Managing Long-Lived COTS-Based Systems

Vigder, Mark; Dean, John

13 August, 1998

Managing Long-Lived COTS Based Systems

Dr. M.R.Vigder
J.C. Dean
National Research Council of Canada
{mark.vigder|john.dean}@nrc.ca

Abstract

Managing long-lived computer systems containing a large number of Commercial Off-The-Shelf (COTS) software components requires a significant investment of effort. This effort is spent on a number of activities, including adding, replacing and deleting components, integrating and customizing components, and monitoring and troubleshooting the system. Organizations can reduce the time and cost required for these system management activities and increase the usability and reliability of a system, by planning for system management during acquisition. This paper describes how organizations can use software architecture, software instrumentation, and component-based configuration management to support the ongoing system management activities.

Keywords: OTS, COTS, maintenance, management, software, component, architecture, instrumentation, configuration management

1. Introduction

Modern computer systems can be constructed from many software components acquired from many vendors and distributed over multiple networks. Employing diverse off-the-shelf (OTS) components from so many different sources provides challenges to an organization that must acquire, install, and manage the computer system [1,2,4,7,8]. Commercial OTS software components make this particularly challenging for the following reasons:

- Source code is not available (perhaps a blessing).
- Maintenance and release schedules are under the control of the COTS software vendor.
- Software components provide too little, or too much, functionality.
- You are one of many customers and your voice may be lost in the crowd when requesting changes to the product.

System management of traditional systems and management of COTS based systems share many traits. However, keeping a COTS software based system up and running requires that the system manager successfully undertakes a number of activities that have specific COTS related aspects. Among those activities are:

- Component life cycle management. As systems evolve new COTS

components are added, unused components are removed and components are replaced with newer versions or with similar components from different vendors. Changes in configuration may require software components to be moved to a different platform. Modifying the component set of a system is invariably a labour intensive operation.

- Customization. COTS software must be customized (in ways supported by the vendor, not through source code changes) and components must be integrated to provide new services. This can be done at the desktop level through scripting a set of user applications or on a departmental or enterprise level by customizing and integrating back end server applications.
- Behavioural Analysis. System characteristics must be monitored in order to understand the current status of both the components and the system as a whole. Software systems frequently break down and the managers must identify and rectify the problem. The lack of access to the internal operations of the components makes both monitoring and fault isolation a challenging task.

This paper looks at how these activities can be facilitated through proper use of architectures, instrumentation, and configuration management. Section 2 of the paper describes three of the activities that have implications in the life cycle management of COTS software-based systems. Section 3 describes properties one can build into a system to facilitate these activities. Finally, Section 4 provides some conclusions.

2. Management Activities in a COTS Software Based System

Ongoing maintenance of a software system involves a number of different activities. These activities are needed to maintain the continued operation of the system and to evolve and enhance the system. In this capacity a system manager performs a combination of adding/replacing components, tuning the functionality and services of the system, and probably a great deal of troubleshooting. These activities can consume significant resources but the process can be made more efficient by ensuring the existence of a number of appropriate system properties. These properties support the system management function by supplying information about, or by providing a structural context for, the system.

2.1 Component life cycle management

A computer system contains many software components ranging from full-scale applications down to small-scale components such as DLL's and Java Bean/ActiveX components. Many of these components evolve independently of each other.

Part of a system manager's responsibility is to manage these components. This

involves updating components as new versions become available, adding new components as the system evolves, deploying components at different sites and removing components that are no longer required. Operations such as load balancing may require the reconfiguration of the system by moving affected components to other platforms. The activities involved in performing component management include:

- Evaluation of the new or updated component to determine its suitability within the context of the current system. Evaluation looks at architectural compatibility, interactions with other components, performance, etc.
- When updating components, the manager must test, and possibly rewrite, the "glue" code, scripts, wrappers, etc. associated with the updated component.
- Testing the component within the system.
- Installation, deployment, moving and removing components, and all associated configuration management activities.

2.2 Customization

Many software components require a level of customization. Customization for COTS software does not involve modifying source code (otherwise, it is no longer COTS but rather a custom piece of code.) There are however, many other ways in which a COTS component can be customized, such as:

- Using plug-ins to add functionality to an application;
- Configuration or preference files;
- Scripting an application to add new services or to drive one application from another;
- Combining services from different applications or components to provide new user level services;
- Wrapping components to add or hide functionality, or to provide a new interface;
- Providing standard templates or macros for an application.

Customizing COTS software components allows an organization to use a generic component but to tailor it to the local business needs and workflow of the organization.

Customization can be done at many levels. A user’s desktop can be tailored for their personal requirements. At the enterprise level, an application may have extensive customization and configuration to conform to the workflow, accounting practices, etc. of the enterprise.

A system manager’s responsibilities for the customizations include:

- Determining which customizations must be under configuration

management and verify that the CM processes are followed.

- Implementing a process for defining, implementing and testing the customization.
- Deploying the customizations at the appropriate sites.

2.3 Behavioural analysis

System managers must be able to monitor and analyze the behaviour of a system.

One purpose for system monitoring is to allow a manager to maintain and improve the capability of a system. System monitoring requires the ability to determine which software components are being used, how they are being used, how they are interacting, and what problems are occurring. For example, based on analysis of system behaviour a manager could reconfigure system resources to improve performance.

A second purpose of system monitoring is for troubleshooting. Computer systems running many different applications from many different vendors are particularly prone to problems. When a problem occurs, system managers must be able to identify the source of the problem. The problem could lie entirely within a single component, or it may be the result of unforeseen consequences as a result of component interactions. Regardless, a system manager must be able to identify and isolate the cause of the problem. This is a non-trivial problem particularly since there are many components, they are completely black-box and purchased off-the-shelf, and they have been built by different developers.

Fixing the problem first requires the system manager determining the component(s) at fault and why the situation is occurring. Solving the problem will likely require calling the help desks of one or more vendors who will need to know specific details of why the manager believes it is their component causing the problem.

3. Support for management activities

There are significant challenges for the System Manager in supporting each of the broadly based activities described in Section 2. These challenges, while also occurring in a traditional environment, are more clearly evident in a COTS-based system and require that emphasis be placed in a number of key areas. These areas include, the architectural definition process, configuration management of the system and associated COTS components and instrumentation of the system. This section will examine each of the activities in turn and highlight specific techniques which, if applied appropriately, can help in system management.

3.1 Component Changes

3.1.1 Architectural support

In the area of component changes architectural layout is critical in determining the ease with which a manager can substitute, add, move or remove components [6]. An architecture that is designed to accommodate these activities is one that provides for the isolation and efficient communication of disjoint components.

A number of architectural and design styles can be used to achieve a level of isolation and communication. For example, specific design patterns [3] can be used including *facades* and *adapters* to achieve isolation, and *mediators* to encapsulate interdependencies between components [3,7]. These patterns allow data translation and routing to occur independently of the COTS components.

Once architectural and design properties have been identified it can be verified that these properties exist within the system, and that any changes to the system preserve these properties. This verification can be done through a combination of tools, such as static code analyzers and design inspections.

3.1.2 Configuration Management support

Managing the configuration of a traditional software system has been accomplished through the monitoring of the source code that is produced. This includes a mechanism to control and track changes to the source. Numerous tools are available to support this task and the issues involved are well understood. With the introduction of COTS software , Configuration Managers no longer have access to the source for a COTS product and must supplement traditional methodologies. A tracking method must be initiated at a more global level wherein entire modules are tracked by version. This is in contrast to the traditional micro-management techniques.

COTS software systems allow less control over the availability of updates and therefore the CM must be more responsive to vendor update schedules and paths. Configuration control decisions become less concerned with reacting to engineering changes or bug fixes and more often involve deciding when to incorporate a new version of a COTS product. Often this update activity will take place just to maintain compatibility rather than to enhance the system or repair defects. The scheduling of these updates may not be as critical but because of the multiple COTS products will involve some effort.

Configuration management becomes an activity of:

- Tracking versions and version histories of COTS components.
- Identifying differences between versions of COTS components.
- Identifying which versions of sets of component are compatible or incompatible with each other.

- Tracking which versions of each component are installed at every site.

3.1.3 Instrumentation support

Instrumentation can provide valuable information regarding the status of the components of a system. Among the information that can be gathered at run-time are:

- What components are installed on the different platforms.
- What the history of component replacement has been.
- Current version data.

Tools to assist in, and record, the deployment of components should be integrated into the system during implementation .

3.2 Customization

3.2.1 Architectural support

The capability and ease with which a system can be customized to meet local requirements is dependent on the architecture of the system. A system that has a proper architecture and design will allow for more customization during its evolution and management. Builders and managers can follow design guidelines that allow for customization.

One example of designing for customization is to select customizable components. More software components and applications are being constructed with technologies that allow for integration and customization. Examples of such technologies are the inclusion of scripting capabilities within the application (e.g., tcl, VB, JavaScript), and the ability to include third party plug-ins.

Another guideline that can be followed is to employ an architecture that allows for customization. For example, the customizations that implement business processes can be encapsulated within *mediators* [3,7] that can be easily modified. The mediators can be developed in languages designed for rapid development (such as tcl, perl, or VB) while the components can be selected such that they provide standard programmatic interfaces (e.g., ActiveX, JavaBeans, or ODBC).

An important consideration when using COTS software is that it may be necessary to customize one's business process to match the software capabilities rather than the other way around. Some COTS software is designed to incorporate a particular set of predefined business processes (such as SAP, PeopleSoft HRS). Software of this nature used outside of its intended usage parameters tends to be extremely fragile.

3.2.2 Configuration Management support

The customization components developed must be maintained under

Configuration Management on two levels. First, they are managed at the source code level just as with traditional development. Second, they must be treated as components of the system and maintained under CM as described in 3.1.2.

3.2.3 Instrumentation

The customizations are often the only means by which end-users can design instrumentation into the system, as this is the only source code to which they have access. For this reason it is important for system managers to understand the types of instrumentation that is required to maintain the system, and to have a process in place that verifies that the appropriate instrumentation is being adopted when establishing the customizations.

3.3 Behavioural Analysis

3.3.1 Architectural support

As with traditional development, choosing the correct architecture for a system can provide improved monitoring and troubleshooting capabilities. A key aspect is to minimize coupling and maximize cohesion between components so that problems and behaviour can be isolated. Architectural styles can support this, for example by putting component interactions inside mediator components.

The architecture of a system should be designed to isolate faults. If there is a problem within a component, the problem should be detected as soon as possible and should not be propagated to other components of the system. Architecture can support this by ensuring that the code developed to inhibit direct component interaction has been designed to detect, isolate and log component faults.

3.3.2 Configuration Management support

Many problems within a system are configuration problems. They are often caused by incompatible versions of components trying to interact with each other. Configuration management supports troubleshooting in the following ways:

- It can be determined what versions of components are installed at each site.
- Known sets of compatible and incompatible component versions are recorded.
- Component update history is maintained and can be related to when faults were introduced into the system.

3.3.3 Instrumentation support

Instrumentation plays a critical role in monitoring and troubleshooting a system. It is through the instrumentation that a system manager can determine the performance characteristics, component usage, and resource availability within the system. Organizations must understand their requirements for system

monitoring, and design the system with the instrumentation to permit the monitoring.

Troubleshooting COTS based systems is difficult because there frequently is no straightforward way to determine which product is responsible for the occurrence of a fault. It is impossible to examine the internal operation of a COTS module because the source is not available. End-user organizations do have some control over the wrappers, glue code and customizations. By building the proper instrumentation into these components system managers will have some tools available at their disposal to determine which component, or set of components, is at fault.

4. Conclusion

System management of COTS software-intensive systems is a key part of the process of integrating COTS products. We have shown that three important aspects of system management are component life cycle management, customization, and behavioural analysis. We have described how architectural choices, instrumentation techniques and configuration management practices can and should be adapted to support long-term system management in COTS based systems.

Bibliography

[1] A.W. Brown and K.C. Wallnau. Engineering Of Component-Based Systems. In *Proceedings of the 1996 2nd IEEE International Conference on Engineering of Complex Computer Systems*, pages 414-422, 1996.

[2] J.C. Dean and M.R. Vigder. System Implementation Using Off-the-shelf Software. In *Proceedings of the 9th Annual Software Technology Conference*. Department of Defense, 1997.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley Professional Computing Series, 1995

[4] D. Garlan and A. Robert and J. Ockerbloom. Architectural Mismatch or Why it's hard to build systems out of existing parts. In *17th International Conference on Software Engineering*, pages 179-185, 1995.

[5] M.Shaw and D.Garlan. *Software Architecture: Perspectives on an Emerging Discipline,* Prentice Hall Publishing, 1996.

[6] K.J. Sullivan and J.C. Knight. Experience Assessing an Architectural Approach to Large-Scale Systematic Reuse. In *18th International Conference on Software Engineering*, pages 220-229, Berlin, 1996.

[7] M.R. Vigder and W.M. Gentleman and J.C. Dean. *COTS Software Integration: State of the Art*. National Research Council of Canada, Institute for Information Technology technical report NRC39198, January, 1996.

[8] *Component Based Software Engineering Selected Papers from the Software Engineering Institute*. Alan W. Brown editor. IEEE Computer Society Press, Sept. 1996.