Automated Integration for the Joint Maritime Command Information System

Douglas S. Lange
Naval Command, Control and Ocean Surveillance Center, RDT&E Division
NRaD Code 4222
53140 Gatchell Road
San Diego, California
92152-7466
Phone: (619)553-6534
Fax: (619)553-3926
e-mail: dlange@nosc.mil
URL: http://zaphod.nosc.mil:8085/JeTSET/JeTSET.html

Abstract

Budget is one of the primary engineering constraints for United States Department of Defense (DOD) program managers. In order to satisfy expanding requirement sets with less development money, managers have consolidated projects eliminating systems that are designed for the requirements of a single site or class of sites. As the consolidation approach has risen in the hierarchy, the variety of software being brought together has grown and the amount of software reuse required has been amplified. As a result, a new method of system integration has become essential.

This paper describes the integration standard and tool set being used by major elements of the DOD, including the Defense Information Systems Agency which is responsible for integrating the Global Command and Control System. By automating integration, the DOD has developed an efficient, repeatable process for integrating the work of many developers.

Keywords: automated integration, software reuse, software tools, variant engineering

1. Introduction

"Necking-down" is among the many strategies being used by the United States Department of Defense (DOD) to reduce the costs associated with the development and support of Command, Control, Communications, Computer, and Intelligence (C4I) systems. In "necking-down", projects, agencies, and even groups of agencies attempt to consolidate the systems that they are responsible for. The particular implementation of this approach taken by the Space and Naval Warfare Systems Command (SPAWAR) led to a requirement to perform system integration in a new way for the organization, leading to techniques and tools, and ultimately to a new way of building and fielding C4I systems. This new way of doing business is characterized by the following requirements:

- Software developed by one project or agency that fits the requirements of another must be reusable by them without reprogramming. Not only must software be made as general as possible, but all software developed for C4I systems must be able to coexist and in many cases interact with other software being fielded.
Software developed by multiple developers must come together to work and appear as an integrated system. The users must not be forced to understand what software belongs with which functions. Integration must be seamless and, although performance dictates the use of UNIX based engineering workstations, users must not be expected to understand the use of UNIX.

There must not be a situation where because different software is being used for the same purpose, that two workstations or sites come up with different results from the same data. All users must be able to share information and be certain that they are looking at the same information regarding tactical and strategic situations. This property is termed interoperability.

Integration of software from greater than 100 different development organizations must be possible. Schedules of developments must not need synchronization, and integration must not become a bottle neck.

Recently, the Defense Information Systems Agency (DISA) adopted the same strategy for bringing C4I software developed by all defense services into the Global Command and Control System (GCCS), ensuring that the entire DOD will be using the same set of standards and tools developed for SPAWAR. The publication of the first Joint Maritime Command Information System (JMCIS) Integration Standard was in February of 1994. Already several releases of JMCIS based systems have been fielded using these principles and tools. The first GCCS release built in this manner was fielded in March 1995.

There are three areas of standardization that were addressed by JMCIS.

1. Automated Integration
2. Environment Brokering
3. User Interface

Automated integration will be addressed in this paper, but it is through the synergy of these three standardization efforts that widespread software reuse is being practiced in the DOD, allowing the U.S. Government to reduce development and support costs for several major projects, while increasing the capabilities fielded.

1.1. Automated Integration

In "classic" system development, a systems engineer would be responsible for developing an architecture by which software requirements for integration would be determined. Smaller pieces would come together and roll up into larger pieces until finally all the Computer Software Configuration Items (CSCI) are ready to be integrated with Hardware Configuration Items (HWCI). Then a long careful integration and testing phase commences. This works when one has a situation where a single organization (or a small number of organizations acting as one) is responsible for system development. When one begins "necking-down" from a large number of projects, suddenly the number of different software development organizations becomes large as well.

Automated integration is a technique that allows the integration of UNIX based software with a minimum of human intervention. It is based on the principal of standardizing those elements of software design that relate to the ability to integrate CSCI into a system. These factors were determined by the experience of system integrators who had worked with multiple developers in the past. The approach is also grounded on the fact that when one can describe the attributes of the elements of a domain, one can often automate the tasks relevant to those elements. Automation provides two distinct advantages for JMCIS and GCCS integration.

Efficiency

Integrating by hand what now number over 100 segments (the JMCIS integration unit) developed by a variety of developers working from asynchronous incremental build schedules would cause integration to rapidly become a choke point.

Reproducibility

Automating a process ensures that it will be followed in a repeatable fashion. The software tools used by the integration facility are made available to the developers so that they can use the same tools in the same environment as will be used once their product is submitted. If it works for them, it will work.
Essentially the philosophy is to define up front the integration requirements for software and provide tools that will allow the developer to check for conformance to those standards.

1.2. Variant Engineering

The "necking-down" process affects the manner in which software systems are fielded besides influencing how their components are built. The JMCIS library contains all software either built for or used by JMCIS projects. This collection is known as the JMCIS superset. A JMCIS variant is a system that is built using JMCIS integration tools containing a subset of the software available. The process of defining such a subset from the requirements of a particular project or end user site, building the system and testing it, is called variant engineering.

Conformance to the integration standard and use of JMCIS environment brokering provides assurance that software from the library will either "play well with others" or will not be able to be brought into the subset. Thus, variant engineering becomes an analysis role of matching available applications to requirements, and identifying any shortfalls as potential acquisitions or development efforts. This form of software reuse has already saved the U.S. government a considerable amount of money, improved the systems being fielded, and is easier to implement than source code based reuse techniques, which have the additional problem of being difficult to account for in contracting arrangements. Variant engineering can be done as part of the initial system requirements analysis.

2. Integration Standard

The primary reference for all developers, integrators, testers, and variant engineers in JMCIS and GCCS related projects is the integration standard. This document specifies how software must be built and delivered, thus providing guidelines for both application developers and integration tool developers. In fact, the standard is an open one in that by using the standard, one can build one's own set of integration tools with extended capabilities from those of the JMCIS/GCCS tool set.

Topics covered by the integration standard include:
- base and patch segment development and management
- variant definition
- application registration
- segment structure and submission format
- run-time environment rules and format and content of environment broker control files
- inter-segment relationships
- linking requirements and plug-and-play features

3. Process and Tools

Automated integration and variant engineering define a new process for systems analysis and development. Projects that are using JMCIS and GCCS standards for their development are currently modifying their efforts along these lines. The basic flow of automated integration is shown in figure 1.

3.1. On-line Catalog

Software reuse of any type cannot occur when the potential users do not know of the existence of reusable software that meets their needs. For JMCIS, a repository of software was developed and a tool to automatically create an on-line catalog was written. Since the information necessary for generating the catalog can be found in specified locations in known formats, it became easy to extract it for the catalog. All JMCIS and GCCS projects look to the catalog to see if software that meets the needs of their user sites is already present or planned. If it exists, a copy can be downloaded (if they have adequate permission). Information relevant to the catalog includes:
- Name of application
- Description of capabilities
If a new capability needs to be developed, applications are registered. Registration serves two basic functions. First and foremost it provides the developer with an application prefix. This is a 6 character alphanumeric code that uniquely identifies the application to the run-time environment broker tools. Second it creates the application in the library, causing the information described above to show in the on-line catalog.

3.2. Application Registration

- Dependency, compatibility, and exclusion information from run-time environment and integration control files.
- Points of contact
An application is basically a container class for segments. A particular version of an application is made of a base segment and a series of patch segments that modify it. Base segments have an overall structure depicted in figure 2.

While most UNIX developers will recognize such directories as *lib* or *include*, some of these entries are unique to JMCIS/GCCS development. *SegDescrip* is a directory containing control files used by the integration and run-time broker tools to set the UNIX environment information are the files that indicate inter-segment relationships. Segments may relate to each other in two ways. First, as shown in figure 3, segments may depend on other segments.

Dependent segments can be compatible or incompatible (primarily relating to versions). From these relationships, a natural set of discrete structures behavior develops, making tool development extremely clean. These relationships are specified in the *Requires* and *Compat* files found within the *SegDescrip* directory (see table 1).

<table>
<thead>
<tr>
<th>File</th>
<th>Acct Group</th>
<th>Aggregate</th>
<th>COTS</th>
<th>Data</th>
<th>Patch</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcctGroup</td>
<td>R</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>COEServices</td>
<td>O</td>
<td>N</td>
<td>O</td>
<td>N</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Community</td>
<td>O</td>
<td>N</td>
<td>O</td>
<td>O</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>Comm.deinstall</td>
<td>O</td>
<td>N</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Compat</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>Conflicts</td>
<td>O</td>
<td>N</td>
<td>N</td>
<td>R</td>
<td>O</td>
<td>N</td>
</tr>
<tr>
<td>Data</td>
<td>N</td>
<td>N</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>DEINSTALL</td>
<td>O</td>
<td>N</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>FilesList</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Hardware</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Installed</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>File</td>
<td>Acct Group</td>
<td>Aggregate</td>
<td>COTS</td>
<td>Data</td>
<td>Patch</td>
<td>Software</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>-----------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>Menus</td>
<td>R</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>ModName</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>ModVerify</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>Permissions</td>
<td>O</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>PostInstall</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>PreInstall</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Processes</td>
<td>O</td>
<td>N</td>
<td>O</td>
<td>N</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>ReleaseNotes</td>
<td>R</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>ReqrScripts</td>
<td>R</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>Requires</td>
<td>O</td>
<td>N</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Security</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>SegType</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Validated</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>VERSION</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
</tbody>
</table>

R - Required  O - Optional  N - Not Applicable
I - Created by Integrator or Installation Software

Table 1. SegDescrip Contents by Segment Type

**Dependency**

- A depends on B
- B depends on A
- C is compatible with A

**Compatibility**

- A is compatible with B
- B is compatible with A
- C is compatible with A

- depends on or is compatible with
- implied relationship
- illegal relationship

Figure 3. Inter-segment Relationships
3.4. Patch Development

The standard and tools allow for the development and submission of patch segments. These segments are used to swap files in and out, or even run programs that modify the contents of an installed base in the run-time system. The structure and content of patch segments along with their relationship to base segments has made an automated patch management and maintenance build capability straightforward to produce. Figure 4 shows the patch segment directory structure.

The patch segment has different requirements than the base segment for SegDescrip contents. Table 1 shows the required and optional files for patch segments. Figure 4. Patch Structure

3.5. Conformance Verification and Submission

One of the tenants in JMCIS and GCCS standards development is to attempt to provide an automated conformance verification tool for any standard written or adopted. Integration standard requirements are checked by two tools known as mkSubmitTar and VerifySeg. These verify that structures and to some extent content are correct. If the segment passes the tests, mkSubmitTar packages the segment for submission. The delivery structure is a signed (electronic signature from public key encryption) encrypted compressed tar (tape archive format) file. Like other elements of the integration standard this structure is documented so that other projects can create their own tools if necessary and still submit software for JMCIS and GCCS use. Segments can be submitted using a variety of communications mechanisms. Once submitted, most of the steps that the segment takes are done without human intervention. Since point of contact information is provided as part of registration, automatic notification using electronic mail and news is provided regarding every stage of the segment’s progress.

3.6. Integration Testing and Blessing

Integration testing is currently a manual process.
made based on the category of capability the new segment (or new patch) represents.

### 3.7. Variant Engineering and Delivery

Projects in the DOD play two roles. First they are responsible for providing a set of capabilities to some group of operational sites. If a project can provide those capabilities by reusing software built by others, that is typically the most economical manner in which to do it when using the JMCIS approach. If software must be written to provide a new capability, then the project takes on its second role. The project is responsible for developing software to satisfy a set of requirements. It is the former that is called variant engineering in JMCIS terminology. Any system delivered using the JMCIS approach is a variant of the superset being used. Variant engineering is primarily an exercise of matching requirements to capabilities. To this end, the tools that help the engineer are the on-line catalog and a program that creates the variant descriptor file (VDF) shown in figure 5.

The VDF creation tool is able to read information about segment relationships to ensure that all such relationships are satisfied in the variant being defined. For instance, if segment1 depends on segment2, but segment2 has not been added, the tool can automatically add segment2 to the VDF. Again if other behaviors are required by projects they can write tools to perform them since the structures are all defined in the integration standard and supporting documents.

```
<TITLE>Variant Descriptor File</TITLE>

<H1>Descriptive Variant Name</H1>
<H2>Version X.Y.Z</H2>

A descriptive paragraph goes here describing the variant's mission and basic capabilities. The user will be shown this paragraph when the variant is selected to ensure that the appropriate variant has been chosen.<P>

<H2>Reference Documents</H2>
<OL>
  <LI><A HREF="/Documents/Document_Catalog.html#Document1">Document1</A>
  <LI><A HREF="/Documents/Document_Catalog.html#Document2">Document2</A>
</OL>

<H2>Hardware</H2>
Description of required and supported hardware environment.<P>

<H2>Segment List</H2>
<OL>
  <LI><A HREF="/Segments/Segment_Catalog.html#segment1">segment1</A>
  <LI><A HREF="/Segments/Segment_Catalog.html#segment2">segment2</A>
  <LI><A HREF="/Segments/Segment_Catalog.html#segment3">segment3</A>
</OL>
```

Figure 5. Variant Descriptor File (VDF)

By creating an integration standard and working with program managers, the DOD has been able to provide the techniques and tools essential to match the programmatic requirement to put an end to "stove-pipe" system development and make software reuse in the form of variant engineering possible.

Now that there are more than a few projects using the integration standard and tools, the repository concept is being expanded to encompass what is called the distributed software support activity. The current
implementation of a centralized library encourages software sharing among communities, but still allows the central site to become a choke-point for integration testing. By distributing the submissions, blessing functions, and repository, this constraint will be removed. No changes to the standards will be required to support this new capability. The tools merely must be upgraded.

Remote installation and updates are also being added. This capability will allow user sites to pull software from repositories. In addition, where desirable, the repository will be able to automatically push new versions and patches of software to sites. This can be done automatically upon the software being "blessed" just as automatic notification is currently done over electronic mail and news groups.

Variant engineering and reuse searches will be enhanced through interfaces with the Defense Software Repository System. This reuse tool provides keyword and facet searches using a relational database.

The next version of the integration standard will also include a specification for structuring and placing on-line documentation and help files. These will be based on hypertext markup language. By placing them in a known location within the segment, a variant user's manual can be created automatically by creating links to developer produced files.

5. References


[4] The JMCIS On-line Library (JOL) can be found at URL: http://perch.nosc.mil:8095

[5] The GCCS On-line Access Library (GOAL) can be found at URL: http://hakita.nosc.mil/gccs/goal.html