PRACTICAL TECHNOLOGIES FOR IMPLEMENTING DISTRIBUTED APPLICATIONS AS EVOLVABLE SOFTWARE SYSTEMS (ESS)

Kendall O. Conrad
Air Force Research Laboratory
Wright-Patterson AFB, OH 45345

Vincent A. Schmidt
Air Force Research Laboratory
Wright-Patterson AFB, OH 45345

Abstract - The Evolvable Software Systems (ESS) concept allows software to keep up with the changing requirements in work and reduces costs associated with upgrading software. Many technologies exist that can be used to create ESS-based enterprise software, and selecting the appropriate technologies can be difficult. We propose a classification scheme intended to divide software technologies into categories that support certain aspects of ESS. Representative technologies are surveyed to determine their strengths and weaknesses with respect to supporting ESS.[1]

Keywords: Evolvable software systems, Progressive Software Systems, network-based, work-centered, software development

1.0. Introduction

The United States Air Force (USAF), other US Government agencies, and many commercial organizations spend a great deal of time and money upgrading and managing enterprise software, adding new and enhanced functionality to keep the system relevant as the work environment and requirements change. The substantial initial investment in enterprise software underscores the importance of extending the useful life of custom software systems. It is not practical to refactor or reengineer entire fielded systems due to time, cost, and other constraints. However, integrating a mechanism that enables software to be modularly modified in the field may be a practical way to extend system usability and lifetime by allowing the software to continually adapt to changing work needs and requirements. In some cases, the software is programmed to evolve dynamically, often autonomously. Various terminology exists to describe such systems, generically referred to as Evolvable Software Systems (ESS) [1].

Features and capabilities supporting an ESS must be explicitly programmed into the system. However, ESS is the realization of a concept, not a technology or software design methodology. Implementation of ESS is achieved using a collection of technologies integrated into a software system. The result is a system capable of keeping up with the changes in work as it evolves. This paper categorizes contemporary technology options that could be used in the practical development of Evolvable Software Systems, and comments on their role in ESS design. The categorization will be helpful in guiding the selection of appropriate technologies as ESS-capable applications are implemented.

2.0. Background

Unlike most desktop applications, where data is locally available and accessed by an individual user, contemporary enterprise applications often involve collaborative work and frequently require access to information distributed across remote locations. These applications are being developed to take advantage of networking capabilities in order to access remote data and promote collaboration within the work environment.

The US Air Force understands the advantages of deploying fully integrated, networked, enterprise-wide command and control (C2) applications. In this net-centric vision, various systems and devices, such as satellites, aircraft, PDAs, laptops, and existing C2 systems fully collaborate, forming the basis of new USAF enterprise applications. The inclusion of ESS concepts also adds value to these new systems. In addition, components used by ESS directly take advantage of net-centric concepts and help further extend the capabilities of a software application.

ESS concepts are supported by not only large frameworks, but also a variety of networking technologies. It would be futile to try to assess every technology to determine its applicability to ESS, and even when the list is narrowed to just network-based technologies there are still many to choose from. This paper strives to show a mechanism for categorizing potential ESS technologies. A small sample of specific technologies is chosen to demonstrate the categorization scheme, providing an example so others will be able to select appropriate new technologies when designing ESS-based applications.

3.0. Categorizing the Technologies

The development of Evolvable Software Systems is not about using a specific type of programming language or technology, but deals with the concept of creating software that is useful longer than typical software, adapting to the changing requirements in functionality and work. However, just because ESS does not specify a certain technology doesn't mean all technologies will work equally well in designing such a system. An assortment of technologies was examined to find out which features

[1] Paper cleared for public release AFRL-WS 06-0442

might best support the creation and maintenance of an ESS, and how that support is provided.

With so many technologies available, a categorizing scheme is needed to help differentiate the technologies and indicate how they assist the needs of an ESS. The technologies were divided into *framework*, *architecture*, *communication*, and *support*. To help understand how each of these levels supports ESS, the main components of ESS (self-maintaining, self-testing, and evolve-ability [1]) were identified with the categories they were most likely to support. These assignments are not absolute, but some technologies seem to match certain ESS requirements better than others. The categorization is shown in Table 1.

The first category, *framework*, contains technologies that contribute to the maintenance and evolve-ability of the system. Maintaining an ESS from a framework standpoint can be easily achieved as long as the framework is well structured. Also, in order to incorporate evolve-ability into an ESS the framework has to be structured to handle evolutionary changes to the system. The framework technologies create high-level views of the overall system architecture and outline the key interfaces and structure of the software system. Such technologies include TBone and ConstellationNet. Frameworks describe the system without designating specific programming languages or committing to specific computing technologies.

The second category, *architecture*, touches on all of the ESS components, testing, maintainability, and evolving. Architecture technologies can help with testing of the systems by enabling creation of benchmark tests and establishing system metrics. The architecture, like framework, provides a definition for structuring a system, which helps with the ability to maintain and evolve the ESS. This category provides a deeper view of the software system and is where more fine-tuned details are defined for specific ESS software. Depending on the features and capabilities needed for the system, specific types of architectures are determined and described. Some of the technologies in this category include CORBA, AJAX, and Macromedia. Selection of these technologies begins to limit other design and implementation options, such as choice of programming languages. Technology decisions here also guide hardware platform and operating system selection. AJAX, for instance, is generally used for online web applications, whereas CORBA has a reputation for being used in developing desktop applications (although it is now used for enterprise applications as well).

The third category, *communication*, supports ESS with testing and maintaining. Networking is dependent on its ability to communicate with other systems, and therefore must also have ways to test communication status within an ESS. There is also a need to maintain appropriate communication requirements to ensure ESS availability. These technologies tend to focus on providing reliable communications and data messaging services. Data messages may include information used within the application, or metadata to monitor the application's status.

The final category, *support*, is for technologies that fit into multiple aspects of ESS, providing low-level assistance. This category includes technologies that transfer data within the system as well as communicating with other systems. Part of that communication deals with using common data formats and transformations to promote interoperability. When deciding upon technologies at this level, common interfaces and data formatting are primary considerations. Various standards exist for using common formats, such as XML.

4.0. Technology Discussion

The following sections and subsections discuss the specific technologies from Table 1 in more detail.

4.1. Framework Technologies

The framework technologies in Table 1 include TBone and ConstellationNet, but could also be expanded to include similar frameworks. These frameworks help to define the high-level ESS architecture without fully committing to programming languages and message formats.

4.1.1. TBone

The vision of the Theater Battle Operations Net-centric Environment (TBone) is to "Create dynamic planning, execution, and assessment capability that links requests, effects, operational guidance, and supporting tasks within a temporal and geospatial unified database environment." [2] TBone advances collaborative planning and supports all Air Operations Center (AOC) processes such as strategy, targeting, planning, execution, and assessment.

TBone aims to provide a single resource that is dynamic and event-driven, and promotes temporal, geospatial, and nodally linked visualizations. It is also scaleable from a single point laptop to an enterprise environment. TBone is

Table 1. Categorization scheme for the various technologies that can be used to develop an ESS.

Categories	***Technologies***
• Framework • Maintaining • Evolving	• TBone • ConstellationNet
• Architecture • Testing • Maintaining • Evolving	• CORBA • AJAX • Macromedia
• Communication • Testing • Maintaining	• Java – RMI • RPC • HLA / DIS
• Support • all categories	• XML • XML XUL • Java Applets (WAR) • Java EJB

intended to be intuitive and interactive, and produces a common look and feel for user content.

TBone replaces older systems with new state-of-the-art technologies such as XML-enabled database, and uses network architecture. This architecture remains consistent with tools used in the systems it is replacing: it provides web services for third party applications, is not proprietary, and has been built net-centric from the ground-up. The TBone framework design fits well into the ESS concept because it strives to handle changes that occur in the work. TBone could thrive as an ESS framework because of its ability to define and communicate the system's architectural definition.

4.1.2. ConstellationNet

ConstellationNet (C2 Constellation) is a communications network overlooking the air, space, and ground allowing a free flow of information to be rapidly accessible to the users (for the USAF, the warfighters) at the right time and right place [3]. ConstellationNet allows users a mechanism to access a globally distributed common knowledge base of shared information, providing a consistent level of understanding and situational awareness between all users. ConstellationNet was created to assist the Air Force in building a network-centric, peer-based system of systems, which can operate in a seamless and fully interoperable framework. ConstellationNet takes a two-tiered approach; the top tier defines C4ISR enterprise integration and at the bottom tier solves near term, quick turnaround integration solutions for the USAF.

ConstellationNet eliminates stovepipe system implementations by providing decisive information superiority, collaborative planning, and synchronized operations for users. It exploits advances in open systems architecture, information processing, expanded bandwidth, and sensor technologies. It is a subset of the USAF Enterprise Architecture, following the precepts established by Joint (DoD-wide) Architecture. ConstellationNet supports ESS by helping various systems work together flexibly.

4.2. Architecture Technologies

Architecture technologies represented in Table 1 include CORBA, AJAX, and Macromedia. These help to establish the functional details of the ESS.

4.2.1. CORBA

The Common Object Request Broker Architecture (CORBA) is a part of Object Management Group's (OMG) open, vendor-independent architecture and infrastructure that computer applications use to work together over networks [4]. CORBA achieves interoperability through use of the OMG Interface Definition Language (OMG IDL) and the standardized protocols GIOP and IIOP. With these, a CORBA-based program from almost any vendor, computer, operating system, programming language, or network can interoperate with another CORBA-based program across similar or different platforms.

One benefit of CORBA is the ability to quickly generate interoperable code, allowing applications to be deployed rapidly. CORBA has a better architecture than traditional software due to the applications being designed around discrete services. The architecture divides applications into modules or object groups based on functionality. The IDL interface definition is independent of the programming language used, and maps to all of the popular programming languages via OMG standards. The learning curve for CORBA and its standards (such as IDL) can be steep, and using IDL for a programming language that isn't supported can be labor-intensive. Fortunately, most languages are covered, including C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript. Using automated tools to map IDL to language creates code stubs based on the interface, but some of these tools may not integrate new changes with existing code well [4]. Also, CORBA does not support the transfer of code or objects. Some of CORBA's specifications are still in a state of flux so continuing training is to be expected.

CORBA is a great choice for enterprise development because of its ability to integrate different machine hardware architectures, such as mainframes, desktops, minis, handhelds, and embedded systems. Within a server it is able to handle a large number off clients, at high throughput, with high reliability, leading to its extensive use for web sites. CORBA's modular design allows different modules to be transitioned for work support tasks. This capability fits well into the ESS concept.

4.2.2. AJAX

Asynchronous JavaScript and XML (AJAX) is a method for building interactive web applications that have a fast interactive desktop feel. AJAX itself is not a technology, but a development approach using existing technologies to create a specific effect. AJAX generally combines several programming tools: JavaScript, dynamic HTML (DHTML), Extensible Markup Language (XML), cascading style sheets (CSS), the Document Object Model (DOM), and the XMLHttpRequest object [5].

Traditional internet-based applications generate events when the user clicks buttons or makes other selections. These events trigger communications back to the server, and the user is forced to wait as results are fetched and the page is reloaded. With AJAX techniques, these server calls are performed asynchronously; the page doesn't have to reload when the server call returns. This significantly reduces the time the user has to wait for information and actions to occur. AJAX improves the user experience, making internet application behave more responsively, like desktop applications. (This can also be a downfall since many users will not be expecting this style of interaction, and may not realize the page has been updated. Developers must take this into account and give visual feedback to the user as event actions are occurring.)

AJAX is still a new way to create web applications, and with new territory comes risks. AJAX techniques rely on

browser support for the XMLHttpRequest object. Although most modern browsers have this support, some security settings may disable the support. Therefore, AJAX-based applications meant to be used by the general public needs to have a fallback option when XMLHttpRequest support is not available. Another concern is how AJAX "breaks" the back button: when the page is updated asynchronously the URL in the browser does not change, and browser history isn't affected. One technique developed to correct this problem is to create an artificial history that rewrites the URL in the browser with information telling how this page was created, so the back button would work as expected. Yet another drawback for AJAX is there is no guarantee that XMLHttpRequest events will be completed in the order they were dispatched, so applications must be designed to handle this accordingly.

AJAX does not have to strictly use JavaScript and other programming tools mentioned here: it can be integrated with PHP and other utilities as needed, taking advantage of web standards so that cross-browser viewing will yield common results. AJAX concepts, such as asynchronous server calls and page updates without reloading, give the user a responsive "local desktop application" experience. This makes AJAX a great tool for ESS applications executed over the network.

4.2.3. Macromedia

Macromedia consists of a web-based suite of software, providing a way to create Rich Internet Applications (RIA). This suite consists of Macromedia Flex, Flash, ColdFusion, and Flash Remoting. Other Macromedia products can be used in conjunction with these, but these four are the mainstay. Although the suite acts as an entire framework, architecture and communication solution when used together, each component has its own strength.

Flex is the presentation tier for RIA. There are four key benefits that allow Flex to help developers build and deploy more effective applications: improved data display and visualizations, more versatile skinning and styling, multiplatform deployment options, and enhanced performance. Flex makes use of technologies such as MXML and ActionScript, and has class libraries to interact with J2EE and .NET. Flex uses a standards-based architecture that is able to compliment current enterprise developer tools, methodologies, and design patterns. Flex makes it possible to give web applications a desktop-application feel by providing immediate responses and smooth transitions between screens and displays. When Flex is executed within the Flash player, the application can interact with server-side functionality, such as Java objects, SOAP web services, and other services.

Macromedia Flash is an authoring tool enabling designers and developers to generate presentations, applications, and other interactive content. Flash projects can contain simple animations and video, complex presentations, and everything in between. Interactivity is programmed with ActionScript, a proprietary language syntactically similar to JavaScript, but movie-centric rather than browser-centric. Using Flash on the web works well because it compresses content into very small files. This is achieved by using vector graphic technology. It is also able to make use of common graphics files in presentations, such as GIF, JPG, and AVI. Flash presentations are viewable in a stand-alone Flash player or with a Flash-enabled internet browser using a plug-in. Both of these options are available for free for a variety of major browsers and operating systems.

ColdFusion is the underlying architecture for creating RIA with Macromedia, allowing for production and delivery of products more quickly than applications created in C++ and other high-level programming languages. ColdFusion uses CFML, a markup language based on HTML, and has a shallow learning curve for those already familiar with HTML. ColdFusion has powerful support capabilities that help integrate structured business reporting within web applications. It can also be used to create instant messaging (IM) applications by leveraging IM presence services, which identify when specific users are online.

Flash Remoting is the connection between Flash and the web application server. It has a powerful yet simple programming model that can easily be integrated with Flash content in applications using ColdFusion, Microsoft .NET, Java, and SOAP-based web services. Flash Remoting provides easy access to business logic. It accesses XML documents and web services using ActionScript, and provides a communications interface to other Macromedia products so data can be easily transmitted to different areas.

The main drawbacks to the Macromedia suite are the expense and a potentially steep learning curve for users who are unfamiliar with the proprietary languages the suite uses. Once mastered, these programming languages allow for changes to be implemented quickly, though it is unclear if the capabilities would support dynamic changes. ESS features could be implemented using Macromedia-based web services.

4.3. Communication Technologies

The communication technologies examined are Java RMI, RPC, HLA and DIS. These technologies wouldn't typically be used to directly establish an ESS, but are useful in the overall design, since communication is a critical function.

4.3.1. Java Remote Method Invocation (RMI)

Java Remote Method Invocation (Java RMI) is used by programmers to create communications links among Java-based applications, allowing methods of remote Java objects to be invoked from other Java virtual machines. RMI uses object serialization to keep track of parameters and does not truncate types, supporting true object-oriented polymorphism [6]. The RMIRegistry component tracks distributed objects, so persistent connections are maintained between clients and servers residing on different machines and processes. The server defines objects for remote client access via RMI setup. Clients use

these objects as if they are local objects running in the same virtual machines. The client RMI keeps the underlying mechanisms of transporting method arguments hidden, returning values across the network.

Since RMI is tightly coupled with the Java language there is no need for separate IDL mappings to control invocation of remote object methods. However, using RMI applets on the internet is impractical because of the instability of the internet and lack of client support. There are also security threats due to remote code execution and limitations on functionality enforced by security restrictions. Java RMI is limited by interoperability only with other Java systems. RMI by itself does have capabilities that could help in creating an ESS, but is a much more useful tool when coupled with Java Enterprise Java Beans (EJB).

4.3.2. Remote Procedure Call (RPC)

Remote Procedure Calls (RPC) is a technique for constructing distributed, client-server based applications. It uses the notion of conventional local procedure calls, so the called procedure need not exist in the same address space as the calling procedure [7].

XML-RPC is a specification and a set of implementations that extend RPC to allow procedure calls to be made over the Internet to machines with potentially different execution environments and operating systems. This makes use of HTTP as the transport and XML as the encoding. XML-RPC was designed to be simple, but also be powerful enough to allow complex data structures to be transmitted, processed, and returned. One benefit of using RPC is its ability to hide the network code within the stub procedures. Generally RPC-based programs are easy to learn and can be implemented in a number of languages and formats such as XML, Java, C, C++, Perl, and Python. RPC calls are made synchronously so the program has to wait for the response before moving ahead. Within an RPC program only one transport can be used (TCP/IP). An enterprise network with several connections may have trouble at times because every simultaneous active RPC connection requires its own distinct connection. RPC does not always have ways of communicating with older legacy systems due to different paradigms. This may limit its value to ESS developers.

4.3.3. High Level Architecture (HLA)

High Level Architecture (HLA) is a general-purpose protocol for managing distributed computer simulation systems, enabling communications between computer simulations regardless of their computing platform [8]. HLA is able to achieve interoperability through its ability to publish and subscribe to attributes and interactions. Communication is managed by its Runtime Infrastructure (RTI). The Interface Specification document defines how HLA-compliant simulators interact with the RTI, and the Object Model Template (OMT) specifies what information is communicated between simulations and how it is documented [9]. HLA simulations must obey published HLA rules in order to be compliant with the HLA standard.

HLA doesn't have to be used with simulated data: it can manage real-time data. This makes HLA valuable in an ESS by giving users a forecasting mechanism within a simulation by using actual data. Third-party RTI vendors provide APIs in C++ and Java, but an RTI can also be custom-built to meet the needs of the specific situation.

4.3.4. Distributed Interactive Simulation(DIS)

Distributed Interactive Simulation (DIS) is a standard for conducting real-time platform-level war-gaming across multiple host computers. The design concept of DIS is that each simulator node is autonomous and simulates a single battlespace entity, or a group of entities in the case of semi-automated forces (SAFOR) systems. DIS uses a distributed concept of simulation in a similar fashion to HLA; each node in the configuration supplies all the resources necessary for its own processing, and has its own responsibilities. These include communicating changes in the entity's state, responding to user inputs, modifying entity state, maintaining a local view of the battlespace environment (including all non-node entities), updating the view, and presenting the user with a view consistent with the state of the entity and the battlespace environment.

DIS does not need a centralized computer to manage the interaction detection and resolution. This helps prevent single-point failures from disrupting the overall exercise or mission supported by a DIS configuration. Like HLA, DIS can be useful within an ESS by allowing users to forecast data, and sharing these forecasts with other individuals by using distributed capabilities.

4.4. Support Technologies

The support technologies that have been reviewed are XML, XUL, Java Applet Web Archives (WAR), and Enterprise JavaBeans (EJB). These technologies may not directly help in development of an ESS, but they provide ways to create a good software system, and are able to support other technologies.

4.4.1. Extensible Markup Language (XML)

Extensible Markup Language (XML) is a simple and flexible text-based format derived from SGML. XML is used to exchange a wide variety of data on the web and between applications. One XML utility is the Extensible Stylesheet Language (XSL), a family of recommendations defining XML document transformations and presentations. This is accomplished through three parts: the XSL Transformation (XSLT) language translates between formats, XML Path Language (XPath) is an expression language used by XSLT to access or refer to parts of an XML document, and the XML Formatting Objects (XSL-FO), is an XML vocabulary for specifying formatting semantics [10].

An XSLT stylesheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses a formatting vocabulary, such as XHTML or XSL-FO. XML validators parse XML and ensure the code is well-formed.

This structure allows XML documents to be generated efficiently from existing information. The documents are stored in plain text and can be edited or used by any platform or operating system. XML is also language independent, allowing it to work with heterogeneous systems. When interfacing with a non-XML-capable system, XSLT can transform the XML into HTML or plain text. XML's extensibility allows developers to create custom tags for their own data formats, so web servers can pass application parameters and return values as XML.

XML is still young in comparison to other formats, but is gaining popularity. Browser support for XML is not completely standardized, but this can easily be overcome by using server-side technologies to handle XML before it gets to the browser. Since XML is in plain text, file sizes are typically larger than binary formats, but as network performance improves this will become less of an issue. XML would be useful as a protocol for data exchange in ESS. Using different stylesheets, XML can be transformed into many different layouts depending on the need.

4.4.2. XML User Interface Language (XUL)

XML User Interface Language (XUL) is a markup language, like XML and HTML, that can be used for creating dynamic user interfaces. Since it is platform-agnostic, XUL can be used to create applications on Windows, Macintosh, or Linux. XUL is based on XML, so it inherits XML's standards and capabilities: XSLT, XPath, and DOM functions can manipulate the user interface. Even though these technologies are not individually very powerful they can be combined to create interfaces for full fledged applications, such as the Mozilla Firefox browser and Mozilla Thunderbird email client. XUL can also be used to create stand-alone applications or installed as a browser extension. These extensions modify existing browser behavior and graphical interfaces. For example, XUL can make modifications to the menu structure, or simply change the look of "Back," "Forward," and "Refresh" buttons within the browser. XUL is a part of Gecko engine used in many Web browsers. Gecko supports various Web Services technologies such as XML-RPC, SOAP, and WSDL.

Having XUL based on standards such as XML allows it to easily work across browsers and operating systems. These web-based languages are also much easier to learn than standard programming languages such as C++. Using XUL to develop web applications as extensions is much easier than creating stand-alone applications. XUL has great flexibility for being used on web sites because it can make use of server-side architectures such as PHP and JSP to display dynamic content [12]. Its overall flexibility allows Gecko to be a two- or three-tier application model depending on the needs of developers. XUL makes use of existing standards, but XUL itself is not yet a standard. (It has been submitted for approval.) Other drawbacks of the current state of XUL are that creating stand-alone applications with XUL can be tricky, but progress being made to simplify this process.

When installing, or uninstalling XUL extensions, the browser must be restarted for the changes to take effect. Even so, this ability to add extensions so easily makes it a great tool for ESS. An assortment of extensions can be created to supplement user's work in different ways; the users have the choice of which extensions they want to use, and can remove them later when desired. Since the XUL code is based in XML, the code can be updated easily to make adjustments for changing work requirements, reducing the development cycle for any necessary changes to the application.

4.4.3. Java Applet Web Archive (WAR)

Java Applet Web Archive (WAR) files allow Java 2 Platform, Enterprise Edition (J2EE) Web modules that include Java Servlets and JavaScript Pages (JSP) to be packaged together with the application, associated files, and a required XML deployment descriptor file. J2EE WAR applications include client components consisting of dynamic web pages, applets, and a web browser for the client machine. The web components include servlets and JSP, running within the browser. Business component modules implement a particular enterprise domain, and enterprise infrastructure software runs on legacy machines.

WAR files have the same format as JAR files, but also include a deployment descriptor file within the JAR. Each WAR file contains the servlets and JSP along with related resource files. The web components tier in the Access Manager model can be customized to meet the needs of an organization. WAR files allow for an easy way to make updates to the system, providing new functionality. This makes the Java Web Archive a good tool for an ESS.

4.4.4. Enterprise Java Beans (EJB)

Java's Enterprise JavaBeans (EJB) technology is a server-side component architecture for the Java 2 Platform. EJB enables rapid and simplified development of distributed, transactional, secure, and portable applications based on Java technology [13]. EJB development encapsulates business logic in a component framework that manages the details of security, transaction, and state management.

EJBs come in three varieties: an entity bean has persistent data with a specific bean representing database record information, a session bean simply provides a service and no data, and a message-driven bean provides a simple method of asynchronous communication (rather than using Java Message Services (JMS)). Java's practice of "write once, run anywhere" holds true for EJB as well, and works on most operating systems.

EJBs make is possible to build distributed applications by combining bean components from different vendors. This flexibility promotes ESS generation by offering programmers a variety of development components. EJB applications are easier to write because developers don't have to deal with low-level details of transactions, state management, multi-threading, or resource pooling. It also allows experts to gain low-level access through APIs.

Buying commercial EJB containers can be expensive. Using them can also create large and complicated specifications and increase development time when compared to straight Java programming, but the beans are reusable for future projects. EJB limitations include lack of support for local files, limited GUI support, no capability to act as a network server. Nevertheless, EJB modularity may make EJB technology useful for ESS.

5.0. Impact of Technology Selection

The classification scheme is not intended to be a strict code for selecting technologies, but should be looked upon as a set of guidelines for choosing the foundations of the enterprise application. Even within this classification, the survey of technologies presented clearly demonstrates that not all options are equally suitable for the development of ESS applications. Some of these, such as XML, RPC, and HLA seem to have characteristics allowing them to serve best in a support role. Others, such as XUL, AJAX, and TBone, offer structure to the ESS architecture. Regardless of the specific system, classification should be useful for selecting frameworks, architectures, communications, and support components as the system design solidifies.

The goal of an ESS is to be able to evolve as a user's needs change due to the changing nature of the work. Technologies directly supporting this ability to evolve will offer the best immediate value to the development team, and ultimately to the end user. This inherent adaptability separates ESS from typical contemporary software systems. ESS software maintenance and testability issues are probably not as central to the initial selection of technologies, but must be considered early in the software design. Essentially, a technology is beneficial in the development of an ESS if it is able to help, directly or indirectly, with making evolutionary changes to the system. Classification of technologies simplifies selection, with more well-defined options and fewer decision points.

6.0. Conclusion

A classification scheme has been introduced in an effort to compartmentalize the technologies that could be used to build ESS-based enterprise applications. Within each class, a number of technologies were surveyed, with a brief discussion of strengths and weaknesses with respect to their usability in ESS development. There are many more specific technologies than those presented here, but these are a representative cross-section for the research occurring in the Cognitive Systems branch at the Air Force Research Laboratory.

This paper has divided technologies into four main categories to help identify the features they promote for effective ESS designs. The *framework* category focuses on software system structures for the high-level view of the system. The *architecture* category describes the details and underlying mechanics of the software system providing the heart of the ESS. The *communications* category contains technologies enabling the most effective ways to transmit between its local and remote systems. The *support* category concentrates on interoperable message formats and low-level support issues common to other categories. This classification scheme is expected to help identify appropriate components for implementing ESS.

Continuing research in ESS will focus on the creation of ESS-based enterprise software, especially with respect to the selection of specific technology components suitable to software evolution. The technologies surveyed here are likely candidates to be used in this work.

As software becomes network-integrated, we anticipate an increased interest in ESS concepts and practical implementation approaches. Evolvable software provides users with a powerful tool for accomplishing their work.

References

[1] Conrad, K.O., Schmidt, V.A. (2005). Progressive Software Systems: Dynamic Software for a Dynamic Workplace. *Proceedings of the Ninth IASTED International Conference SEA*, Nov. 14-16, 2005, Phoenix, AZ, USA.

[2] AFC2ISRC (2005). *Theater Battle Operations Net-centric Environment (TBONE)*. http://www.afc2isrc.af.mil/tbone/.

[3] Sweet, N., Kanefsky, S. (2004). The C2 Constellation A US Air Force Network Centric Warfare Program. *Command and Control Research and Technology Symposium*. San Diego, CA, June 15-17, 2004.

[4] Object Management Group (2006). *OMG Specifications and Process*. http://www.omg.org/gettingstarted/.

[5] McCarthy, P. (2005). *Ajax for Java Developers: Build Dynamic Java Applications*. Retrieved from IBM site: http://www-128.ibm.com/developerworks/library/j-ajax1/?ca=dgr-lnxw01Ajax.

[6] Reilly, D. (2000). *Java RMI & CORBA: A Comparison of Two Competing Technologies.* http://www.javacoffeebreak.com/articles/rmi_corba/.

[7] Wikipedia (2005). *Remote Procedure Call*. http://en.wikipedia.org/wiki/Remote_procedure_call.

[8] Dahmann, J.S., Fujimoto, R.M., Weatherly, R.M. (1997). The Department of Defense High Level Architecture. *Proceedings of the 1997 Winter Simulation Conference*. p.142-149.

[9] Buss, A., Jackson, L. (1998). Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI. *Proceedings of the 1998 Winter Simulation Conference.* p. 819-825.

[10] W3C (2005). *Extensible Markup Language (XML)*. World Wide Web Consortium (W3C) site: http://www.w3.org/XML/.

[11] XULPlanet (2005). Welcome to XULPlanet. XULPlanet site: http://www.xulplanet.com/.

[12] Hosanee, M. (2002). *Manually Creating a Simple Web ARchive (WAR) File*. Sun Microsystems site: http://access1.sun.com/techarticles/simple.WAR.html.

[13] JavaBeans (2005). *JavaBeans*. Sun Developer Network site: http://java.sun.com/products/javabeans/.