoD. This trend will continue and likely will become more pronounced.

It also does not address the substantial portion of direct or assembly code that is embedded in many CMS-2 programs. Much of this exists because the Navy standard computers were not fast enough and/or did not have sufficient memory when the code was written in CMS-2 and compiled. For all intents and purposes memory is no longer a constraint that needs to be addressed by hand written assembler, nor for that matter is processor speed.

Having the ability to generate object code for other ISAs from CMS-2 code still requires that the assembler be rewritten or translated to CMS-2, or converted to a new assembly language, or converted to another high level language.

A further argument can be made against propagating CMS-2 to new processors. That is that new graduates will need to be trained in CMS-2 to replace programmers that leave the workforce. This training expense could be saved if the CMS-2 were converted to a more widely accepted language with which new graduates were already familiar.

Due to the above and in order to make use of current and future technology in hardware and software it is desirable to move off of these platforms to commercial processor families and from CMS-2 to a more widely supported, contemporary
language. At present the language of choice would appear to be Ada. Should the goal change from Ada to C or to another language much of the technology inherent in converting CMS-2 to Ada could be applied to converting CMS-2 to C. In many cases it could also be applied to converting Ada to C, or to some other language.

**Future and Related Work**

Three tools for automating the translation of code written in the CMS-2 programming language to the Ada language were tried on a significant portion of code from the Navy's Aegis Weapons System. At this point, it can be said that each of these tools work and automate a significant part of the translation process. There is still a significant amount of work that needs to be done after translation to make the code executable, but it is felt that automating part of the process is a very useful step [4]. Moreover, it may help expedite what has been described as reusable component recovery [5]. It may also be possible to combine automated translation with tools which help to extract architectural features from code[6], thus giving some insight into design rationale.

Future papers will discuss in more detail what is required to make the translated code executable but the crux of the matter is that CMS-2 programs generally rely on an executive tailored to a specific application. This functionality needs to be mapped to a Ada run time environment and/or operating system.

Further work is in progress to more extensively evaluate these tools and try to quantify the results to make the information more usable.

Other work is in progress to integrate translation oriented reengineering tools with other tool environments [7] to try to help integrate them into an overall process.

Should the goal change from Ada to C or another language much of the technology inherent in converting CMS-2 to Ada could be applied to converting CMS-2 to C or Ada to C, or between some other language combination. In fact a variant of one of these systems has already been built to handle Fortran to Ada.

**References**


