# Software Reuse Between the RCAS and SBIS Programs

Jeffrey S. Poulin

16 February 1995

## 1.0 Introduction

This paper will present lessons learned from a coordinated reuse effort between the U.S. Army Reserve Component Automation System (RCAS) program [1] and the U.S. Army Sustaining Base Information Services (SBIS) program [6]. These two major Department of Defense (DoD) Management Information Systems (MIS) contracts provided the opportunity for a significant multi-million dollar Reuse Cost Avoidance. This paper illustrates the difficulties and benefits of program-to-program reuse as well as changes in current acquisition practices required to institutionalize this approach. Finally, it discusses the successful transfer of some reuse assets (such as lessons learned and software architecture) across RCAS and SBIS.

**Keywords:** Software Reuse, Software Architectures, RCAS, SBIS

## 2.0 Background

In 1991 the Army Reserves awarded the Reserve Component Automation System (RCAS) contract to Boeing Computer Systems. RCAS spans 10 years with a potential value of $1.8 billion. RCAS will provide an automated information system that will support both the United States Army Reserve and the Army National Guard. RCAS will supply these organizations with office automation tools such as word processing, electronic mail and spreadsheets as well mission support tasks; i.e., Personnel Administration, Force Authorization, and Mobilization. The RCAS program has entered its third year and has developed approximately 1.2 million source lines of Ada code.

In June 1993 the US Army awarded Loral Federal Systems in Owego, New York (then IBM Federal Systems Company) a contract potentially worth $474 million over 10 years to improve and standardize the active Army's day-to-day business functions. This Sustaining Base Information Services (SBIS) contract will overhaul the installation management functions carried out at over 100 Army installations, including areas such as logistics, finance, personnel, training, and office automation (electronic mail, word processing, and spreadsheets). The contract will help the Army move from proprietary operating environments to open systems standards-based configurations. SBIS will soon complete the implementation of its initial suite of applications with delivery scheduled for the spring of 1995.

This paper describes the cooperation between the SBIS team and members of the RCAS team who had primary responsibility for development of the RCAS system software, application software, and databases.

### 2.1 System architectures

Although this paper primarily deals with software reuse, it first helps to introduce the system environments in

which each program operates.[1]

The RCAS system architecture hinges on independent site/installation local area networks (LANs) using the TCP/IP protocol; National Guard and Reserve sites connect to a wide area network (WAN) asynchronously via modems. Individual users locally process office automation, applications, and database software running on one or more computers, depending on the size of the site. This processing will primarily take place on DEC 5000 series servers running MLS+, an operating system based on the DEC version of the UNIX, Ultrix 4.2. Users obtain access to RCAS via X Terminals from the HDS Corporation or via personal computers, such as the Zenith 486. RCAS currently operates multi-level mode security at the "confidential" level with the requirement to operate at the "secret" level upon delivery of the first application code.

The SBIS system architecture hinges on an installation LAN using the TCP/IP protocol; installation LANS connect to an Army WAN. Individual users locally process SBIS functions on two or more computers, depending on the size of the site. This processing will primarily take place on NCR 3450/3550 servers running the UNIX System V.4 operating system, with users obtaining access via X Terminals from the HDS Corporation. Users may also access SBIS via personal computers, such as the AST Premium 486.

The programs met their contractual requirements for standards-based solutions with diverse suites of commercial products. Both programs must comply with Open Systems standards and must deliver applications with a MOTIF/X Window interface. RCAS and SBIS will develop their applications in the Ada programming language, for which both chose the Verdix Ada compiler. For requirements and high level design both use the Cadre Teamwork environment. For database support, both use Federal Information Processing Standard (FIPS) 127 Structured Query Language (SQL) compliant databases, with SBIS choosing Oracle and RCAS choosing Informix. The two programs also use many similar or identical products, tools, supporting application packages, and software development environments.

## 2.2 Software architectures

Although components may have certain attributes which make them easier to reuse, external factors, such as *context*, also determine reuse potential. In other words, the ability to reuse a given component depends on how well the target software environment matches the environment which produced the component. Although RCAS and SBIS have certain mission requirements in common and run on open system architectures, their software architectures also influence the opportunities for large scale reuse between the programs. The following sections give a quick overview of the RCAS and SBIS software architectures as a prelude to identifying reuse opportunities.

### 2.2.1 RCAS

As part of its pre-award demonstration the RCAS program initially developed prototypes of their applications which ran as stand-alone Ada programs. However, they quickly found that the size of the applications and the network load required to support the transfer of these applications seriously effected the cost and performance of their systems. These experiences benefited the production system by leading to several key design decisions. Recognizing that most of the applications consisted of common Graphical User Interface (GUI) and data management software that remained virtually the same independent of application, they separated these functions into executable processes that each of a user's applications can share. This led to the RCAS application architecture shown in Figure 1 [2].

[1] The configurations and data presented in this paper reflect the state of the RCAS and SBIS programs circa June 1994. Both programs have since evolved and altered these configurations.

In the RCAS architecture, the *Window Object Manager (WOM)*, manages all GUI functions and user interaction with RCAS applications. Every application that a user runs shares one copy of the WOM, thereby relieving application developers of having to design, develop, or maintain that code. The WOM validates user input by field (e.g., allows only valid dates in a date field), checks every input the user attempts (e.g., preventing numeric entries in a alpha field), and enforces a limited semantic verification on forms (e.g., ensures "start" dates precede "end" dates).

Likewise, a shared *Data Object Manager* (DOM) manages all accesses that a user's applications make to the RCAS database. This shields applications from the effect of any changes to data and relieves the applications from each having to write SQL. Furthermore, this WOM/application/DOM design allowed RCAS to assign their best Xwindow/MOTIF developers to the WOM, their best database/SQL developers to the DOM, and their best Ada application developers to the applications. This architecture greatly reduced application size, network traffic, and improved development of RCAS applications.

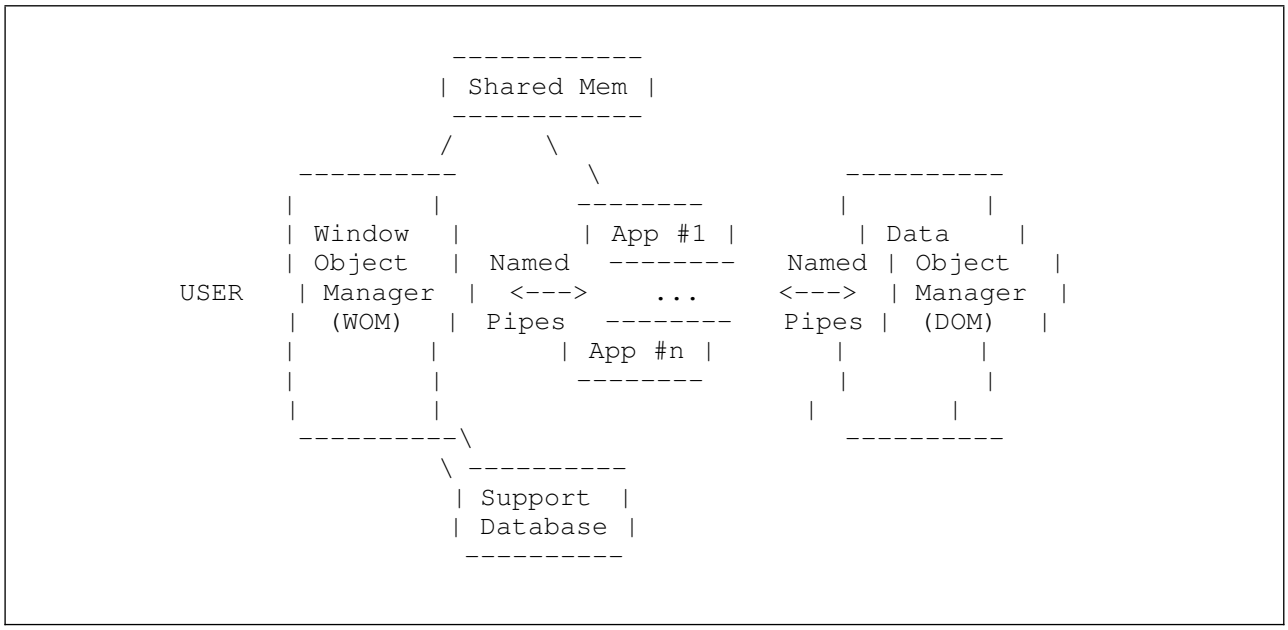


Figure 1. The RCAS Application Architecture

The WOM, DOM, and applications communicate via named pipes for routine messages and via shared memory for other messages. The use of shared memory obviates the need to transmit large messages; rather, the WOM can simply transmit a message to the application saying that the WOM stored the required information at memory location *M*. The WOM makes use of a support database which defines the rules and details of the behavior it must take based on message types it receives from the applications.

Note how this architecture takes an object-oriented approach at the system level since each object in the system runs as a separate process and communicates via messages. Also note how the well-defined structure provides a common model for understanding the roles of each component in a standard RCAS application. Finally, RCAS receives large amounts of reuse from the WOM and DOM which provide the domain-independent GUI and data operations for all RCAS applications.

### 2.2.2 SBIS

The SBIS contract specifies delivery of approximately 80 MIS applications. Within those applications, SBIS has identified numerous functions and processes that many SBIS applications will need. The SBIS software architecture calls for the SBIS team to develop these common functions, or *Service Objects*, one time and make them available to all the application developers. Table 1 contains a partial listing of the SBIS Service Objects.

Table 1. Partial listing of the SBIS Service Objects

| | | | |
|---|---|---|---|
| Scheduling | Help | Data Validation | Access Control |
| Adhoc Query | Mail | Correspondence/Forms | Error Handler |
| Audit Trail | Print Manager | Print Queue Manager | Data Abstraction |
| Batch Processor | File Transfer | Directory Services | External Devices |
| Initialize/Terminate | Report Generator | Common Screens | Business Graphics |
| Data Conversion | Communications | Forms Generator | External Interface Mgr. |

The next step in the SBIS software architecture involves the observation that the SBIS can view the 80 applications as coming from eight primary domains:

1. Resource Management (10 applications)
2. Logistics (13 applications)
3. Operations and Plans (11 applications)
4. Personnel (18 applications)
5. Information Management (16 applications)
6. Installation Management (2 applications)
7. Engineering (4 applications)
8. Legal (10 applications)

Since applications will share functions within a domain, SBIS will provide a set of common domain-specific objects, which SBIS calls *Business Objects.* Application developers will reuse these Business Objects in all or most of the applications within a SBIS domain but will not necessarily find use for them outside the domain. Once SBIS completes development of all Service and Business Objects, a developer only needs to write *Application Objects* to provide the remaining software needed to implement the application.

Figure 2 shows how the objects work together in this architecture. The typical SBIS application consists of a set of separately running processes which collaborate to deliver the function required; the figure depicts two applications running on a processor. The first application, *AIMS-R,* belongs to the *Operations and Plans* domain because it consists of training management software. The second application, *Safety,* belongs to the *Information Management* domain. Both applications make use of several existing SBIS Service Objects, such as the Printer and Data Management Service Objects. Notice that many other Service Objects may run concurrently on the processor whether or not explicitly used by our two example applications.

The figure also shows that the AIMS-R application uses three existing Business Objects from the domain of *Operations and Plans* and that the Safety application uses two existing Business Objects from the domain of *Information Management.* In this example, the developer must only write the remaining AIMS-R and Safety Application-Specific Objects to implement the respective applications.

Figure 2. Two Applications Running in the SBIS Architecture

Like RCAS, SBIS also took a system-level object-based approach in the design of its application architecture. Inter-object communication in SBIS takes place in one of two ways. The first method uses Remote Procedure Calls (RPCs) and has the advantage of conforming to the Open System standard for distributed process management.

Most objects in the diagram communicate via RPCs. The second method involves physical linking of objects and works best when performance requirements dictate real-time response. In the diagram, the *Audit Trail* Service Object gets linked into the two Business Objects shown because the audit functions need to take place quickly and linking will have the least performance impact on the normal operation of the application.

## 3.0 Opportunities for Reuse

In some cases RCAS and SBIS delivered common architectural or functional components to their respective customers. Because of this, there existed a tremendous opportunity to reuse significant amounts of software and thereby save the government a correspondingly large amount of money. Because RCAS has a two year lead over SBIS and because of a Congressional mandate for exclusively new development on RCAS, more of the opportunities for reuse occur from RCAS to SBIS. Therefore, SBIS could mostly benefit through reduced costs and chances to improve its product delivery schedule. However, both programs could benefit from the prestige of sharing an architecture and working together in a manner unprecedented in the DoD.

Numerous opportunities for software reuse of low-level functions could take place over time given a cooperative effort. Supporting software developed by RCAS could serve in the Service Object and Business Object layers of the SBIS model. Application code for RCAS could transfer directly to the Application Objects of SBIS. Several of the RCAS architectural components mirrored those identified for development by SBIS.

After several technical exchanges between RCAS and SBIS, the SBIS software architects became convinced that SBIS could benefit from the WOM, DOM, and some of the other designs and software developed by RCAS. In May 1994 SBIS requested the following from the RCAS Program Office:

1. Window Object Manager (WOM)
2. Data Object Manager (DOM)
3. External Interface Manager (EIM)
4. Data Synchronization Manager (DSM)
5. Print Management System (PMS)
6. Navigation Application Broker (NAVAB)
7. Electronic Signature Function
8. Continued technical liaisons on topics such as legacy system interfaces.

Note that the value of this effort extends far beyond code. Other benefits include simply exchanging experiences. For example, RCAS solved numerous problems or limitations of the Ada compiler, database, and other development tools that SBIS would also have to solve. RCAS established interfaces to the many Army legacy systems that SBIS would have to establish, and RCAS created high-performance images of many Army and DoD forms that SBIS would need to duplicate.

### 3.1 Business Model

In financial terms, estimates showed that the Army could benefit from $1-2 million [4] in Reuse Cost Avoidance (RCA) as defined in [3], [5]. Independent of the assumptions made, there existed a tremendous business case for large scale reuse. The WOM alone consists of 21k lines of source code; this component represented just one of many large reuse opportunities.

### 3.2 The benefits of experience

Although difficult to quantify, some members of the RCAS team very cooperatively and successfully transferred numerous helpful insights to the SBIS team. Over the course of several technical meetings, the management and developers of the RCAS application software revealed how they handled many technical issues the SBIS teamed currently faced and warned SBIS of situations which would eventually arise. Although the RCAS program had to solve these issues, their openness prevented SBIS from having to do so. Examples include:

1. Each program sought to ensure consistency both within and across applications so every application has the exact same look and feel. To accomplish this, RCAS developed a thorough GUI style guide. However, they discovered programmers still found ways to interpret the guide differently even though the guide appeared clear and obvious. They recommended SBIS make early and concrete steps to define and control access to allowable X widgets and GUI code.
2. Despite detailed interface agreements with legacy systems, RCAS found that legacy systems owners change their interfaces (often without notice) faster than developers can reasonably write software to accommodate them. To deal with this situation, RCAS coded a separately running *External Interface Manager (EIM)* that inputs a description of the legacy system's data format. This allows RCAS to rapidly adapt to most interface changes without writing any additional Ada code.
3. Extensive discussions focused on user interface considerations. In the RCAS design, a single module manages all data integrity checking for the applications (e.g., only allows dates input to a date field). Keeping integrity checking in one place guarantees consistency to the user, allows all applications to share the integrity checking software, and ensures all applications only work on valid data.
4. Separating the GUI access to the WOM means application programmers (Ada specialists) do not have to learn the X event loop process. RCAS stressed separating not only applications and data, but also applications from the database, applications from security software, etc. Not only does this lead to well-engineered software, it allows management to focus their organization's software skills by assigning the most qualified developers to a component.
5. Due to multiple problems with Ada shared libraries, RCAS recommended limiting their use. SBIS had planned to use this technique.
6. Both programs experienced Ada task-signal and Xwindow event signal conflicts. RCAS discussed several approaches they had tried, including forcing Xwindow events through UNIX sockets and writing code to replace the Verdix interrupt service queue at runtime.

## 4.0 Lessons learned from attempting reuse across large programs

Many organizations have made attempts to promote software reuse within the DoD through the use of centralized reuse libraries and technology transfer programs. To date, these attempts have not produced large scale reuse opportunities nor associated savings. This paper describes an attempt to bring a different paradigm to promote software reuse. The effort planned organized and coordinated activity across two programs that could produce predictable savings with low risk. Rather than designers and developers randomly searching for suitable components, program-to-program reuse could result in an engineered, direct exchange of plug-compatible assets. This paper offers the following lessons for those attempting a similar effort:

Lesson #1 - Start early

To succeed in program-to-program reuse one must start early; it requires 6-12 months to establish a formal agreement between two programs. While many visionaries readily support the fundamental concepts of software reuse and its associated benefits, the majority of people actually working on DoD contracts remain highly skeptical of this new way of doing business. These people recognize the important unresolved issues in working with other programs and experience shows that they have well-founded concerns.

Lesson #2 - Address technical and managerial issues simultaneously

Many experts and organizations have addressed the technical and managerial barriers to software reuse. In program-to-program reuse these issues overlap. For the technical staffs to freely exchange ideas or software they first must have management support at many levels. The hierarchical culture within DoD contractors and the DoD itself demands that one address the concerns of each person who may see any possible effect of the effort. In general, staff members seeking to initiate reuse should not do so without the consent of their management and the management of their customers. Unfortunately, in the time it takes to obtain support and finalize a formal exchange agreement, architects may have to make design decisions which would inhibit reuse between the two programs. One must start early on *both* issues.

Lesson #3 - Start with architectures, not software

While most reuse programs focus strictly on software components, experience shows that other assets should come first. Reusing software from another program requires an understanding of the decisions that directed the design of that program's software architecture and its associated components. First, the use of individual software components often depends on the overall context in which they originally appeared. Exchanging these domain-specific design decisions must precede any exchange of software. A shared, common architecture for MIS would facilitate the reuse of software across programs. Second, technical exchanges and transfer of "lessons learned" will assure the engineers on the receiving program that the assets meet their needs and will help mitigate the tendency to reject assets due to the "not invented here" syndrome.

Lesson #4 - Build technical rapport

Program-to-program reuse works best with face-to-face working groups between developers and technical team leads. The agreements of upper level management and formal Memorandums of Understanding (MOUs) make the *exchanges* possible but do not necessarily make *reuse* possible. Technical rapport leads to understanding of each program's software, confidence in each other's products, and trust. Placing assets in a reuse repository in the hope that another program with find and use them cannot substitute for rapport between professionals. Future reuse efforts should focus on encouraging *technical* exchanges.

Lesson #5 - Acquisition policies must support reuse

The use of the RCAS software architecture on SBIS could go beyond short term savings. Having a common, proven architecture would open the door for high levels of reuse on many future MIS systems. However, the tight schedules on programs and the natural tendency to focus internally indicates that these programs need support, encouragement, and incentives. Changes in acquisition policies need to address how the DoD approaches large scale reuse by providing guidance on how to implement and encourage reuse across future software development programs. For example, contracts should clearly state financial incentives (and penalties) for sharing resources with (or not supporting) similar programs.

Lesson #6 - Provide independent, continued support

In addition to written contractual agreements, program-to-program reuse initiatives need vigorous support from an independent arbiter outside both program offices. Because of increasing competition between programs for funds, parochial interests will still arise. An independent agency or executive office can ensure the programs continue to act in the best interests of the government.

Lesson #7 - Education

Education on program-to-program reuse issues may help resolve concerns but the education must take place well before making contact between programs. If this does not happen, the education will only consume time and not have an impact. The current DoD reuse education and courses offered by organizations such as the Defense Information Systems Agency (DISA) and the Army Reuse Center (ARC) can help by educating the all levels of program management of both government and government contractors.

## 5.0 Conclusion

In attempting program-to-program reuse between RCAS and SBIS, many people asked questions for which we had no easy answer. These questions often represent unacceptable risks to the programs involved in such a venture [4]. Program Managers find these risks unacceptable because they encompass the DoD core metrics of schedule, cost, and quality. Once two programs agree to exchange assets they become dependent on each other. Who retains responsibility for a schedule slipping? Which program bears the cost of maintaining the assets? Will assets that become difficult to integrate reflect badly on one program or the other? Changes to acquisition policies can help ameliorate these questions.

The technical exchanges that provided the foundation for this effort succeeded in passing valuable experiences between programs. These experiences helped validate and influence the decisions that ultimately will lead to better, cheaper solutions for the Army customer. Future programs can also benefit from the lessons presented here to help make large scale reuse in the DoD a reality.

## 6.0 References


[1] Arya, Pamela K., "Software Reuse on RCAS," *Proceedings of the 6th Annual Workshop on Software Reuse*, Owego, NY, 2-4 November 1993.

[2] Arya, Pamela K., "The RCAS Software Architecture and Its Relation to Reuse," *Tri-Ada'94*, Baltimore, MD, 6-11 November 1994, pp. 388-395.

[3] DISA/JIEO/CIM, "Software Reuse Metrics Plan," *Defense Information Systems Agency, Joint Interoperability Engineering Organization, Center for Information Management*, Version 4.1, 4 August 1993.

[4] Endoso, Joyce, "Just say yes? RCAS execs decline to share code with SBIS rivals," *Government Computer News*, Vol. 13, No. 15, 18 July 1994, pp. 1, 69.

[5] Poulin, Jeffrey S. and Joseph M. Caruso, "Determining the Value of a Corporate Reuse Program," *Proceedings of the IEEE Computer Society International Software Metrics Symposium*, Baltimore, MD, 21-22 May 1993, pp. 16-27.

[6] Poulin, Jeffrey S., "SBIS Reuse Strategy," *Loral Federal Systems Sustaining Base Information Services documentation*, 22 March 1994, available from the Army Reuse Center (ARC) Defense Software Repository System (DSRS), (703)275-6368.

Dr. Jeffrey S. Poulin
Loral Federal Systems
MD 0210
Owego, NY 13827
Voice: (607)751-6899
Fax: (607)751-2597
Internet: poulinj@lfs.loral.com