System Implementation Using Commercial off-the-shelf (COTS) Software

Dean, John; Vigder, Mark

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

*System Implementation Using Commercial off-the-shelf (COTS) Software**

Dean, J.C., and Vigder, M.
April 1997

* published in Proceedings of the 1997 Software Technology Conference (STC '97), Salt Lake City, Utah, USA. April 26 - May 1, 1997. NRC 40173.

Canada

Title: System Implementation Using Commercial Off-The-Shelf Software
Presenter(s): John C . Dean, CD Dr. Mark R Vigder
Track: Track 4 - Technology Adaption
Day: Thursday
Keywords: COTS, software, reuse, requirements, large-scale systems
Abstract: In an attempt to reduce cost and delivery time of software systems Commercial Off-The-Shelf (COTS) software components are being used in increasing numbers. The source code for these components is not available to the system developer nor does the system developer control the specification, release schedule and evolution of the components. The Software Engineering Group of the National Research Council of Canada is undertaking a series of experiments concerning systems that use COTS software components. In order to better understand the issues facing system users and developers, (as opposed to developers of the COTS software components), we are experimenting with architectures, technologies and processes which may be applicable to such systems.

The experiment consists of developing a prototype of a distributed database management and communications system. This system is representative in that it integrates a significant number of COTS software products under one umbrella, including data acquisition, data conversion, data manipulation, communication, database, and messaging. Many of these functions are provided by COTS software components from different vendors. The purpose of this experiment is to try to understand, as we go through the development, the effect the use of COTS software components has on the procurement process, the development process, and the architecture of the system.

System Implementation

Using

Commercial Off-The-Shelf (COTS) Software*

Abstract

In an attempt to reduce cost and delivery time of software systems Commercial Off-The-Shelf (COTS) software components are being used in increasing numbers. The source code for these components is not available to the system developer nor does the system developer control the specification, release schedule and evolution of the components. In order to better understand the issues involved in system implementation using COTS software, the Software Engineering Group of the National Research Council of Canada is undertaking a series of experiments concerning systems which use COTS software components. In order to better understand the issues relative to system users and developers, (as opposed to developers of the COTS software components), we are experimenting with architectures, technologies, and processes.

This paper describes preliminary results obtained from an ongoing research experiment. The experiment consists of developing a prototype of a distributed database management

---

* NRC Report Number 40173

and communications system. This system represents the need to integrate a significant number of COTS software products under one umbrella, including data acquisition, data conversion, data manipulation, communication, database, and messaging. Many of these functions are provided by COTS software components from different vendors. The research extends our understanding of the problems involved in implementing these types of systems. This research is supported by the Chief, Research and Development (CRAD), Canadian Armed Forces, Canada.

1. Introduction

This paper describes an ongoing study to investigate software engineering techniques to be applied to projects which involve the use of COTS software products as integral components. The systems produced are typically long lived, that is we expect the life cycle to span several revisions of the system and thus the system will evolve over time. We define COTS software as any pre-built software component that is used as a part of the system under development. A COTS software component could include complete applications or services, subsystems, libraries of subroutines, abstract data types or classes. We place the following restrictions on the use of COTS software:

1. The COTS software is not modified in any way but used "as is" within the product to be developed;
2. There is generally limited or no access to technical documentation, including source code; and
3. There is no way to control the release of updates to the COTS software.

As part of this research effort we have previously examined the current state of the use of COTS software within the Canadian Armed Forces[8]. As a result of this study we found:

1. The procurement and development process makes it difficult to maximize the use of COTS components within a system.
2. Then using COTS software, there is a strong temptation to customize the component by contracting with the developer to modify the source. In our view, this is no longer COTS software.
3. Integration of COTS software components can be expensive.
4. New releases of COTS software components arrive regularly and are difficult to re-integrate.

The next phase of the research is to apply some proposed alternative development techniques to a representative system and study the effect COTS software components have on the implementation of the system. To accomplish this task, we examined various National Defence Department projects for suitability. The Canadian Forces Photo Unit (CFPU) Imagery Document Transfer and Management System (IDTMS) was chosen because it was at the appropriate stage of implementation and because the system could be easily partitioned into manageable subsystems. These subsystems consist of a comprehensive centralized Image Distribution system located at the Canadian Forces Photographic Unit and a second subsystem which contains a subset of the components located at each Canadian Forces Base. We chose to implement a non-deliverable prototype of the base subsystem as a functional test bed.

We use a prototype development process to examine new strategies for initially gathering and defining system requirements and of evaluating components. We also employ the prototype itself to assist in evaluating technologies that support a COTS software based development environment. This includes "glue" technologies and component "wrapping".

Building a system by integrating components requires a communication interface between the components in the form of an appropriate "glue"[2,7]. Gluing components requires interchange of data plus an added layer of functionality to control the information flow among the components. The glue assists in combining the functionality of the individual components to provide the required system functionality. We are interested investigating languages (Java, Visual Basic, C++, Perl) and technologies (development environments, CORBA, ActiveX, etc.) that can be used to perform this glue function.

Component wrappers are a mechanism for isolating a component from other components of the system. Component wrappers serve many purposes, for example, mapping data representations, adding functionality to (or masking unneeded functionality of) components, or providing a higher level of abstraction to the component. The wrappers permit the inclusion of critical functionality which the COTS components do not provide internally. The wrappers serve to isolate the components by acting as the interface to the glue code. Should it become necessary to substitute a new or updated COTS component for an obsolete one, most of the code modifications required to support the new or updated component will occur in the wrapper. We wish to look at the best wrapper technologies (scripting languages, frameworks) and architectures that provide the most flexibility in extending and hiding the underlying component.

2. Background

The traditional methodologies in Software Systems development[5] all agree with the concept of the need for the early establishment of clearly defined, detailed requirements. Subsequent refinements of the traditional waterfall based techniques persist in this particular outlook. Gathering, analyzing and accurately recording these requirements occupies a significant part of the process and has initiated an entire field of research, Requirements Engineering. Supporters of these methodologies argue that without detailed requirements the development process is in jeopardy because it is impossible to ensure that the production system will exhibit the needed functionality and will avoid “gold-plated” solutions. These methodologies were applied to both complex and simple systems, with varying degrees of tailoring of the procedures to fit the project.

Most of the traditional methodologies were developed during the era of government sponsored mega-projects. These projects were usually stand-alone systems that were constructed as a unique development and much of the software was custom built. Project managers and suppliers had complete control over the system and had full access to documentation and code for each individual component. Program managers believed that detailed specification of requirements permitted more direct control over the process and a better probability of a successful project. The participants had visibility into the complete software documentation throughout the entire development process and thus could effect changes to the system at the microscopic level. The state of the industry during this period was such that there was a symbiotic relationship between the developer and the customer. The customer (DND, Government) defined each system uniquely and selected a single supplier to produce or manage the production of the system. The developer assigned a significant portion of available company resources to each project and during the development life cycle depended on the customer to support current operations. This dependency allowed the customer to assume a role of preferred client because, for each unique system, there was only one customer. The customer had the power to both direct the activities of the developer and to control the process.

Most systems to be developed were submitted to the competitive bidding process. The depth and consistency of the level of detail of specifications were seen to be an advantage in the tender process because it leveled the playing field among bidders. In addition the criteria for the selection process could be quantified at a microscopic level because of the

detail presented in the requirements. It was believed at the time that this would allow greater visibility for project managers and thus enhance the chances of selecting the best proposed system. The level of detail was also believed to contribute to reduced maintenance effort because the maintainers would acquire the same visibility. This was based on the assumption that the majority of these systems would be maintained by an internal organization and that the maintenance involved significant evolutionary changes to custom built code.

Many of these concepts are not valid when applied to COTS software based systems. The reality of a COTS software based development is that detailed control at the microscopic (code modification) level is not possible. Most COTS software suppliers enjoy a broad customer base that has been developed over time. The developer who wishes to incorporate a COTS software component becomes just one of many customers and has limited or no input to the supplier's development process. Thus it is impossible to enforce unique requirements or constraints on the COTS software component. The project manager must deal with unneeded functionality as part of the product under consideration.

It is obvious that the capabilities of the available COTS software components have a profound effect on the resulting product. Defining a set of detailed requirements establishes an artificially high baseline on the selection and evaluation of COTS software components. This leads to the unjustified elimination of candidate components that might provide a reasonable subset of the proposed functionality of the system.

The primary task in maintenance of COTS software based systems involves solving integration problems as opposed to changing internal code of components. The detailed specification of the individual components is not available and maintenance does not involve modifying the internal code of a COTS software component. Concentration on the definition of interfaces will therefore enhance maintainability.

3. Effect of COTS on the Process

Within the procurement/development process, we believe using COTS software components has the following implications:

1. Initial software requirements must be defined at a very high level of abstraction. Detailed requirements can only be determined in conjunction with COTS component selection.
2. Significant up-front effort is required for COTS component selection and evaluation.
3. Software architecture must be suitable for component wrapping and gluing.
4. Maintenance involves replacing components with new versions of old components, or with new components.

We believe that achievable software systems functionality is dependent on the available COTS software components and should not be fully defined before the developer surveys and evaluates the available components. Under this hypothesis, a detailed description of the system must wait until the final COTS software components are selected. This approach requires a different process from traditional software development as component evaluation must occur early in the system development. We undertake component evaluation with a view to modeling behaviour relevant to the application and developing tailorable system architectures that enhance the interchangeability of components. This is not to say that requirements need not be defined; rather we suggest that we initially describe them at a much more abstract level than with traditional methodologies. We tend to establish much more qualitative requirements because the available COTS software

components determine the quantitative aspects of the system. For example, in the case of the IDTMS project, we state that one of our requirements is that the system be capable of transferring data from the CFPU to the Base in a batch mode using the Internet. We do not state the data format requirements for the data packet because the software chosen to perform the transfer will control that aspect. Similarly we would not designate a particular protocol before investigating the availability of transport software.

We also suggest that, with COTS software based developments, we define requirements with different goals in mind. The major goal is to attempt to describe the real world functionality of the system as opposed to describing the technical aspects. This leads us to conclude that the needs of the end user take on a much more important role in COTS software based development as these tend to closely model real world issues. In addition, user stated requirements are usually more qualitative in scope than those provided by a technical authority. We also allocate greater weight to these requirements because they allow us more flexibility in the initial selection and analysis of COTS components. Qualitative requirements do not allow for the detailed definition of absolute performance criteria (response times, memory requirements) for the system. Within the framework of our approach, this is a desirable quality because the components and not the requirements drive these issues.

Effective use of COTS software requires a large up-front effort to identify where in the system COTS software components can be used, and to evaluate and select the appropriate components[3,4,6]. Developers must first analyze the functionality of the system to determine the classes of COTS components that are applicable. A survey of COTS components available in the marketplace must be performed, and criteria established for selecting the appropriate components. The criteria range across a broad spectrum, including run-time characteristics, documentation, vendor support, etc. This evaluation can include writing test-harnesses in order to determine qualities such as performance, resource requirements, and reliability, information that may not be available from the vendor of the component.

We believe that COTS software components cannot be pre-qualified as generic components for use in these systems. The effort expended to establish generic lists of permitted COTS components is not cost effective. For each project, components must be re-evaluated against criteria that are valid only for each individual system. No pre-qualification process can anticipate the effect of the criteria defined for each specific system. The evaluation of the competing components must determine the component that most closely meets the requirements.

We believe that the development process as a whole must change to reflect the emphasis on the use of pre-determined COTS software components. A new, more important emphasis must be placed on the component evaluation process during the analysis and design phase. Component testing and selection must take place early in the process since we must understand the limits of each individual component and tailor our design to react to these limits. Where no component meets all the established criteria then the most qualified candidate is selected. The criteria are then re-evaluated with a view to eliminating as many of the unfulfilled criteria as possible. Alternatively it may be necessary to incorporate some of the required added functionality into the code wrappers. The resulting system may be less capable than first anticipated but where the use of COTS software is mandated this may be the only viable solution.

With COTS components, the maintenance activity must be expanded to include upgrading and replacing of components. As new versions of components are released by the software developers, and as superior components become available in the marketplace, system maintainers must evaluate the costs and benefits of integrating newer versions of the component into the system. This requires re-evaluating the component and, if it is to be

integrated, rewriting the wrapper to the component so any differences with the original component are masked out. Software maintainers must also have in place an appropriate configuration management process that allows for tracking third-party components within delivered systems.

4. The Experiment

We are examining our thesis by constructing a prototype of the subsystems contained in the IDTMS. The purpose of the prototype is to better understand the approaches needed to ensure an appropriate functionality and architecture for an operational system. The prototype system will be used to manage a set of databases of photographic images and to distribute them on demand over a wide area network. Users of the system will be able to add images to the database as well as search the database and retrieve images. The system will also allow the users to manipulate the images and will provide a suite of integrated tools (graphic, database) to support the process. This prototype system represents the need to integrate a significant number of COTS software products under one umbrella system, including data acquisition, data conversion, data manipulation, communication, database, and messaging. Many of these functions will be implemented using COTS software components from different vendors.

4.1 Overview of the system

The IDTMS is composed of a set of distributed applications. which are physically located within the Photo Units on each Canadian Forces Base. The general overview of the system is illustrated in Figure 1.

Every Base Photo Unit (BPU) subsystem will have the following functionality:

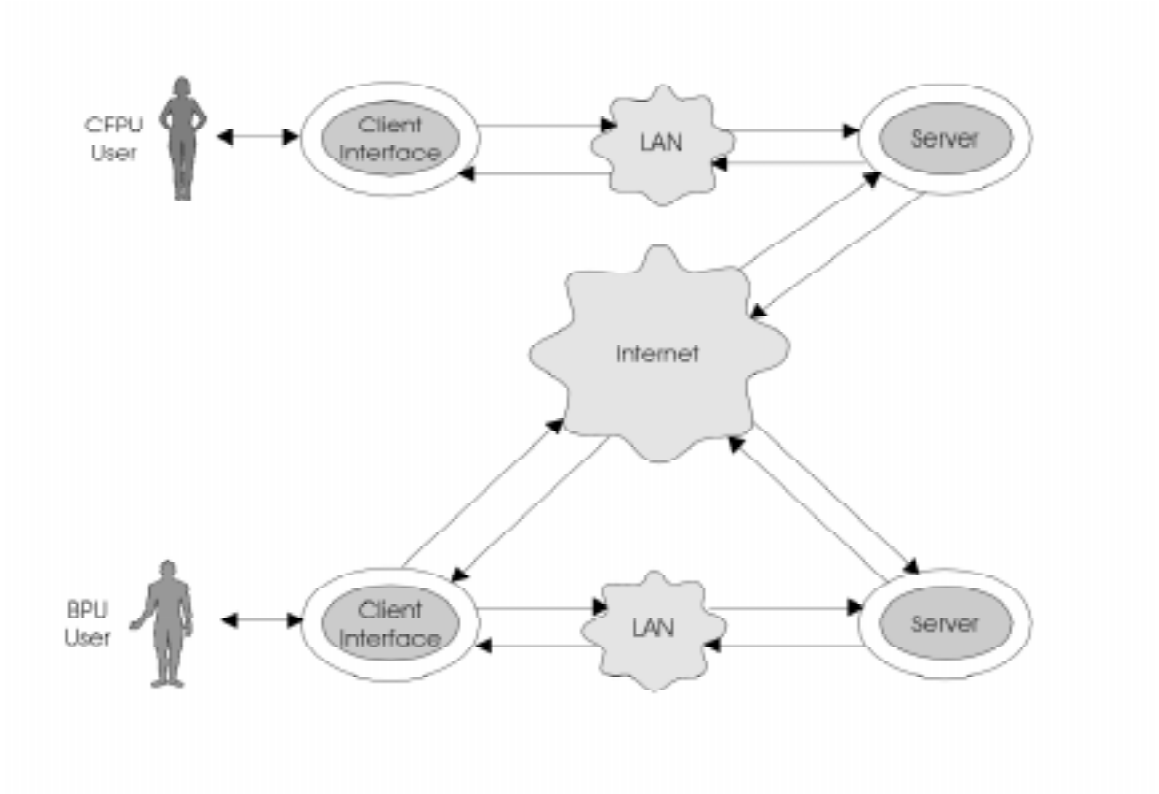

Figure 1. CFPU-BPU Interaction

1. A local image database. The image database will include cataloguing information for all the images. Where appropriate, the images themselves will be stored on-line. The databases permit only local search and retrieval. Although there may be shared data among the different databases, they will not be replicas of each other.
2. Image acquisition. Workstations with special equipment and software (e.g., scanners, digital cameras) will be used to generate images for the database. Images can also be acquired from remote databases.
3. Product preparation. The images in the database are used to generate products for delivery, such as training information, publicity material, and briefing material.

The CFPU subsystem located at National Defence Headquarters (NDHQ) has a special role as a central repository for all material of general and historical interest. It will have all the functionality of the base plus the following added functionality:

1. Wide area access. Users connected through the Internet (or Intranet) will be able to search, retrieve, and order items from the CFPU.
2. Record uploading/downloading. A BPU is able to replicate local records into the CFPU database, or download records from CFPU to the local database.

The Base system will initiate both uploading and downloading of data and images. Browsing requests will be initiated at Base level and the CFPU system will interpret these requests and respond appropriately.

4.2 Description of Process

We are attempting to demonstrate a modified development process throughout this project, but one that is consistent with current standards that recognize the use of COTS components [1]. We describe the overall development process that we are following as evolutionary. The system is developed in a series of steps or phases, each of which represent a complete functional system with clearly defined deliverables. As the project matures, we extend the capabilities of the system via the addition of a new set of subsystems. The completion of each phase represents a point at which we can re-evaluate the system and modify the process. This process is applicable not only to prototype development but also to the project as a whole. We believe that the quality of the final product depends on the usability and acceptance of the product by the user community. This evolutionary approach enables the introduction of enhanced functionality requested by the users at each stage.

Within each evolutionary phase we attempt to follow a number of specific activities, each of which have specific goals and deliverables. These activities are not applied in sequential order but show significant overlap and they exhibit the normal cyclical aspects of any development. The activities are described in the following paragraphs.

4.2.1 Requirements gathering

Initial requirements are gathered at a very high level of abstraction. The purpose is to define the context in which the system will operate and the major functions that the system will provide. The output of the initial requirements gathering is a brief description of system functionality. Our goal for this activity is to determine requirements which will assist in establishing a basis for evaluating and selecting appropriate COTS software candidates. We use a number of techniques including interviews with key personnel, observation of current

practices within the organization and defining short high level scenarios which model the major aspects of the workflow. In establishing requirements we pay particular attention to the constraints which current equipment and practices place on the new system. This is consistent with the stated goal of incorporating changes to work practices in a series of stages.

More detailed requirements are established concurrently with the architectural identification activity. The output is a more detailed description of functional requirements. However, much of the detailed functionality will be dependent on the specific COTS components included within the system.

4.2.2 Architectural identification

Using the data from the requirements gathering exercise, we identify major subsystems or subsystem groups. These correspond to functional groupings of software components. We map the scenarios generated during requirements gathering onto these subsystems, adjusting the subsystems definitions to reflect the scenario guidelines. This enables us to identify specific areas where COTS software can potentially be effectively integrated into the product. We define supporting software platforms and frameworks based on both the requirements and the availability of commercial technologies. Examples of platforms are the operating and communications systems while frameworks include component technologies such as ActiveX or the Common Object Request Broker Architecture (CORBA). The deliverable in this phase consists of a list of major subsystems and a list of candidate environments.

4.2.3 Design phase

We evaluate a number COTS Software components in each major subsystems group and, from these prospective candidates, we select those which will be incorporated into the final product. These components are then subject to detailed analysis and testing to determine exactly what capabilities are available. We concentrate only on those capabilities which are interesting in the context of the current system as determined by the requirements. Interconnections between subsystems and components are identified. The extensions to the components which must be provided by wrappers and glue code are determined and design criteria for these established. Wrappers are designed for each COTS software component. As shown in Figure 2, the interface between each COTS software component and other parts of the subsystem consists of wrapper code which provides translation services for the glue code.

This allows us to isolate components and ensures that, in the case where we must substitute components, that only the wrapper code needs to be modified. This phase results in a design document containing the detailed design of the code wrappers and glue. We also provide a record of the evaluation process for the various COTS software components.

4.2.4 Implementation

The implementation phase consists of the coding of the glue and wrappers and the verification and validation of the system. We also implement any dedicated components which need to be developed to provide functionality not available from the COTS software. We also may need to tailor individual COTS components using any inherent scripting capabilities. The testing concentrates on the interfaces between various COTS software components because the components themselves have already been extensively tested during the design phase of the system. The wrappers and glue code require extensive testing using currently accepted testing methodologies. The final documentation consists of the source code of the glue and wrappers, the design specification and the test results.

4.2.5 Maintenance

The maintenance of the system will concentrate on upgrading and substituting COTS components. This may occur when a COTS supplier issues a new release of a component or when a new COTS component is found that exhibits some enhanced capability that better fulfills a work task. This maintenance will usually require modifications to the code wrappers and glue code.

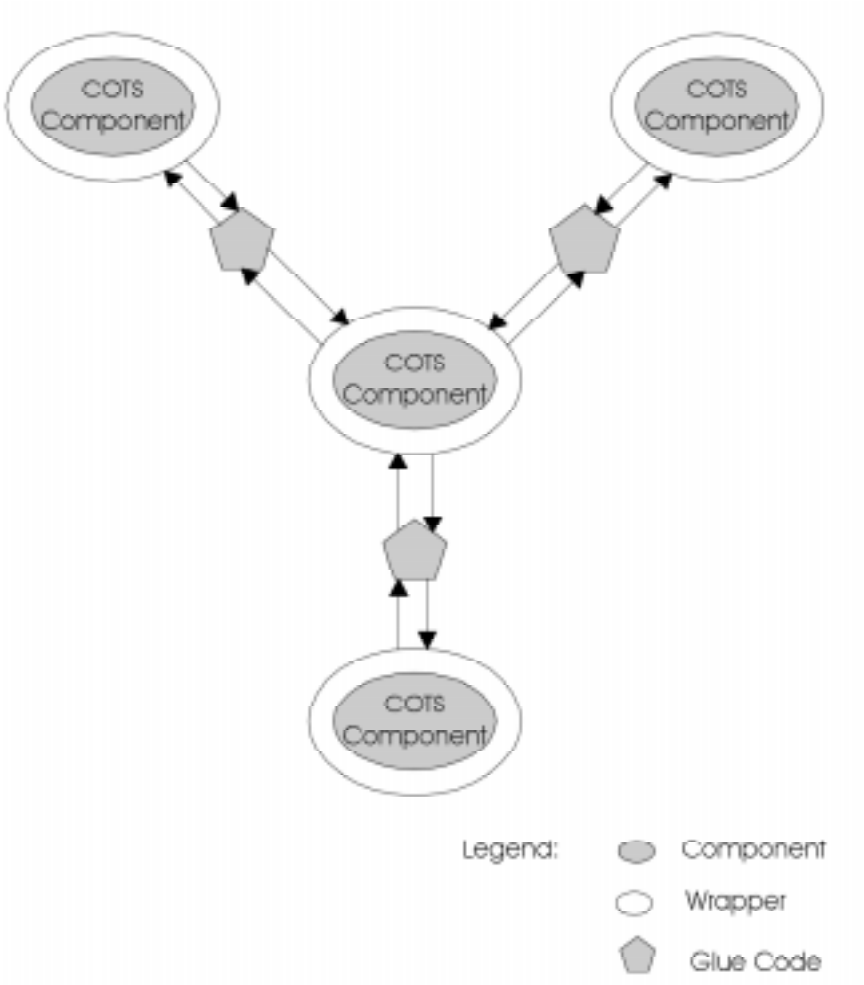

Figure 2. The Wrapper and Glue Paradigm

5. Current Work

Development of the prototype began in December 1996 with requirements gathering and a selection of the functionality to demonstrate in the prototype. At the time of writing, we have completed initial requirements gathering, performed technology evaluations on and selected candidate COTS software components. We have begun preliminary design of the prototype by defining the architecture and commenced limited implementation of portions of the prototype. At each stage we attempted an analysis of our success in adhering to the proposed process. The results will be used to modify our approach in follow up projects.

We present here a description of our examination of the requirements gathering, architectural and the design activities.

5.1 Experiences with requirements gathering

We began the development with a series of interviews with key Photo Unit personnel. These interviews were conducted in an informal setting and usually were group interviews with from two to three subjects attending. The groups included representatives from both user groups and managers. We found that group interviews were effective in that the subjects rationalized opposing views "on the spot". The interviews covered the current state of the Canadian Forces photographic system, both at CFPU and on the base. We looked at what the users proposed for improvements to the current system. The entire interview process concentrated on qualitative aspects of the system. This allowed us to quickly gain insight into the major aspects of the proposed system.

The data gathered was collated and an attempt was made to separate the data into two broad categories of requirements, major (necessities) and minor (wants). We added a category for constraints as well. This category handled the constraints on the system such as the need to access PCD format images. We determined that many of the more specific statements from the interviews could be translated into constraints. This category had a significant impact on selecting image manipulation components. We attempted to observe and record current work practices as part of the analysis. We found that acceptance of new concepts was more readily attained if we could describe them in terms of familiar current practice.

We then defined a set of high-level user scenarios. The scenarios were defined at a functional level and concentrated on global descriptions of functionality. Samples of these scenarios include:

1. A base acquiring images and storing in the local database.
2. A base uploading a set of images to the CFPU database archive.
3. Auser preparing material by searching for images in a database (which could be local or remote) and downloading them into an application suitable for product preparation.

For the initial prototype we concentrated on the communications and image handling aspects of the system. For example, we looked at how an image would be manipulated from initial acquisition to transfer to the main CFPU database, to inclusion within a finished product. One of the areas that we did not consider included security. We will research security issues at a later date. Major functional areas in which COTS software components can be applied include the following:

1. Binary and textual data transfer.
2. Image data acquisition and manipulation.

3. Database operations.
4. Message creation.
5. User interface.

Based on the above functions, and the proposed platform to which we were building (Windows NT), we could begin trying to identify COTS components available in the marketplace that could eventually be integrated into the system. For example, we examined database operations and chose a number of commercial database systems. We evaluated each of these against the criteria developed from the requirements. We established criteria for selection based on the general outline of the overall system and on other development criteria (the constraints). Among the criteria we used for acceptance were the following:

1. Availability of documentation and information on the product.
2. Maturity of the product.
3. Conformance to the appropriate standards and protocols (ODBC, HTTP, etc.).
4. Ability to be tailored through the use of scripting and plug-ins.
5. Ease of integration through the use of the platform architecture (e.g., the ActiveX technology of Microsoft.)

For example, one of the criteria is the scriptability or connectability of the component. While under traditional methodologies this might not even be considered, in this case it is critical to the success of a COTS software driven development. Since we have no control over the update or support of components, we need to be able to substitute a new or updated component easily. This evaluation of components against criteria led to the selection of specific components which are currently being incorporated into the prototype system.

5.2 Experiences with system architecture

The first phase of the prototype implementation concentrates on the Base subsystem and the communication protocols between the Base and CFPU. The Base subsystem is a comprehensive subset of the final CFPU system in that it contains the majority of the components which will perform essentially the same functions. At present some of these functions are simulated but we will implement these if we need to demonstrate enhanced functionality.

The prototype is being designed using a client server architecture (Figure 3). Communication between client and server may be local (both existing on a single machine) or distributed (multiple machines in multiple locations). All communication between the client and the server employs the standard HTTP protocol. This allows us to use Web clients such as Internet Explorer or Netscape as a user interface. It also permits us to use commercial components that implement HTTP and HTML to build specialized interfaces and to integrate with other software components on the client machines. Server to server communication is done using either HTTP or other open standards such as ODBC. This architecture is replicated at each base as well as at CFPU.

There are three classes of servers in the system. These servers supply the database capability as well as providing functional support. The server classes are:

1. Imagery management - manages the storage of all the images and the cataloguing information.
2. Order processing - tracks and fulfills user requests. This includes for example: a base user archiving a set of images to CFPU; or a user requesting a set of hard-copy images to be produced and delivered.
3. User access management - keeps track of all authorized users of the system, their access rights, and their profile.

The server classes are not defined in terms of physical location. Server classes may be co-located or may be distributed over a number of machines. Data or applications contained within a server class appears to the user to be resident on the local machine. Because we anticipate that capabilities will vary from base to base, some server applications may not be universally accessible.

There are up to three classes of clients:

1. Image acquisition - a client that supports the acquisition, cataloguing, and uploading of images to an imagery database. Depending on the acquisition method, special hardware and software may be required.
2. Database access - provides basic support for searching, retrieving, and maintaining the databases.
3. Production - a workstation application with full image manipulation capability for generating finished products.

As much as possible, the integration of the functionality should be seamless to the user.

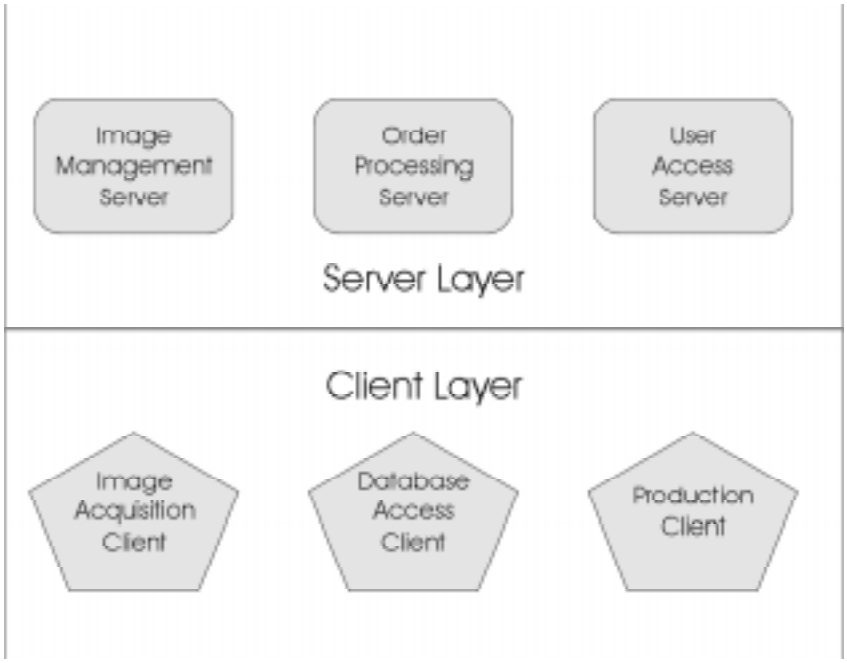

Figure 3. Client-Server Architecture

An example of a user scenario using the glue code is:

1. Search the database via the image management server; select images for downloading;
2. Launch the product ion client (PhotoShop, PhotoPaint, etc.);
3. Create a new project within the product preparation package; and
4. Download the images into this project .

Once this sequence has completed the user is using the product preparation application's user interface. Upon quitting the application, the user is returned to the application.

Most of the server and client classes will be constructed of COTS software components supplemented with custom components. All access to the COTS software components will be accomplished through code wrappers. These wrappers will be external to the application ('scriptable' by the OSA definition rather than 'attachable'). The wrapper adds to the development effort, and adds to the performance overhead, but has the following advantages:

1. New components (or new versions) can be substituted for the old components simply by modifying the wrapper.
2. The wrapper performs data translation functions between incompatible component interfaces.
3. Components can be distributed. The initial prototype will be a single workstation. Future versions will have the components distributed. By using wrappers, a component can be moved to a different workstation and the wrapper modified.

The following paragraphs describe the design of the servers, clients and their corresponding components in greater detail.

5.3 Experiences with design

Component-based design treats a component as a black box and deals with the issue of integrating components to provide the required functionality. In addition to traditional design principles, two elements are further emphasized by the use of COTS components:

1. Components are placed within a wrapper and all interactions with the component are done through the wrapper. The wrapper hides the details of the interface of the component from external component and facilitates component upgrading and substitution.
2. Gluing components together. The glue code controls the flow of execution, data conversions, etc.

The high-level design of the IDTMS server is illustrated in Figure 4. This high-level design, as well as much of the code, will be shared between the large CFPU establishment as well as the smaller base establishments. The major elements consist of the Web server and databases. The Web server we are using is the Microsoft Internet Information Server (IIS).

The specific databases used will depend on the site requirements. A small base with few users can use a desktop database such as Microsoft Access. A large installation requires a higher performance database. Database technology is sufficiently mature that a standard API and query language has been developed (ODBC and SQL). These standards in effect form the wrapper around the database and hide the specifics of the database from the external components . Many database vendors now provide ODBC drivers; thus not only is the database a COTS component but the wrapper is a COTS component as well.

A number of smaller commercial components are used to provide other functionality such as image processing (for format conversion and resizing) and TCP/IP protocol implementation (HTTP, SMTP, etc.). There are numerous COTS components on the market that provide the level of functionality we need for image processing and TCP/IP protocol implementation. In line with the Windows NT™ platform, we elected to use ActiveX technology in whichever components we bought. The components available for these purposes are sufficiently simple and cheap that we did not feel an intensive analysis of the options was warranted . A quick scan of the ActiveX components available allowed us to choose appropriate components. For the prototype we are using Catalyst Socket Tools™ for TCP/IP protocol implementation, and ImageMan from Data Techniques™ for image manipulation.

There are different options from the vendors as to how to glue together the server components. For simple database access this can be done using different technologies (e.g., Active Server Pages from Microsoft, standard CGI). For more complex functions,

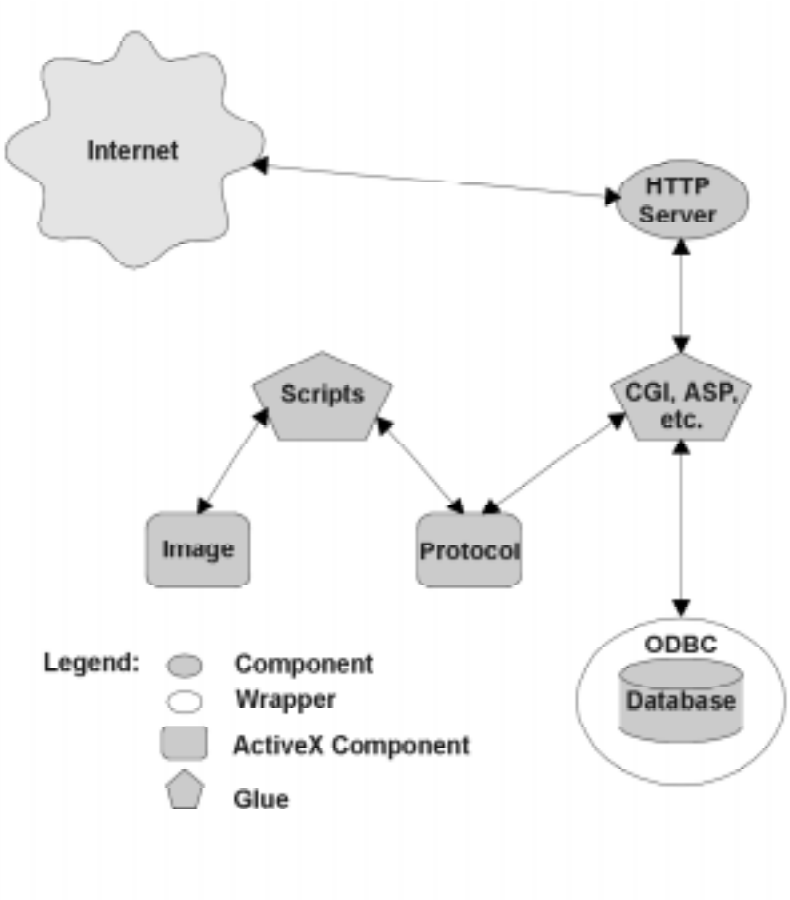

Figure 4. Server Functionality

or functions that are not synchronous with client requests, program scripts can be written that are invoked by the Web server, either when a client HTTP request is received, or that run as stand-alone programs. The languages we are using for this type of glue code is Visual Basic and/or Perl.

The clients interact with the server using the HTTP protocol. For basic interaction such as database searching and submitting orders, any standard commercial Web browser that supports file uploading is sufficient.

Workstations will optionally have software installed for image acquisition and processing. Two goals we are trying to achieve in the design are the following:

1. Moving between the different functions, such as downloading database items and manipulating those items locally, should be as seamless as possible to the user.
2. Substitution of COTS components must be possible without reprogramming. For example, a system should be configurable to use either PhotoShop or PhotoPaint for image manipulation and product preparation.

The design mechanism for achieving these goals are shown in figure 5. A Visual Basic shell provides some of the user interface and the glue code for tying the applications together. Through the Visual Basic interface the user can search and select images from a database server. Upon initiating a download of the images, the glue code launches the product preparation package, creates a new project within this package, and downloads the images into this project. Once this sequence has completed the user is interacting with the product preparation application's user interface. Upon quitting the application the user is returned to the Visual Basic application interface.

The glue code interacts with the product preparation application through a wrapper. The wrapper provides a standard interface to the underlying package, allowing different systems to be configured with different product preparation applications without changing

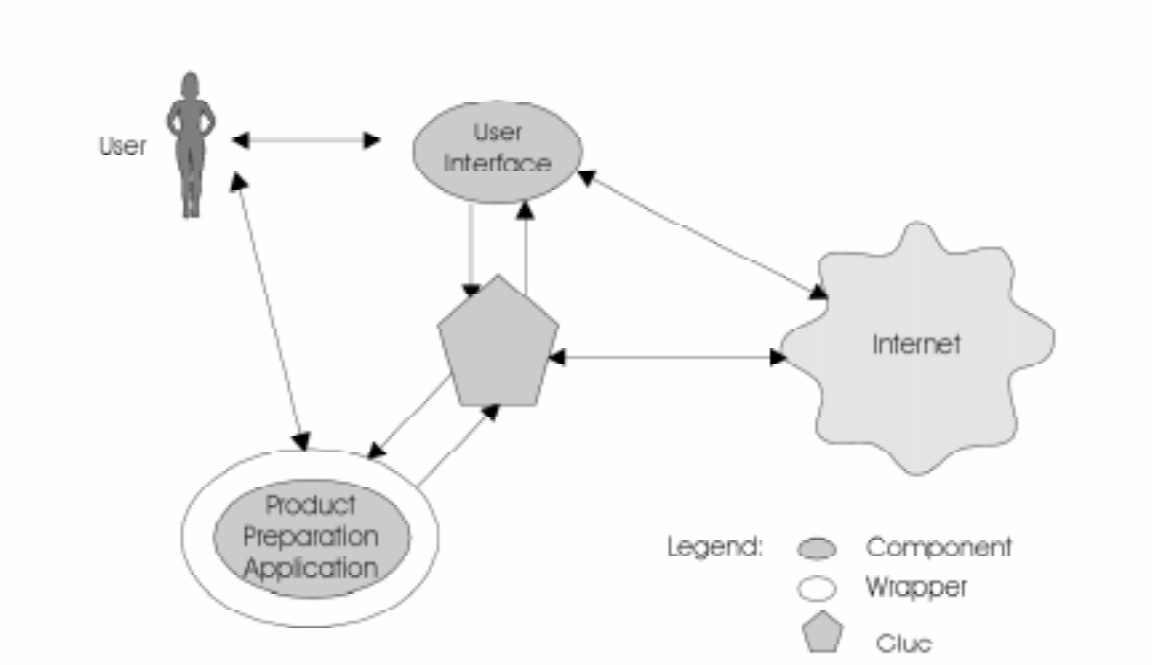

Figure 5. Client Functionality

the glue code.

Other functionality within the client, such as TCP/IP protocol implementation, is achieved through the same ActiveX components that are included within the server. A Visual Basic shell provides some of the user interface and the glue code for tying the applications together.

The interface to the product preparation package is done through a wrapper, also written in Visual Basic, to perform operations such as launching an application, creating a project, etc. Substituting a different package involves providing a different wrapper. The main Visual Basic application does not change. Other functionality, such as TCP/IP protocol implementation is achieved through the same ActiveX components that are included within the server.

6. The Results

Our experiences in defining requirements were not unexpected. Using the evolutionary approach we were able to quickly develop criteria for component selection. In the case of ActiveX components we found that, while component availability was limited, the capabilities we required were provided equally by multiple components. We then made a choice based on perceived ease of integration. We also saw that many of the choices we made in selecting other COTS components were based on personal preference and platform support rather than in a distinct difference in capability. This was especially evident in our choice of web browser since both candidates appeared to provide similar capabilities. We chose Internet Explorer mainly because we were dealing with a Microsoft platform.

The other aspect that we noted is that we tended to change the requirements much more frequently as we gained knowledge about a particular components capabilities. We incorporated additional functionality and also modified our initial design to reflect the capabilities of the COTS software. We believe that we have demonstrated that a more generalized statement of requirements is an effective tool in COTS based development project. This enhances the selection of specific COTS software components later in the project by leaving flexibility to examine a more representative subset of products.

In the case of the design of the system, to date we have implemented portions of the database query system based on the ODBC model. We have also implemented methods to transfer imagery data Internet using HTTP protocols. Our design has been guided by the availability of COTS components so we have begun to confirm a portion of our hypothesis.

7. Future Work

The current functionality of the prototype is confined to a small subset of the overall system. The effort in this direction has to this point been limited. We need to further refine the evaluation and testing of components. We would like to incorporate more functionality into the prototype and then examine the consequences of component modifications and substitutions in more detail. We will develop a set of guidelines based on our experiences for distribution within the Canadian Armed Forces. We also will look at the security issues involved in COTS Software based systems.

References

[1] MIL-STD-498, "Software Development and Documentation", United States Department of Defense, December, 1994.

[2] D. Garlan, R. Allen, J. Ockerbloom, "Architectural Mismatch or Why It's Hard to Build Systems out of Existing Parts", *In Proc. of the 17th Int'l Conference on Software Engineering*, IEEE Press, 1995.

[3] J. Jeanrenaud, P. Romanazzi, "Software product evaluation metrics: a methodological approach", *In Software Quality Management II: Building Quality into Software*, Edinburgh UK, 1994.

[4] J. Kontio, "A Case Study in Applying a Systematic Method for COTS Selection", *In Proc. of the 18th Int'l Conference on Software Engineering,* IEEE Press, 1996.

[5] Roger S. Pressman, "Software Engineering: A Practitioners Approach, 3rd Edition", McGraw-Hill, 1992.

[6] Hyunsik Shin, Jinjoo Lee, "A Process Model of Application Software Package Acquisition and Implementation", *Journal of Systems and Software*, vol 32, pp. 57-64, 1996.

[7] Kevin J. Sullivan, John C. Knight, "Experience Assessing an Architectural Approach to Large-Scale Systematic Reuse", *In Proc. of the 18th Int'l Conference on Software Engineering,* IEEE Press, 1996.

[8] Dr. Mark R. Vigder, Dr. W Morven Gentleman, John C. Dean, "COTS Software Integration: State of the Art", *NRC Report Number 39198*, January 1996.

Biography

John C. Dean, CD1

John C. Dean is an Associate Research Officer in The Software Engineering Group of the Institute of Information Technology, National Research Council of Canada, located in Ottawa, Ontario, Canada. He is also a Professor in the Computer Studies Department of Algonquin College of Applied Arts and Technology.

Mr. Dean enjoyed a twenty-two year career in the Canadian Armed Forces. Among other achievements, as an Aerospace Engineer he developed real-time flight simulation software systems and designed the first Open-loop Record and Playback system for simulators. He was employed as a Systems Engineer for the Dash-8 Navigation Training System. After leaving the Canadian Armed Forces he worked as a Project Manager on various military projects with Adga Consulting, and was a Principal Software Engineer with Unisys Canada. He has extensive experience in Project Management, Software Engineering and Configuration Management and has developed and presented courses in DOD-STD-2167A

and MIL-STD-498. Since joining the Software Engineering Group his interests include Software Engineering education, Software reuse and WorldWide Web applications.

Mr. Dean earned a B. Sc. in Mathematics and Physics (Honours) from the Canadian Forces Military College (Royal Military College) and a Master of Mathematics (Computer Science) from the University of Waterloo. He also graduated from the Canadian Forces School of Aerospace and Ordnance Engineering as an Aerospace Engineer.

John C. Dean, CD1
Associate Research Officer
Software Engineering Group
Institute for Information Technology
National Research Council
Building M-50, Montreal Road
Ottawa, Ontario, Canada
K1A 0R6
Voice: 613-991-6975
Fax: 613-952-7151
Internet: John.Dean@nrc.ca
WWW: http://wwwsel.iit.nrc.ca/~jcdean

Dr. Mark Vigder

Dr. Mark Vigder is an Associate Research Officer in The Software Engineering Group of the Institute of Information Technology, National Research Council of Canada, located in Ottawa, Ontario, Canada.

Dr. Vigder has extensive experience in research, development, and education related to software intensive systems. Among the experiences Dr. Vigder has had prior to arriving at the National Research Council are: the design and development of system software for network based multimicroprocessor systems; definition of protocols, standards, processes, and systems for use in the library community; and visiting professor and lecturer at the Universidad del Valle and Carleton University.

His current research interests include component based software systems, software process, and personal networking solutions.

Dr. Vigder has a Ph.D. and an M.Eng. in Computer and System Engineering from Carleton University, Ottawa, and a B.Sc. (honours, mathematics) from Queen's University, Kingston.

Dr. Mark Vigder
Software Engineering Group
Institute for Information Technology
National Research Council
Building M-50, Montreal Road
Ottawa, Ontario, Canada
K1A 0R6
Voice: 613-991-6972
Fax: 613-952-7151
Internet: Mark.Vigder@nrc.ca
WWW: http://wwwsel.iit.nrc.ca/~vigder

Presentation Summary - STC '97

Title System Implementation UsingCommercial Off-The-Shelf (COTS) Software
Presenters John C. Dean, CD1, Dr. Mark R.Vigder

In an attempt to reduce cost and delivery time of software systems Commercial Off-The Shelf (COTS) software components are being used in increasing numbers. In order to better understand the issues involved in system implementation using COTS software, the Software Engineering Group of the National Research Council of Canada is undertaking a series of experiments concerning systems which use COTS software components. This paper is of interest to conference participants because it provides purchasers with timely guidance in the specification and acquisition of COTS-based software systems. In addition, the research extends our understanding of the problems involved in implementing these types of systems.