Technological Evolution Affecting the Maintenance Horizons

Department of Computer Science Technical Report 05/00, December 2000

Claire Knight and Malcolm Munro
Visualisation Research Group,
Research Institute in Software Evolution.
Department of Computer Science,
University of Durham,
Durham, DH1 3LE, UK.
{C.R.Knight, Malcolm.Munro}@durham.ac.uk

Keywords: Software Maintenance, Distributed Systems, Components, Internet

Abstract

Rapid change is nothing new in a discipline that only has a lifespan of approximately fifty years. What is unique at the moment however, is the speed at which industry is adopting and then utilising these changes. The Internet is the primary driver in this rapid evolution, and this requires a step change in the thinking behind software evolution. Logically, and financially, it makes sense for them to do this so that they do not lose market-share, but by doing so they are creating a wealth of maintenance issues. These are issues that the academic communities as well as the industries involved need to address if they are to continue to be considered an asset and to be making useful progress in furthering the field. It is no longer the case than maintenance and legacy only applies to monolithic, one machine, one data-store transaction systems. It also means that large systems that may be more flexible can no longer be considered entirely in isolation. There is then a final concern that the newer systems being developed and deployed are wholly unsuitable for use with existing tools and analysis.

1. Introduction

In the era of the Internet there must be a step change in the concepts and theories of system and software evolution. In this new era the technologic evolution is advancing so rapidly that the maintenance horizons have to also adapt. This paper sets out to introduce some of the major factors and the related technologies, which have caused this mass explosion of new and as yet unmaintained (certainly in the long term) systems. It is not a tutorial in any of the specific languages or frameworks presented, nor is it an exhaustive survey. What it does do, however, is make clear the areas in which maintenance activities should be focused to take account of this new phase of software systems and the attitudes towards them.

Because software is rapidly changing, then the associated engineering and maintenance also need to respond in that same time frame. This is not just about the evolution of the systems, which is one particular aspect of maintenance. It is also to do with the real world implementations already in use but often ignored by maintainers.

The maintenance process itself may not be changing all that much, but the activities that go towards that process have to change to accommodate the software and systems that need to be maintained. There is a huge difference in attitudes by software development companies, and also by the businesses that use their technology, products, and services when compared with academic attitudes. There is a pressing need to embrace and acknowledge new technology, and to adapt the approaches and techniques to cater for these changes. This is particularly true of international software engineering related conferences where a radical and fundamental change of agenda must occur in order to address the technological issues caused by the increasingly ever-pervasive Internet.

Maintenance itself needs to evolve as the environment around it changes. There is no reason to

consider “maintenance” to be any different to the things it is applied to. It is no longer enough to claim that detailed analysis of large homogenous systems (such as a COBOL transaction processing system, or an application comprising a large amount of C code) are the only valid maintenance activities. Rapidly changing, possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,

possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising various component techniques are prominent. Database
technologies now link with the web (even at the vendor .7( )127(act)6.83(a l)6one activities. Rapidly changing,
possibly distributed, heterogeneous systems utilising

provides access to services similar to the others under the name of CORBAServices, with CORBAFacilities providing less generic services. The benefit of a standard architecture is that it provides developers with interfaces described at a high level so they are then free to implement across languages, platforms, and operating systems. CORBA was developed by the Object Management Group (OMG), which is actually an industry group rather than one specific vendor, although implementations that adhere to this specification will obviously be specific to some company.

A generic ORB is capable of handling language, system, and vendor boundaries by acting as a form of lookup service, which then handles any necessary connections between the requesting component and the remote object that fulfils the results of the lookup. The ORB has to provide all of the necessary distributed communication transparently. It also acts a shield between the two components in that their implementation has no need to be revealed, and therefore is (as it should be) of no concern to the other. Therefore it should be obvious that access to services such as naming directories, event handling, lifecycle and persistence control, transaction management, and concurrency are necessary whatever the actual implementation.

As well as the CORBA ORB, the other main technology is Microsoft's COM/DCOM. This is their Component Object Model (and Distributed COM) [10]. Much of Microsoft's work is tied to PC platforms and their own operating systems, although that is starting to change through third party implementations. COM is used to refer to both the specification and the implementation by Microsoft. From a specification point of view it defines an API for component creation and interaction, but all components have to adhere to the same binary structure to interact. This does allow the use of different languages, but obviously places a restriction due to the binary structure that must be compiled from the code. DCOM allows for the components to be distributed across machines and networks. COM and DCOM can be considered together because the supporting infrastructure incorporates the two together. This lower level technology is then capable of supporting Microsoft's OLE and ActiveX components. A possible side effect of this technology this is not prevalent with CORBA is that because of the closeness of this technology to the operating system code of the various Windows incarnations and their APIs, even certified code could potentially cause damage. COM+, an updated view of the component model makes it easer to write and use components compared with COM. It incorporates more of the distributed services required by component applications running across machines and networks. There is also much better support for languages other than C++, which is highly utilised in Microsoft operating systems.

A final technology that exploits ORB like capabilities is the RMI (Remote Method Invocation) model that has now been part of the standard Java API for the last few releases. This is less generic in that it only permits Java objects to be executed remotely (so only covering different platforms where Java Virtual Machines exist), but it is a native part of the extensive APIs that come as the standard reference edition of Java from Sun. In order to utilise middleware and to leverage existing (possibly legacy) applications RMI also interoperates with IIOP (Internet InterORB Protocol) to permit CORBA connectivity. This is a much more portable version than pure RMI because it allows the integration and utilisation of many CORBA services, and also permits non-Java code to be incorporated. RMI-IIOP incorporates full CORBA ORB functionality and thus combines the best of RMI with the best of CORBA without requiring other levels of software.

2.2 Internet Interfaces

Not only has the use of middleware technology accelerated the take-up of client-server architectures whilst still being able to leverage legacy applications, there has been a move towards online information presentation, access, and transactions. These all require some form of Internet (or at the very least Intranet) interface, the most common of which is through a browser.

There are a number of scripting languages that can be utilised when creating web pages, beyond that of standard HTML. Apart from SGML and XML which are intended for mark-up of text in a static fashion, (at least in the context of the web) there are several ways in which dynamic content and interaction can be built into pages. These are of significance here because of the freedom it gives application developers in linking web front-ends to what would previously have been considered non-Internet relevant applications. This then has a direct impact on systems that utilise such technologies and they are therefore part of the maintenance and evolution cycle. An added burden with pages that are generated based on specific (and thus quite feasibly varying) parameters is that they can have an existence span of seconds. The speed at which such artefacts come into being and are then replaced

obviously has an impact on maintenance and evolution, not least because of the speed of change.

The first dynamic content to universally appear on the web, connected with server-side applications, was based around CGI (Common Gateway Interface) programs, often combinations of Perl and C code that then returned HTML to the web server, to be passed to the client. Since then there have been various adaptations to HTML such as DHTML and embedded scripts like JavaScript (nothing whatsoever to do with the Java programming language) that allow client side dynamic pages. In reality most client-side dynamism is only for interface effects. This is of limited use anyway, and would not be necessary when actually using the web as a user interface to an application.

It could be asked as to why one would want to have dynamic web pages. Several reasons that relate to industrial use of the web as an interface are:

- To respond to user data and/or selection
- To represent content that is derived from source data that is subject to rapid change
- To utilise server side resources, such as data warehouses, transaction databases, etc.

There now seems to be four main techniques for creating dynamic web pages that permit server-side interaction as well as allowing the pages to be written as HTML with embedded extra actions or tags; PHP, ASP, JSP (also encompassing Servlets), and ColdFusion. Each of these will be covered briefly, and ASP and JSP revisited in Section 3 due to their significance in wider specifications. Java applets are not covered because they are another application embedded into the browser rather than at the HTML level.

PHP

PHP is often referred to by just these three initials, but the full name is PHP Hypertext Processor [8]. It is a scripting language that is embedded into the HTML script used to describe the web page. PHP shares syntax with C, Java, and Perl. Whereas CGI scripts require that there is program output that conforms to HTML structure, PHP is embedded into the HTML to do something. The PHP code is then enclosed within start and end tags. PHP is a server-side scripting mechanism because it is executed, and therefore resolved, by the server before being sent to the client. This enables the programming and logic to be hidden from users, as all they see is the completed HTML file (which comes from the original HTML and the results of the PHP code). PHP is similar to both ASP and JSP. PHP is capable of doing anything that can be done from a server side program, but the strongest and most significant feature is that it has good support for a wide range of databases, and therefore writing a database linked web page is made very simple. PHP does support other services through the appropriate protocols, and it even allows for network sockets to be opened and the low-level interaction handled.

ASP

ASP is a Microsoft initiative that stands for Active Server Pages. Its initial limitation was that it required an investment in Microsoft technologies (such as operating systems and web servers) rather than any that implement services (which is how PHP can be integrated with many servers, especially under UNIX and Linux) or that have a Java Virtual Machine available (thus enabling JSP pages). This is changing as Microsoft seeks to open the market to third party vendors, and has therefore labelled the newer initiative as ASP+. ASP allows for untyped variables and also provides easy access to the Windows API through the COM standard. This power also adds to the possible complexity of ASP code. Many ASP pages are created using HTML and VBScript (essentially embedded Visual Basic), although there are several other specific languages supported by ASP.

Microsoft Transaction Server (MTS) was integrated into Microsoft web technologies a few years ago, but the COM+ initiative saw such functionality merge with that of COM/DCOM. ASP+ is a response to this to update ASP to become more component-oriented, and therefore modularised thus creating runtime component objects. This provides greater scalability (through the built in COM+ features for distributed and transacted applications, pooling etc.) and more scope for creating new components within ASP+. ASP+ has also moved from a pure scripting language to allowing compiled components. This is much like the JSP concept of compiling the script into a Servlet as a server-side action before serving the user.

JSP

JSP are JavaServer Pages, and they enable static HTML to be combined with dynamic content. There are actions that can be called directly from the HTML embedded tags, but they are also a convenient mechanism for working with Servlets (covered later). JSP allows for a separation of concerns between content provision (and the dynamic nature of it) and the HTML used for presentation and layout. The majority of the page is written as for any other HTML page. The logic and code that is necessary for the dynamic part of the page is then dealt with elsewhere (for example, a Servlet). The web server is then able to generate the

page from the HTML tags and then insert the generated responses from the programmed section into the page where the tags indicate.

Servlets were mentioned in the previous paragraph. They are often combined with JSP to create solutions for dynamic web content, and online front ends to applications and databases. The first reason is that JSP was never intended for complicated logic and doesn't have the API to allow accesses to databases (for example). The second is an implementation one, as JSP pages are actually compiled into Servlets by the web server, and then this allows for connection pooling and other application niceties. Servlets are a Java alternative to CGI programming. They run on web servers and act as middleware; converting browser (client) requests to databases and applications running within a business (server-side), and then converting the results into a presentable form to be sent back to the browser. Servlets need to be capable of reading data sent by the user (if any), looking up information that forms part of the HTTP request such as cookie data or hostnames, generating the result (database access, RMI/CORBA calls, legacy application connections, or directly), formatting the result for the browser, setting the appropriate HTTP response parameters (via headers) and then finally sending the document back [6]. Theoretically Servlets can be embedded in any type of server, as long as the FTP (for example) headers and responses are formulated correctly. In reality industry take-up has been for Servlets combined with JSP and HTML to work on web servers.

Just to muddy the waters there are JSP Scriptlets. These can be used when there is a need to do something that is more complex than simple expression results being inserted into the HTML, but not complex enough to warrant the effort required to implement a full Servlet solution. An example of this sort of use is the conditional include of JSP/HTML code.

ColdFusion

ColdFusion has been added to this section for completeness. Unlike the above technologies which can be integrated with several servers should the appropriate module be available, ColdFusion is a complete web application server [9]. The product is targeted at scalable e-commerce and enterprise web based applications.

The product (as well as the server) provides:

- Rapid development through visual tools and tag-based programming
- Connectivity to databases, Java code, legacy systems, and other such enterprise applications
- A high performance multi-threaded architecture that is scalable through such techniques as load balancing
- Clean integration with network and web server security

These features distinguish ColdFusion from the other technologies presented. There are tools available to help with PSP, JSP, and especially ASP (from Microsoft) but neither of these three are sold as web application servers. They are utilities that can be integrated into other servers. ColdFusion is the same in the respect that it utilises a mark-up language (CFML; ColdFusion Markup Language) that integrates with HTML for the interface aspects.

2.3 Summary

The above sections, as with the following one, do not claim to give a complete overview of all of the technologies presented. Nor are they capable of providing comparisons and the advantages and disadvantages of each. They have been introduced to make clear the areas in which software development has moved. This immediately should become an issue for maintainers, as once a system has been rolled out, then it becomes subject to the same problems of maintenance and evolution that have been known to affect software for years. The difference here is the nature of the development and deployment process, and it is a difference that software maintenance has to address.

3. Server-Side Component Architectures

This section covers initiatives from two of the major players of the computing industry; Microsoft and Sun. These have been included because of their prominence on the marketplace, and they relate to some of the technologies introduced in Section 2. An overview of the aims, and the major component parts of these initiatives will be presented to show both the overall similarity and also to make clear any differences. It is evident from this sort of arrangement how industry sees it should be moving. This ought to be an indication to the maintenance arena that these should then become areas that require maintenance, and that appropriate tools and techniques need to be researched to deal with this.

The underlying theme behind these ideas is that they are ways of providing suitable enterprise solutions to the problem of server-side components. They are essentially addressing the whole of the middleware theories. It has been seen as useful to do this because of the move towards a minimum of three tiers in client-server solutions for much of the business software produced today. Be creating schemes such as .NET and J2EE (.NET is from Microsoft, and J2EE is the Java 2 Enterprise Edition from Sun) these companies look to provide coherent frameworks that address everything necessary and hopefully tie customers in to using at least some of their products. The solution described by both is a server-side component architecture.

Component architectures require tools for developing components, a container to manage the deployed components, and obviously tools for deploying and maintaining components. A well defined component architecture supplies the necessary standards for organisations to then be able to write such tools, and this then allows component developers to be sure their work will be focused on the components and not the environment in which such components will be used.

3. 1 .NET

There are (according to Microsoft [1]) four main components of .NET, which are considered to be core to the whole ideal. These are:

- A set of building block services for (again a Microsoft wording) the Internet operating system, as well as for local services such as file handling.
- An infrastructure with accompanying tools for building a new generation of services.
- Device software for smart Internet devices
- .NET user experience

What may not be explicit from the above text is that Microsoft now hold the view that software in the future will be delivered as a service, and .NET is part of their delivering on the promise on enabling such software. An integral part of the .NET initiative in the immediate future has been the development of tools to allow applications to be built that utilise this new ideal. The COM specification has been updated to COM+, and ASP to ASP+ to reflect such changes. The Internet Information Server (IIS) provided by Microsoft web servers previously provided ASP services, DCOM, and transaction services. COM+ is a step forward in integrating some of these technologies and providing a framework in which the .NET ideal can be achieved.

The other side of the adoption of a .NET view of the world is that the user's conceptual view is different. Hardware and specific machines become less important, as their own data becomes more pervasive and accessible from desktops, laptops, or phones, and integrated across applications. This in theory provides a user's data for them in a format they choose at that time (or is appropriate for the medium being used), and then allows them to use it how they wish. The ideal of service provision rather than products is encompassed in this view because computers, devices and services work together in different flexible configurations to give a richer solution than is currently possible.

3.2 J2EE

The Java 2 Platform, Enterprise Edition, is generally known as J2EE and as with most Java technology names has been trademarked by Sun [11]. In short J2EE can be considered to be a collection of enterprise technologies. In full there are many parts to the J2EE specification; the first distinction being that it is split into two higher-level concepts; architecture and services. The J2EE architecture consists of:

- Application components
- Containers
- Resource manager drivers
- Database

The services provided by J2EE are

- HTTP
- HTTPS
- Java Transaction API (JTA)
- RMI-IIOP
- JavaIDL
- JDBC
- Java Message Service (JMS)
- Java Naming and Directory Interface (JNDI)
- JavaMail
- JaveBeans Activation Framework (JAF)
- Java API for XML Parsing (JAXP)
- J2EE Connector Architecture
- Java Authentication and Authorisation Service (JAAS)

To cover all of these in detail would require several books, but suffice to say that JSP and Servlets covered in Section 2 are Application Components, as are the Enterprise JavaBeans (EJB) components that could be considered to be similar to COM objects within the Microsoft technologies. Many of the services are provided to allow for ease of integration and interoperability with other systems.

An integral part of using J2EE from a development point of view, assuming other vendors have produced the supporting middleware services according to Sun's specification, is to utilise Enterprise JavaBeans and Servlets to create distributed enterprise applications. Enterprise JavaBeans, allow enterprise-class distributed applications to be built in Java and deployed without the need for a complex distributed object framework. This therefore enables rapid server-side development with "built-in" scalability, reliability, and security [7]. Enterprise JavaBeans are coarser grained than JavaBeans. The latter can be used to build entire applications, or even just be assembled into larger components. Essentially these are development components, rather than the deployable components created with EJBs. EJBs can also be composed, but each has to be deployable in its own right.

4. Maintenance Implications

Should any distributed systems researcher, or indeed, many currently working in industry, read the above text in the context of this paper they would wonder why it was necessary. These technologies are already used to deliver real applications and systems and are therefore old news in that field. Since they are being utilised in industry then it is necessary that solutions and issues are considered and implemented from a maintenance and evolution perspective.

XML is a recent technology that has been quietly incorporated into .NET and J2EE, and that in itself requires support and maintenance. Since XML can provide both the specification of what is valid data (through a DTD) and the data itself in one file, the maintenance of the utilisation of that data becomes an issue.

The ideal of these component-oriented systems is that since they are generally multi-tier architectures maintenance should be easier. The basic n-tier system (where n is three or greater) separates concerns by having at least a presentation layer, a business-logic/application layer, and a data layer. This means that replacement of components at these levels should be easier, and that through interfaces, implementation issues should not impact so widely. Again it would be wise to consider the other maintenance issues that such systems may cause, such as the overhead in transactions and messaging passing and the problems in tracing errors through these links. There is no silver bullet solution in software engineering [12], and to think that enterprise computing through distributed components will solve all legacy issues would be naive. Object orientation has provided us with the ability to develop new application styles and implement software in different ways, indeed it is central to the current changes, but it didn't take over as was promised in it's early years. It may have solved some problems but caused plenty of others, such as dealing with comprehension with dynamic binding of methods.

Another issue surrounds the use of existing legacy applications. The improvements in component technologies and the middleware allow for these to be integrated into the newer web-enabled systems. They can be leveraged into such systems through the use of connector technologies as supplied by hardware and software vendors such as IBM. This does not solve the problems of errors within the old systems, and certainly doesn't address all of the connectivity issues but goes along way towards allowing systems into which there has been significant investment to be retained.

Transactions, security, traceability, for example, are all features of an online society in which failure along those lines results in a massive backlash. Allowing bank details to be viewed if they are online and only implementing security after someone notices will not and should not be tolerated. The same goes for any enterprise application. Some of these issues are at a development and testing level, but many therefore have follow-on effects into maintenance. What happens when the commerce model has to change, or that flaws are found?

Another issue is the length of time that a system exists for. Through the middleware concept and servers systems will exist for a long time, but actual instantiations of aspects of that system (for example dynamic web pages written in JSP or ASP) may only exist for seconds. There is also then a secondary issue of JSP and ASP pages actually existing (as long as their contents does not change) for as long as the host server is running, but providing an outward appearance of being generated for each request.

The provision of services rather than products means that components will be gathered together, utilised, and possibly disbanded, although the components themselves will still exist to be re-utilised in other combinations. Where does the maintenance have to be directed as a component needs to evolve? The communication may never be replicated, and even if it were there is no provision for it ever being reused in exactly that combination again.

5. Conclusions

Once a piece of software or a system has been developed it is deployed. Once something has been deployed then it will at some point in the future (although the concept of time is somewhat varying with the Internet and modern software) require maintenance. Thus anything that is in development at the moment, or has recently been deployed, such as the many web aware enterprise distributed heterogeneous component systems, is a target for maintenance in the very near future. The current trends in development and deployment need to be addressed from a maintenance perspective if maintenance and evolution are still to remain viable.

One must consider the view that with the ease with which component parts of systems (at varying levels of granularity) can be replaced without any other part of the system having to know about that change surely has an impact on the reuse and maintenance ideas of the past. The rapid response to change required in today's market place dictates that the quicker a system change can be made the more profitable the organisation. Development tools are so much more sophisticated, and many have been tailored to work with this new concept of software and service provision. All of these factors then point to rewriting a component rather than repairing it. The maintenance community needs to address these issues to see where component maintenance can be of use, and where a component rewrite to benefit system maintenance would be better.

This paper has provided an overview of the newer technologies being used for application development and raised many issues for the maintenance community to think about. The academic maintenance community especially, must therefore change its mindset to bring it into line with the reality that is happening in industry in this Internet era. This has to be reflected in the research being carried out and thus in the publications in conferences and journals.

Much more detail on any of the information presented here can be found online and in some books that are now starting to appear. There is, however, enough information presented here to show that the playing field is changing and that in order to remain effective maintenance must also change.

Acknowledgements

This work is financed by an EPSRC ROPA grant: VVSRE; **V**isualising **S**oftware in **V**irtual **R**eality **E**nvironments.

References

[1] http://www.microsoft.com/net/

[2] http://www.service-oriented.com/

[3] Bereton, P., Budgen, D., Bennett, K. H., Munro, M., Layzell, P. J., *The Future of Software: Defining the Research Agenda*, Communications of the ACM, Vol.42, No.12, December 1999

[4] Layzell, P. J., Bennett, K. H., Budgen, D., Bereton, P., Macauley, L. A., Munro, M., *Service-Based Software: The Future for Flexible Software*, Asia-Pacific Software Engineering Conference, 5-8 December 2000.

[5] The Common Object Request Broker: Architecture and Specification, Version 2.0, http://www.omg.org/

[6] Hall, M., *Core Servlets and JavaServer Pages*, Prentice Hall, 2000.

[7] Roman, E., *Mastering Enterprise JavaBeans and the Java 2 Platform Enterprise Edition*, Wiley, 1999.

[8] http://www.php.net/

[9] http://www.allaire.com/Products/ColdFusion/

[10] http://www.microsoft.com/com/

[11] http://www.javasoft.com/j2ee/

[12] Brooks, F. P., *No Silver Bullet*, IEEE Computer, pp10-19, April 1987.