MSc Information Systems

Component Based Development : Methodology Proposal

Methodology Proposal

Abstract

Enterprise architecture is undergoing considerable change with recent developments in client server technologies and middleware. Each in turn has a significant impact on the way systems are designed, and a more component-based approach to development is beginning to emerge.

While we now have the Unified Modelling Language as universal notation for design modelling, there is currently no consistent standard for the definition of components. A pragmatic architecture for application development is needed that delivers business benefit without the need for significant investment in tools and training. It should minimise risk and maximise return on investment by leveraging investment in legacy systems, and provide the means to more closely relate business requirements to each phase of the development process.

This paper suggests that a better way of controlling technology is by adopting a service based approach to design and development, concentrating on pragmatic techniques and models that add value through reuse within a sound architectural framework. Given a set of business requirements, the focus is on business oriented component modelling techniques (e.g. process models, use cases, service allocation), and the delivery of a complete component design specification (e.g service definitions, service package architecture). Unusually, this does not involve the definition of a domain class model, but rather a definition of implementation independent, and therefore reusable, services (or contracts) that component packages will deliver. The component package is regarded as a 'black box' from which components will be designed and built by specialists in the technology of the component domain. This approach also provides the means for legacy and packaged applications to be reused in the same way.

The methodology was evaluated by a peer group of six senior IT professionals from the insurance and IT services sectors, who together represent over 110 years of industry experience. The methodology was presented in the form of a case study and questionnaire, and from the feedback it was concluded that there was merit in the approach. Reservations over how it would scale to larger systems have been addressed by the agreed need for a suitable repository for the documentation of data and business rules, and the need to separate the definition of technical non-functional requirements.

Author: Paul Botel

Table of Contents

Table of Illustrations

i. Dedication

I dedicate this work to my dear departed Parents, and particularly to my Mother who saw the start of this research, but sadly did not live to see its completion.

ii. Acknowledgements

I would first like to offer my thanks to my tutor Garfield Southall, Deputy Head of University College Chester's Department of Computer Science and Information Systems, for his advice and guidance throughout the course of this project.

I would also like to thank my target audience for taking the time to study my research proposal and complete the associated questionnaire, namely: Len Hall, Patricia Johnson, Michael Lowcock, Martin Morris, Neil Peachey and Steve Walley. I especially appreciated the timely return and quality of their feedback, given that they are all senior IT professionals with project responsibilities and schedules to maintain. This is a reflection on the caliber of the individuals involved, their degree of interest in this field of work, and their willingness to support my research objectives.

Without the support of these individuals, I would have been unable to complete this research. Their help, encouragement and support has been deeply appreciated.

1. INTRODUCTION

1.1 Background

Enterprise information systems have undergone considerable change with recent developments in client server technologies, middleware, and development languages. Each in turn has had a significant impact on the way systems are architected and designed, and while most recent software has been developed according to the object-oriented paradigm, a more component-based approach to development has begun to emerge.

The Butler group described the component as: "… a black box accessed only via interfaces, and has only superficial similarities with an object. Developing an object-oriented application requires high skill levels, and can deliver a monolithic, object-oriented application. We advise our clients to deliver software applications as components and then the internals and design approach can be object-oriented or conventional" (Goodwin, 1998).

If we accept this statement on its merits, it would appear that the definition of the black-box interface is all-important. The method of implementation is then a separate choice, and this is by no means simple. If an Information Technology (IT) department made the wrong decision, this could have a significant impact on the profitability of the business.

IBM (1998) observed that from a business perspective the technology never seems to be there when it is needed, and software frequently lags behind business needs. They posed a number of questions including:

- "How do you keep up?"
- "How do you make the right guess about what you will need before you need it?"
- "How do you preserve the viability of software investments you have already made?"

Again, if we accept this statement on its merits, trying to predict which technology to invest in can be a bit of a gamble. The key statement here concerns the preservation of existing investments in information technology.

Taking the key elements of component based development, black-box service definition, advances in technology and architecture, and the need to protect legacy investment, this led me to consider how best I could contribute to the debate with meaningful applied research.

1.2 Components and Information Systems

In order to determine the scope of my research, my preliminary investigations were to determine the role that components play in the development of information systems, and in order to carry out that role, what processes needed to be there to support their development. The resulted in the relationships profiled in Figure 1-1. It can quickly be seen from this diagram that component-based development is central to the development of information systems, and are influenced by technology and architecture, and require support from appropriate methodology.

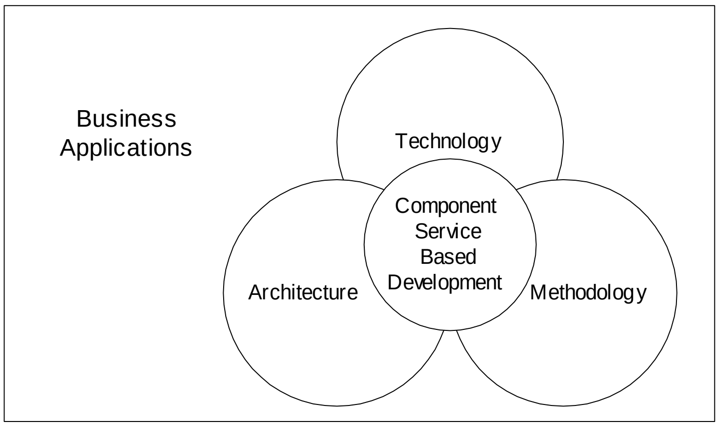


Figure 1-1 : The Relationship of Components in Information Systems

Further investigations identified that while the design and construction of components such as Java applets and ActiveX controls for internet Web-enabled applications was increasingly understood, there had been little progress in terms of the methodology needed to define the black-box component interfaces, and more importantly manage the whole process. This led me to focus on the requirements for an appropriate framework and methodology within which the component interfaces could be designed as reusable services for development as components.

1.3 Scope and Literature Review

In order to determine the requirements for an appropriate methodology to support component-based development, I first had to understand the influences of current technology and systems architecture on component build and deployment, and review current methodology and techniques in order to establish a firm foundation for my proposals. I undertook this task within the Literature Search of Section-2.

I first looked at current technical architectures and developments in client-server, and the middleware components and languages required to support distributed applications in Section-2.1. This was necessary in order to establish the requirements for the definition of service layers and their distribution, and recommend appropriate techniques in the methodology approach.

I followed this with a review of the current use of distributed technology including internet/intranet and smartcard type developments in Section-2.2. This was required in order to establish the sort of integration between old and new technologies that might be required, and that the methodology would need to satisfy.

I then reviewed the current state of systems development methodology in Section 2-3. This was necessary in order to establish why object-oriented methods were seen to be at too low a level of granularity to deliver the promise of application re-use, and to establish the advantages of the component approach based on the provision of system services as contracts between the user and the system. I then established how component techniques were grounded in earlier approaches and that the Universal Modelling Language could satisfy the modelling requirements. I also reviewed the process frameworks used by several of the leading object-oriented methodologists to determine a suitable framework for the component-based development approach.

I concluded the Literature Survey with an analysis of system architecture in Section-2.4 in order to determine why system architecture is important to the understanding and development of application systems. This also established the architectural requirements that the methodology had to satisfy in order that an optimal environment for component management and reuse could be achieved.

Supplementary notes on technology are provided in the appendices and include: Technology and Application (Appendix-1), Client-Server (Appendix-2), and Industry Experience with Distributed Objects (Appendix-3). A Glossary is also provided to further describe some of the terms introduced in this section.

1.4 Original Material and Methodology Proposal

The conclusion from the Literature Review was that while we now have the Unified Modelling Language as universal notation for design modelling, there is currently no consistent standard for the definition of components.

I am proposing, therefore, that a better way of controlling technology can be achieved by adopting a service based approach to design and development, which concentrates on pragmatic techniques and models that add value through reuse within a sound architectural framework.

While the proposed framework includes all phases of the systems development lifecycle, the starting point for the proposed techniques is a set of business requirements. How they are obtained and documented is outside of my scope. Likewise how the components are actually built and deployed is also outside my scope.

My focus is on business oriented component modelling techniques e.g. process models, use cases, service allocation, and the delivery of a complete component design specification e.g. service definitions and service package architecture. From an object-oriented viewpoint my approach is unusual in that I do not advocate the definition of a domain class model, but rather the definition of implementation independent, and therefore reusable, services (or contracts) that component packages will deliver. The component package is the ‘black box’ from which components will be designed and built by specialists in the technology of the component domain. This approach also provides the means for legacy and packaged applications to be reused as component packages in the same way.

In terms of project team structure, this approach also provides more flexibility. It enables business oriented non-technical analysts to define system services, reserving potentially more scarce technical specialists for the work of technical component design and build. However, that is not to say the team should be structured this way, merely to illustrate greater role flexibility, and potentially better use of resources in this respect.

The methodology should deliver business benefit without the need for significant investment in tools and training. It is designed to minimise risk and maximise return on investment by leveraging investment in legacy systems. Since the use case is the focal point for analysts, designers and developers, this approach also provides the means to more closely relate business requirements to each phase of the development process.

My methodology proposal is defined in Section-3. I have used the Objectory project phases of Inception, Elaboration, Construction and Transition to establish a framework within which to define the processes and techniques of the methodology. I have also defined a simple case study based on the registration and payment of an insurance claim. This is used to illustrate the application of the techniques throughout the proposal.

1.5 Discussion, Evaluation and Conclusions

1.5.1 Discussion and Evaluation

In order to test the feasibility of the methodology I chose to present my arguments in the form of a number of claims, and design a questionnaire in order to obtain appropriate feedback for evaluation. The questionnaire and covering note explaining the reason for the research (included in Appendix-4) was issued to my target audience together with a copy of my methodology proposal (Section-3).

The methodology proposal is a comprehensive document and introduces numerous techniques with which many IT practitioners may not be familiar. Since I am presenting the methodology as an architecture for the development process, I wanted to target the proposal on a small number of senior IT staff who could take a more holistic view of the component development process, and its implications both to IT and the business. In order to obtain this quality feedback, I selected a small peer group of six experienced and senior IT professionals, representing a stratified sample from the insurance and financial services sectors. I refer to this group as my Panel, which collectively represents over 110 years of industry experience with over 58 years at a senior and management level. The Panel are introduced in Section-4.0, together with my rationale for the design of the questionnaire.

Following the return of all completed questionnaires I consolidated the results into one document as shown in Appendix-5. I removed the identity of the respondents in order to present and evaluate the feedback collectively, and not at a personal level. My analysis of their feedback and evaluation of the results follows from Section-4.1.

1.5.2 Conclusions

The conclusions on the research are documented in Section-5.

I first summarise in Section-5.2 how the methodology relates to the issues raised in the Literature Review of Section-2. I then review the suitability of the Objectory framework in Section-5.3, before presenting a summary of the main findings on the suitability of the proposed methodology in Section-5.4. This is based on a summary of the key elements from my evaluation in Section-4.

I then present an updated process model in Section-5.5, which incorporates updates to address several of the concerns raised by the Panel. I conclude with suggestions for future research on this topic in Section-5.6.

I finish this introduction with an overall roadmap for the project as illustrated in Figure 1-2 below.

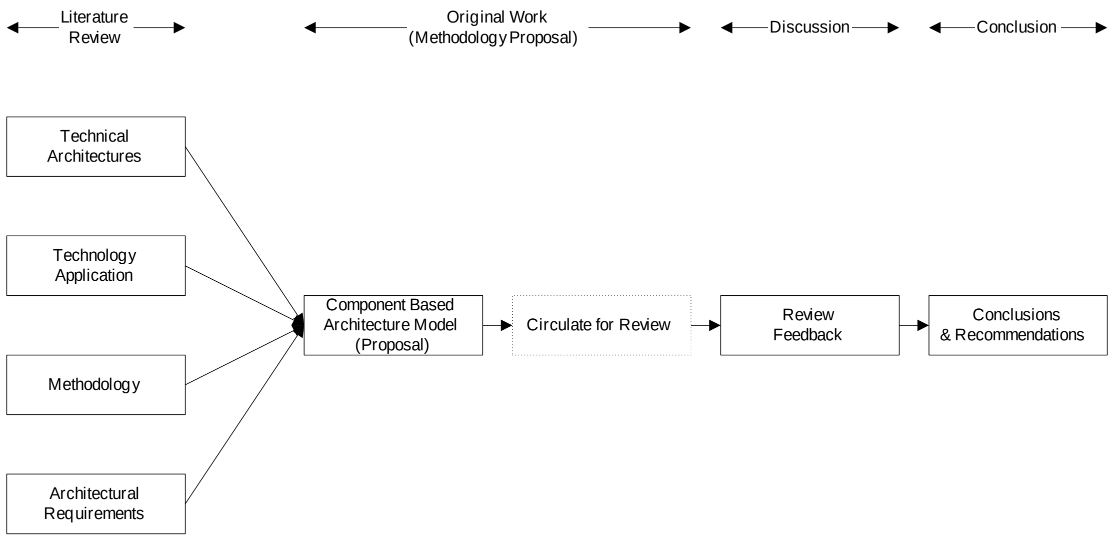

Literature Review
Original Work (Methodology Proposal)
Discussion
Conclusion
Technical Architectures
Technology Application
Methodology
Architectural Requirements
Component Based Architecture Model (Proposal)
Circulate for Review
Review Feedback
Conclusions & Recommendations

Figure 1-2 Research Roadmap

2. LITERATURE SURVEY

2.1 TECHNICAL ARCHITECTURES

2.1.1 Introduction

The purpose of this section is to review the evolution of technical architectures in the distributed environment. Each evolution defines a new architecture within which systems have to be built. Such technology advances can bring significant business benefits and allow companies to re-engineer their business processes e.g. electronic commerce over the Internet. However, a legacy of business applications is always left behind, and has to be managed as the next wave of new opportunities and technologies are explored.

This technology review will establish the complexity of this area, and the importance of seeking a stable enterprise architectural framework within which emerging technical architectures can be managed. The general structure of this review is illustrated in Figure 2-1 below.

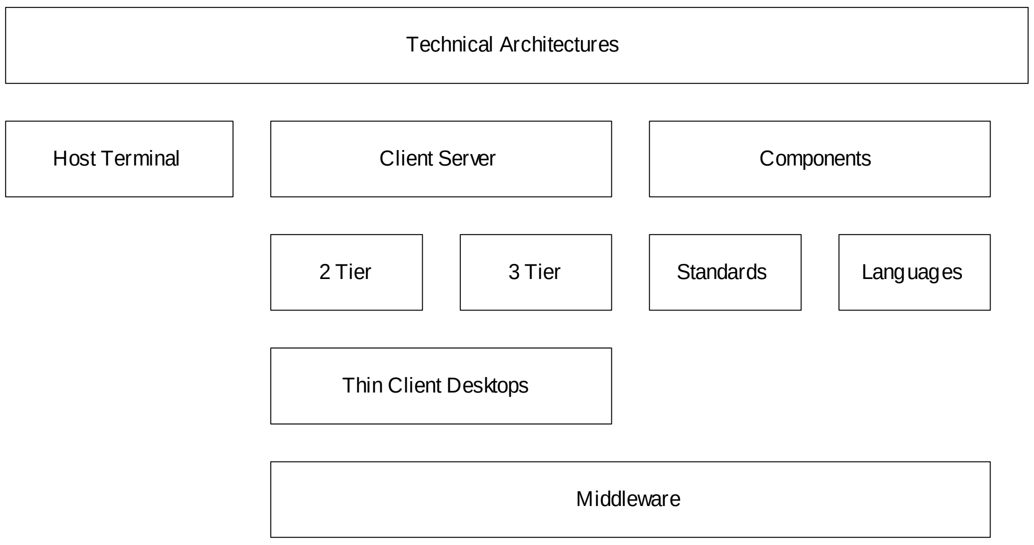


Figure 2-1 : Technical Architecture Survey

I will first introduce the host terminal based architectures common to the earlier mainframe and minicomputer based technologies, before moving on to the significant developments in the area of client server systems, the architectural layers, and the different types of client desktop. I will then examine the development of middleware technologies which are so fundamental to the evolution of distributed systems, before progressing to the current component based technologies, and the standards and languages applicable to them.

2.1.2 Terminal Based Architectures

Traditional mainframe and minicomputer architectures relied on dumb terminals using emulation software to access programs on the host machine, from which all transaction and data processing was performed. Applications were designed on a functional basis, using the approaches common to structured analysis and design, and information engineering techniques (Gane & Sarson (1979), Martin & Finkelstein (1981), Jackson (1983), Rock-Evans(1992)).

A major constraint of the host based system was that the user interface, business and data manipulation logic tended to be mixed throughout the host based code (Allen & Frost, 1998). This resulted in systems that became difficult to use, hard to change, and inflexible given the advances in desk-top computing, storage technologies, databases, and network communications.

2.1.3 Client Server Architectures and the Distributed Systems Model

2.1.3.1 Client Server Architecture

In order to manage the technical complexity of large-scale systems, a number of layered architectures have emerged which separate the application model into discrete tiers. I will reserve my discussion in this paper to the more popular 2-tier and 3-tier layered architectures (Allen & Frost (1998), Bleasdale (1998), Pressman (1997)), although 4-tier and 5-tier have been defined (Mowbray & Malveau, 1997).

The 2-tier model can be referred to as being ‘user-driven’. An event driven GUI puts control in the user’s hands, and the GUI can integrate data from a server with desktop productivity tools like spreadsheets etc. It also enables some reuse of business rules and processes.

The major problem with this approach is where you put the business logic : on the client or on the server? Only business logic on the server is reusable across several clients (e.g. database stored procedures). However, if the logic is embedded in a particular database (e.g. Sybase, Oracle, DB2 etc) then you limit what is otherwise general business logic to that particular database server.

With the 3-tier model, there is clear separation or layering, between the presentation logic, business logic and database logic. This provides a closer match between the architecture of the business and that of the technology. However, this is an architecture primarily driven by the technology, and implies a strict client-server interaction between software layers. These logical layers can then be applied across the network consisting of Services, Communications Middleware, and Client(s) as shown in Bleasdale’s (1998) illustration in Figure 2-2, and further explored in Appendix 2.

Many new technologies are beginning to emerge and co-exist with older technologies, and such integration is essential given the legacy of hierarchical and relational databases, transaction processing systems and applications, which are at the heart of most commercial organisations.

2.1.3.2 Alternatives to the PC - New types of desktop

The client element of the architecture has been traditionally provided by the Personal Computer. However, Gartner (1997) estimate the total cost of ownership for a PC within a business organisation at $10,000 per annum per user, including the costs of procurement, maintenance and upgrade.

As a result, several cheaper alternatives to the PC known as ‘thin-clients’ have emerged where processing power can be more closely matched to the requirements of the user. As Gartner (1997) also observed: “It is almost as inappropriate to provide a worker with too much functionality as not enough. Functionality adds complexity, complexity adds cost”.

The Ovum Report “Network Computers: Risks and Rewards for Business, 1998” identified four types of ‘thin client’ : the Network Computer, NetPC, Intranet Terminal and Windows Terminal (Griffiths, 1998). They all run across a network, depend on a server and differ in functionality - see Appendix 2 for a more detailed description.

Ovum concluded that Windows terminal and NetPCs will dominate over the next 5 years because they serve Windows users, whereas intranet terminals and network computers only designed for Java will achieve: “only niche success”. This is also due to the lack of Java applications compared to 10,000 Windows applications (Griffiths,1998). Gartner confirm this view that a hybrid model will continue to prevail: some applications being better for fat clients, others for thin (Computing, 1998b).

Just to complicate the issue even further, and possibly negate one of the reasons for a PC alternative in the first place, “Emachines”, a Korean PC manufacturer in a joint venture between KDS and Trigem, has just announced plans to produce PCs for under £300, designed to be “…NetPC and network computer killers” (Clarke, 1998b). This reduction will undercut network computer prices and allow users to continue with their Windows based applications, rather than having to adopt the NC’s Java operating system.

One thing that is for certain, Gartner confirm that network computing will be a major part of an increasingly complex enterprise architecture, requiring middleware to tie the differing systems together, with message brokers becoming increasingly popular by 2001 (Computing, 1998b).

2.1.3.3 Middleware and Component Architecture in the Distributed Environment

Middleware is the software layer that lies between the application and the underlying complexities of the network, and provides the developer with an easy to use API in order to access the resources that the application needs (Linthicum, 1997). This provides transparent access not just to database access operations, but to other systems and services by using a naming service, and invoking a remote process which then returns a response to the calling process. This is shown in Figure 2-2 below.

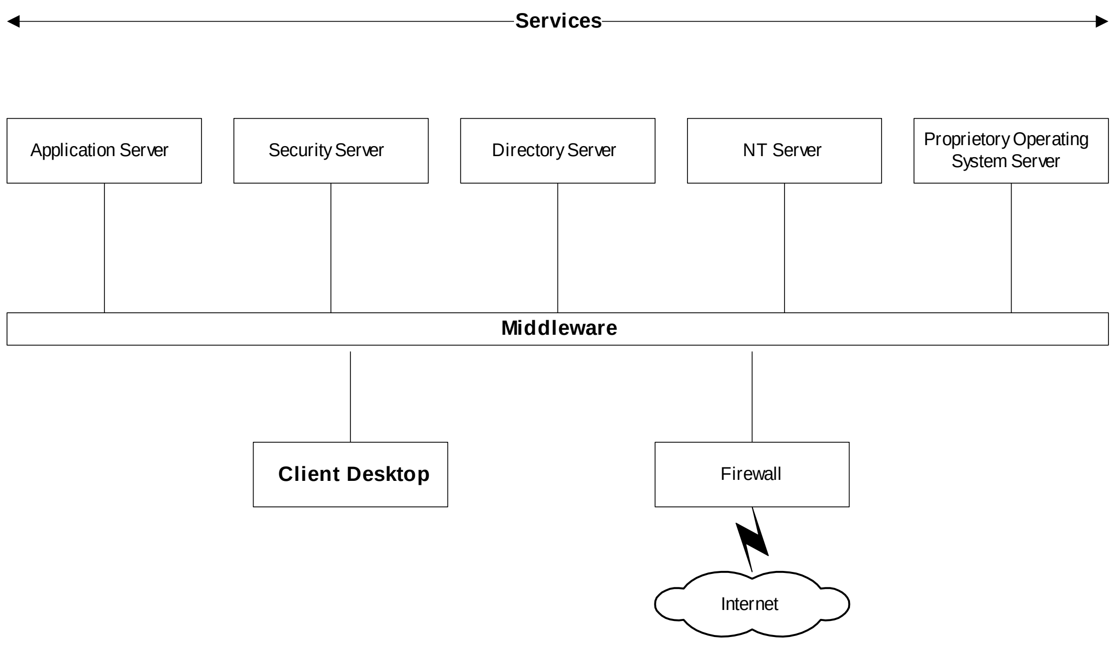


Figure 2-2 : Middleware Perspective

The development of middleware and especially the emergence of Microsoft's DCOM, and the Object Management Group's CORBA industry standard models for object communication, have had a dramatic effect on the approach to component distribution. There is no longer a single address space, and an application can span different technical configurations, operating systems, and across multi-language environments (Allen & Frost, 1998), (Anderson, 1998), (Pressman, 1997).

To enable this interoperability between components across a network, it is the component's interface that is defined to the middleware. Language neutral interface standards are used to do this, with Microsoft using their proprietary vtable standard, and OMG the open IDL standard.

The CORBA framework is an example of a good open architecture which has enabled suppliers to compete without jeopardizing the compatibility of components e.g. VisiBroker from Visgenic Software, Orbix from Iona Technologies. (Morris & Ferguson,1993), unlike DCOM which is the proprietary reserve of Microsoft.

Which is the right choice? Butler Group chairman Martin Butler has branded CORBA the "Betamax of object technology" claiming that business would adopt DCOM since it is free with Windows NT (Kelly,1998d). However, research by Neil Ward-Dutton, senior analyst at Ovum indicated that while the installed base of NT servers (3.1million) will outnumber Unix (0.5million) by six times in 2002, revenues for DCOM/COM on NT will be under $1m, against those of Java and CORBA of $2.5m. In their defence, Richard Soley chairman of OMG suggested that: "DCOM will be increasingly used as people move to NT, but existing legacy systems are not going to be replaced. Diversity is here to stay in operating systems and CORBA is the only bridge". I have noted recent industry experience with this middleware in Appendix 3.

The key issues here are whether you are committed to the Microsoft NT route, or require a more open approach, with greater options for the integration of legacy systems, and choice of operating system. The choice may be governed by what components are available. DCOM has evolved from the desktop and has only recently emerged as a model for distributed applications, whereas CORBA and Java were originally designed for network use. The facilities they provide, however, are fundamental to the ability to deploy components at execution time in order to optimise the technology to deliver the most business benefit. A key example is the Internet which applies the concept on a global scale.

2.1.4 Component Development Languages

While the DCOM and CORBA technologies provide for clear component interfaces, separate from implementation and therefore isolating change, we now need to consider how components are actually implemented, and how they relate to the middleware.

The two dominant component technologies are Microsoft's ActiveX - machine dependent binary for Windows applications, and Sun Microsystems Java - machine independent byte compiled code (Allen & Frost(1998), Anderson(1998)). Using this technology, families of components can be developed based on a published interface without knowledge of their internal details. Such components can range from user interface icons and controls to complete software products. How the components attach to the middleware is shown in Figure 2-3 below.

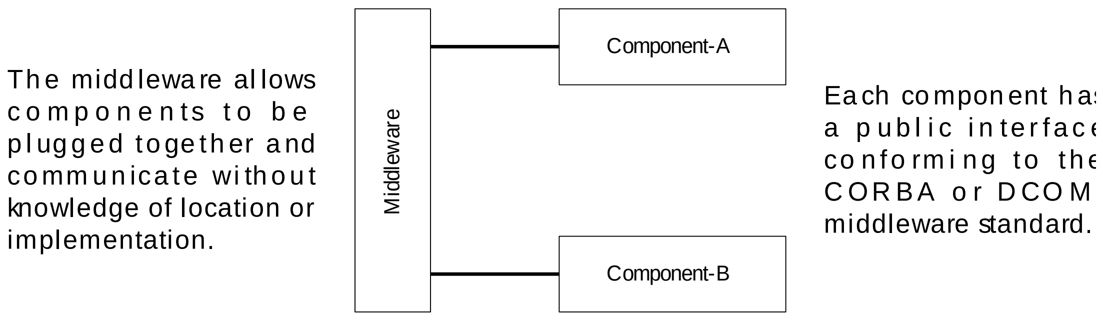

The middleware allows components to be plugged together and communicate without knowledge of location or implementation.
Middleware
Component-A
Component-B
Each component ha a public interfac conforming to th CORBA or DCOM middleware standard.

Figure 2-3 : Component Connectivity

Java has become tremendously popular as a development language based on its 'write once, run anywhere' philosophy. Research by International Data Corporation revealed an average payback for Java developments of 14.6 months (Kelly, 1998a). This was endorsed by ObjectShare who experienced a two to four times cost reduction over similar development in C++. It is also being adopted for desktop and server strategy by manufacturers e.g. Hewlett Packard (McGinn, 1998).

JavaBeans is the component architecture for the Java platform, and most Java development tools now use JavaBean components. Enterprise JavaBeans is a newer specification that runs on the server in a distributed application (Anderson,1998). While Java has its own RMI technology for communication between distributed objects, Sun will standardise on CORBA's IIOP (Anderson, 1998), which clearly shows Sun Microsystems commitment to this industry standard.

Being a new technology some problems are still in evidence. When the "Liffe" automated brokerage service was implemented for the London Futures and International Petroleum Exchanges, it was discovered that the Internet Explorer and Netscape Browsers used different versions of the Java Virtual Machine, and that the download of an application applet could take 10 minutes (Kelly, 1998b).

Poor performance was also identified by a Bloor Research survey of 100 UK sites across industry sectors, together with a lack of support for platforms, and insufficient development tools (Kelly, 1998c). On the other hand its strengths included ease of deployment, built in security, and the portability of the application. More significantly, Bloor identified that 9% of respondents reported doing something strategic or significant with Java (estimated 41% by 2002), and 79% were seriously looking at the language.

This was in contrast to the Butler Group whose research showed that less than 5% of corporations were using Java, and there were no cases of large successful server based systems (Goodwin, 1998). Clearly, there is a conflict here, and the research criteria would have to be examined to determine the discrepancy. While the problems with performance of server based systems are reflected by the Liffe example, this could be due to many factors and development in network technology will minimise these problems.

Regardless of choice, across a network Java Beans and ActiveX are memory hungry component architectures and demand considerable processing power. Providing the network performance can be managed, the Java approach is characterised as thin client, fat server with a wider network bandwidth requirement. Microsoft's approach is to load more features onto a fat client, with ActiveX managing the interworking of Microsoft applications both locally and over a distributed network of servers (Pal, Ring, & Downes, 1996). While this will find favour with Microsoft sites and should deliver better local client performance and integration with other Microsoft products, it carries the increasing costs already identified for a PC desktop solution. I have included further information on this topic in Appendix 1.

2.1.5 Conclusions on Technical Architectures

The technology is now available to deploy objects where they are needed, and not where traditional host based solutions would otherwise have dictated. This has resulted from the development of the CORBA open component based middleware, and Microsoft's proprietary DCOM middleware, together with client server and distributed network technologies. Together, these technologies enable the interoperability between components. There may still be some performance problems to overcome e.g. Java download times, but as Butler Group(1998b) observe, this should not be an excuse to ignore component based development.

2.2 TECHNOLOGY APPLICATION

2.2.1 Introduction

Having now reviewed the increasing complexity of technology, and identified where we are in terms of client-server, middleware, components and component development languages, I will now illustrate how that technology is currently being applied.

IBM (1998) describe electronic commerce (or e-commerce) as systems designed and developed using internet technology and access methods, to allow companies to conduct business both internally using intranets or externally via the Web. In terms of the next steps with Java technology, Bloor Research predicted that: “The next wave of computing will involve networks and distributed processors linked into the web. These will use Java to manage machinery, buildings, vehicles and the home” (Farrell, 1998a). However, I will limit my review to the world of electronic commerce, following the approach shown in Figure 2-4 below.

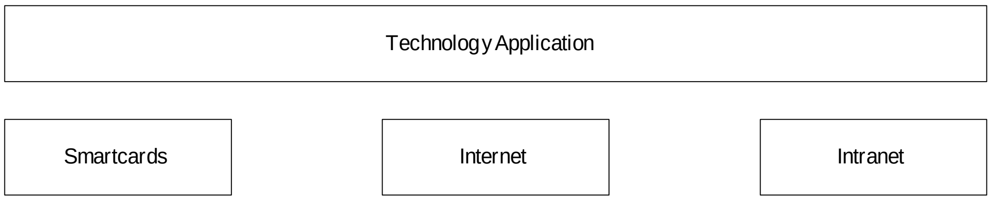


Figure 2-4 : Technology Application Survey

I will first explore the application of Java in smartcard applications, and then the impact of the internet and intranets on business practices, opportunities for growth and new ways of doing business in the global environment.

This will again highlight the requirement for a stable architectural framework in order to manage application development from a business perspective, while now *applying* these new technologies. I will then illustrate in Section 2.3 what methodologies that have been used to develop these type of systems, their limitations, management issues and the requirements for an architectural framework to better manage the development process.

2.2.2 Smartcards

Where Java has been notably successful is in the development of smartcards. Smartcard technology has been identified by International Data Corporation as one of the fastest growing and dynamic branches of the IT industry in Western Europe today, with 308 million microprocessor based cards expected to be sold in 1998 (Tober,1998).

Mark Stevenson, independent consultant and co-author of an Ovum report on smart cards, predicted that: “By 2003 there will be around five billion smart cards in use – as many cards as there are people on earth” (Tober, 1998). Indeed the number and scale of smartcard developments seems to bear this out. Visa Cash have 60,000 cards in circulation in Leeds (Tober, 1998). London Transport anticipate six million contactless smart cards in circulation (Clarke, 1998a), Co-operative Bank is working on an ISA system (Tober,1998), and Aberdeen Council will soon have 40,000 cards in circulation (Phillips,1998). Take up will no doubt accelerate further when Microsoft release their equivalent development technology in 1999 (Saran, 1998). I have provided more background information on these topics in Appendix 1.

2.2.3 Internet and Intranet Developments

Access to the internet is provided by a Web browser, which is also a good example of a component container. Netscape and Microsoft's Internet Explorer both support Java, and IE supports ActiveX. (Anderson, 1998). As a result, applications are increasingly being developed with the client component running in a Web browser, with middle tier components running on servers, supported by back end data sources of various kinds (Anderson, 1998).

Bloor indicated that electronic commerce via the Internet was growing at the rate of 8% per month, and as a result predicted that banking, retailing and insurance will undergo dramatic change (Farrell, 1998b). One of the drivers for this is that web based business costs are about one tenth of those of traditional business methods. In the United States Bloor indicated that 20% of all stock market transactions were web-based, but that people were less happy to buy and sell insurance over the internet. Given the level of competition in the banking, retail and insurance sectors, Bloor predicted dramatic change in this area. This would result in the retail and insurance sectors seeing levels of competition exceeding anything experienced before, forcing rationalization on a world wide scale.

Gartner (1998) predicted that by Year 2003, 80 percent of enterprises will be using the Internet for telecommuting, accessing mobile users and communicating with trading partners (0.8 probability). He also noted that due to focusing resources on Year2000 development, that most organisations will have failed to invest in Internet technologies, and will be forced to use systems integrators throughout 2001 to overcome the shortage of skills and lack of internal expertise (0.8 probability).

Examples of successful Internet developments in the financial sector include Equitable Life with an investment tracking system for 500,000 clients (McLeod, 1998), and Lloyds TSB with an online service to allow Lloyds customers to pay bills, transfer money between accounts, view statements and adjust standing orders over the Net (Poston, 1998). These developments are detailed in Appendix 1.

Intranets have increasingly grown as the means to simplify document publication and distribution within an organisation. While they can reduce costs and improve information flow, they are not viewed as mission critical (Pal, Ring, & Downes, 1996). They provide added value, rather than major business benefits.

2.2.4 Conclusions on Technology Application

This new technology has been applied under several guises including electronic commerce (or e-commerce), and includes the emergence of smartcards, and the Internet.

Over the Internet, applications are being developed for deployment on a global scale with networks and distributed processors linked into the Web. Intranets are also being developed for internal use within organisations, and use the same Internet technology and access methods. Most of the ‘thin’ client processing in this environment is performed using a simple to use Web Browser which is a good example of a component container.

The emerging challenge is to integrate these applications into the core business processes of the organisation. Indeed, as Web technology offers more capabilities, its complexity will increase – as will the job of managing the infrastructure (Pal, Ring, & Downes, 1996). This again identifies the need for a comprehensive software component architecture within which the distributed functionality of Web clients and servers can be managed, and integrated with other applications. Further information on Java and Microsoft’s ActiveX components is provided in Appendix 1.

2.3 METHODOLOGY

2.3.1 Introduction

Having now reviewed the technology and its application we now need to examine the parallel progress of methodology, and I have used the steps shown in Figure 2-5 below.

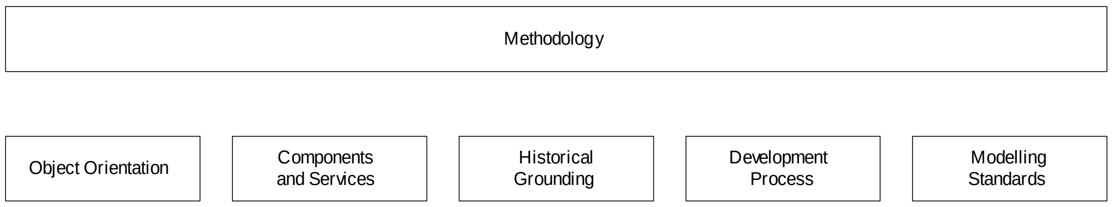


Figure 2-5 : Methodology Survey

Jayaratna (1994) defines Methodology as: “… an explicit way of structuring one’s thinking and actions”, and this will be the driving force behind my proposals in Section 3. He goes on to say that: “Methodologies contain model(s) and reflect particular perspectives of ‘reality’ based on a set of philosophical paradigms.”, and it is the paradigms of object orientation, and components that I will now examine.

I will first discuss how the object-oriented approach to development has failed to deliver the promises of application reuse, and then introduce how the component, as a higher level of abstraction than the object, will achieve this. I will then discuss how this service based component approach is grounded in earlier work. I follow this with a review of the latest thinking on the organisation of the development process from the perspective of project management, and review recent developments in object-oriented modelling standards that can be used to support the component approach.

2.3.2 Object Orientation

Many claims have been made for the benefits of object orientation. Objects are meant to reflect the real world in software. Allen & Frost (1998) observed that they are meant to: "… provide the reuse, clarity and adaptability that are lacking in client-server applications and the interoperability required by distributed technology". However, they also identified that implementation dependencies can often be exposed through the programming language interfaces. It is recommended that technology such as CORBA's IDL is used to define component interfaces (which are separate to the implementation), and thereby provide the necessary isolation from change.

Mowbray & Malveau (1997) also identified problems in a lack of distinction between interface and implementation in object oriented analysis and design, suggesting that most of the modelling relates to the characteristics of the implementation and not to its interface. This can result in a lack of reuse outside that particular environment, since objects are not treated as service providers but as parts of applications that often fail to address the wider needs of the business. In practice: "…the application-based mind-set is indeed "alive and kicking" in many organizations today" (Allen & Frost, 1998).

While reuse has often been heralded as another major benefit of object orientation, a recent survey at Object World UK revealed that only 5% of respondents were managing to reuse more than half their code (Goodwin, 1998).

It is evident, therefore, that while object oriented approaches can be effective in the construction of software, they can be misused. Indeed, how object oriented a design is does not guarantee the quality of the deliverable. As Mowbray & Malveau (1997) suggest: "…there needs to be something more sophisticated than 'objects are things you can touch', in order to design effective OO systems."

New paradigms, therefore, need to take account of the difference between interface and implementation. It is this separation of implementation from interface that will guarantee reuse. Providing the interface is well defined and specified, it can be reused to build components specific to whatever target environments are required e.g. C++ or Java. This separation is also the basis for the architectural boundaries, which is a feature of both the CORBA and DCOM distributed component technologies.

2.3.3 Objects and Components

Allen & Frost (1998) draw the distinction between objects and components as follows:
"Although it is objects that form the underlying fabric of software solutions, it is components that provide the effective granularity of reuse through consistent published interfaces that encapsulate the implementation…A component-based approach removes the tight coupling of an application's parts and eases reuse of parts across different applications. In fact the application has disappeared: instead model items are packaged as components, which are configured in network fashion to meet business needs."

Butler Group (Goodwin, 1998) similarly referred to the component as: "… a black box accessed only via interfaces, and has only superficial similarities with an object. Developing an object oriented application requires high skill levels, and can deliver a monolithic, object-oriented application. We advise our clients to deliver software applications as components and then the internals and design approach can be object-oriented or conventional".

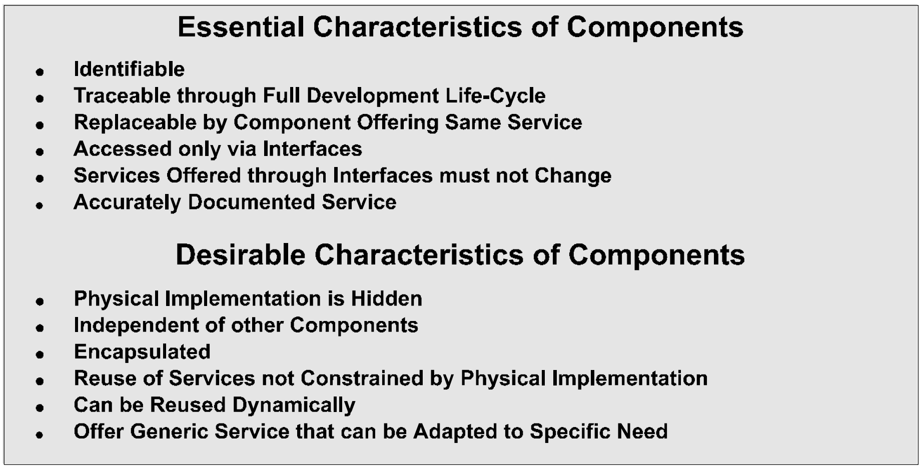
This has the advantage that legacy code and third party software can be wrapped up and treated as a component, as long as they have a standard interface with the outside world.

To summarise in the words of Allen & Frost (1998): "A component is an executable unit of code that provides physical black-box encapsulation of related services. Its services can only be accessed through a consistent, published interface that includes an interaction standard. A component must be capable of being connected to other components (through a communications interface) to form a larger group.".

2.3.4 Components and Services

The focus on services is seen to be the key to providing reusable software components. Allen & Frost (1998) define service as: "…cohesive collections of related functionality, accessed through a consistent interface that encapsulated the implementation. Each service has a published specification of interface and behaviour. Components provide the interfaces."

Services offered through interfaces should not change, and such interfaces are often referred to as being 'immutable' (Butler Group, 1998a). While the physical implementation of the component might change, the service offered via an interface should not, i.e. the *how* can change but not the *what*. The essential and desirable characteristics of components as described by Butler Group (1998a) are illustrated in Figure 2-6 below.


Essential Characteristics of Components

- Identifiable
- Traceable through Full Development Life-Cycle
- Replaceable by Component Offering Same Service
- Accessed only via Interfaces
- Services Offered through Interfaces must not Change
- Accurately Documented Service

Desirable Characteristics of Components

- Physical Implementation is Hidden
- Independent of other Components
- Encapsulated
- Reuse of Services not Constrained by Physical Implementation
- Can be Reused Dynamically
- Offer Generic Service that can be Adapted to Specific Need

Figure 2-6 : Characteristics of Components - Source: Butler Group (1998a)

The impact of the component based approach has been researched by Ovum in their report "Componentware: Building It, Buying It, Selling It" (Mannion & Keepence, 1998). They suggest that the software component approach will make a “silent coup”, with dramatic impact on both user companies and software suppliers, and see sales of component products reaching $3bn in 1998, taking off to $64bn in 2002. Ovum does not see the component approach immediately becoming an alternative to traditional development for IT departments, but as a new technology for in-house development. They see most companies gradually adopting componentware: “…as a packaging, re-facing and integration technology rather than as an application development technique”.

2.3.5 Historical Grounding

An analysis of Wirfs-Brock, Wilkerson, & Wiener (1990), Coad and Yourdon (1991), and Jacobson, Christerson, Jonsson, & Overgaard (1992) has identified that the concepts of the component, and the service based approach, are firmly grounded in their work.

Wirfs-Brock, Wilkerson, & Wiener (1990) introduced the concept of a subsystem as: “…a set of classes (and possibly other subsystems) collaborating to fulfill a set of responsibilities.” They affirmed that although subsystems do not exist as the software executes, they are useful conceptual entities treated as black boxes by other subsystems, and are another example of encapsulation. The interface of the component was described as a “contract” supported by the subsystem.

Coad & Yourdon (1991) also introduced two key concepts. Firstly that a service should carry out one, and only one function. Secondly, the concept of “packaging” as a means for the designer to organise a design so that it is resilient to change, the objective being for the package to remain stable over time.

Jacobson, Christerson, Jonsson, & Overgaard (1992) also used the subsystem description as: “… a way of structuring the system for further development and maintenance…At the lowest level we call these units service packages”. This was to be performed on the basis of objects with a strong mutual functional coupling being placed in the same subsystem. Reference was also made to the work of Embley & Woodfield (1988), and Yourdon and Constantine (1979).

We can see, therefore, that the service based approach has been used as a means of abstracting system detail into manageable components for the purpose of application design and packaging. However, the focus here was on the packing of application objects, and not on the provision of a service based architecture. It is in the context of an architecture that the component and service based approach is now being applied.

2.3.6 The Development Process

Many approaches to object oriented systems development have emerged in recent years, the most popular including those of:

- Shlaer and Mellor (1988)
- Wirfs-Brock, Wilkerson and Wiener (1990) – Responsibility Driven Design
- Coad and Yourdon (1991) - Object-Oriented Design

- Jacobson, Christerson, Jonsson, and Overgaard (1992) - Object Oriented Software Engineering (OOSE).
- Rumbaugh, Blaha, Premerlani and Eddy (1991) - Object Modelling Technique (OMT)"
- Martin and Odell (1994)
- Booch (1994) – Rational

All assume objects to be stable over the lifetime of the system, but each differ in the proportion of time, and the degree of importance of the activities of analysis, design and implementation (Mowbray and Malveau, 1997), and in the notation of system models (Fowler and Kendall, 1997).

The most recent developments have appeared since the three famous methodologists: Grady Booch, Jim Rumbaugh and Ivar Jacobson, joined forces at Rational Software. The initial most significant event from this merger was publication of their Unified Modelling Language (UML). This has since been adopted as an industry standard and Version 1.1 was released by the Object Management Group in November 1997. This is discussed further in Section 2.3.7.

They are currently, however, working on a unified process which will be known as the 'Objectory Software Development Process', and undoubtedly builds on the work of Jacobson (1992)'s 'Objectory' process. Like Fowler and Kendall (1997), I do not believe that you can be prescriptive and standardise on one development process, because the nature of information systems are inherently different e.g. real time, management information, GUI, together with their scale e.g. single user or enterprise wide. The benefit of the Objectory process is that it provides a framework for development, yet allows developers to have the flexibility to use different techniques appropriate to their particular project.

The process is iterative and incremental, and allows activities to span phases recognising that problem solving is a non-linear activity (Cantor, 1998). The process as described by Fowler and Kendall (1997), and Cantor (1998), begins with an Inception phase where the business rationale and scope of the project is agreed, and commitment is gained from the sponsor. This is followed by the Elaboration phase where detailed requirements are obtained, high-level analysis and design is undertaken, the baseline architecture established and the Construction plan developed. Construction builds the system in a series of iterations, each consisting of the analysis, design, coding, testing and integration of the increment. The final phase is Transition, where beta testing, performance tuning, and user training is conducted prior to the rollout of the system.

It is this framework that I will use when I define my methodology proposal in Section-3.

2.3.7 Modelling Standards

Fowler and Kendall (1997) define a modelling language as: "…the (mainly graphical) notation that methods use to express designs", and each of the object oriented analysis and design methods of the 1980s and 1990s prescribed their own.

Unlike a methodology, the UML does not define a process. As the name suggests it is a 'standard' modelling language for which Butler Group(1998a) describe the benefits as:

- providing a common mechanism for communication;
- a basis for tool interoperability;
- reduced training needs;
- increased opportunity for standard patterns and frameworks to become accepted.

From the component based perspective, Butler Group(1998a) saw the UML:

- enabling improved communication between a component supplier and the consumer.
- facilitating reuse.
- allowing component catalogues to use a single notation to describe their contents.
- enabling the exchange of data between modelling tools.

Their survey results shown in Figure 2-7 below, also confirms a rapid and widespread take-up of the UML which more or less guarantees it achieving its objective as a universal standard.

Figure 2-7 : UML Adoption - Source: Butler Group (1998a)

How rigidly people will apply the standard notation is, however, debatable. There are many more methods and techniques which embellish the UML for various reasons. It is enhanced in the business process modelling of Select's Perspective (Allen & Frost, 1998), and in various forms in the Catalysis approach of D'Souza and Wills (1998) reflecting the methods of their respective companies: Icon Computing and Trireme International Limited.

Butler Group(1998a) also indicate that database design is another area of weakness, and more relevant here, that the UML currently treats components only as a means of packaging objects for deployment. However, where the UML must be applied is when a CASE tool is being used, and particularly if it is to generate code.

The principle models and techniques covered by the UML (Fowler and Kendall, 1997) and Cantor (1998) include: use cases, class diagrams, interaction diagrams, package diagrams, state diagrams, activity diagrams, and deployment diagrams. I will later introduce those applicable to my work in this dissertation, (subject to the constraints of my modelling tools), and indicate where and why I have deviated from the standard. It is also worth highlighting the Allen and Frost (1998) guideline to keep modelling simple: “A modelling notation must add value to the overall process”.

2.3.8 Conclusions on Methodology

In terms of development methodology it has been identified that object-oriented approaches often fail to deliver the promises of application reuse. This is seen to be the result of a lack of distinction between interface and implementation throughout the process of object-oriented analysis and design. It is the component, or component sub-assembly, that is seen as increasingly more useful, reusable, and marketable (Butler Group, 1998a).

This higher level of abstraction is seen to be achieved by service definition, which is clearly defined by Allen & Frost (1998) as: “…cohesive collections of related functionality, accessed through a consistent interface that encapsulated the implementation. Each service has a published specification of interface and behaviour. Components provide the interfaces.” It is this ‘black-box’ encapsulation of services that will enable software to be defined as a component, for which the internals and design approach could be object-oriented or conventional. This will provide system architects and designers with a truly flexible approach to development.

The available process and modelling approaches are developed from the disciplines of object orientation. The majority of the required modelling constructs are, however, provided by UML Version 1.1, and the ‘Rational Objectory Process’ provides an appropriate development lifecycle framework.

The architecture and design of component-based systems is an emerging discipline.

2.4 ARCHITECTURAL REQUIREMENTS

2.4.1 Introduction

Having now examined the technical architecture, the application of technology, and methods used in application design, I want to examine the attributes required of enterprise application architecture, following the approach shown in Figure 2-6 below.

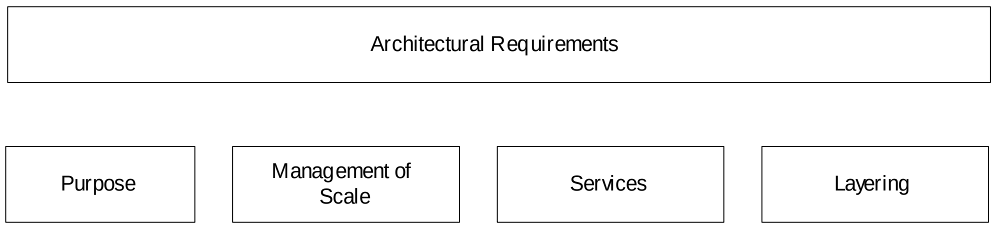


Figure 2-8 : Architectural Requirements Survey

I will first review the basic purpose for architecture, before looking at the various levels or scales of architecture that apply in order to construct an information system. I will then look at system architecture from a service based perspective, before examining how the system can be partitioned or layered for better understanding.

2.4.2 Purpose : Why do we need an architecture?

An architecture is needed to follow on from the object-oriented paradigm, and must take account of the essential distinction between interfaces and implementation. It must improve the management, development and deployment of application systems, and define the boundaries within which complexity and change can be more effectively managed.

This is crucial if IT Departments are to maximise the return on investment in information systems, which the business expects. The architecture must therefore allow for the integration of legacy systems (IBM, 1999). Likewise any enhancements made should cause as little disruption to the business as possible. Indeed, while the term legacy is normally associated with older systems with separation of data and function, the term can equally be applied to current object oriented systems that may need to be ‘wrapped up’ for reuse.

As Morris & Ferguson (1993) suggest: “Architectures impose order on the system and make the interconnections possible.” Architecture must protect the investment in existing systems, and be both scalable and extensible in order take advantage of new technologies and facilitate future development.

These principles equally apply to both systems and business organisations. A company needs an environment and architecture within which it can customise its own technology and systems in order to gain competitive advantage in the marketplace (Perspectives, 1993). For the re-engineering of business processes, it would appear that a less radical and more evolutionary approach is being taken.

Research by Currie & Willcocks (1995) showed empirically that the ‘all or nothing’ approach to business re-engineering and organisational change as promoted by Hammer (1990), often failed to achieve the claimed benefits. This was partly due to the lack of a suitable multi-disciplinary methodology, and lack of management of the political dimension (Willcocks & Smith, 1995). This has led to the conclusion that like many other large scale change programmes, they rarely produced change (Beer, Eisentat and Spector,1990). As a result, the outcome of a re-engineering project was frequently difficult to predict.

What this identifies is that the architecture must provide a clear description of the intent of its constituent parts in order to prevent the misinterpretation of a project’s objectives.

2.4.3 Management of Architectural Scale

Shaw & Garlan (1996) noted that as systems grow and become increasingly more complex, the structure of the overall system becomes more significant than the detailed definition of business algorithms and data structures. They suggest that the system is a composition of components including: global control structures, protocols for communication and data access, functional design elements, and their physical distribution.

Mowbray & Malveau (1997) propose a solution and structure for such architecture that can be defined according to its scale. They suggest that at every scale of a software system there is implementation and architecture. For example, an application needs an internal architecture to support its integration, reuse and evolution. Whereas implementation is the realization of some finer grained component modules, architecture is the abstraction defining the inter-module interfaces that help developers understand how the modules are put together. They see this as a key concept since few practitioners understand system level forces and how to resolve them effectively. The scalability model proposed by Mowbray & Malveau (1997) is shown in Figure 2-9, which also suggests the external ‘primal’ forces, or issues, dominant at each level.

Looking at implementation, this will be language dependent and reflected in the collection of software modules that provide functionality and performance. This is encapsulated by the interface, for which Mowbray & Malveau (1997) recommend the open CORBA IDL standard at the system level. This concurs with Morris & Ferguson’s (1993) view that open-systems and a well-designed and open-ended architecture permit evolution along with critical technologies. This serves to reduce complexity, enables the management of unpredictability and change, and allows individual technologies, components, or products to be changed or replaced with minimal impact on the rest of the system.

Scales		***Description***	***Primal forces (issues)***
7	Global Architecture	Design and system coordination issues across all organisations sharing communications and information. This includes languages, standards and policies across multiple enterprises e.g. Internet, Corba etc	Leveraging existing standards
6	Enterprise Architecture	Coordination and communication across multiple systems with multiple applications, within a single organisation. This may cover many locations with heterogeneous hardware and software systems This is where an organisation controls its resources and policies e.g. network standards, databases, common operating environment etc	Managing resources
5	System Architecture (OO Architecture)	Communication and coordination across applications, and the interoperation between applications. It is the enduring structure that survives the modification and replacement of component applications over the lifecycle of the system. Includes the impact of communication mechanisms and distributed processing. Manages complexity by the right amount of abstraction for a system architecture. Manages change by the definition of common interfaces i.e. services. Implements the systems internal model.	Managing change and complexity in order to reduce lifecycle costs.
4	Application Architecture (Subsystem)	Focuses on the organisation of application development to meet a set of user requirements. This involves numerous classes, multiple microarchitectures, and one or more frameworks. It satisfies end user requirements including user interface and visible system functionality.	Functionality and performance
3	Macroarchitecture (Frameworks)	The organisation and development of application frameworks. This involves one or more microarchitectures, and attempts to use large portions of design and software in a particular domain (e.g. IBM San Francisco)	Reuse of code and design
2	Microarchitecture (Design Patterns)	Development of software components to solve software problems. This will involve the combination of multiple objects, the design of which is typified by the work of Gamma (19nn) as recurring design patterns .	Object cooperation
1	Object	Development of reusable objects and classes (at the code, rather than design level). Defines the object attributes and signature of operations. This is very much language dependent.	Primitive functionality and reuse

Figure 2-9 : Architectural Scale - Source: Mowbray & Malveau (1997)

2.4.4 Service Based Architecture

In a distributed system a service-based architecture is needed to provide the independent reusable services necessitated by the lack of a single address space. By this method distributed objects are accessed purely through their interface, without any knowledge of the implementation details that support the interface (Mowbray & Malveau, 1997).

The benefit of a service based architecture is that the services are completely de-coupled from the clients that access them. This de-coupling allows services to be shared by multiple clients, which allows a service to be used and reused within the distributed environment. If the need arises, a service can be simply replaced by a new one with no ill effect on the system, providing the same interface is maintained. This is a significant step forward in enabling the migration to new products, applications and technologies with minimal systems impact.

2.4.5 Layered Architecture

Architecture should be flexible enough to enable the distribution of system functionality according to actual business requirements, rather than those of hardware and software suppliers. Allen & Frost (1998) define an architecture that partitions models into: user interface, business and data storage service layers. These are stereotypes, and as services can be collected together into cohesive packages (Fowler & Kendall, 1997) for which implementation object classes will later be designed. By layering the interfaces, a clear separation of services can be provided.

The interface User Services layer performs the activities of the user interface, including control and reporting functionality. User Services are implemented by User Objects.

The Business Service layer provides services to meet the needs of specific business processes or may be generic e.g. client services used by underwriters or claims handlers. It encapsulates business policy, and applies business rules to transform data into information. Business Services consists of Business Objects that can call User Objects as well as Data Services, and may alert User Objects where necessary. Business Services can also be used to wrap legacy assets and legacy transactions.
The Data Services via Data Objects provide data services across many Business Services. They translate persistent Business Objects into appropriate storage, including the wrapping of databases.

This architectural layering is used by Allen & Frost (1998) to partition application services into component packages for a particular business domain e.g. Claims, Accounts, Quotations, Underwriting. This is then subject to further object oriented analysis in order to allocate responsibilities to user, business and data objects for implementation as component packages, treated as black boxes, and accessed via defined interfaces to service classes within them. These packages of objects are then deployed onto the appropriate technology platform e.g. Client to an Enterprise Server, Quotations to a Direct Sales departmental server, Accounts to the Accounts Departmental Server etc.

This supports Mowbray & Malveau’s (1997) view that if the architect partitions (i.e. abstracts) the business domain into meaningful groups of related objects during object-oriented analysis, this should ensure the reusability of the software component. If specified correctly, this domain model grouping of components, their description and interaction should result in an ‘evolvable’ model.

2.4.6 Conclusions on Component Architecture

Mowbray & Malveau’s (1997) classification of architecture by scale is a significant contribution to the architectural debate. Likewise the layered service based approach (Allen & Frost, 1998) where components are treated as black boxes, accessible only by their interface.

Providing component interfaces are well defined within a suitable framework, it should be possible to trace the design back to the originating business requirements – a major key to reuse. If this can be achieved, system changes should be accommodated more easily, and it should be much simpler to identify components for reuse. Considerable savings can be made in this respect, and it has been estimated that the cost of reusing a component is one-fifth that of building it from scratch (Mannion & Keepence, 1998). The component-based approach can also be used irrespective of the technology. Butler Group (1998b) observed that some of the most successful implementations of componentisation that they had seen did not use any of the new distributed technology, simply traditional environments. Nothing technically, therefore, should inhibit progress towards a component-based design and development environment.

However, while the body of knowledge on object orientation is large, little has emerged to relate service-based components to application architecture, and the management of the development process.

3. ORIGINAL MATERIAL

3.1 FRAMEWORK INTRODUCTION

The purpose of this Section is to introduce my approach to service based component development, from the perspective of an overall architected design framework. In scope it covers the whole of the systems development lifecycle, but my intention is to focus on those elements specific to the approach (e.g. service definition), rather than the more traditional elements that are commonly understood (e.g. system testing).

In order to position the approach and its techniques, I have adopted the methodology of the Objectory Software Development Process, which defines the system development phases of Inception, Elaboration, Construction and Transition. Each phase will define a number of design and specification activities, which I will introduce by means of a framework road map.

I propose to explain the merits of this methodology by means of a simple insurance-based case study. The diagrams that I use will correspond to the Unified Modelling Language (UML) standard, albeit within the constraints of my Visio diagramming software. As the case study progresses, I will explain the meaning and purpose of the models used, together with a description of the notation relevant to their interpretation.

The overall methodology framework and roadmap structure is illustrated in Figure 3-1 below.

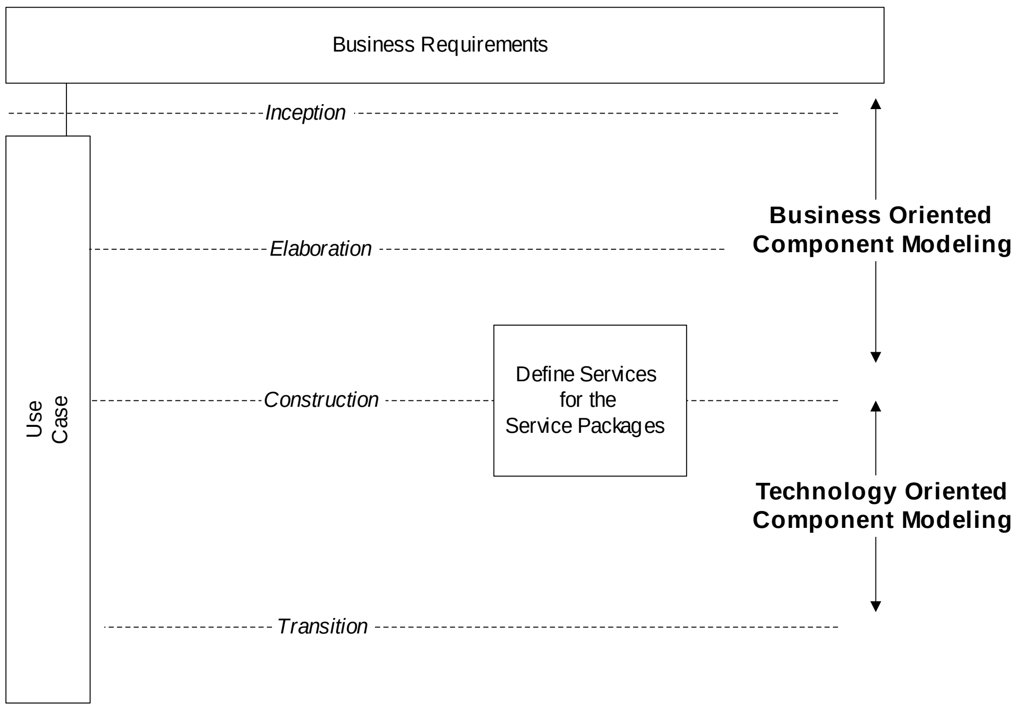

Business Requirements
Inception
Use Case
Elaboration
Business Oriented Component Modeling
Construction
Define Services for the Service Packages
Technology Oriented Component Modeling
Transition

Figure 3-1 : Methodology Framework

The key features that need to be highlighted are as follows:

a) The Inception and Elaboration phases are business oriented, while Construction is technology dependent.

b) The bridge between the business-oriented component modelling and the technology-oriented component modelling, is the activity described as “Define Services for Service Packages”. It is here that Service Specifications are defined in business terms at the end of the Elaboration phase, and become the main input to technical design in the Construction phase.

c) The use case is the repository for all business requirements documentation, and its definition is fundamental to the approach. Initially, Business Requirements are translated into Use Cases during the Inception phase. They then evolve during the Elaboration phase where more detail is progressively added, until they are fully defined. They then provide a central reference for the business requirements throughout the final Construction and Transition phases of development.

   Jacobson, Ericcson and Jacobson (1994) describe the Use Case as “a behaviourally related sequence of interactions performed by an actor in a dialogue with the system to provide some measurable value to the actor”. This is more simply stated by Fowler and Scott (1997) as: “ a typical interaction between a user and a computer system”, and this is the essence of the technique.

   The collection of Use Cases represents the external view of the system, and individually provide a useful means for the planning and control of iterative development.

The following Section 3.2 describes the Inception phase, and introduces the case study. This is then progressed through the Elaboration phase in Section 3.3. Only those elements of the Construction phase relevant to my methodology will be discussed in Section 3.4, and Transition, while essential to the development process, is largely outside of my focus and is included only for completeness in Section 3.5.

3.2 INCEPTION

3.3.1 Introduction

The roadmap for the Inception phase (Figure 3-2) shows the Business Requirements spanning all of the Inception activities (I have not drawn all the relationship lines in order to avoid clutter).

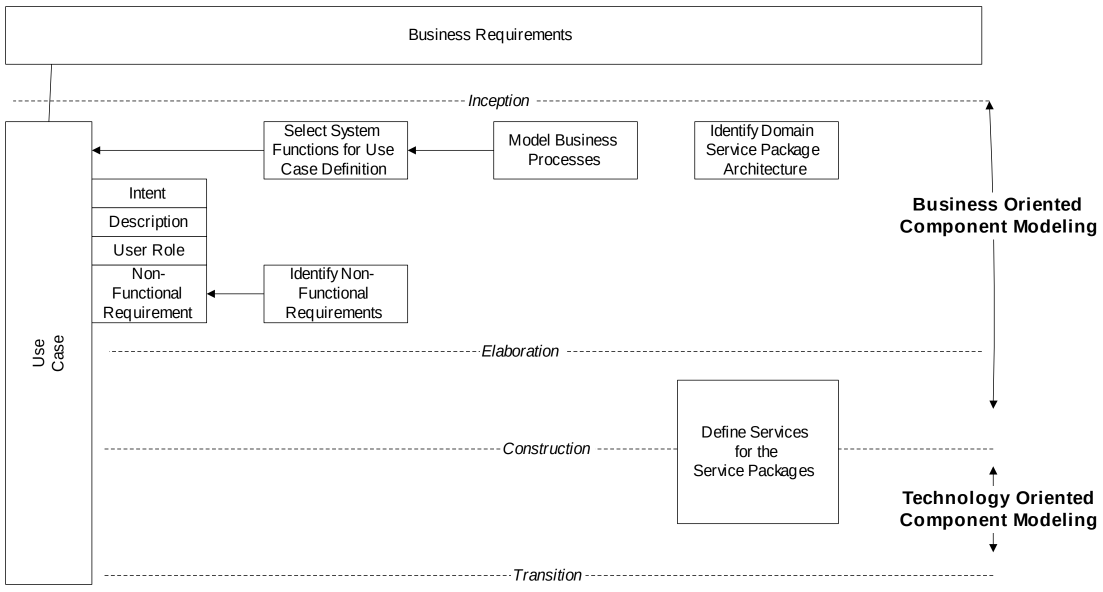

Business Requirements
Inception
Use Case
Select System Functions for Use Case Definition
Model Business Processes
Identify Domain Service Package Architecture
Intent
Description
User Role
Non-Functional Requirement
Identify Non-Functional Requirements
Business Oriented Component Modeling
Elaboration
Construction
Define Services for the Service Packages
Technology Oriented Component Modeling
Transition

Figure 3-2 : Inception Phase Activities

The overall objectives of this phase are to:

- translate Business Requirements (functional and non-functional) into Use Cases for development.
- identify the high-level Service Package Architecture for the domain

Use Cases can be identified either directly from the Business Requirements, or from a Business Process model, if one has been prepared (See Section 3.3.3). However, I see the modelling of the business processes and events as a key deliverable to aid the understanding of the process, and this should be recommended.

3.3.2 Business Requirements for the Insurance Claims Case Study

The business requirements for the case study can be simply stated:

‘A computerized system is required to enable the online data capture of insurance claims. Checks should be undertaken to ensure that the claim is valid, and when Authorized, made available for payment by the Accounts system.’
(Requirements Reference : Claims01).

It should be noted that this is an oversimplified example for the purpose of the case study. It requires no knowledge of insurance, and does not even consider the type of insurance product e.g. motor, household, corporate etc

3.3.3 Model Business Processes

I will begin with a model of the business processes (Figure 3-3). The model illustrates the user roles of Claim Handler and Claim Supervisor, and the activities they each carry out. These processes are illustrated in columns known as swimlanes, which help to highlight workflow.

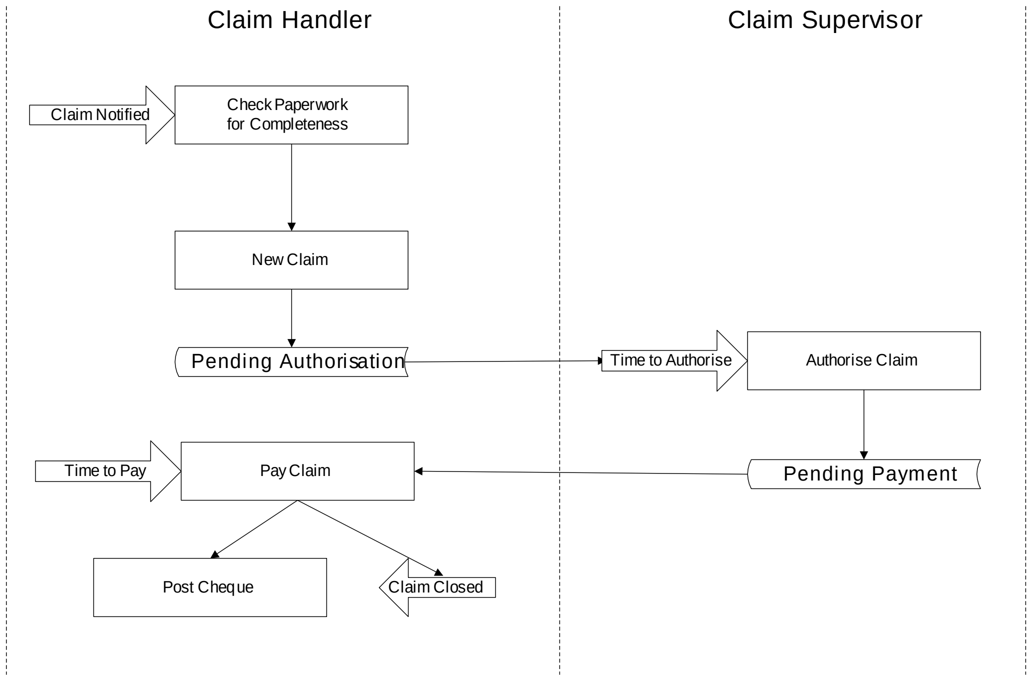


Figure 3-3 : Claim Business Process Model

In terms of the notation:

Feature	*Meaning*
rectangular box	a business process e.g. Pay Claim
right-hand arrow	an event initiating the process e.g. Claim Notified
left-hand arrow	an outcome of value from a process e.g. Claim Closed
narrow rectangle	a delay prior to another process e.g. Pending Authorization
vertical dotted lines	separate the processes into ‘swimlanes’, one for each user role

3.3.4 Select System Functions for Use Case Definition

Looking at the process model (Figure 3-3) it is clear that New Claim, Authorize Claim and Pay Claim are the processes to be computerized and in need of use case definition, while those of Checking Paperwork for Completion and Posting Cheque are not.

3.3.5 Identify Non-Functional Requirements

Non-functional requirements may relate to the technical environment (e.g. hardware platform, network constraints), or usability criteria such as response times, type of user: novice or expert etc. For the purposes of the case study, I have limited my description simply to a response time e.g. less than 10 seconds.

3.3.6 Use Case Definition

In general a use case is triggered by an external system event which assumes the existence of a human computer boundary. It is supported by one or more business and user services. The Use Case is defined in the Inception phase, and fully defined during Elaboration. The completed Use Case will record the information shown in Figure 3-4

Use Case Name	Use case name
User Role	The name of the role that the user is playing in this use case.
Intent	A sentence describing the overall intention of the use case.
Description	A paragraph describing the use case in more detail.
Detailed Steps	Detailed steps described in a structured English style.
Pre Conditions	The state of the system before execution of the use case. This corresponds to a process dependency where business process modelling has been used.
Post Conditions	The state of the system after execution of the use case. This corresponds to a process dependency where business process modelling has been used.
Non-functional Requirements	Operational constraints and quality criteria e.g. response time of less than 3 seconds is required
Business Rules	Business conditions governing the behaviour of the use case e.g. Claim is only valid during the period of Policy cover.
Alternative courses	Names of other related Use Cases
Sample screen layouts	Demonstrate function and not the user interface. Include sample user data
Requirements References	Cross reference to original requirements to ensure traceability

Figure 3-4 : Use Case Template

During the Inception phase, basic information about the Use Case is defined. This is what Jacobson, Ericcson, and Jacobson (1994) would describe as a requirements use case. I would expect to define at least the Use Case Name, the User Role and Intent, but would aim to have a Description of what it was meant to achieve and some idea of the Non-Functional requirements. A reference back to the original Business Requirement document is also made.

For the Inception phase of the case study, our use cases may appear as shown in Figures 3-5, 3-6 and 3-7 below.

Use Case Name	***New Claim***
User Role	Claim Handler
Intent	Register a new Claim against a Policy.
Description	On notification of a Claim, the system will check that the claimant is an insured Policyholder, and that Cover is available for the type of Claim. For a valid Claim, a new Claim will be registered on the system.
Non-functional Requirements	The system will confirm the creation of the new Claim within 10 seconds.
Requirements References	Claims01

Figure 3-5 : New Claim Use Case – Inception Phase

Use Case Name	***Authorize Claim***
Role	Claims Supervisor
Intent	Authorize the payment of a valid claim.
Description	After checking that the Authorizer has the required level of seniority to Authorize the Claim, the Claim will be Authorized for Payment.
Non-functional Requirements	The system will confirm Authorization within 3 seconds.
Requirements References	Claims01

Figure 3-6 : Authorize Claim Use Case – Inception Phase

Use Case Name	***Pay Claim***
User Role	Claim Handler
Intent	The Claim will be paid within a period of time according to company Best Practice (a non-computerized procedure).
Description	The Claim is Authorized for Payment and sent to the Accounts system for processing.
Non-functional Requirements	The system will confirm the payment request within 3 seconds.
Requirements References	Claims01

Figure 3-7 : Pay Claim Use Case – Inception Phase

3.3.6 Use Case Diagram

We can now illustrate the Use Cases for purposes of communication as shown in Figure 3-8. This diagram includes an additional use case called Update Claim, which could be identified after a more detailed analysis of the Authorize and Pay Claim Use Case. As each need to update the Claim and set a status, it makes sense to extract this common processing into a separate Use Case to be used whenever the Update Claim functionality is required. (Note: I have not further defined this Use Case for the case study.)

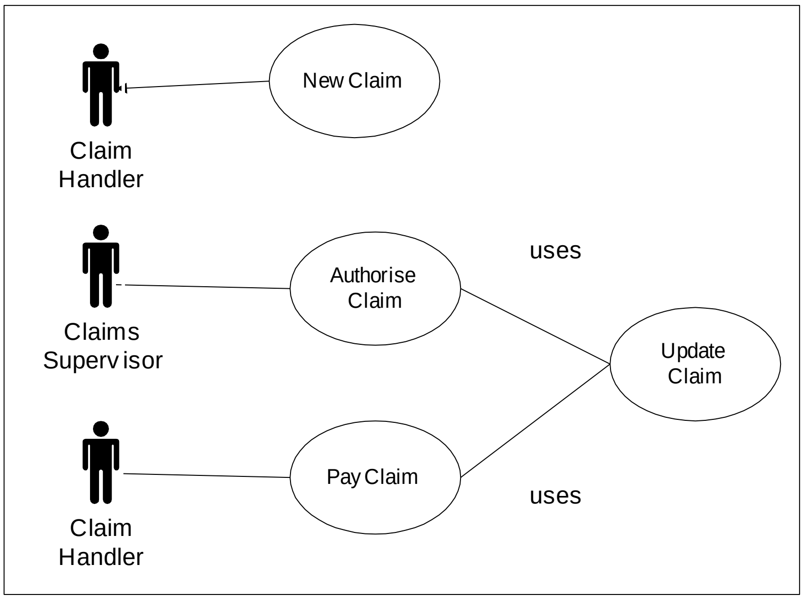


Figure 3-8 : Use Case Diagram

The notation used is as follows:

Feature	*Meaning*
ellipse	use case
person graphic	the role the user plays in relation to the use case e.g. a Claims Supervisor can also act in the role of Claim Handler. Note: With this technique the user is more formally referred to as an Actor, but this term is often alien to the user community!
uses	illustrates that Authorize Claim and Pay Claim both use the services of Update Claim

3.3.7 Identify Domain Service Package Architecture

One further architectural step now needs to be considered: the relationship of the Claim system with other information systems. The new Claims system is to be developed as a Service Package, and it is necessary, therefore, to consider its relationship with other Service Packages. This is illustrated in the form of a Domain Service Package Architecture Diagram (Figure 3-9). This uses the Package Diagram notation from the UML, which is a key tool for maintaining control over the structure of a system (Fowler and Scott, 1997).

To prepare this diagram, the catalogue of available component packages is reviewed and those likely to play a role in the Claims system are selected. In this case, the Policy, Administration Services and Accounts packages are all seen to play a part.

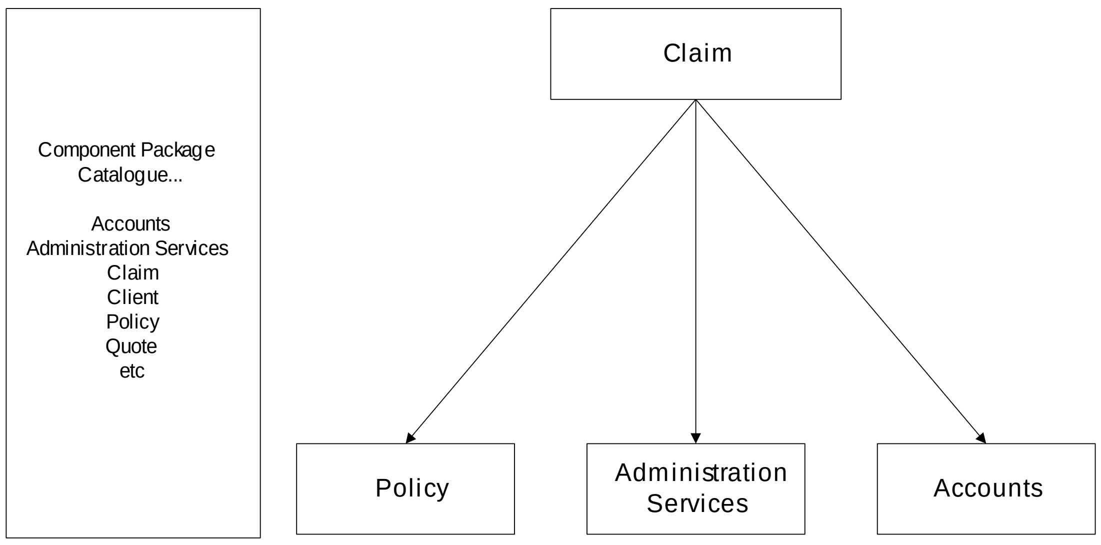


Figure 3-9 : Domain Service Package Architecture – High Level

In terms of the notation:

Feature	*Meaning*
rectangular box	component package
arrowed line	illustrates the relationship between the packages. The arrow on the target package indicates the reliance on the services of that package. In our case Claims is reliant on services from Policy, Administration Services and Accounts packages. If a service in any of the target packages is changed, this may have an impact on the Claims package, which will need to be checked.

3.3.8 Architectural components in place on completion of the Inception Phase

At this point the Inception phase is largely complete. We have begun the translation of Business Requirements into component terms, and from the project planning perspective have also identified three separate ‘chunks’ or sub-projects for development i.e. New Claim, Authorize Claim and Pay Claim. We have also established two modelling components within the development architecture: the Use Case model to which detail will now be progressively added, and the Domain Service Package Architecture that positions the Claims component in relation to others in the enterprise.

While the Service Package is a simple diagram, it provides a useful means of communication, being an abstraction of component details. The use case diagram also provides a simple yet meaningful view of the services to be delivered.

3.4 ELABORATION

3.4.1 Introduction

The roadmap for the Elaboration phase (Figure 3-10) builds on the Use Cases and Service Package architecture identified in the Inception phase, and defines the activities leading to the definition of System Services and their allocation to Service Packages.

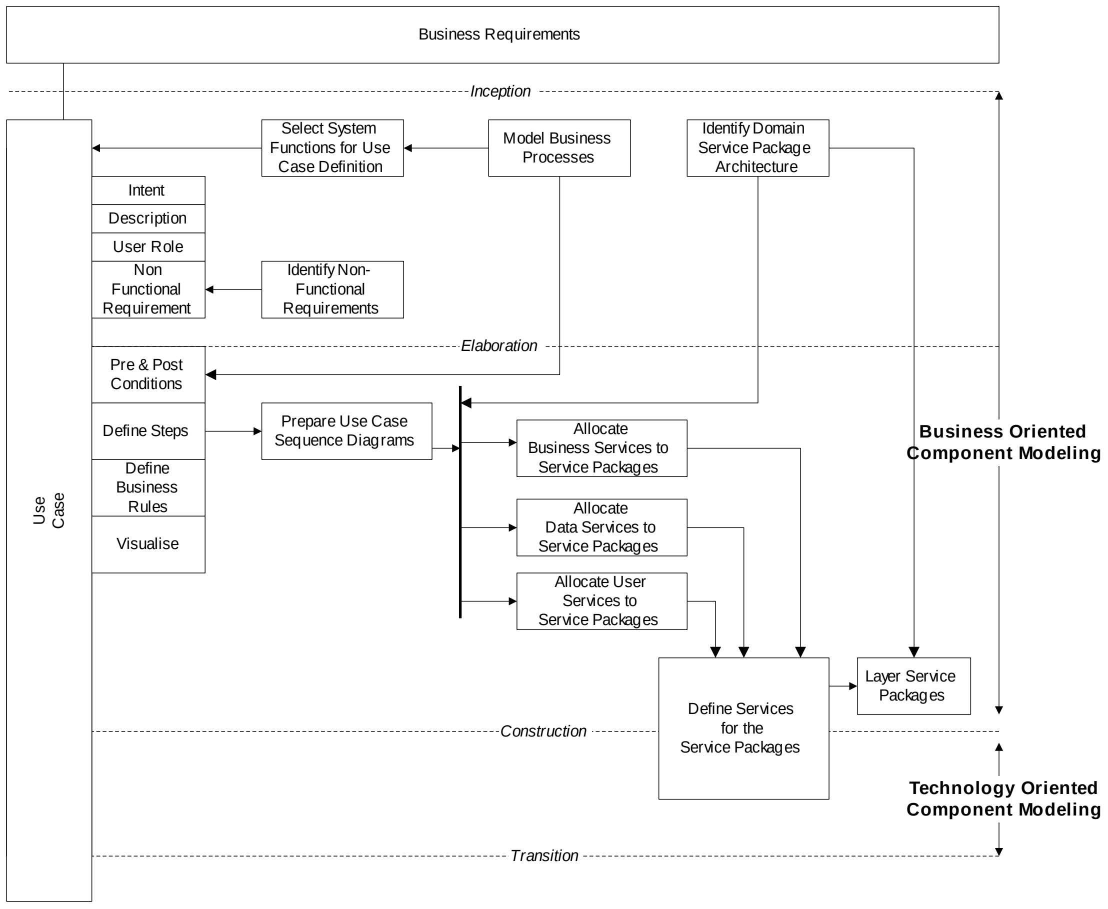

Business Requirements
Inception
Select System Functions for Use Case Definition
Model Business Processes
Identify Domain Service Package Architecture
Intent
Description
User Role
Non Functional Requirement
Identify Non-Functional Requirements
Elaboration
Pre & Post Conditions
Define Steps
Prepare Use Case Sequence Diagrams
Define Business Rules
Visualise
Use Case
Allocate Business Services to Service Packages
Allocate Data Services to Service Packages
Allocate User Services to Service Packages
Business Oriented Component Modeling
Define Services for the Service Packages
Layer Service Packages
Construction
Technology Oriented Component Modeling
Transition

Figure 3-10 : Elaboration Phase Activities

The overall objectives of this phase are to:

- add detail to the Use Cases (including business rules) to achieve a series of defined steps from which Services can be identified.
- define Business Services and supporting Data Services (see Section 3.4.6)
- define the User Services for the purpose of transaction control and invocation of Business Services.
- allocate all Services to Service Packages in their respective layers.

3.4.2 Use Case : Pre & Post Conditions

The pre and post conditions can be determined by reference to the process model (Figure 3-3), and should now be documented in the Use Case (Figures 3-11, 3-12 and 3.13). Also refer to the Use Case Template in Figure 3-4.

Use Case Name		***New Claim***
User Role		Claim Handler
Intent		Register a new Claim against a Policy.
Description		On notification of a Claim, the system will check that the claimant is an insured Policyholder, and that Cover is available for the type of Claim. For a valid Claim, a new Claim will be registered on the system.
Detailed Steps	1.	Enter key fields and retrieve Policy details
	2.	Check that the Policy is valid for the Claim Date, and that the Policyholder is covered for this Type of Claim
	3.	Create a new Claim with details as supplied on the Claim Form, and set the Status to Pending Authorization
Pre Conditions		Policy Found
Post Conditions		Status : New Claim Pending Authorization
Non-functional Requirements		The system will confirm the creation of the new Claim within 10 seconds.
Business Rules	1.	The Claim Loss Date must be within the period of Cover.
	2.	The Claim Loss Type must be covered by the Policy.
Alternative courses		None
Sample screen layouts		Prototype attached below. *(beyond scope of Dissertation)*
Requirements References		Claims01

Figure 3-11 : New Claim Use Case – Elaboration Phase

Use Case Name		***Authorize Claim***
Role		Claims Supervisor
Intent		Authorize the payment of a valid claim.
Description		After checking that the Authorizer has the required level of seniority to Authorize the Claim, the Claim will be Authorized for Payment.
Detailed Steps	1.	Check the user authorization limit
	2.	Open the Claim
	3.	Open the Policy
	4.	Check that Amount to be paid is within the limits of Cover
	5.	Authorize the Claim, and set the Status to Authorized.
Pre Conditions		Status; : Claim is Pending Authorization
Post Conditions		Status : Claim is Authorized
Non-functional Requirements		The system will confirm Authorization within 3 seconds.
Business Rules	1.	The value of the Claim must be equal or less than the Authorization Limit for the User.

	2.	The value of the Claim must be equal or less than the Sum Insured covered by the Policy.
Alternative courses		Authorize Partial Payment
Sample screen layouts		Prototype attached below. *(beyond scope of Dissertation)*
Requirements References		Claims01

Figure 3-12 : Authorize Use Case – Elaboration Phase

Use Case Name		***Pay Claim***
User Role		Claim Handler
Intent		The Claim will be paid within a period of time according to company Best Practice (a non-computerized procedure).
Description		The Claim is Authorized for Payment and sent to the Accounts system for processing.
Detailed Steps	1. 2. 3.	Open the Claim Pay the Claim Close the Claim, and set the Status to Closed.
Pre Conditions		Status : Claim Authorized
Post Conditions		Status : Claim Paid
Non-functional Requirements		The system will confirm the payment request within 3 seconds.
Business Rules		None
Alternative courses		None
Sample screen layouts		Prototype attached below. *(beyond scope of Dissertation)*
Requirements References		Claims01

Figure 3-13 : Pay Claim Use Case – Elaboration Phase

3.4.3 Use Case : Define Business Rules

The business rules associated with the Use Case should now be determined and added to the Use Case (Figures 3-11, 3-12 and 3-13).

Normally it is recommended that only those rules relevant to User Services are defined e.g. a Claims Handler can view payments for which they are not eligible to Authorize. The definition of other rules would be defined elsewhere (often unspecified), and allocated to relevant object classes in a domain modelling exercise.

However, I consider that all rules should be specified in the first instance with the Use Case. It is then possible to view all rules corresponding to the use case, and they can still be effectively allocated to appropriate services later in the Allocate to Service Package activities. Ideally, a repository of Business Rules should be built which cross-references the Use Case, the Service Package that uses them, and the Component Package that implements them.

3.4.4 Use Case: Requirements Visualization

A useful aid to the communication of business requirements is by means of visual representation. A prototype is often used, and window layouts can easily be created using modern GUI development tools like Delphi, PowerBuilder, Microsoft's VisualStudio or IBM's VisualAge range of products. The key thing to remember, however, is that while these layouts can be reused in the actual development, it is the data that is being presented for agreement and not the visual behaviour at this stage.

It should be noted that this exercise could also be part of the Inception phase. However, I have left it until early in the Elaboration phase to cater for the emergence of more detailed requirements, which will provide a more meaningful insight into the workings of the required system.

3.4.5 Use Case : Define Steps

Further analysis of the Use Case is now required in order to explicitly define the steps that are being carried out. This should be described in the form of structured English using the constructs of sequence (a,b,c…), choice (if/else, case of…) and iteration (for each…). This is what Jacobson, Ericcson, and Jacobson (1994) would describe as a system design use case, whereas Allen and Frost (1998) consider this the separate task of object interaction modelling (with sequence diagrams).

I consider that the detailed steps belong with the use case and agree with Jacobson et al (1994). It is also more natural to question the detailed steps at this point, rather than when you come to allocate the steps to Services, which is the main purpose of the sequence diagramming activity in this methodology, and indeed also that of Allen and Frost (1998)! I would suggest that the structured English is reviewed during the sequence diagramming, and would not be too critical of the construct syntax at this stage in the elaboration of the Use Case.

However, the activities of use case definition and sequence diagramming are very closely related, and it is recommended to iterate between the two. This is necessary if we are to define layered system services in terms of Business Services, their supporting Data Services and the controlling User Services that implement the Use Case. If we build them in this order, it is easier to verify that the appropriate services are in place by enacting a scenario of the use case.

3.4.6 Service Identification

The key to reuse is the definition of services to support the requirements of the Use Cases. The starting point for this is an examination of the detailed Steps of the Use Case from which the following type of service will be defined:

Business Services

"A business service normally only exists to support other services and is largely 'black box'" (Allen and Frost, 1998). It does not usually have a purpose other than to respond to a request from a User Service (or another Business Service) in support of a Use Case. In addition to the application of business rules, the Business Service communicates with the Data Service for the management of persistent objects.

Data Services
“Data Services are used for manipulation of data in a way that is independent of the underlying physical storage implementation” (Allen and Frost, 1998). Data Services are requested by Business Services. The only exception to this is for a housekeeping task, when the User Service can directly address a Data Service.

User Services
“A user service normally implements a significant part of a use case if not the whole use case, confers some kind of business capability, and involves significant user-interface design.” (Allen and Frost, 1998). In addition to control of the visual interface with the user, User Services may also be responsible for batch reporting, transaction control and the provision of search facilities.

The proposed Services are described in Section 3.4.8, after discussing the nature of the Service Package diagram in the next Section.

3.4.7 Service Packages

The UML (1999) defines a package as “A general purpose mechanism for organising elements into groups”. This construct was first introduced in the Inception phase when the high level architecture was designed. In this case it represented the abstraction of lower level Service Packages.

We now need to introduce the concept of Service Packages in relation to the detailed steps of the Use Case.

My view is that all identified services should be targeted at a Service Package, implemented in object oriented design terms as a Service Class, sometimes known as a Control Class. As a design pattern this is known as a ‘façade’ which: “Provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to use.” (Gamma, Helm, Johnson, and Vlissides, 1995).

I believe that interacting with the façade is more appropriate for business oriented modelling than dwelling on the design of a business oriented object model, which is the approach in most object oriented methods, and in the component based approach of Allen and Frost (1998). I have seldom found this to bring much greater insight into the working of the system than that brought from the techniques previously introduced, for a number of reasons:

- it does not form a suitable vehicle for communication with the business user, since an understanding of object oriented class design and modelling is required.
- it places an overhead on the analysis phase, and requires business analysts to have a thorough understanding of object oriented class design and modelling.
- it is of little value during physical design, since platform specific considerations will have to be made, and indeed off-the-shelf class libraries provided by the vendor may satisfy a number of these requirements.
- the model is seldom maintained – the concentration is on the physical design once the system has been built.

3.4.8 Service Allocation to Service Packages

This activity is modelled using a Sequence Diagram, which is a type of object interaction diagram, and as such, a standard component of the UML. The layout is shown in Figure 3-14.

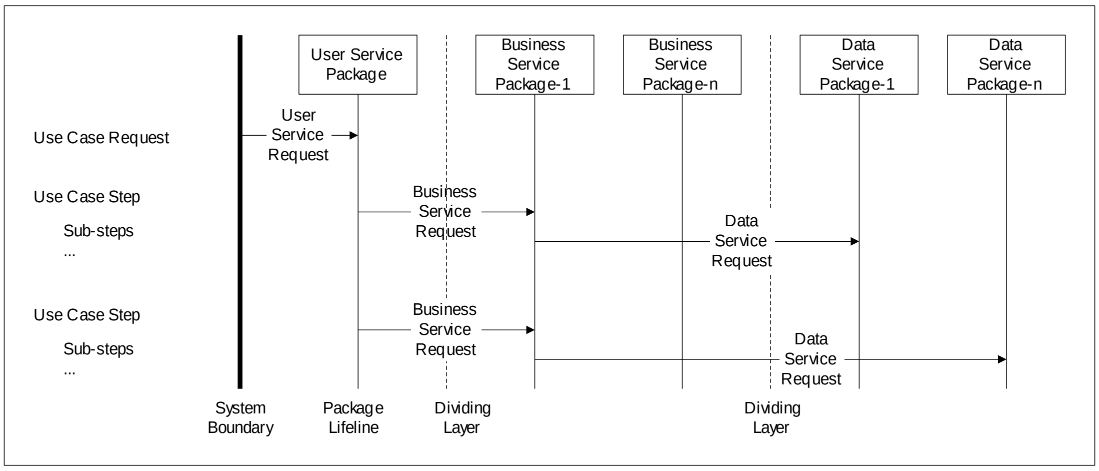


Figure 3-14 : Sequence Diagram Template

Notation:

The Steps and sub-steps of the use case are detailed down the left hand side of the diagram

The solid vertical line represents the System Boundary.

The square boxes are the Service Packages, below which a line is extended to the end of the use case, it's Lifeline.

The dotted vertical lines represent the boundary of the system layers, which are read left to right and reflect the User, Business and Data Layers of the system.

The arrows from the System Boundary or Service Packages represent the request for a Service, or Service Operation, and display only the name of the Service. The parameters supplied and returned are fully defined in the Service Definitions (see Section 3.4.11).

A Sequence Diagram is created for each Use Case, to which is added the steps of the Use Case in structured English. The related Service Packages required by the Use Case are added along the top.

Unlike Allen and Frost (1998), who start this exercise with only Business Service Packages on the diagram, my approach differs in two respects:

- add the Use Case as the initiating User Service – we know this anyway, and it adds meaning to the diagram.
- add the Data Service Packages in support of the Business Services – we tend to know what they are, and it is useful to indicate the type of package being used e.g. a legacy system (under the guise of a wrapper). Again, it adds more value to the diagram.

Now, the Services are defined and added to the diagram, linking each Step of the Use Case with the Service Package deemed responsible for its execution. Figures 3-15, 3-16 and 3-17 illustrate the allocation of Business Services and their supporting Data Services to the Service Packages.

At this stage it is again possible to validate the allocation of Services by running a scenario through the Use Case e.g. My Claim for £800 notified on 1st March 1999.

Assuming the relevant Services are deemed to be in place, we can then focus on the identification of User Services to control the display of windows and the flow of control for the transaction. This is illustrated for the New Claim Use Case in Figure 3-18.

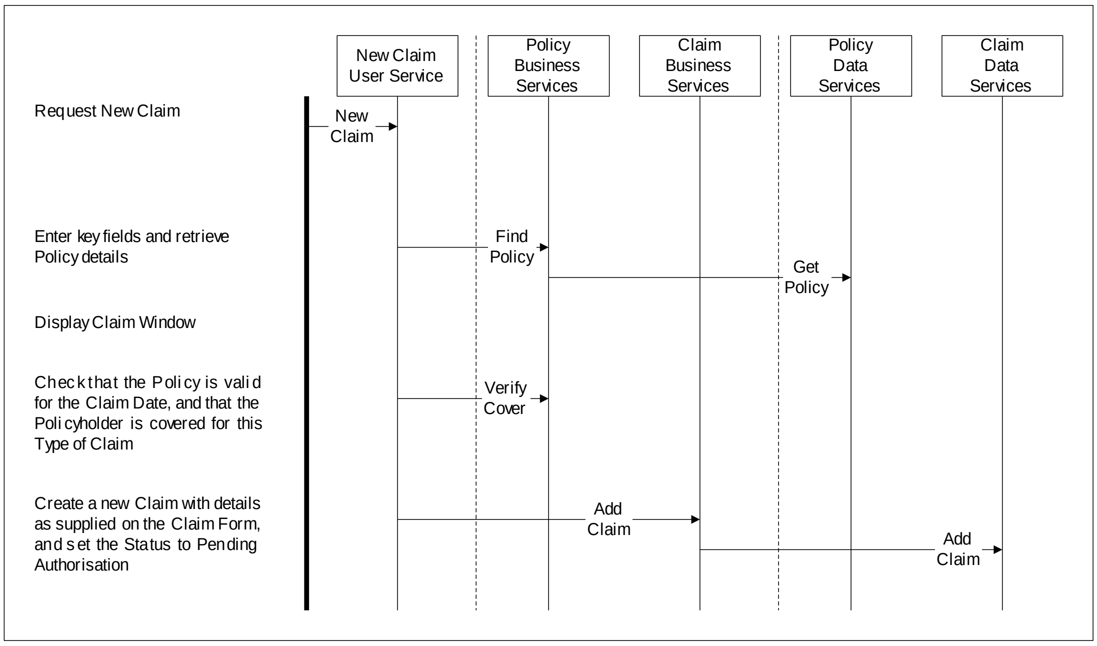

New Claim User Service
Policy Business Services
Claim Business Services
Policy Data Services
Claim Data Services
Request New Claim
New Claim
Enter key fields and retrieve Policy details
Find Policy
Get Policy
Display Claim Window
Check that the Policy is valid for the Claim Date, and that the Policyholder is covered for this Type of Claim
Verify Cover
Create a new Claim with details as supplied on the Claim Form, and set the Status to Pending Authorisation
Add Claim
Add Claim

Figure 3-15 New Claim : Business and Data Services

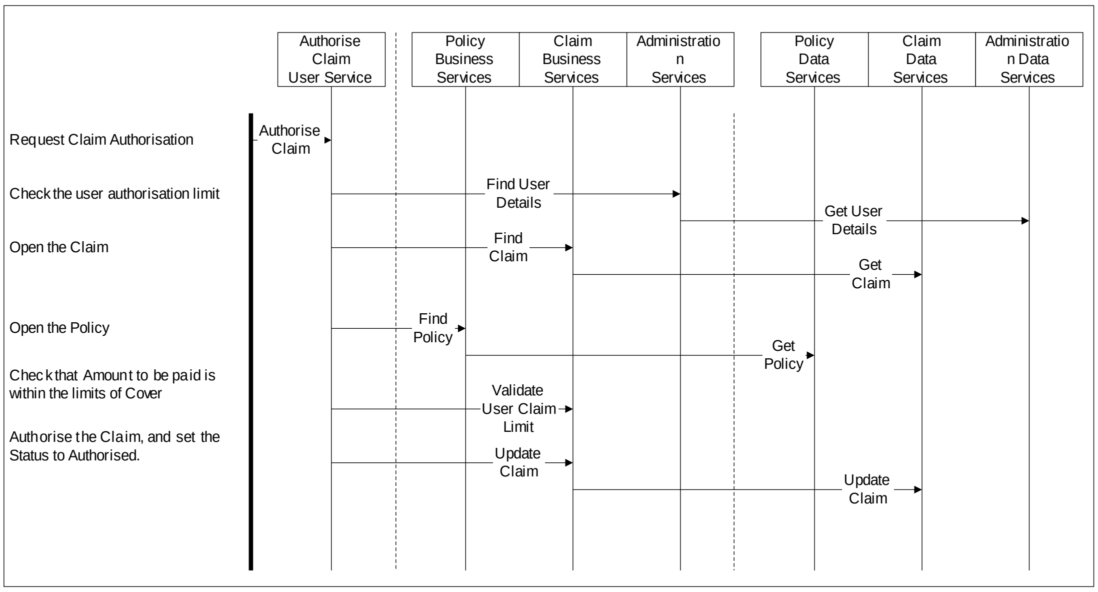

Authorise Claim User Service
Policy Business Services
Claim Business Services
Administration Services
Policy Data Services
Claim Data Services
Administration Data Services
Request Claim Authorisation
Authorise Claim
Check the user authorisation limit
Find User Details
Get User Details
Open the Claim
Find Claim
Get Claim
Open the Policy
Find Policy
Get Policy
Check that Amount to be paid is within the limits of Cover
Validate User Claim Limit
Authorise the Claim, and set the Status to Authorised.
Update Claim
Update Claim

Figure 3-16 Authorize Claim : Business and Data Services

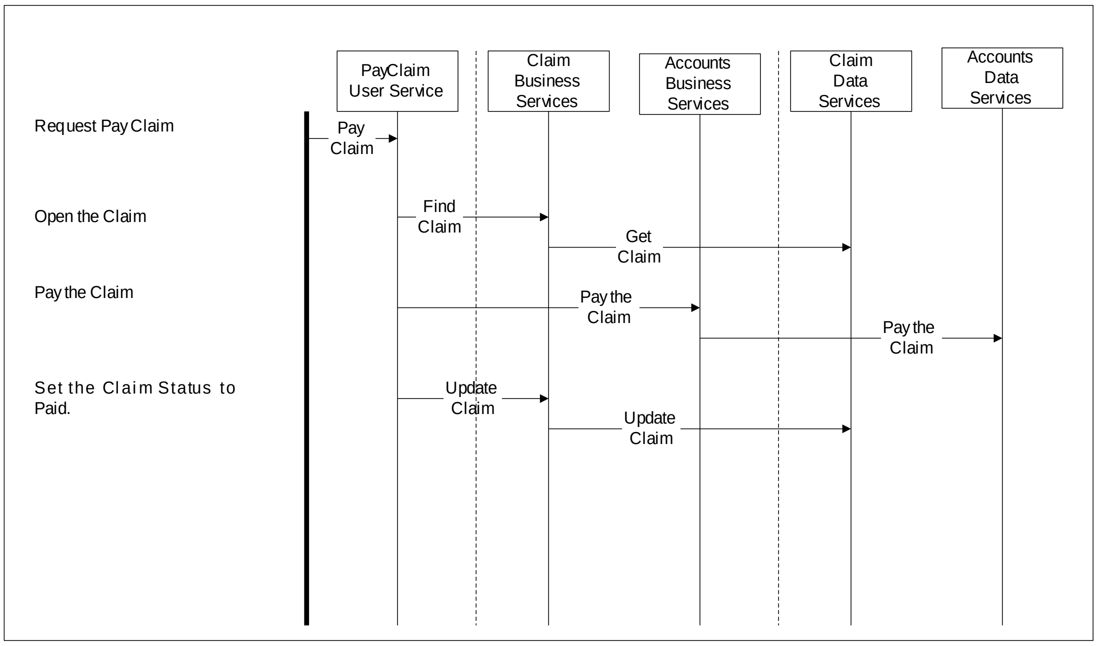


Figure 3-17 Pay Claim : Business and Data Services

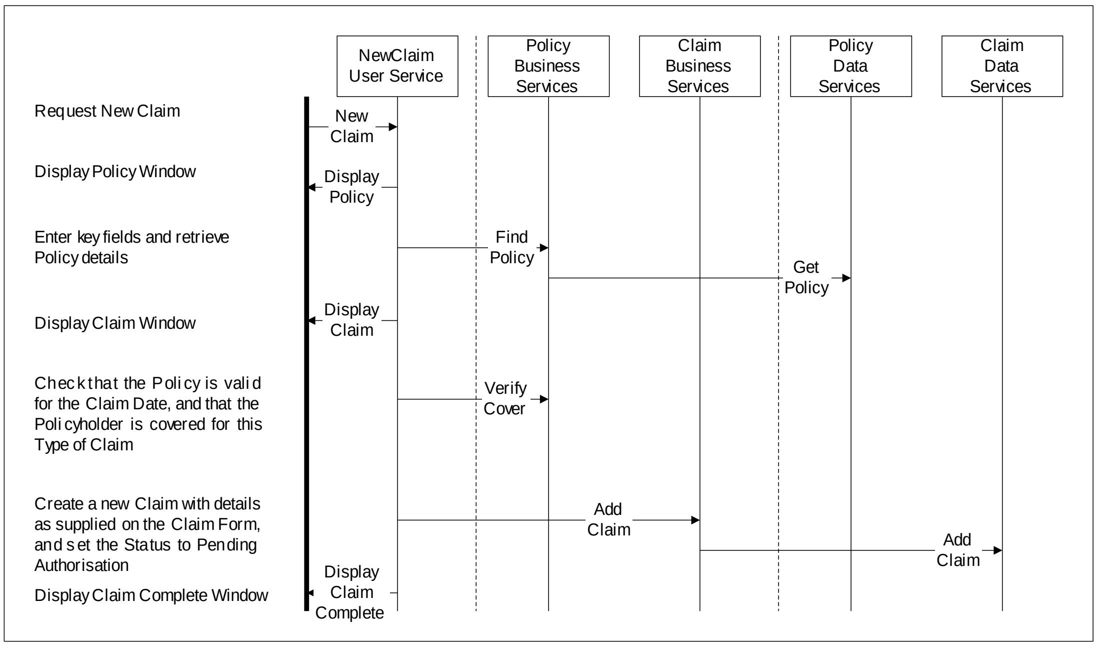

NewClaim User Service
Policy Business Services
Claim Business Services
Policy Data Services
Claim Data Services
Request New Claim
New Claim
Display Policy Window
Display Policy
Enter key fields and retrieve Policy details
Find Policy
Get Policy
Display Claim Window
Display Claim
Check that the Policy is valid for the Claim Date, and that the Policyholder is covered for this Type of Claim
Verify Cover
Create a new Claim with details as supplied on the Claim Form, and set the Status to Pending Authorisation
Add Claim
Add Claim
Display Claim Complete Window
Display Claim Complete

Figure 3-18 : New Claim : User, Business and Data Services

3.4.9 Service Package Layering

At this stage we can illustrate the layering of the proposed application by revisiting the Domain Service Package Architecture (Figure 3-9), to which we can now add the Services which have been identified for this application (Figure 3-19).

Again, this is a useful diagram for communication of the system architecture both to the business user, technical architects, and system developers. It clearly identifies the

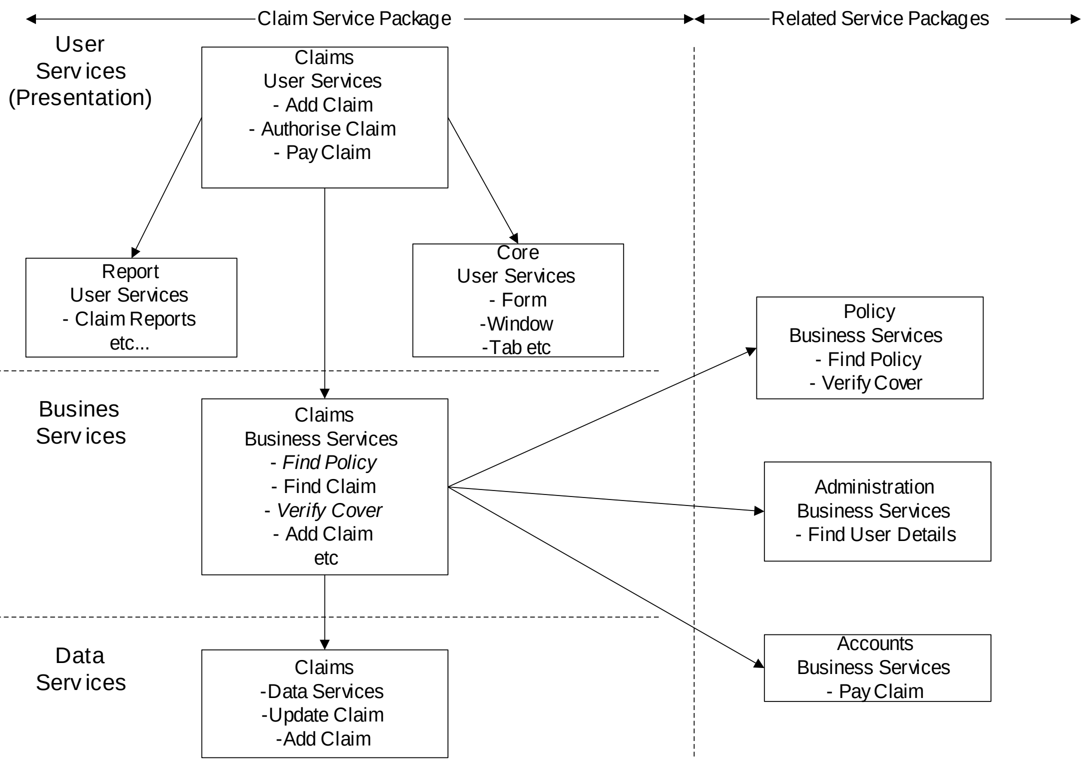

Claim Service Package
Related Service Packages
User Services (Presentation)
Claims User Services - Add Claim - Authorise Claim - Pay Claim
Report User Services - Claim Reports etc...
Core User Services - Form -Window -Tab etc
Busines Services
Claims Business Services - Find Policy - Find Claim - Verify Cover - Add Claim etc
Policy Business Services - Find Policy - Verify Cover
Administration Business Services - Find User Details
Accounts Business Services - Pay Claim
Data Services
Claims -Data Services -Update Claim -Add Claim

Figure 3-19 : Domain Service Package Architecture - Layered

User Services implementing the Use Case in the presentation layer, the Business Service in the business object layer, and the supporting Data Services in the data layer.

3.4.10 Define Service Package Deployment

Now that the Services have been identified, the location and implementation of the Service Packages can be considered. The layers in Figure 3-19 would be referenced to determine how this was to be achieved. The non-functional requirements must also be considered at this stage, especially any physical constraints that may impact the choice of nodes e.g volume of transactions, network response times, current infrastructure (Windows NT, Unix etc).

For example, deployment in a thick-client architecture may involve deployment of the:

- User Services for Claims (with core user, reporting and administration services), and Business Services for Claims, Policy and Administration to an NT based desktop PC.
- Data Services for Claims to a Claims departmental server using a SQL Server database.
- Data Services for Policy and Administration to a central corporate server using an Oracle database.

This is illustrated in Figure 3-20 where, again, the UML notation has been used.

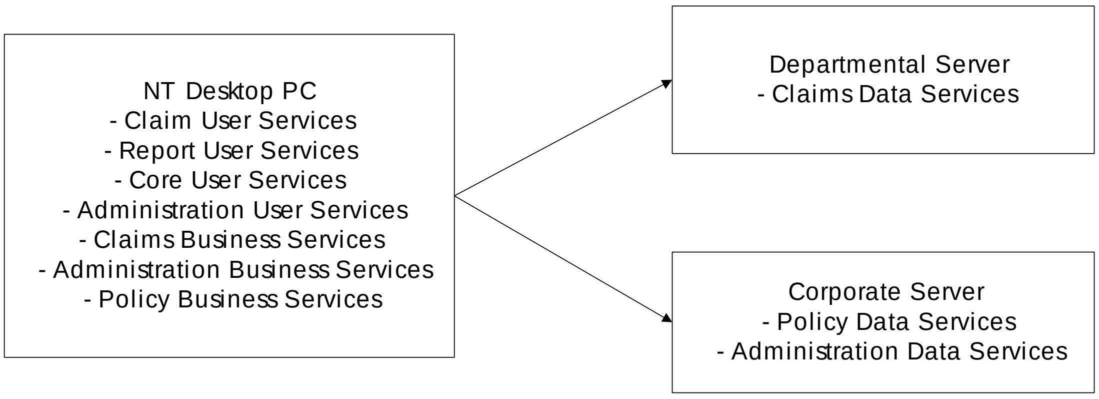


Figure 3-20 : Service Package Deployment - NT Desktop PC

Notation

Boxes	Boxes repesent the nodes in a network. They are shown as 3-D boxes in the UML, but as just plain boxes in this model The name of the node is listed at the top, and the Component.Packages listed underneath.
Lines between boxes	The lines represent the connection between nodes

Alternatively, with a thin client architecture (Figure 3-21) the same service packages could be deployed as follows:

- User Services for Claims (with core user, reporting and administration services), to a Browser (Netscape or Internet Explorer) on a thin client eg netPC
- Claims Business and Data Services to a Claims departmental server using a SQL Server database.
- Policy and Administration Business and Data Services to a central corporate server using an Oracle database.

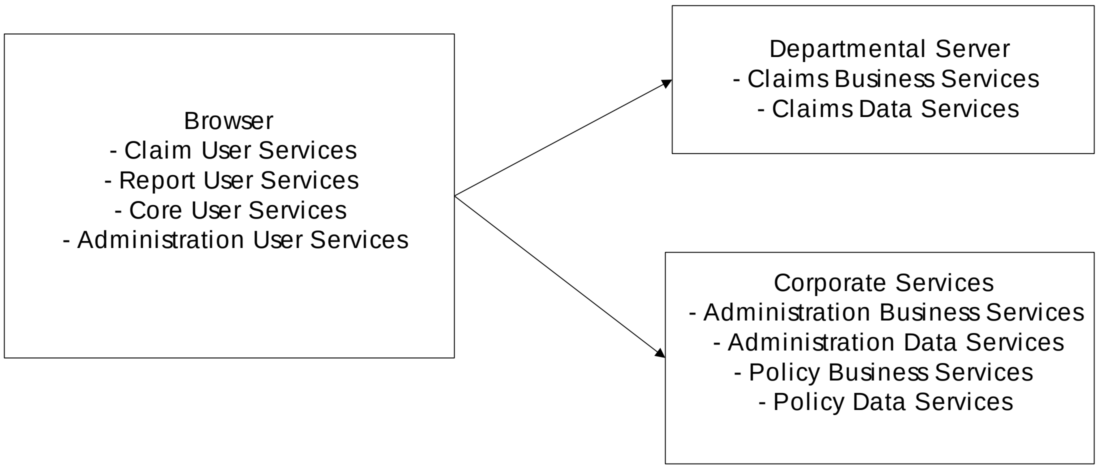


Figure 3-21 : Service Package Deployment - Browser Environment

3.4.11 Service Catalogue

At this stage we should also register the Services in the Service Catalogue. Figures 3-22, 3-23 and 3-24 illustrate a simple Catalogue, which lists the Services according to the Package stereotypes of User, Business and Data Services. This type of Catalogue would be referenced in the Inception phase to determine whether existing Services could be reused, prior to defining a new one. The detailed description of the Service is described in the next Section.

User Service Package	*Service*
New Claim	New Claim
	Display Policy
	Display Claim
	Display Claim Complete
Authorize Claim	Authorize Claim
Pay Claim	Pay Claim

Figure 3-22 : Service Catalogue – User Service Packages

Business Service Package	*Service*
Policy	Find Policy
	Verify Cover
Claim	Add Claim
	Find Claim
	Update Claim
	Validate User Claim Limit
Accounts	Pay the Claim
Administration	Find User Details

Figure 3-23 : Service Catalogue – Business Service Packages

Data Service Package	*Service*
Policy	Find Policy
	Verify Cover
Claim	Add Claim
	Get Claim
	Update Claim
Administration	Get User Details
Accounts	Pay the Claim

Figure 3-24 : Service Catalogue – Data Service Packages

3.4.12 Service Definition

Having now identified and allocated the Services to Service Packages we now need to define the service detail. This is done in the form of a Service Specification, the template for which is illustrated in Figure 3-25. This defines the Service in technology neutral terms. It allows the specification to be reused, and enables the Component Package to be built for deployment on different platforms, or using different languages.

Service Name	Name of the operation in the form *PackageName:ServiceName* eg *Claim:FindClaim*
Stereotype	The type of service : User, Business or Data
Intent	A short description of the operation.
Parameters	The parameters consist of a list of those supplied to invoke the service, and the expected result. The type of each field is specified in each case eg date, integer, character string (char), boolean (true/false) etc. For example: <u>Supplied</u> (PolicyClass : Class ,PolicyNumber : Char ,ClaimValue : Integer) <u>Returned</u> (ClaimValid : Boolean) (Note: In technical terms this could be described as the signature of the service consisting of an input argument and a return type expression.)
Pre / Post Conditions	A means of specifying assertions that describe software contracts (Meyer 1988, 1995)
Specification	The specification of the operation
Called Operations	Any operations required by this operation.
Attributes	List of attributes updated
Non-Functional Requirements	List of any non-functional requirements eg response times.
Use Case	Reference to Use Cases using this Service
Business Requirements	Reference to originating Business Requirements

Figure 3-25 : Service Definition Template

My approach is to maintain the terminology of business oriented component modelling, and continue to call this a Service for consistency, unlike Allen and Frost (1998) who would now call this an Operation, which is a more object-oriented term. The Service forms the contract between the business and technical boundary and is, therefore, immutable.

Again, unlike Allen and Frost (1998), I would maintain cross-references to the Business Requirements and Use Cases as can be seen in the Add Claim Service example in Figure 3-26, and Verify Cover Service in Figure 3-27.

Service Name	***Claim : Add Claim***
Stereotype	*Business Service*
Intent	Create a new Claim
Parameters	<u>Supplied</u> (PolicyNumber : Integer ,ClaimType : Char ,ClaimDate : Date ,ClaimValue : Integer) <u>Returned</u> (ClaimNumber : Integer)
Post Conditions	New Claim added to the system Policy Claim List updated to include this Claim
Specification	This service will :- • add a new Claim to the system • add the Claim to the list of Claims associated with the Policy • increment the last Claim Number by one, and return the new Claim Number
Called Operations	None
Transmitted events	None
Attributes Set	Policy: PolicyNumber Claim: ClaimType Claim: ClaimDate Claim: ClaimValue Claim: ClaimNumber
Non-Functional Requirements	Response time < 3 seconds.
Use Case	New Claim
Business Requirements	Claims01

Figure 3-26 : Add Claim Service Definition

Service Name	***Policy : Verify Cover***
Stereotype	*Business Service*
Intent	Verify that the Policyholder is covered for the Claim notified.
Parameters	<u>Supplied</u> (PolicyNumber : Integer ,ClaimType : Char ,ClaimDate : Date) <u>Returned</u> (isCovered : Boolean)
Pre Conditions:	Policy is found
Specification	Check that :- • the ClaimDate is within the Policy effective date range • the ClaimType is covered by the Policy If both conditions are true, return isCovered as True, otherwise False.
Called Operations	None
Transmitted Events	None
Attributes Set	None
Non-Functional Requirements	Response time < 3 seconds.
Use Case	New Claim
Business Requirements	Claims01

Figure 3-27 : Verify Cover Service Definition

3.4.13 Architectural Components in Place on Completion of Elaboration

On completion of the Elaboration phase we have completed the translation of Business Requirements into Service Definitions that will now be used for detailed system design and build in the Construction phase. We have not thrown away any of the previous development phase documentation, but reused and progressively added detail and content to the Use Cases, and the Domain Service Package Architecture.

It is the Service Definition that provides the contract and interface between the business oriented component modelling of the business domain, and the technically oriented component-modelling domain. In order to achieve this bridge, we have introduced Sequence Diagrams to verify the allocation of Services to Service Packages, and only when considering how the system will be deployed do we move into the physical architecture of the system.

Nowhere have we considered objects and object design. In my development approach, it is the Service Definition that is the input to the Construction phase, which will now be introduced.

3.5 CONSTRUCTION

3.5.1 Introduction

The roadmap for the Construction phase (Figure 3-28) builds on the Use Cases, Service Package Definitions, and Deployment architecture defined in the Elaboration phase, and leads to the construction of Component Packages for deployment.

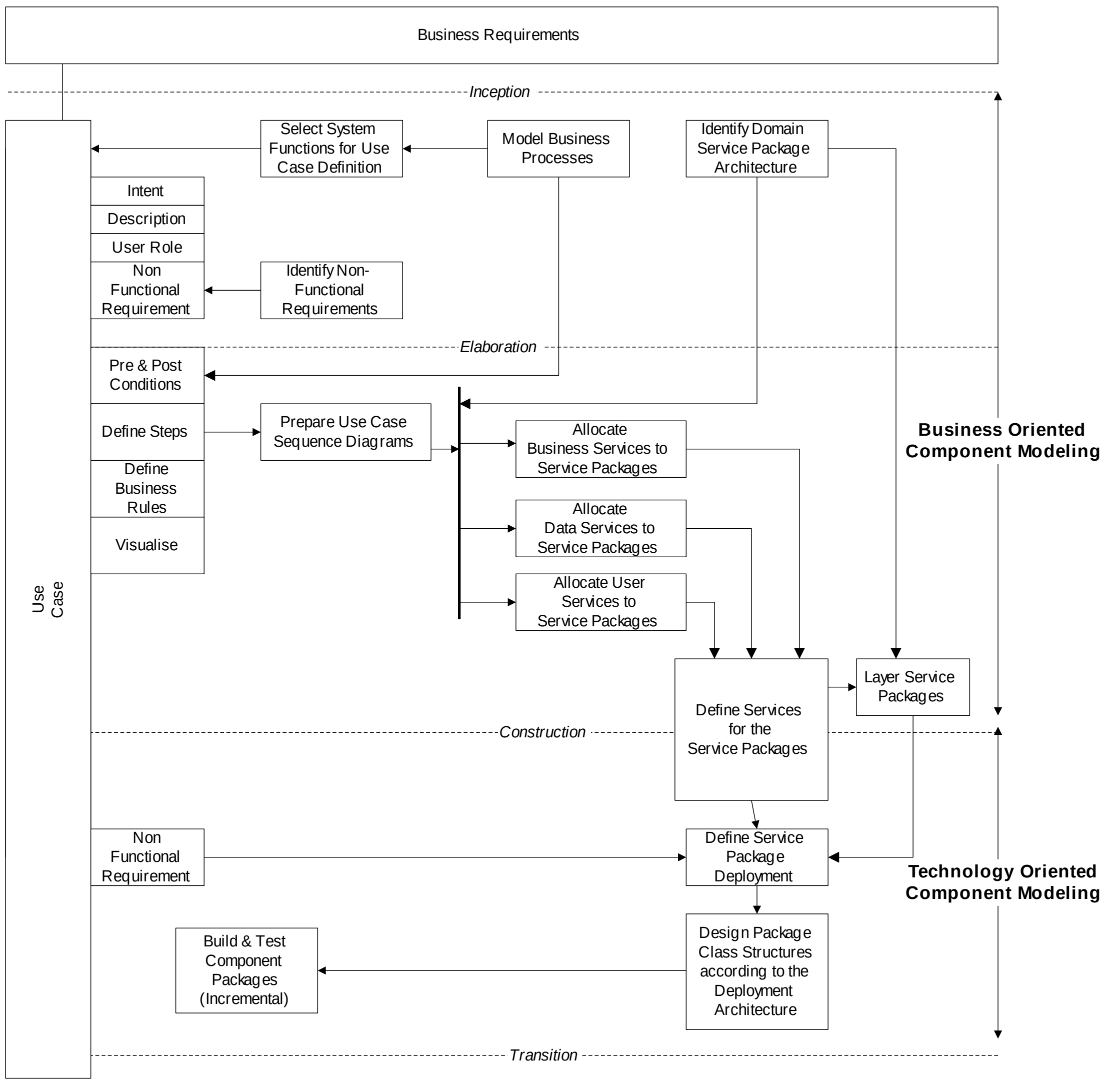

Business Requirements
Inception
Select System Functions for Use Case Definition
Model Business Processes
Identify Domain Service Package Architecture
Intent
Description
User Role
Non Functional Requirement
Identify Non-Functional Requirements
Elaboration
Pre & Post Conditions
Define Steps
Prepare Use Case Sequence Diagrams
Define Business Rules
Visualise
Allocate Business Services to Service Packages
Allocate Data Services to Service Packages
Allocate User Services to Service Packages
Business Oriented Component Modeling
Use Case
Define Services for the Service Packages
Layer Service Packages
Construction
Non Functional Requirement
Define Service Package Deployment
Technology Oriented Component Modeling
Build & Test Component Packages (Incremental)
Design Package Class Structures according to the Deployment Architecture
Transition

Figure 3-28 : Construction Phase Activities

The overall objectives of this phase are to:

- Design the object model for the chosen development language for this component.
- Design the user interface with the appropriate GUI software.
- Design the data objects to interact with the chosen database.
- Build and test the software.

It is not my intention to expand on the process of object-oriented design in this Section, but simply to illustrate how the Service Definitions for the Service Package are translated, by the same techniques of Sequence Diagramming used in business oriented modelling during the Elaboration phase.

3.5.2 Design Package Class Structures for the Deployment Architecture

Looking at the Service Package Catalogue (Figure 3-23) for the Policy Service Package, we can see that two Services are involved: Find Policy and Verify Cover. This is shown with a Sequence Diagram using the same Policy Use Case Steps, but instead of the Services pointing to the Service Package, they are now allocated to the controlling (or service) class façade for the Policy Component Package (Figure 3-29)

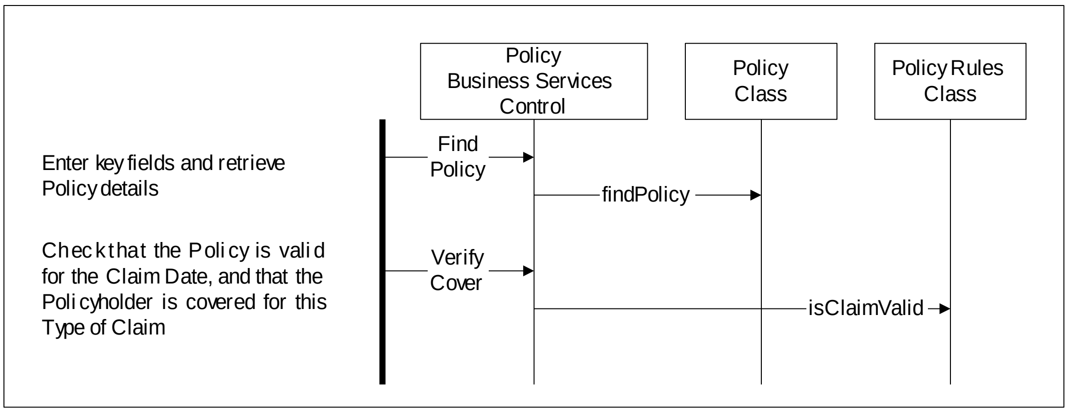

Policy Business Services Control
Policy Class
Policy Rules Class
Enter key fields and retrieve Policy details
Check that the Policy is valid for the Claim Date, and that the Policyholder is covered for this Type of Claim
Find Policy
findPolicy
Verify Cover
isClaimValid

Figure 3-29 : Policy Component Package - Business Classes

It is the controlling class that is now responsible for delegating activity to other business classes within the Policy Component Package. In this case two classes are defined, the Policy Class where the Policy is managed and where the findPolicy service will be implemented; and the Policy Rules Class where all Policy Rules are defined and are checked, in this context with the isClaimValid service request.

Unlike Allen and Frost (1998) who introduce domain class modelling as part of the analysis phase, I consider the class design to be the job of the package designer, who in my view is best placed to carry out this task. This is beneficial for a number of reasons:

- the designer is familiar with the intricacies of the technical environment for which they are designing the component, including any frameworks available to simplify the application build.
- the class design is governed by the Service based documentation, and does not involve interpretation / misinterpretation of a business domain model.
- the designer has responsibility for the technical design and development of the component.

- one designer or team can be made responsible for the design, development and maintenance of the package.
- the Service Definition is the interface between the designer and the business analyst.

An example of the class design for the New Claim service user interface is also illustrated (Figure 3-30). This shows similar interaction between the New Claim User Service Control Class, and to its left, the classes responsible for window control: Policy Key Window and Claim Window. The requests from the control class for business services are also shown for completeness.

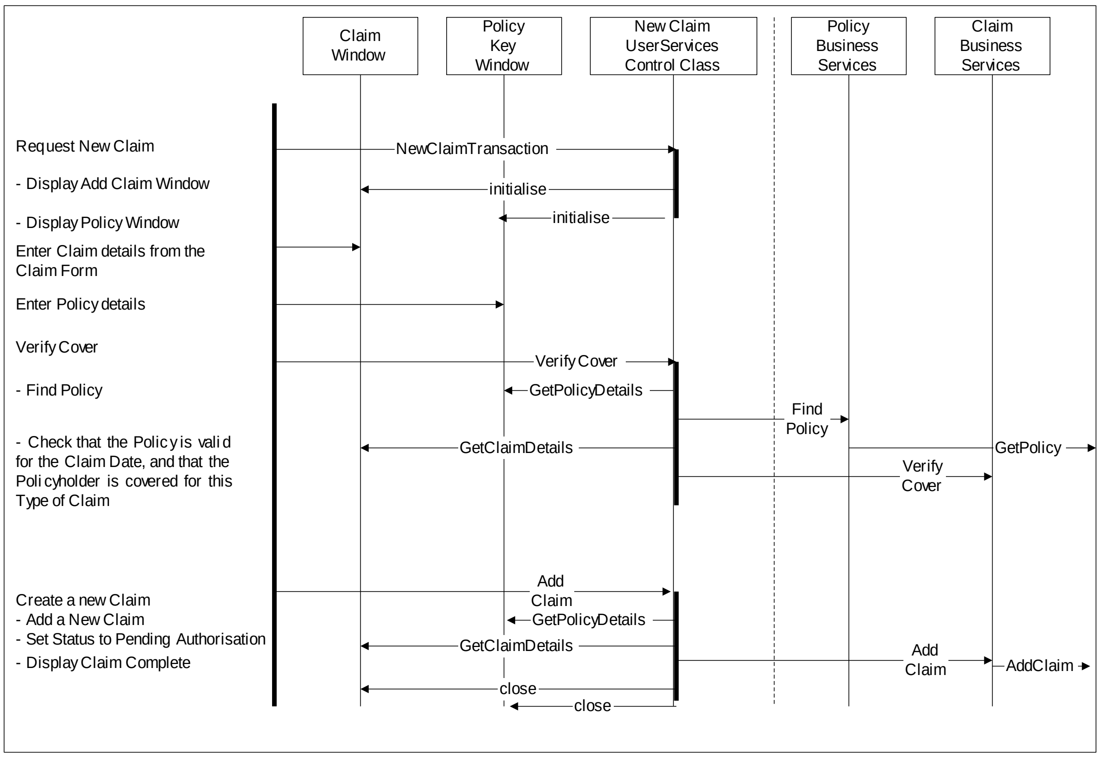


Figure 3-30 : New Claim Component Package - User Classes

3.5.3 Component Packages Build and Test - Incremental

While beyond the scope of this work, it should be noted that an incremental build and test cycle would be adopted as recommended by the Objectory Process for the Construction phase.

3.5.4 Architectural Components in Place on Completion of Construction

On completion of the Construction phase we will have completed the translation of Business Requirements via Service Packages into deployable Component Packages, accessible only as a black box via their service interface. The Service Definition defines this interface, and contract, between the analyst’s domain of business oriented component modelling and designer’s domain of technology oriented component modelling. The Component Packages are now available for Transition into the production environment as defined in the next Section.

3.6 TRANSITION

The Transition phase within the Objectory Process takes the tested Component Packages in preparation for production rollout. Again, this phase is outside of my remit and is referenced here for completion. The complete process, including the Transitional activities of System Test, Performance Test, User Acceptance Test and Deployment are illustrated in Figure 3-31 below.

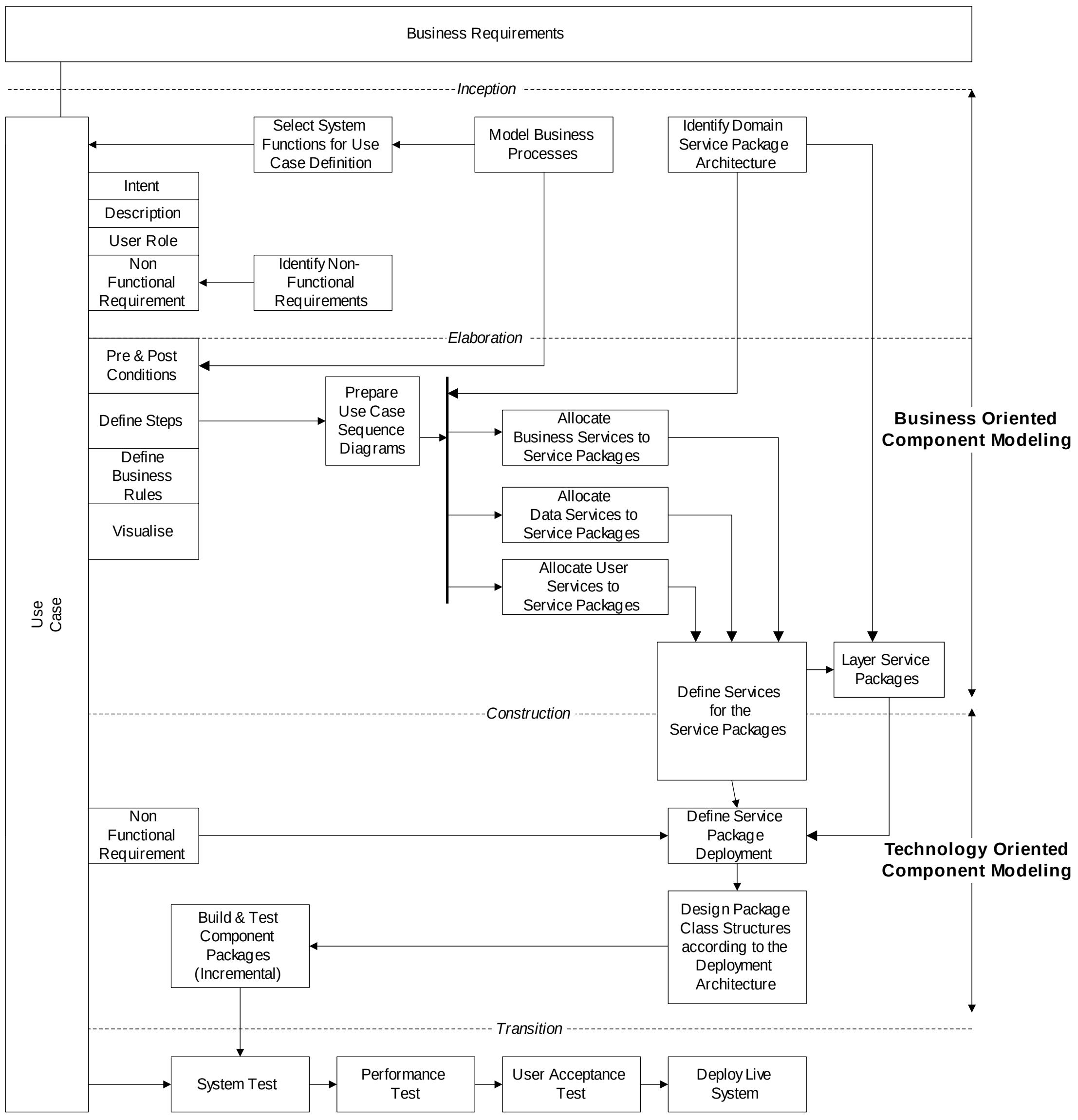

Business Requirements
Inception
Use Case
Select System Functions for Use Case Definition
Model Business Processes
Identify Domain Service Package Architecture
Intent
Description
User Role
Non Functional Requirement
Identify Non-Functional Requirements
Elaboration
Pre & Post Conditions
Define Steps
Define Business Rules
Visualise
Prepare Use Case Sequence Diagrams
Allocate Business Services to Service Packages
Allocate Data Services to Service Packages
Allocate User Services to Service Packages
Business Oriented Component Modeling
Define Services for the Service Packages
Layer Service Packages
Construction
Non Functional Requirement
Define Service Package Deployment
Technology Oriented Component Modeling
Design Package Class Structures according to the Deployment Architecture
Build & Test Component Packages (Incremental)
Transition
System Test
Performance Test
User Acceptance Test
Deploy Live System

Figure 3-31 : Transition Phase Activities

4. DISCUSSION and EVALUATION

4.0 Evaluation Methodology

In order to test the feasibility of the methodology proposed in the Original Material of Section-3, I chose to present my arguments in the form of a number of claims, and request feedback in the form of a questionnaire.

I opted to obtain quality feedback from a small peer group of six experienced and senior IT professionals, representing a stratified sample from the insurance and financial services sectors. I will hereafter refer to this group as the Panel, for whom the member details and experience is summarized in Figure 4-1 below. ***It should be noted that the comments are strictly those of the Panel members and not necessarily those of the organisations to which they belong.***

Name	**Job Title**	**Organisation**	**Years in IT**	**Years in IT Consultancy, Project or Departmental Management**
Len Hall	Senior Systems Specialist	Royal & Sun Alliance : Commercial IT – UK Regions Division	21	12
Patricia Johnson	IT Project Manager (Strategy)	Royal & Sun Alliance : Commercial IT –UK Regions Division	10	4
Michael Lowcock	Senior Consultant	CAP Gemini	25	15
Martin Morris	IT Manager	Royal & Sun Alliance : UK Commercial –London Market & Multinational Division	20	15
Neil Peachey	Systems Consultant	Royal & Sun Alliance : Personal Financial Services – Direct Division	14	4
Steve Walley	Project Teams Manager	Royal & Sun Alliance : Personal Financial Services – Direct Division	20	8

Figure 4-1 : Questionnaire Recipients & Feedback Panel Members

As it can be seen, this represents 110 years experience in information technology, 58 years being at a senior level. The fact that 100% of the target sample returned their feedback within the prescribed time limits, given their already busy schedules, is a reflection on the caliber of the individuals involved, and their degree of interest in this field of work. Five of the Panel members are senior IT personnel in three autonomous divisions of Royal and SunAlliance[1], and the sixth a member of the leading services company CAP Gemini[2]. I consider that this blend of insurance and services sector

[1] Royal & SunAlliance is one of the world's top ten international general insurers by premium income and the UK's largest general insurer. The Group is in the top 30 of the 'FT-SE 100' companies quoted on the London Stock Exchange. At the end of 1998, Royal & SunAlliance employed over 42,000 people in over 55 countries worldwide, and covered risks in more than 130 countries.

[2] The Cap Gemini Group is the leading European management consulting (Gemini Consulting) and IT services companies (Cap Gemini). Its businesses are those of management consulting, IT consulting, systems integration, software development, outsourcing and training. At the end of 1998 the Group employed over 38,000 people located in more than 20 countries.

experience will provide a good indication of how the methodology would be received if it was to be tested in the wider marketplace.

Section-3 was issued to each member of the Panel together with the Questionnaire, which included a note describing the background to the research, and instructions for completion of the form. A copy of the Questionnaire and its covering note is included in Appendix-4.

I designed the Questionnaire in four sections in order to:

a) test the understanding and reaction of the Panel to the design activities in the proposed component process (leading to the definition of service packages, as a pre-requisite for effective component design and build.)

b) test the understanding and reaction of the Panel to the potential for reuse of design artifacts (as opposed to just code), and the provision of requirements traceability.

c) consider what is needed to make the approach work.

d) provide an opportunity to raise any questions or comments.

The Questionnaire consists of 45 structured statements, designed to test my claims and form the basis of my argument (Booth, Colomb and Williams, 1995) in support of the significance of the approach. Each statement included a rating for the Panel to indicate the strength of their reaction to the claim, and increase the reliability and consistency of response. To increase the validity of the response (Bright, 1991) I also included space for comments where respondents could qualify their reaction. This proved to be beneficial where in some cases, respondents preferred to comment on an item rather than indicate the strength of an opinion at that point.

The consolidated responses are shown in Appendix-5. I have removed the identity of the respondents in order to present and evaluate the feedback collectively, and not at a personal level. All references will be to the Panel. It should also be noted that the comments are sometimes brief, and in the form of short notes. I have maintained this brevity when quoting the Panel, since it is the message that is important here and not the grammar.

In the following sections I will re-state each claim, and evaluate the Panel’s feedback, reactions and perceptions. I will then qualify or endorse my argument according to the evidence presented. It should be noted that the section numbers correspond with those of the Questionnaire, in order to simplify cross-referencing and analysis.

4.1 Activities in the Component Process

4.1.1 Process Modelling

***Q1. Business process modelling improves our understanding of the business processes, their sequence and dependency, the events that initiate them, and the outcome on completion. They provide an easily understood and visible means to communicate this understanding, and should be adopted as a standard part of the development process*.**

Half the Panel agreed, and the other half strongly agreed with this claim.

It was noted that to build any system you have to understand it, and the use of process modelling, even at a high level, is seen as a good basis for doing this. One respondent noted that : “To build any system you have to understand it. This is the basis for gaining that understanding even if at a high level. Common processes have to be identified also, as they may point to common components.”

Another respondent highlighted that: “Modern commercial IT is all about gaining competitive edge and business advantage from employing different and/or slicker processes. Business Process Re-Engineering (BPR) has been tried and in many examples has failed because it tends to be carried out in a vacuum. Integrating the IT design activity with the business process modelling activity provides the right environment for genuine BPR to happen and provides the opportunity for quantum leaps rather than incremental improvement. The biggest failing with a lot of IT projects is that the end result is simply an electronically improved version of old business processes and there has been no significant change made to those processes.”

It should be noted that the proposed process modelling technique caters for ‘solution’ projects (application specific), corporate projects (services to be reused in many applications e.g. a client system), and BPR exercises. It includes the use of swimlanes (as shown in Figure 3-3) which were seen by one respondent as beneficial in BPR exercises.

However, the Panel raised the problem of how the business would react to this approach, given that existing documentation may already be available, and questioned why “existing papers need to be reworked just because they’re not quite in the required format”, and “what the business requires is immediate pay-back”. This is indeed an issue, which I suspect can only be resolved by the education of business users. When changing a system, the opportunity should always be taken to improve things for ‘next time round’, and process models can be prepared by the development team with minimal effort. However, as I stated in Section 3.3.1, while process models are recommended, they are optional, and much will depend on the nature and scale of the development.

Q2. Process models provide an effective means to communicate business processes with the user.

The majority of the Panel agreed, one strongly, and another unsure, that process models provide an effective means of communication with the user. As one respondent suggested: “…What is required is single means of communication between business, IT managers and the various IT developers which explains not only what is done but also why it is to be done”. The question of terminology was raised as a disincentive to the adoption of such approaches by the business, a reaction prompted I suspect, from experience of IT bombarding the business with numerous methods in the past. Again, I would consider that education is key, since the techniques within my proposal represent the maturing of various previous techniques and approaches, and should not be too prone to change in the future.

Q3. Process models provide the means to enable computer and non-computer system processes to be easily identified.

The Panel agreed that process models provide a means (not necessarily the means) for computer and non-computer processes to be identified, but made the point that the identification/assignment of computerized and non-computerized is not necessarily easy. It was also stated that this distinction was not self-evident from my example in Figure 3-3 referenced in Section 3.3.4, but it was meant to be a simple and hypothetical example.

Q4. The processes to be computerised typically identify the Use Cases to be developed. Since the development of each Use Case is an activity on the project plan, this approach more closely integrates the requirements of development with those of project planning and control.

One half of the Panel agreed with this, one strongly, but another disagreed. The question was designed to establish the link between the process model, the use case, and the project plan, and was meant to emphasize the integrated nature of this approach, and the degree of consistency throughout the development process. The response from the Panel ranged from strong agreement, to more common uncertainty of this claim.

Questions were raised about knowing when all use cases were defined. While this is a valid question, at this stage the point is that the main use cases will be identified from the process models (whether for a solution, corporate or BPR project), providing process modelling has been performed. What is not known is how many use cases might be required to ‘extend’ the main use case to cater for exceptions and alternate scenarios, and when common processes can be identified and converted to ‘use’ relationships between use cases. This is the product of more detailed use case modelling in the Elaboration activity.

Where process modelling has not been performed, then the use cases will not be identified from this form of model. However, the technique can still be used, since by nature and definition the use case represents an interaction between a user and a computer system. As Fowler (1997) points out: “In its simplest usage, you capture a use case by talking to your typical users and discussing the various things they might want to do with the system. Take each discrete thing they want to do, give it a name, and write up a short textual description (no more than a few paragraphs)”.

The approach can also apply to the maintenance of an existing system. The starting point in this case is to establish a skeletal use case for the user/computer system interaction, then document the change, and note the links to existing documentation. Future maintenance then starts with the use case, which then becomes a ‘living’ document.

4.1.2 Use Cases

Q5. Use Cases are the way forward to capture business requirements. They provide a simple yet comprehensive framework to capture business requirements in user terms, and provide views of this information at appropriate levels of detail as you progress from Inception to completion of the Elaboration phase. They also provide a single source of reference to the Construction phase for purposes of design, build and test, and provide the means to plan and initiate system testing.

The response from the Panel was divided between agreement, and uncertainty of this claim. While it was agreed that the use case suited the documentation of functional requirements, the suitability to document non-functional requirements (NFRs) was questioned. It was felt that NFRs should be abstracted-out into a separated document, since the audience for these could be specific architectural teams on large projects, where a framework may have to be designed and built for use by developers.

Q6. The Use Case is an effective means of capturing business and system requirements from the user's perspective

All respondents from the Panel agreed or strongly agreed with this statement.

It would be more correct to say that use cases allow you to document rather than 'capture' requirements.

Q7. My interpretation of the contents of the use case is comprehensive?

The response from the Panel was divided between agreement, and uncertainty of this claim. The main issues included the treatment of business rules, exceptions, and the inclusion of screen layouts in the use case, which in themselves do not contradict my claim.

Given that the use case represents the interaction between the user and the computer system, I would restate my claim (supporting Allen and Frost (1998)) that some representation of screen details should be included. The focus is on data, and not on usability. Screen prototypes, whether on paper or produced using a GUI tool, could be included to help visualise what the system is dealing with.

In consideration of exceptions, I would classify types as system (e.g. network error), application (e.g. invalid claim notification date) or framework (e.g. no history found). I would not expect to define system or framework errors in the application use case. The main use case (derived from the process model) defines the principle processing path, and lists the exceptions. The exceptions identify the need for separate use cases to 'extend' the main use case.

The treatment of business rules is a more contentious issue. Strictly speaking, the only rules that should be documented in the use case should refer to the visual i.e. user facing aspect of the system. Business rules governing the behaviour of the system will ultimately be allocated to the business objects responsible for actioning them. The approach I have adopted is to document the rules in the use case as you discover them, but only on a temporary basis, moving them to the appropriate business service definition at a later stage in the Elaboration process. This does involve more work, but caters for the situation where a rules repository is not available. I would accept that, should a rules repository be available, then it would be more appropriate to populate it and cross reference as appropriate.

Q8. Both visual and non-visual business rules should be captured in the Use Case during analysis in the Inception and Elaboration phases

Most of the Panel agreed with this, and one member emphasized that “Gaining as complete an understanding as possible of business rules as early as possible is always key to avoiding problems later in the development cycle”.

However, another strongly disagreed, following up the issue of business rules (which I accept - see final paragraph of Q7 above): “In our experience there are far too many ‘business rules’ (hundreds) and they are far too complicated to go into use cases without overloading the use cases. A repository is the only way to ensure completeness, consistency and traceability”.

Another interesting point was made here, in qualifying agreement with the claim: “Except, that if purely technical and irrelevant to User, then don’t confuse them!”. This reinforces the view that technical NFRs should be defined in a separate document. Given that I have not differentiated between business and technical NFRs, this may be a good criterion to use.

Q9. On completion of Elaboration, all rules should be allocated to services in the Service Package. The Use Case should then be updated, replacing the rule descriptions with cross-references to the Services having responsibility for implementing them

Most of the Panel agreed with this, other than for the recurring issue over the treatment of business rules (see final paragraph – Q7), and in this case the volume of them in certain applications. Again, it is agreed that a separate rules repository should be used.

One respondent suggested that you: “Could take the approach that each stage is a progression and forget what has gone before”. However, I would disagree strongly with this, given that the aim of my Proposal is to propagate ‘living’ documentation, and eradicate the waste from previous methodology where intermediate levels of documentation is produced, only for it to be thrown away later.

Q10. Sequence Diagramming in the Elaboration phase restructures the general process description into the detailed steps of sequence, choice and iteration.. This provides a clear and simple, yet more formal and detailed means of describing the business requirements within the Use Case.

The response from the Panel was divided between agreement, and uncertainty of this claim. Some concern was expressed over the visual representation and notation of the steps, but unfortunately this is the UML presentation style for sequence diagramming.

In agreeing with the need for sequence diagrams, one respondent raised the question of the relationship of requirements to the low-level behaviour of the use case: “Requirements are ‘statements’ which if not ‘met’ by the completed systems will have a detrimental effect on the CBA/Business Case. Low-level behaviour discovered during sequence diagramming is not ‘requirements’ and it is dangerous to let ‘users/sponsors’ believe it is. “ While this is a very valid statement on the verification of business requirements, it is not my intention to present detailed use case steps as such. Rather, the use case documents business requirements at the 'Intent' or 'Description' level early in the Elaboration phase. The steps of the use case are the result of more detailed analysis and design later in the Elaboration phase. The more detailed steps are indeed not part of overall business requirements, but the anticipated implementation of them.

Q11. Have you / your team used Use Cases to capture business requirements? If Yes, what did you like / dislike about them, if not already mentioned above?

All bar one respondent had had exposure to use case driven development. One respondent summed their benefits with word like: "Crucial, Pivotal, Concise, Understandable etc".

Another respondent summarised use cases as: "Formal and standard approach readily understood by all parties. Uses terminology the User understands rather than IT gobbledegook - once that is, they understand the term 'Use-case'. Breaks an overall process into smaller more manageable chunks. Users are becoming more and more computer literate - however a little knowledge can be a dangerous thing sometimes and care is required to get the right level of detail"

While previously mentioned in Q4, it was reiterated that some guidance was needed on the number of use cases that need to be defined. However, there is no easy answer to this question. Fowler and Scott (1997) quoted Ivar Jacobson saying that he would expect 20 use cases for a 10 man-year project, while they themselves defined over 100 for the same size of project. They concluded: "I don't think there is one right answer at the moment, so be flexible and work with whatever seems comfortable."

4.1.3 Domain Service Packages

Q12. The identification of the main component packages during the Inception phase is a useful means of abstracting system detail from the architectural perspective, and illustrating the dependency between components. The intended use of legacy systems can also be identified and tabled at this stage of architectural design.

All members of the Panel agreed with this, with one respondent commenting that: "At a high level can simplify a complex system and give a good overview and clarify understanding and basic assumptions", and another that: "We define both logical and physical sub-systems early on".

One point raised concerned the naming and description of the service packages. Indeed, the name and purpose of the domain service package needs to be clearly defined in the component catalogue. It should also be noted that service packages can, and do, include the services for 'wrapped' legacy systems.

Q13. Reusing this diagram in the Elaboration phase for the allocation of services to Service Packages is a good way to ensure that the architectural scope has not been exceeded.

This point raised a mixed response. One third agreed, half were unsure, and one disagreed. The main issue concerned the definition of scope.

The point here is that the architectural scope is defined in terms of high-level service packages in the Inception phase – see Section 3.3.7 and Figure 3-9. This diagram is then referenced throughout the Elaboration phase as more is learnt about the required system. Ultimately, all required services will be added to the service packages previously defined, resulting in the layered architecture shown in Section 3.4.9 and Figure 3-19. If not, then the original scope has been exceeded. As one respondent indicated: “It ensures that all scope, not just architectural scope, has not been exceeded”.

4.1.4 Allocation of Services (User, Business and Data) to Packages

Q14. Concentrating on Service provision (via Service/Control class) by the Service Package simplifies the structural definition of the system without sacrificing meaning.

This received a mixed agreed and unsure reaction from the Panel. The point here is that it is the service that defines the functionality of the system, the contract to be supported and the focus for design. The service package itself is treated as a black box.

It was questioned whether in a large system this would be meaningful, but I believe the claim holds true. In a larger system you would partition the service packages into a more manageable size based on system sub-functions rather than larger functional areas. This is something that would be considered during the architectural design discussed in Q13 above.

Q15. This is a more effective use of development resource than spending time developing domain class models that require specialised skill to design and understand, which are generally not maintained, and ultimately fall into disuse.

The majority of the Panel were unsure of this claim, with several respondents not understanding the meaning of domain class model, one preferring the definition of such a model, and another on how the maintenance of this type of documentation would be different to any other.

I use the term domain class model to mean the definition of an object oriented class diagram, which represents the objects of the business domain, their behaviour, and the methods or operations that implement this behaviour on the class or its attributes. It is usually produced as part of an analysis exercise to verify understanding. It is then used for input to the design activity, whereupon it is made redundant, since designers may meta-model the requirements, use design patterns or whatever, to produce a physical design. Apart from the need for domain class modelling skills (which are not in plentiful supply), you are producing redundant documentation, which is something my approach avoids.

In terms of the maintenance of this documentation, the use case and service definitions are designed to be ‘living’ documentation, and therefore, the first port of call when a system change is required. In the case of the Year2000 scenario as raised by one respondent, it is unlikely that any documentation would be changed at this level, since this is a change to the definition of the date data type domain definition, and is largely an issue of testing. I am unsure of the situation on the Euro, but I would suspect that some functional, and therefore, documentation changes would be required in this case.

Q16. The Service defines the interface to the component 'black box'. There is no loss of specification detail using the Service Package approach, compared to other approaches which would require the preparation of a domain class model.

The Panel were generally unsure of this claim (possibly because of the lack of domain class modelling knowledge as discussed in Q15 above), apart from one who strongly disagreed on the basis of its complication. If this meant that my approach was complex, this is not the case, since by definition, you define the service contract without worrying about how the internals of the 'black box' are to be designed. As one respondent noted: " I would agree that interfaces need to tightly defined and openly published", and another that: "This has to be detailed, signed off, publicized interface to enable components to be independent and reusable".

4.1.5 Service Definition for Service Packages

Q17. The Service Definition contains the interface specification for the Service Package, and therefore, the Component. It is defined from the detail held in one or more Use Cases. This definition provides an effective framework for the reuse of system services.

The majority of the Panel agreed with this, other than one whom disagreed on the basis that the parameters needed to be more fully defined. I would agree with this, since data definitions should be part of a data dictionary, which is a requirement, but outside of my scope.

Another respondent was keen to see how this scaled to a larger system, but I will conclude with another observation: " It is the best chance we have for reuse."

Q18. The proposed structure for the definition of services is comprehensive

This met with a mixed agreed, unsure and one disagreement based on the parameter definition issue raised in Q17.

4.1.6 Layering of Service Packages

Q19 The Domain Service Package high-level diagram is reused with layering detail added. The layering of Services is a simple way to illustrate the architectural structure of the systems in terms of User, Business and Data requirements.

The Panel agreed and strongly agreed with this. While the question of how it would scale to larger systems was again raised (answered in Q14), it was noted to be a: "Useful architectural communicator". Another respondent suggested that the diagram should be produced automatically with provision for explanatory text tailorable to the target audience, which would be ideal if there was a case tool to support it.

Q20. For technical design and development, teams can be assigned specific areas for development. For example, teams could be created as centres of expertise for Policy, Claims etc. The Service Package view clearly illustrates the services provided by the various packages, and is a useful means of communication.

The Panel generally agreed with this. Teams can indeed be allocated specific areas to support using traditional techniques, but using this approach, responsibility is clearly assigned to the defined architecture (see Section 3.4.9 Figure 3-19).

Two other interesting alternatives were also suggested: “Alternatively team expertise may be developed across type of package, for example, Data Services across Policy, Claims and Administration”, and “Useful means of allocating Business User effort and Business Analysis expertise also”.

4.1.7 Service Package Deployment

Q21. The deployment diagram is a useful means to illustrate the allocation of Services to nodes on a network, and highlight the impact of deployment choices involved in the chosen architecture.

The majority of the Panel agreed with this, other than one whom disagreed, questioning the level of detail and audience (intended to be mainly technical) for the diagram. Again, the diagram was noted as a : “Useful architectural communicator”, but its significance was identified by another respondent: “However, a simple diagram but some big decisions in here. Maybe a lot of work to get this agreed / defined.” This is the main purpose for the diagram: a proposal for discussion and agreement.

The diagram capitalizes on the layered service definitions, which can be allocated to the various client-server architectures without having to be reworked or redefined. The choice and options for technical architecture would not impact this methodology.

4.1.8 Package Class Structure and Component Design

Q22. The Service Definitions are the specifications for the detailed class design and code development. While reference is made to the Use Case for non-functional information, and to the Use Case and Layered Service Package Diagram for contextual information, no other documentation is required.

Comments from the Panel ranged from not sure to strong disagreement with this claim. The main objections concerned the completeness of specification for developers to do their work, and related to this, how errors are handled.

The claim focuses on the functionality of the application, and the definition of the business and data services it is to provide. I consider that the service definition is a complete specification of this functionality, in the same way that a traditional module specification furnishes the developer with a description of what is to be coded and unit tested. In each case, the designer / developer would work within the constraints of the relevant technical architecture, be that at the framework or system architecture level, and capitalize on whatever common components are available to handle any errors appropriate to these levels.

It should be noted that the GUI design and development will be driven from the use case and its visual rules. It is the services questioned above that will be invoked by the screen / window based dialogue that results from the visual design.

In terms of contextual information, it was queried whether the original business requirement is needed at this stage. It may not be, but since it is captured as part of the use case, it is available for reference.

Q23. It is at this stage that the detailed class models are designed to support the functionality of the Service / Control Classes (Façade). The allocation of class design to technical rather than business specialists is a more effective use of resources.

This resulted in a mixed response from unsure to strong disagreement, but with little qualification by the Panel. I would question the respondent whom: “Would prefer to see this as a co-operative joint effort”, since the point of my claim is that you do not need to mix the skill sets involved. I am suggesting that those who have good business and application knowledge concentrate on the application in terms of the services the black-box provides, leaving the content of the black-box to the technical designers and developers who are expert in the potentially diverse and highly specialized technologies involved.

4.1.9 Others Techniques

Q24. Are there any other techniques that you would like to have questioned or seen used?

In terms of the techniques already discussed, apart from the use of a Data Dictionary, it was questioned whether the process model was comprehensive enough. In the absence of evidence of what is missing, and given that it is based on the UML with the addition of elements from the Select Perspective (Allen & Frost, 1998), then I would suggest that it is.

The only technique that was questioned was the use of Task Hierarchy Diagrams, which I suspect is an element of the KPMG Structured User Interface Design (STUDIO) methodology. This is a valid technique for the purpose of task analysis at the detailed level, but I chose to concentrate on process for consistency, rather than engage in the semantic debate over whether a task is a low level process, a process another word for an activity, and a function a high-level process etc.

4.2 Use, Reuse and Traceability

4.2.1 Use Cases

Q25. To gain the maximum advantage from Use Case documentation, it should be considered in the same way as a relational table with a number of available views. For example, a business facing view containing user role and intent at the Inception level, plus Description in the early stages of Elaboration etc. Each view would be tailored for communication with a specific audience.

Most members of the Panel agreed with this although two were unsure. One respondent highlighted the importance of interpretation as follows: “Use cases need to be tailored to the person using them e.g. technical designer. Important that although a different view the interpretation / understanding is exactly the same”. As another respondent commented, however, the full view should be available to everyone if required.

Another respondent questioned: “Can you have general ‘use case’ templates e.g Add Claim etc which can then be modified?”. However, while this might be possible, my approach would be to consider why the two are different, and re-factor the existing one into a corporate ‘Add Claim’ use case which could then be reused / generate reusable components if possible.

Q26. The ability to use one form of documentation i.e. the Use Case for complete and shareable requirements definition is a significant step forward in terms of reusable documentation.

The majority agreed or strongly agreed with this, apart from one unsure response which was not further qualified.

Q27 Continual reference to, and development of, the Use Case throughout the development lifecycle will ensure that the business requirements are always considered and correctly addressed.

The majority agreed, with some unsure of the claim. In one case it was held true: “But only in respect of what is documented within the use case itself”. Another suggested that: “Other business focussed documents may be required”, but did not suggest examples. Concern was also expressed that it: “May become unwieldy the deeper you go into the development lifecycle”, but given that I accept that non-functional requirements will be abstracted into a separate document, this may satisfy the author of this comment.

The question of consistent terminology was again raised in relation to the impact on the business user: “Terminology is often the barrier rather than the actual content. Some business users object automatically to new terms introduced by IT. If the business are happy with the term ‘use case’ then OK. If not then discuss alternatives, decide and use it throughout.” I suspect that again, this is down to the culture of the business and the education of those users involved. Use cases are widely used in the commercial environment, but if the terminology does not fit, then an alternative should be used, and ‘consistently’.

4.2.2 Services

Q28. The Service Package provides the interface definition to the Component Package. This is an appropriate deliverable from business oriented component modelling, and complete specification for the designers and developers of the appropriate middle ground, and enabling unit of reuse?

One third of the Panel agreed, another third were unsure, one disagreed. Only one comment was made, and this quite rightly suggested: “However, there will be other, more granular, levels of reuse”. My claim is that there is greater business benefit to be gained from the reuse of business services, rather than the more design and build oriented components of the development process, essential though that is.

Q29. The implementation independent Service Package is a more effective and enabling unit of reuse than other forms of module or functional specification.

Only one Panel member agreed with this, with the majority unsure. One respondent strongly disagreed, but in doing so stated that this was a: ”Useful starting point and useful architecturally”. I would suggest that this overall feedback is due to lack of exposure to this approach as summed up by another comment: “Haven’t really seen any others aimed at component based development / reuse”.

Q30.The Service concept provides the means to define appropriate services to enable the reuse of a legacy system. While a suitable transaction interface will need to be written inside the Component, the Component is the wrapper for the legacy system. (Note: I would include traditional Cobol and object oriented systems as examples of legacy systems.)

Half the Panel agreed with this claim, with one disagreeing, and one uncertain. In qualifying the non-agreement, it was suggested that: "Can't necessarily reuse a legacy system as usually not object oriented but rather they are convoluted spaghetti?" I will agree that such a system over time may have become spaghetti-like, but not that object oriented systems are a prerequisite for packaging as a component. The point is that irrespective of how the legacy (or heritage) system has been developed, it delivers a set of services. Butler Group (1998b) observed that some of the most successful implementations of componentisation that they had seen did not use any of the new distributed technology, simply traditional environments.

Providing we know what the interface is, then we can develop a wrapper for it, and package the application according to the services it provides. Another advantage is that it is possible to re-use elements of the legacy application that would otherwise be too expensive to redevelop. For example, you may no longer be interested in its data capture and reporting transactions, but very keen to reuse a complex set of algorithms e.g. those of a rating engine, which would be of significant benefit. This could be packaged as a service(s).

4.2.3 Specification and Model Reuse

Q31. The Use Case and related Sequence and Service Package Diagrams, plus the Service Definitions for the Service Packages, are designed to be highly reusable and implementation independent. They achieve the objective of design reuse.

As for Q30, half the Panel agreed with this claim, with one disagreeing, and one uncertain. A comment was made that this might be true in theory, not having seen how this might work in practice. Another, suggested that: "They *enable* achievement of design reuse". I would not argue with this, since this is agreeing that the objective will be met.

Q32. Design and model reuse is key to successful and economic system development. The layered approach to service definition provides a sound architectural context, and greatly facilitates redesign for deployment in multi-tier environments, for example when implementing a thick-client PC system for central offices and a Browser enabled version for remote use.

The majority of the Panel agreed with this, with two uncertain. One respondent felt that because my question embedded a number of statements, that it was difficult to answer, questioning the layering and architectural element. In response, I would say that what I am suggesting is no different to current practices in multi-tier client-server design. What I am doing, is suggesting that the service framework is the optimal means of *defining* the services that have to be provided in the client-server environment, and that this approach is implementation independent, and therefore, reusable. As another respondent noted: "Design and model reuse is certainly a key to future economic development".

Q33. The relationship from Business Requirements to Use Case, Use Case to Service Definition, and Service Definition to Component provides for full requirements traceability.

The majority agreed with this claim. One respondent strongly disagreed on the grounds of complication, which based on previous comments, reflects concerns over the issues of non-functional requirements, dictionary, and the provision of technical and software frameworks, which I have already commented on, and agreed need to be in place.

Another respondent agreed that traceability would be achieved, “But only if maintained”, and indeed this is a key issue. I will answer this together with another point raised: “Change control procedures? What is effort per ‘change’ if documentation updated compared to traditional techniques?” One of the benefits of this approach is that the use case should always be the focus for development. If there is a change in business requirements, the use case is updated, and the required system changes radiate from it. It is the focal point for all business oriented system activity.

Traditionally, a variety of documents may be involved: functional specifications, requirements documents, outline designs, change specifications etc, most of which are only a transitory document to the next stage of development, and are never updated. The use case and service based approach, provided it is controlled and is always the first port of call for a system change, will provide the reusable design documentation that is lacking in previous techniques. The required changes will be made to new versions of these documents. There will still need to be a design document that presents the use cases and system components impacted by the change, but this will mainly involve the assembly of documentation for the purposes of planning and control.

4.3 What is needed to make this work?

Q34. Organisation and culture are often more critical than component technology in making reuse, or new process adoption work.

All of the Panel agreed with this statement, all bar one in strong agreement. As one respondent commented: “Strong management backing is required to make any such change work which is why some prototype projects fail. Lip service is paid rather than whole-scale support”

On the subject of project organization, another commented: “To organise teams around components would be the hardest thing. Projects are usually based around business functions and costed that way. Would need a cultural shift to fully embrace this approach”. However, service packages are usually designed to support business functionality, and there is no reason why this should be a problem.

Q35. A good opportunity is to pilot the component development process together with a new technology project in order to stimulate interest in what some developers might see only as a ‘documentation’ exercise. The process will then achieve a similar profile as the success of the technology project (unless it fails of course!).

This resulted in a mixed response from the Panel, with one third disagreeing, the others unsure, agreeing or strongly agreeing.

One respondent queried whether it had to be a new technology project, and in essence it doesn't. It was then suggested that: "You would have to compare two similar developments one using traditional methodology one using new in order to achieve accurate results." Ideally this is true, but only if you are starting with the same degree of knowledge in both approaches, which is unlikely given that the component approach is new, and as another respondent pointed out: "The benefits would only really come from the next project unless large level of reuse of legacy systems in this pilot project".

While willing to try the approach another respondent suggested: "Should definitely be tried out, but I don't think it will scale". Again, consideration of the non-functional requirements, data dictionary and rule repository should help solve the possible scalability problem.

Another respondent in disagreeing with the pilot approach, suggested another very interesting alternative: "Don't agree - Need to explain Benefits of this approach to those who need to be motivated to use it, rather than using it passively". This raises the cultural issue, and how you motivate people towards documentation rather than technology. However, this was the aim of my claim. If the method is sold as an essential part of the utilization of the technology, then it is more likely to be adopted, and a deliverable from the pilot exercise would be new sample documentation that would be viewed on its merits. The technology and the corresponding methodology would go hand in hand.

Q36. Assuming the pilot is a success, a policy statement is required to institute the process, and the objective of reuse, as a departmental objective. Reuse is not free use, and management commitment and the right cultural conditions are a precondition for its success.

Apart from one unsure comment, the rest of the Panel agreed. The question of cost justification and the need for motivation to use the techniques was again raised as a concern (covered in Q35 above). Another respondent also cited this: "Definitely difficult to quantify and cost. Would need high-level management backing". Such backing is essential, but I believe the merits of any technique have to be demonstrated by example, which is why I suggest the pilot project approach.

Q37. As a minimum, a Service / Service Component / Component asset catalogue is required with search facilities in order to maximise reuse opportunity

One third were unsure, the majority agreed or strongly agreed with the need for an assets catalogue. The question of granularity of component was also raised: "Depends on the level of granularity. Would you expect the catalogue to contain high-level business objects and also detailed VisualAge objects (for example) of which there are thousands". My intention would be to document the service definition and component package, with links to the physical component package. GUI development tools, including VisualAge, usually have highly efficient class browsers for the purpose of managing design objects in their native environment. In this case there is little point in documenting them, and doing so would add little value.

Two respondents questioned the use of the word minimum. By this I meant the availability of some form of repository, but not a full-blown case tool. An integrated case tool would be ideal, providing that all component documentation and relationship requirements are met (while I have seen several, none appear to be designed to do this, rather 'adapted' from previous versions). It was also questioned whether a graph of stages and benefits could be plotted. This is a good point, which I have not included in any of my documentation.

Q38. While a case tool is desirable for modelling purposes, and provides the corresponding rigour, it is feasible to achieve a workable and economic framework using basic tools. For example, a word processor (e.g. Word 6.0), or better (e.g. a Notes Client linked to a standard Notes Library Management Database) can be used for the definition of Business Requirements, Use Cases, and Service Definitions; with a diagramming tool (e.g. Visio) for the models.

This achieved the widest distribution of answers from the Panel, with one in every box from strong disagreement to strong agreement. All the comments were very valid, and ranged from: "Just not possible without a CASE tool", to: "Don't get hung up on fancy Case tools, these are a means to an end only. Attitude is key". Much will depend on the size of the project, and I think it is accepted that while minimal tools can be used, for a larger scale project they are essential. This is effectively summarized by another respondent as follows: "Whereas basic tools could prove adequate for a pilot they could very quickly become laborious without some level of dynamic linking and automatic cross referencing were not available, especially if the documents and diagrams were subject to iterative cycles, being used as working tools for the design process. A custom built Notes based system could prove adequate as a working proof of concept".

Q39. Appropriate training in the approach and techniques is required. Following an introduction, this is best achieved by just-in-time training, and regular consultancy support throughout the project.

All bar one member of the Panel agreed with the concept of just-in-time training and regular consultancy. The objection was that an: "Experienced person is integral part of project team plus need to sell benefits". My suggestion was the minimum, and the ability to be an integral part of the team is more preferable.

The question of communication with the team and "level-setting" was also raised, and indeed, this is an important part of project work. Planning the frequency and level of the training must relate to both the project schedule, and the available skills within the team.

There was also a question about whether the trainer should be a practitioner or trainer. My own preference would be for a practitioner to give the course, providing they had the presentation skills and personality to do this. If not, then they should at least be familiar with the course content in order to relate the demands of the job to the topics covered by the course. If there is a gap, then the course is likely to have been ineffective. If the consultant gave the course, they at least have noone else to blame!

I will conclude this claim, with a quote from one Panel member that who effectively described the critical nature of training: "Education and Training requirements should be no different to any other area. People learn more by 'doing'. According to the ancient saying 'I hear - I forget. I see - I remember. I do - I understand'. Training is all-important as is the cost of making mistakes i.e. the savings of fewer formal courses will be met by the increased cost of re-working when mistakes occur".

4.4 General Feedback

Q40. Do you think that there is merit in the methodology and approach described?

The majority of the Panel agreed that the approach had merit with one unsure response.

One respondent indicated that : “I think it represents a logical view of an approach”, and another observed that:“There is very little available which specifically focuses on the component delivery angle although componentisation receives a lot of column inches at present.”, and this is certainly true and the inspiration for this research and proposal.

The respondent who was unsure commented: “However, I would like to see it tried, but I don’t think it will scale, so may be difficult to prove”. Again, hopefully, I have earlier addressed these concerns in answer to Q35. The same respondent suggested that my approach was “waterfall” which it is not. To clarify this, as I said in Section 3.1, my aims were to focus only on those elements of the lifecycle specific to this approach. I adopted the Objectory Software Development Process as a framework within which to do this, and demonstrate the component-based design activities. The whole process is iterative with detail being progressively added as more knowledge is gained, hence ‘Elaboration’. The incremental nature of the approach is also referenced in relation to ‘Construction’ in Section 3.5.3.

Another respondent, who had previously chosen the ‘comment’ route to feedback, provided an effective summary of concerns about the approach and its merits. I have included the complete text here since it represents a good summary view:

“Before this gets anywhere near the Business Community then IT management need to be certain that it will the bring the <u>company</u> additional benefits above and beyond what is currently in place. Primarily this will mean benefit to IT with secondary benefit to the User community. Both User and IT management are, and indeed must be, sceptical. Too often have there been claims for improved service, improved future benefits, jam tomorrow etc. Fourth Generation Languages were once the panacea together with RAD, prototyping, relational databases, object orientation, BPR…… Having said that, we cannot remain stationary and we need to progress through evolutionary means. Revolution is difficult because of areas mentioned elsewhere e.g. existing systems, existing documentation and all the complexities and inconsistencies which have grown over the years. Whereas it’s nice to start from scratch on a green-field site, most large organisations do not have this luxury. It’s certainly possible in these new sites though as TIS [The Insurance Service - a telephone based insurance service, and Division of Royal & SunAlliance] and others bear witness.

It’s very difficult to prove additional benefit over and above the ‘current’ approach especially since no-one actually knows categorically what the current cost is,/ was / or would be in future. Each step through the process may be entirely logical and may get agreement but that doesn’t imply that the end-result will get acceptance.

A couple of further points:

As mentioned the document is the contract between the various involved parties. This is a good formal way of describing it and describes the process better than ‘specification’. If the use-case were to go out to a 3rd party to code then it’s a contract - or at least part of a contract - so this word should be used internally as well.

It will be interesting to see how this develops and how it copes when confronted with the normal nitty-gritty, error processing and added complexity of a real-life medium scale development rather than a simplified case study. The right tool will be a key factor."

Several of the above points have already been addressed, including cost and benefit, technical and software frameworks, and tools. I will however, address the question of bringing benefit over "…what is currently in place".

I would say that the documentation in place today is the product of any number of piecemeal design and development approaches from the past. My proposals are forward looking, and will maximise design reuse, eliminate the need to produce intermediate 'throw-away' design documentation, and provides the means to integrate old with new. My approach applies to legacy as well as new systems design and development, and offers an evolutionary path, mostly using techniques of the UML, but applied to the service / component environment.

Q41. Are there any other areas that you would like to have seen covered / included?

The Panel suggested the following, which could form the basis for future research – See Q44 also.

- More detail or potential benefits of using methodology e.g. degree of reuse, potential decrease in documentation. Needs to be brought out.
- Approach across legacy applications – we now very rarely get the opportunity to go for "green field" development and hence you need to look at any potential for mapping or translation of what is there already.
- Measurements – How big could it get – how feasible to share amongst teams of developers e.g. for large system like UKRIS how long would it take?
- How would this methodology apply to the more popular development software?
- What is relative saving by using this method?
- Is there a difference between maintenance and development?
- Can user resources be used more than IT with the method?
- Platform independent business process modelling using reusable library?

Q42. Are there any other areas that you felt were unclear, and would have liked me to expand on, or explain better?

One respondent suggested that I went: "…into the methodology quite cold", and another that: "Section 3 needs structuring to make it clearer". I'm not quite sure what I could have done to make it clearer, since I followed a phase and activity approach using the case study for my examples. This may partly be related to the "quite cold" comment, however, since the Panel did not have the benefit of my Section-2 Literature Survey. This is where: Technical Architectures, Technology Application, Methodology and Architectural Requirements were reviewed and discussed.

Business rules was the only other area of interest, but following previous discussion, I agree that they can be more effectively stored in their own repository.

Q43. From what you have read, would you consider a pilot to use this approach?

Only four respondents answered this question of which only one agreed: “Yes, if the opportunity arose and it was accepted as worthwhile by others involved.” This may reflect the caution of the people involved, the issue being the need for demonstrable benefit e.g. “No. You need to present more on the potential savings / benefits from this”, and “No. Would be interested in evaluating from other’s experience, but not as “First penguin in the water”.

Q44. Are there other areas of research related to this topic that you would like to see undertaken?

The Panel suggested the following:

- The follow through into delivery, and methods for identifying and managing the next, more granular, level of reuse, those components which would be reusable across service packs, for example code subroutines.
- What level of detail can be ‘got away with’?
- See Q41 for related topics of interest.

Q45 Any other comments?

The following are direct quotes from the Panel:-

- “I thought your methodology was reasonable, logical and analytical in approach. I’m sure it would represent a sound way of representing developments in terms of a “component approach” – but for me a proposal needs more about how it would work and be managed in practice and the potential benefits.”
- “Interesting. These views need airing and exploring - things are not all rosy at the moment. However I think that most systems are too complicated to be ‘pulled-off’ by this method.”

5. CONCLUSIONS

5.1 Areas covered in the Conclusion

I will first endeavour to summarise how the methodology satisfies the architectural requirements discussed in the Literature Survey of Section-2. I will then briefly comment on the lifecycle used as the phased framework within which the techniques of my approach are defined, before then summarising the key findings from each phase of the methodology, and further consider the feedback from the Panel. I will then summarise what has been achieved and present the revised methodology model (Figure 5-1), before concluding with an indication of future areas for research.

5.2 Meeting the Requirements of the Literature Survey.

The concept of service definition is totally independent to the technology of the deployment environment. The benefit of this approach is that whichever technical architecture, middleware (Section 2.1) and standard (e.g. CORBA, DCOM) is used, the services are only defined once. How they are implemented depends on the specific technology of the environment in which the component is to be deployed. Alan Cameron Wills[3] described components as being "…assembled from kits of parts". For example, internet Web based applications may consist of components built from kits of Java or ActiveX parts which are then plugged into the environment in which they are to be deployed. However, regardless of how many environments the components are deployed into, they are each built from the same service definition. This demonstrates the reusable nature of the service definition, and how the diverse nature and application of technology (Section 2.2) can be more effectively managed.

As a concept of methodology (Section 2.3), the service provides a higher level of abstraction than the object in the object-oriented paradigm, and promises to deliver a more valuable level of reuse (2.3.2 and 2.3.3). It takes account of the essential distinction between interface and implementation, and is considered a better means to achieve an optimal return on investment (2.4.2). My proposal allocates services to the appropriate service package, which treated as a black box, effectively encapsulates the services. In terms of technique, the concept of packaging is grounded in tried and trusted methods (2.3.5), processes (2.3.6) and standards (2.3.7),

In terms of the management of architectural scale (2.4.3), the proposed methodology satisfies the requirements of application architecture at Level-4, and the system architecture at Level-5 as defined by Mowbray & Malveau (1997). The abstraction of requirements also satisfies the need for independent reusable services, which in a distributed system is necessitated by the lack of a single address space (2.4.4). It also caters for the layering of services into user, business and data layers (2.4.5).

It has been noted that while the body of knowledge on object-orientation is large, little has emerged to relate service-based components to application architecture, and the management of the development process (2.4.6). I consider that the proposed approach is a workable option and means of satisfying this requirement. The methodology also satisfies the criteria for "Essential and Desirable Characteristics of Components" as identified by Butler Group (1998a) and specified in Figure 2-6.

[3] Alan Cameron Wills was the speaker at a British Computer Society Methods and Tools Specialist Group meeting in Manchester on the 18th January 1999, on the subject of "Managing a CBD Project". He is also co-author of the D'Souza and Wills (1999) book referenced in the Bibliography.

5.3 Suitability of the Development Lifecycle

The phased approach of the Objectory methodology which I used as my system development framework proved sufficiently flexible to meet my needs, and the phases of Inception, Elaboration, Construction and Transition worked well. Given a lack of feedback to the contrary (subject to it not being a waterfall approach as confirmed in my response to Q40), it can be recommended as a suitable project lifecycle for the service based approach to system design.

5.4 Suitability of the Proposed Methodology Activities

5.4.1 Inception Phase Activities

Process Modelling (Q1-4) was introduced as a method to assist in the identification of use cases, and effectively relate them to business processes. It is not meant to be the only means, and it is perfectly acceptable to define use cases from business requirements documented in a traditional word-processed form. The feedback from the Panel suggested that the business would need evidence of benefits in order to buy-into the technique, but it was agreed that process models provided a single means of communication and understanding, which is essential, between IT and the business.

The Use Case (Q5-11) approach provides a workable means to document requirements throughout the development process, subject to a number of provisions including the treatment of non-functional requirements (e.g. Q5, Q8), data (Q17), and business rules (e.g. Q7, Q9). While the proposed use case template provided a framework to record each of these, it was felt that due to the complexity and number of requirements in each of these areas that a separate repository(s) would provide a more effective solution. Under these circumstances, the use case will remain the key functional document and ‘point’ to the relevant areas in the repository.

The concept of Domain Service Packages (Q12-13) was seen to be a useful architectural abstraction, and a useful means of documenting the scope of a project early in the development process and to verify it later in the Elaboration phase.

5.4.2 Elaboration Phase Activities

The Allocation of Services to Packages (Q14-16) in the Use Case Sequence Diagram met with a mixed response from the Panel, and based on the feedback alone would seem inconclusive. However, the feedback on the related Service Definition for Service Packages (Q17-18) activity contradicted these reservations. The Panel agreed that service definition does effectively define the interface specification (subject to the availability of the data repository), and in the words of one Panel member is the: "…best chance we have for reuse".

I would anticipate that the initial reservations were largely due to the Panel’s lack of understanding of domain class models, and therefore, the point of my claim that this approach should make them redundant. I had anticipated at least an awareness of this activity, but I respect the fact that the individuals who were unaware of the technique were honest enough to say so. If doing this research again, then obviously I would have to stipulate that knowledge of this technique would be required in order to obtain a more informed answer.

The need to tightly defined and openly published interface definitions to maximise reuse was agreed. While it was also questioned whether there was sufficient information for a designer/developer to begin the component design at this point, I would consider that this question has been resolved subject to the availability of the information repository discussed in Section 5-4-1 above.

The Layering of Service Packages[4] (Q19-20) was seen as a useful architectural communicator, providing a simple illustration of the distribution of User, Business and Data services within the Service Packages. It also enables the project team to verify that project scope has not been exceeded by reference to the Service Package Architecture diagram prepared in the Inception phase, and also provides an effective means of allocating work packages to teams.

5.4.3 Construction Phase Activities

The Service Package Deployment (Q21) diagram capitalises on the layered service definitions, which can be allocated to the various client-server architectures without having to be reworked or redefined. The choice and options for technical architecture would not impact this methodology.

The Panel did not agree with my claim that the Use Case and Service Definition documentation was sufficiently complete in order to commence Package Class Structure and Component Design (Q22-23). Concern was also expressed about the treatment of error handling. However, I will restate my claim (from Q22) that this: "… focuses on the functionality of the application, and the definition of the business and data services it is to provide. I consider that the service definition is a complete specification of this functionality, in the same way that a traditional module specification furnishes the developer with a description of what is to be coded and unit tested. In each case, the designer / developer would work within the constraints of the relevant technical architecture, be that at the framework or system architecture level, and capitalize on whatever common components are available to handle any errors appropriate to these levels." I would hope that with this clarification, and the availability of the above repository (see Section 5.4.1 above) that the Panel would reflect differently on their response given the opportunity to do so again.

My claim that it was preferable to allocate the task of class design to technical rather than more business oriented specialists (Q23) also met with an uncertain response, but again I would reiterate that: "…you do not need to mix the skill sets involved. I am suggesting that those who have good business and application knowledge concentrate on the application in terms of the services the black-box provides, leaving the content of the black-box to the technical designers and developers who are expert in the potentially diverse and highly specialized technologies involved."

[4] Note: In Section-3 Figures 3-10, 3-28, 3-31 the "Layer Service Packages" activity is shown solely in the Elaboration phase, whereas it should form part of the bridge into Construction in the same way as 'Define Services for Service Packages.

5.4.4 Use, Reuse and Traceability

The Use Case (Q25-27) is the accepted focal point for design and development activity throughout the systems lifecycle, and provides the link back to business requirements. This is subject to the recommendation that a separate dictionary and rules repository is used (Section 5-4-1). In addition, it was also agreed that separate non-functional requirements documentation will cater for more technical and cross-Use Case requirements, and will run in parallel with the Use Case.

Provided that team best practice disciplines are in place to ensure that this documentation is maintained (Q33) and is the source for requirements change control, then this will deliver the consistency of definition, terminology and understanding that is essential for IT to effectively support the business.

The ability to reuse Services (Q28-30) met with a mixed response, but I suspect this is more to do with lack of exposure to, and the newness of this approach. It was also unfortunate that the Panel was not generally aware of the object design activity, and may, therefore, not have fully appreciated the advantages of service abstraction and definition at the package level (to a service class), rather than at the object level (many objects in the package, each possibly with private and public methods).

Contrary to reservations over the reuse of Services above, the Panel generally agreed that the activities of Specification and Model Reuse (Q31-33) delivered the benefits claimed for an implementation independent design, and that the achievement of design and model reuse was key to future economic development.

5.4.5 What is needed to make this work?

The Panel strongly supported the view that organisation and culture were more important than component technology in making reuse, or new process adoption work (Q34).

In agreeing that the approach should definitely be tried out (Q35), the Panel flagged the need to identify costs and benefits, and motivate people to use it. This is why I suggested that it be piloted together with a new technology project in order to generate interest in the accompanying design techniques. While another suggestion was to parallel the approach with an existing method to fully determine the cost and benefit of the approach, I would consider this not to be effective due to the number of variables involved e.g. knowledge and familiarity with the techniques, technology, business domain etc. The benefit of the approach is that it is forward looking, and any component, including a legacy system or function, can be defined as a black-box that provides services. It is perhaps in the integration of old with new that the advantages of the approach can best be demonstrated.

While a case tool was seen to be desirable (Q38), the Panel supported the view that an appropriate tool should be provided, which need not be overly complex. However, for the techniques to scale, then some tools are seen to be essential, especially the provision of a suitable repository for business rules, and data definitions. Basic workable solutions could be simply developed using database tools such as Microsoft Access, or Lotus Notes.

The need for training was also seen to be key (Q39) to the successful adoption of the approach. While just in time training and consultancy would contribute to this goal, it was also suggested that an experienced person should be part of the team in order to sell the benefits more effectively.

5.4.6 General Feedback

The majority of the Panel agreed that the approach had merit (Q40), and should be tried out. It was evident from the responses of the Panel that they were generally unaware, as is most of the industry, of techniques to focus on component delivery as opposed to those of component build. This has been the inspiration for this research and proposal.

Before being willing to pilot the approach (Q43), the need for demonstrable benefit was again the main issue. However, this view originated from those members of the Panel with least knowledge of domain modelling, and I suspect that if they had, then the benefits of service level design and reuse would have been more evident. However, the points are valid, and the opportunity should be sought to introduce new processes and techniques as new business opportunities arise. As Butler Group (1999) recently observed: "…we are somewhat concerned that the demand for component-based applications will result in a replay of the client/server and early object-oriented eras, where technologists convinced business people that the application of new technology is synonymous with business benefit. One of the important lessons, that we have heard from Forum members, is that solving component technology issues is relatively easy - the key issue is, as always, ensuring alignment with business needs"

I would say that today, e-commerce will drive the business requirements and the opportunity to introduce and demonstrate the benefits of this approach. Current systems documentation tends to be the product of any number of piecemeal design and development approaches from the past. My proposals are forward looking, and will maximise design reuse, eliminate the need to produce intermediate 'throw-away' design documentation, and provides the means to integrate old with new. My approach applies to legacy as well as new systems design and development, and offers an evolutionary path, mostly using techniques of the UML, but applied to the service / component environment.

5.5 Summary

In terms of architecture it has been noted that the design of component-based systems is an emerging discipline (2.3.8). I have set out to define a pragmatic service based architecture for application development that delivers business benefit without the need for significant investment in tools and training, that minimises risk, and maximises the return on investment by providing the ability to package and extend the life of legacy systems. It also satisfies the need to relate each phase of the development process to the business requirements as interpreted through the Use Case.

I consider that I have established through feedback from the Panel that there is merit in the approach, and that subject to the inclusion of a rules and data repository, and the provision for a separate non-functional requirements document, that the approach would be considered for adoption on a pilot basis.

I have updated the diagram showing the activities for the complete service definition methodology in Figure 5-1. This includes those recommendations from the Panel with which I agreed during the Evaluation in Section-4.

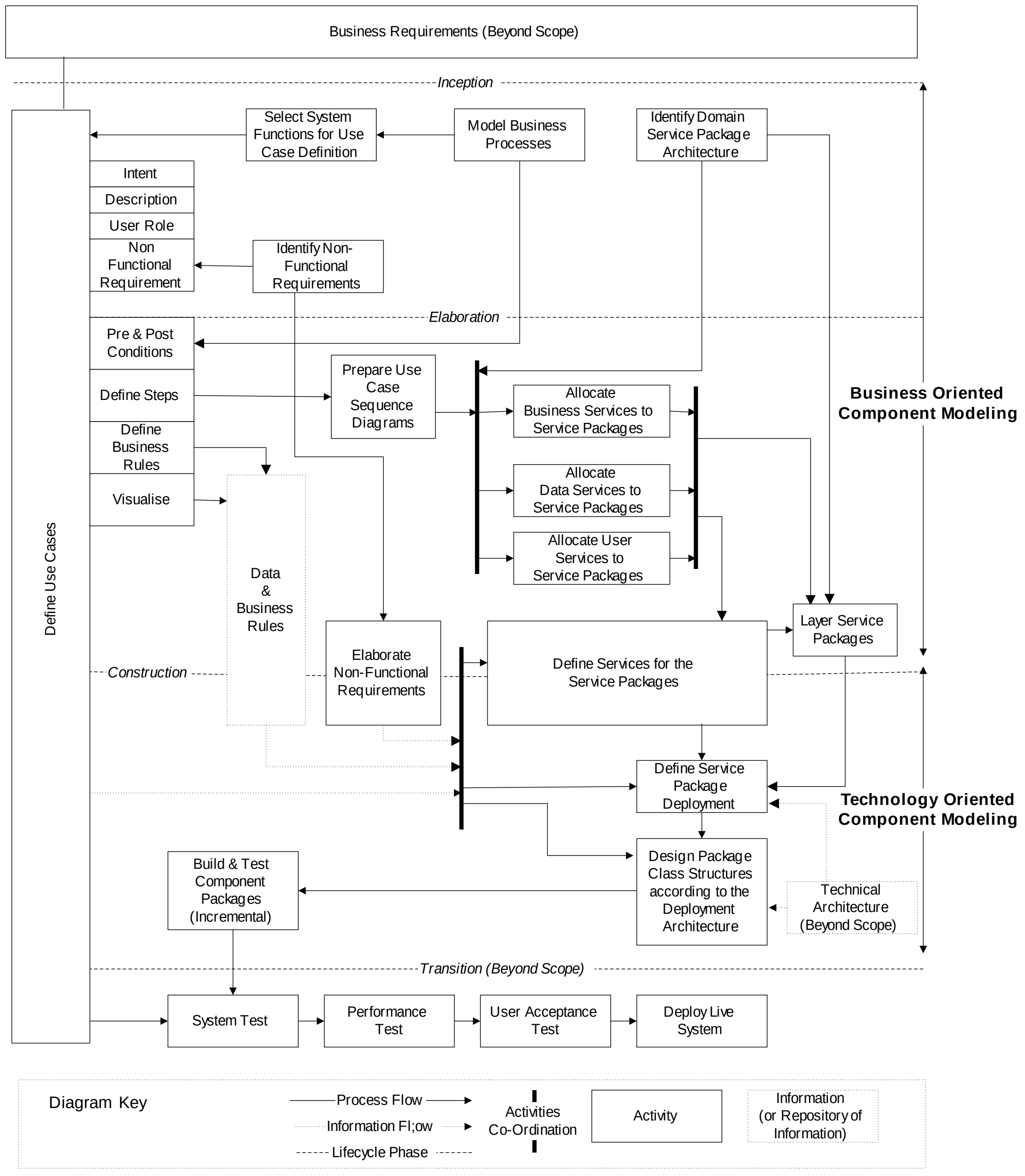

Business Requirements (Beyond Scope)
Inception
Select System Functions for Use Case Definition
Model Business Processes
Identify Domain Service Package Architecture
Intent
Description
User Role
Non Functional Requirement
Identify Non-Functional Requirements
Elaboration
Pre & Post Conditions
Define Steps
Define Business Rules
Visualise
Prepare Use Case Sequence Diagrams
Allocate Business Services to Service Packages
Allocate Data Services to Service Packages
Allocate User Services to Service Packages
Business Oriented Component Modeling
Define Use Cases
Data & Business Rules
Elaborate Non-Functional Requirements
Define Services for the Service Packages
Layer Service Packages
Construction
Define Service Package Deployment
Technology Oriented Component Modeling
Build & Test Component Packages (Incremental)
Design Package Class Structures according to the Deployment Architecture
Technical Architecture (Beyond Scope)
Transition (Beyond Scope)
System Test
Performance Test
User Acceptance Test
Deploy Live System
Diagram Key
Process Flow
Information Fl;ow
Lifecycle Phase
Activities Co-Ordination
Activity
Information (or Repository of Information)

Figure 5-1 Revised Component Delivery Activities – Post Feedback

5.6 Direction of Future Work

5.6.1 The Panel

The Panel identified a number of areas which they would have liked me to have covered in more depth (Q41). I have listed them here since they are also potential areas for further research. They include:

a) More detail on the potential benefits of using the methodology e.g. degree of reuse, potential decrease in documentation
b) Approach across legacy applications
c) Measurements – How big could it get? How feasible to share amongst teams of developers? How long would it take?
d) How would this methodology apply to the more popular development software?
e) What is relative saving by using this method?
f) Is there a difference between maintenance and development?
g) Can user resources be used more than IT with the method?
h) Platform independent business process modelling using reusable library?

In terms of other specific area of future research (Q44), the Panel identified:

i) The follow through into delivery, and methods for identifying and managing the next, more granular, level of reuse, those components which would be reusable across service packs, for example code subroutines.
j) What level of detail can be 'got away with'?

5.6.2 My own interests

The following list identifies the areas that I will focus on in future research:-

a) Integration with existing systems:
   - Adoption of the use case approach as a pre-requisite to service definition and packaging.
   - Introduction of dictionary and rules repository
   - Non-functional requirements template design
   - Legacy wrapping rework and reuse.

b) Organisational Impacts :
   - What is the effect of component and object technology adoption on an organisation?
   - Do you need a reuse architect to channel reuse development/adoption?

c) Reuse:
   - What reuse metrics can be used?
   - How is reuse of a business component measured?
   - Is business reuse a better measure?

d) Business Process Definition:
   - People define business processes in different ways. What can be done to minimise the impact of this, and ensure the reusability of the component?

e) Tools:

Development options for a team based repository and browser to cater for:-

- Use Cases, and allow views to be printed at levels of detail suitable to all parties.
- Domain Packages for identification at the architectural level for reuse.
- Service definitions to facilitate ease of identification and reuse.
- Data / attribute definitions for interpretation and consistency of use.
- Rules, their definition, and classification.

<<<<<<<<<< End of Dissertation – Thank You for Your Interest >>>>>>>>>>

APPENDIX 1 - Technology and Application Notes

A1.1 Java and ActiveX

Both Java and ActiveX promise to integrate Web browsers and servers with legacy applications. They are both examples of component technology, but reflect very different approaches to distributed computing. Java uses machine independent byte-code, whereas ActiveX uses machine dependent binary.

The Java camp is led by Sun, Netscape and Oracle, and is supported by others who wish to see Microsoft's domination of the desktop reduced (Pal, Ring, & Downes, 1996). They support a more open and network centric view where most of the processing is done on servers, and users interact with thin clients: low specification PCs or NCs running a Java enabled browser, intelligence on the client being provided by downloaded Java applets.

A1.1.1 Java

Sun Microsystems Java enables programs to be deployed as Applets which use the Browser as a container (and are therefore limited by Browser capabilities), or as Java Applications which have their own interface and application processing (Pal, Ring, & Downes, 1996). Java combines a set of technologies into a three layered architecture as defined by Pal, Ring, & Downes (1996) comprising:

- an interpreted, object oriented language, translated into Java byte code.
- a virtual machine/runtime interpreter that interprets Java byte code using built in functions. This is done 'on the fly', or can be executed directly on a microprocessor after compilation by a just in time compiler, or on a chip that can directly execute the byte code.
- APIs and class libraries that enable developers to interface the virtual machine with third party systems. This includes Java Beans which are components written in Java that communicate with each other, plus other types of components like ActiveX controls and OpenDoc parts.

Moving forward, development will focus on Java Beans, as componentware typically for use on Internet applications (Pal, Ring, & Downes, 1996). From this perspective Java connections between clients and servers will be handled by:

- Java Remote Method Invocation (RMI) - for Java only environments with no database access.
- Java Interface Definition Language (IDL) - enables Java components to talk to Corba compliant components.
- Java Database Connectivity (JDBC) – a Java API to SQL relational databases

A1.1.2 ActiveX

ActiveX is Microsoft's OLE API based distributed computing and component strategy for the Web (Pal, Ring, & Downes, 1996). While largely confined to the client desktop, it is part of the Microsoft's middleware architecture and interfaces with DCOM. The main components of ActiveX are the Desktop and Server.

The desktop consists of ActiveX Controls, Scripting and COM:

- ActiveX controls are tuned to the requirements of Web applications. Controls can function as clients and servers, and interact with each other via DCOM. They run in ActiveX documents generated by ActiveX enabled applications.
- ActiveX Scripting languages (eg VB Script) can control ActiveX Controls via OLE events. This enables an HTML link (from the Internet Explorer Browser) to open a Word document, and superimpose its icons and menu onto those of the Browser.
- COM (also referred to as ActiveX) The underlying set of services that enables ActiveX controls to interact with each other, and with other components eg Java applets.

The ActiveX Server is a framework of APIs working with its own family of servers to provide a suite of services eg security, transactions etc.

A1.2 Java's Growth

Java has become tremendously popular as a development language based on its 'write once, run anywhere' philosophy. Recent research of 9 companies by International Data Corporation revealed an average payback for Java developments of 14.6 months (Kelly, 1998a). Savings averaging 25% were claimed over comparable C++ developments for multiple platform deployment with coding phase savings averaging nearly 40%. Such savings were also endorsed by services company ObjectShare (which has built systems in C++, Java and Smalltalk) who claimed that it was two to four times cheaper to build a system in Java than C++.

Manufacturers are also adopting the Java standard. Hewlett Packard has changed its strategy and will use Sun's Java Virtual Machine (JVM) for its desktop and server products, as opposed to its own software. They will incorporate Sun's Hot Spot into HP-UX, their version of Unix, and have agreed to support the next release of the Java Development Kit, JDK 1.2 (McGinn,1998).

A1.3 Java and Smartcard Developments

Where Java has been notably successful is in the development of smartcards. International Data Corporation identified smart card technology as one of the fastest growing and dynamic branches of the IT industry in Western Europe today (Tober,1998). The main vendors were expected to increase their revenues by more than 56% in 1998, with 308 million microprocessor based cards expected to be sold in 1998. Mark Stevenson, independent consultant and coauthor of an Ovum report on smart cards, predicted that: "By 2003 there will be around five billion smart cards in use – as many cards as there are people on earth" (Tober,1998).

The two factors most likely to accelerate smartcard usage in the near future are a reduction in costs, and the arrival of Microsoft with an extension to their Visual Studio development environment.

At around £1.50 some consider the current cost too expensive compared to magnetic strip swipe cards of around 50p (Clarke, 1998a). However, as costs reduce, potential applications likely to emerge include: the replacement of paper based security systems (eg fingerprint biometrics), usernames and pass phrases, passports, health care records, and other personal information (Tober, 1998).

Microsoft in 1999 will release SmartCard for Windows, as the programming extension to their Visual Studio development environment. As their rival to Java, this will enable developers to build smartcard applications within the familiar Microsoft development environment (Saran,1998).

A1.4 Examples of Smartcard Applications

Visa Cash - This has been running for12 months in Leeds with 60,000 cards in circulation. It has replaced cash by up to 10% in car parks, and has proved popular in fast food restaurants, sandwich shops, and newsagents (Tober,1998).

London Transport - Awarded the Transys consortium a £1bn 17 year contact to develop an integrated revenue collection service to integrate smart cards for ticketing (Tober, 1998). Six million contactless smart cards will be in circulation once implemented (Clarke, 1998a).

Co-operative Bank - Developing an Independent Savings Account (ISA) scheme (Tober, 1998).

Aberdeen Council - Signed a £500,000 joint venture with smartcard consultancy Smartex to roll out 50 readers and 40,000 cards by April,1999. The council plans to roll out 'Citizen Cards' to all 225,000 residents by March 2001. This will cover a wide range of council services including: leisure and library, school meals, theatre and concessionery bus ticketing, health services, council staff services and retail purchasing by 2004. Digital cryptography will be used to guarantee the integrity of data transmitted from card readers to council servers, and biotechnology like fingerprint recognition is planned within 4 years for added security (Phillips,1998).

A1.5 Examples of Internet Developments

Equitable Life
Equitable Life have implemented a service to allow 500,000 clients to track investments on the Internet (McLeod, 1998). This is believed to be a first in the insurance industry. Facilitieis include the valuation of pension, savings or other life plans; monitoring contributions over any period; the ability to increase monthly payments; change bank details or personal data eg address, or payment date.

The technology includes HTML transactions linking to a Lotus Domino server, with IBM's MQ Series for connection to back office client and administration services.

Lloyds TSB Internet Banking
Lloyds TSB Internet Banking have launched an online service to allow Lloyds customers to pay bills, transfer money between accounts, view statements and adjust standing orders over the Net (Poston, 1998).

Again, the technology uses HTML transactions, and very little will be duplicated (a good example of reuse) when the TSB service is launched in May or June 1999. The same Internet server will be used, with a further proxy server routing transactions to the relevant back office systems. Lloyds initially worked with ICL on a Java solution, but this was dropped due to a slow Java download time, and HTML better suited the banks Microsoft strategy.

APPENDIX 2 - Client-Server Notes

A2.1 Server Options

Whilst the retail, distribution and finance sectors depend heavily on large databases on Unix or IBM mainframes, there appears to be no industry consensus on which platform to standardise. To illustrate this, Lloyds TSB is scrapping OS/2 from 2,700 branches and moving to Windows NT4.0; Royal & Sun Alliance is moving one and a half million life insurance policies off Unisys mainframes onto high-end Sequent Unix servers (rather than NT for enterprise servers); Oddbins and Castrol are moving away from NT based decision support systems to AS/400 systems (Computing, 1998a).

A2.2 Operating Systems

It is generally regarded that Unix will prosper until post Year 2000 before NT5.0 becomes available, with NT posing the biggest threat to Unix in the small to medium sized business sector in the long term (Wallen, 1998). In terms of penetration, Xephon's annual survey of mid-range mainframe sites suggests approximations that Sun Unix has about 27% of the market, against 23% for HP, while IBM's AIX has over 53% (Computing, 1998c).

With the growth in diversity of hardware solutions, even Microsoft has recognised that NT and Unix will coexist, and has developed its own interoperability software called Unix Services (Hunter,1998). Stephen Uden, Microsoft's channel manager was reported to have stated : "…firms are saying it's not a matter of either NT or Unix, but getting them to work better together." Indeed this is the key point.

Standards and protocols are always a problem for users wanting to run legacy Unix based applications from Windows desktops, and security is the biggest issue eg dual userids and passwords may be required. Otherwise, database connectivity (ODBC) and Communications (TCP/IP) standards common to both environments are well established to provide for integration.

A2.3 Client Server Costs

Gartner (1997) estimate that it costs about $10,000 to provide a PC (Windows 95 on a LAN) for a business user. These costs are associated with the planning, acquisition, deployment, operation and replacement of these components. They further define costs as direct and indirect as follows:

Direct (Budgeted) Costs	Indirect (Unbudgeted) Costs
• Hardware & Software • Management (network, system and storage) • Support (help desk) • Development • Communication Fees	• End User IS (cost of users educating & supporting themselves) • Downtime (planned and non-planned)

For a company of 2,500 users this represents an annual cost of $25 million. If only 10 percent of users really need all the PCs flexibility ($2.5 million), then providing a device with simpler but more appropriate functionality would considerably reduce the other $22.5 million. This is has led to the adoption of other devices like the suspect NetPC.

A2.4 Thin Client

Ovum suggests that network computers are supported by those gambling on the supremecy of Java and believe they need a new type of device to support it (Griffiths, 1998). While they predict that sales of the 4 new devices will grow to be 22% of all end-user hardware sales in the next 5 years, it was felt that most organisations were not looking for a replacement for the PC, but to protect their current investments in infrastructure.

The emergence of these devices has resulted from businesses questioning the cost of a PC when used solely to access for business applications and intranets. For this type of usage the PC is an over specified and expensive replacement for a terminal. Ovum (Griffiths,1998) suggests that it is the need to be able to more easily access the company intranet and some Windows applications that will shape the development of the market for end-user devices, this being more significant than consideration of either Java or Web technologies

The Ovum classifications of thin client are described from Griffiths, (1998***)*** below:-

Intranet Terminal

- provides access to a range of applications from a Web browser, but cannot support local processing of Java applications.
- IBM describe it as a replacement for terminals or PCs that are used primarily for accessing applications on various servers, as well as casual browser use.
- only the browser and local operating system are on the hardware
- Wyse, Tektronix, Neoware and IBM are in this market
- 8-64Mb of memory
- 8bit audio
- support for Windows via NT and X-Windows
- IBM 3270 and 5250 terminal emulation (via the browser)

Network Computer

- supports local processing of Java applications loaded from a server
- runs a Web browser, supports Java, and Microsoft Windows applications via NT and X-Windows.
- 8Mb memory and 1Mb video memory
- 16bit audio through a built in speaker
- no disc drive
- suppliers include Sun Microsystems and IBM
- IBM 3270 and 5250 terminal emulation.

General Accident have ordered 4,000 to replace dumb terminals to an IBM mainframe. Most of its key applications being redeveloped in Java by IBM under a Facilities Management contract. Benefits cited include significant hardware cost savings through initial purchase and maintenance, easier installation, faster software updates, improved security support and back up. Overall – lower cost of ownership.

NetPC

Based on a PC but smaller:-

- works across a network with central administration
- floppy disc drive or CD-Rom as options
- promoted by Compaq, Intel and Microsoft.
- Examples include Compaq's Deskpro 4000N NetPC has 166MHz or 233MHz Pentium processor, 16-32Mb of memory and 1.6Gb hard disc

Windows Terminal

Operates using Citrix Winframe, based on its Independent Computing Architecture (Microsoft's own Hydra product will soon be available).

- displays Windows applications held and managed on Microsoft Windows NT computers.
- No local storage or processing

Microsoft's Windows Terminal Server (WTS) is multi-user thin client software running applications centrally on the server, which avoids an expensive fat client architecture. Lloyd's of London are rolling out a back-office application to 20,000 users, and claim significant savings over the cost of a traditional PC, due to the savings that central administration affords (Clarke, 1998c).

A2.5 Technology Adoption

In a recent Compaq UK survey of 384 IT directors (Griffiths, 1998):

87% felt they could save more time and money through better management of their existing PCs than by introducing network computers,

55% did not think network computers would cut their overall IT costs (despite research by Meta Group and Bloor Research which put total cost of ownership savings at between 23% and 30%, and Gartner (Kavanagh, 1998) whose survey of 27 companies proved likewise);

60% of respondents found that the main disadvantage of network computers was the increased dependence on the network and central computers (which had a greater impact on end-users when they went down) than traditional networked PCs with their own processors and storage;

72% did not plan on buying a network computer within the next 2 years (implying 28% might).

APPENDIX 3 - Industry Experience with Distributed Objects

A3.1 Microsoft and DCOM Difficulties

Microsoft themselves have concluded that development using DCOM was currently difficult to use and that most of their customers were currently using XML and the Internet instead (Butler 1998). While the simpler architecture facilitates the requirements for data access using Internet interoperability, DCOM and MTS is still required to manage the integrity of transactions. Apparently COM+ will solve these problems, but this solution will only be available in the millenium.

Numerous deployment issues were identified by IBM including: performance, availability, scalability and security (Butler 1998). These were especially encountered when trying to bring disparate distributed objects together to form an application. Systems management issues are simply compounded by the increased granularity of distributed objects.

A3.2 Distributed Objects : Example Applications

Granville research observed that: “The world is going the way of objects, and financial institutions are leading the way because of their need for globalisation” (Farrell, 1998b).

Example: Dresdner Kleinwort Benson (Investment Bank) adopted the Tibco Enterprise Transaction Express (ETX) CORBA based messaging system for use across 5 world-wide hubs, to provide risk managers real-time financial information across a global network. This has placed the bank at the forefront of object and push technology, and follows the model set by Japanese Investment Bank Nomura International

ETX was adopted for world-wide messaging rather than CORBA, since there were problems running CORBA beyond the LAN. Butler (1998) concluded that this was not to suggest the wrong approach, just that the technology (XML, CORBA and DCOM being simply the transports to make the connections) has yet to mature.

APPENDIX 4 - Questionnaire & Covering Details

(COVER PAGE)

MSc Information Systems

Component Based Development : Methodology Proposal

(Issued in the Form of Section-3 of my Research Dissertation)

Methodology Proposal – Feedback

Author: Paul Botel

Date Proposal Issued	15th March 1999
Date for Return of Feedback	1st April 1999

Feedback Contents

Research Background

With each advance in technology there is a corresponding increase in complexity of design, development, and the management of information systems. We have recently been faced with various client server architectures, beginning with the 2-tier client and server environment. This led to the 3-tier architecture with layers for presentation, business, and data logic, and the thick or thin client choices that have to be made depending on how you want to distribute the system processes.

Developments in networks and middleware (the software layer between the application and the underlying complexities of the network) have made this possible, and opened up opportunities for electronic commerce including the Internet. This has led to the growth in the development of components that can be deployed world-wide onto people’s desktops. There is no longer a single address space.

Current considerations include how these components should be developed. Do you adopt Microsoft’s proprietary DCOM, or the OMG’s (open) CORBA standard for component distribution, and what component language do you use, ActiveX or JavaBeans? While I do not intend to address these choices, this serves to illustrate that the technology is now available to deploy objects where they are needed, and not where traditional host-based solutions would otherwise have dictated. While this is a much more complex development environment, the deployment of components on a distributed and world-wide scale can bring significant advantages to the business.

Changes in technology have been mirrored, albeit neither on the same scale nor with the same degree of success, with changes in methodology. The object-oriented design paradigm has been a great success with its ability to provide controlled public interfaces to private methods for access to an object’s functionality. However, the degree of reuse that this provides is now in question, since such interfaces are constrained by the language of the implementation. While reuse can be achieved at the detailed level of the implementing language e.g. a GUI window or button, business objects cannot be reused outside of their environment. Similarly, object oriented modelling and design tends to target the characteristics of the implementation, and not those of the service or interface.

However, the component has emerged as the most reusable development ‘artefact’. It is at a higher level of abstraction than the object and provides a service according to a defined interface. As such, it is treated as a black box, and is a unit of executable code. It may contain one or more objects, or provide access to a package or legacy system, and even be written in non object-oriented fashion. While the physical implementation of the component may change, the service being provided may not. This separation of component interface from implementation provides the necessary isolation from change.

The question is, what sort of methodology do you adopt to take advantage of this approach, and how can IT departments better position themselves for technology change, while maximising their return on investment in existing systems? While there is no common agreement on such an approach, I consider that a pragmatic method is needed, building on established and proven techniques, but geared towards service based component delivery. My proposal is for such a methodology. It is designed to facilitate the definition, design and reuse of business oriented services for deployment into technology dependant environments. The resulting service-based architecture is designed to be both scalable and extensible to exploit new technologies, and facilitate future development.

Guidelines for Feedback

The statements in the next Section are intended to guide your thoughts in consideration of my proposal. They focus on aspects of the key techniques applicable to the activities of this component based approach.

I have enclosed a printed copy of the Feedback Form for ease of reference, and also a copy on diskette (Word 6.0 format), if you would prefer to return your comments this way.

In completing your feedback, please indicate your strength of reaction to each statement by either ticking or circling your chosen answer, or by ticking or entering (Y)es or (N)o when responding by diskette. There is space below each point for you to qualify your comments where appropriate. If you can afford the time to do this, it will help me to more fully interpret your thoughts.

It is your valued opinion as experienced IT professionals that I need to verify this proposal. Please feel free to be as critical as possible. I do not expect you to agree with everything I suggest!

If you are completing the feedback on paper, you can add separate sheets if there is insufficient room for your comments on the form.

I will consolidate your comments and views and evaluate them in the next stage of the research process. You will be an anonymous contributor from this point of view, but I am sure that you will find the consolidated feedback from your peers to be of interest. My intention is to provide you with a copy of my evaluation of the collated feedback before the end of April.

Thanking you in anticipation of your time and interest.

Activities in the Component Process

Process Modelling

Q1. Business process modelling improves our understanding of the business processes, their sequence and dependency, the events that initiate them, and the outcome on completion. They provide an easily understood and visible means to communicate this understanding, and should be adopted as a standard part of the development process.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q2. Process models provide an effective means to communicate business processes with the user.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q3. Process models provide the means to enable computer and non-computer system processes to be easily identified.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q4. The processes to be computerised typically identify the Use Cases to be developed. Since the development of each Use Case is an activity on the project plan, this approach more closely integrates the requirements of development with those of project planning and control.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Use Cases

Q5. Use Cases are the way forward to capture business requirements. They provide a simple yet comprehensive framework to capture business requirements in user terms, and provide views of this information at appropriate levels of detail as you progress from Inception to completion of the Elaboration phase. They also provide a single source of reference to the Construction phase for purposes of design, build and test, and provide the means to plan and initiate system testing.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q6. The Use Case is an effective means of capturing business and system requirements from the user's perspective.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐

Comments/Observations/Means of Improvement:-

Q7. My interpretation of the contents of the use case is comprehensive?

User Role	Intent	Description	Detailed Steps	Pre / Post Conditions	Non Functional Reqts	Business Rules	Altern ative Course	Sample Screen Layouts	Reqts Refs

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐

Comments/Observations/Means of Improvement:-

Q8. Both visual and non-visual business rules should be captured in the Use Case during analysis in the Inception and Elaboration phases.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐

Comments/Observations/Means of Improvement:-

Q9. On completion of Elaboration, all rules should be allocated to services in the Service Package. The Use Case should then be updated, replacing the rule descriptions with cross-references to the Services having responsibility for implementing them.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐

Comments/Observations/Means of Improvement:-

Q10. Sequence Diagramming in the Elaboration phase restructures the general process description into the detailed steps of sequence, choice and iteration.. This provides a clear and simple, yet more formal and detailed means of describing the business requirements within the Use Case.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐

Comments/Observations/Means of Improvement:-

Q11. Have you / your team used Use Cases to capture business requirements?

Yes	No
☐	☐

If Yes, what did you like / dislike about them, if not already mentioned above :-

Domain Service Packages

Q12. The identification of the main component packages during the Inception phase is a useful means of abstracting system detail from the architectural perspective, and illustrating the dependency between components. The intended use of legacy systems can also be identified and tabled at this stage of architectural design.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q13. Reusing this diagram in the Elaboration phase for the allocation of services to Service Packages is a good way to ensure that the architectural scope has not been exceeded.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Allocation of Services (User, Business and Data) to Packages

Q14. Concentrating on Service provision (via Service/Control class) by the Service Package simplifies the structural definition of the system without sacrificing meaning.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q15. This is a more effective use of development resource than spending time developing domain class models that require specialised skill to design and understand, which are generally not maintained, and ultimately fall into disuse.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q16. The Service defines the interface to the component 'black box'. There is no loss of specification detail using the Service Package approach, compared to other approaches which would require the preparation of a domain class model.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Service Definition for Service Packages

Q17. The Service Definition contains the interface specification for the Service Package, and therefore, the Component. It is defined from the detail held in one or more Use Cases. This definition provides an effective framework for the reuse of system services.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q18. The proposed structure for the definition of services is comprehensive

Stereo type	Intent	Params	Pre / Post Conditions	Specification	Called Operations	Attributes	Non Functional Reqts	Use Case Refs	Reqts Refs

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Layering of Service Packages

Q19 The Domain Service Package high-level diagram is reused with layering detail added. The layering of Services is a simple way to illustrate the architectural structure of the systems in terms of User, Business and Data requirements.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q20. For technical design and development, teams can be assigned specific areas for development. For example, teams could be created as centres of expertise for Policy, Claims etc. The Service Package view clearly illustrates the services provided by the various packages, and is a useful means of communication.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Service Package Deployment

Q21. The deployment diagram is a useful means to illustrate the allocation of Services to nodes on a network, and highlight the impact of deployment choices involved in the chosen architecture.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Package Class Structure and Component Design

Q22. The Service Definitions are the specifications for the detailed class design and code development. While reference is made to the Use Case for non-functional information, and to the Use Case and Layered Service Package Diagram for contextual information, no other documentation is required.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q23. It is at this stage that the detailed class models are designed to support the functionality of the Service / Control Classes (Façade). The allocation of class design to technical rather than business specialists is a more effective use of resources.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Others Techniques

Q24. Are there any other techniques that you would like to have questioned or seen used?

Use, Reuse and Traceability

Use Cases

Q25. To gain the maximum advantage from Use Case documentation, it should be considered in the same way as a relational table with a number of available views. For example, a business facing view containing user role and intent at the Inception level, plus Description in the early stages of Elaboration etc. Each view would be tailored for communication with a specific audience.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q26. The ability to use one form of documentation i.e. the Use Case for complete and shareable requirements definition is a significant step forward in terms of reusable documentation.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q27 Continual reference to, and development of, the Use Case throughout the development lifecycle will ensure that the business requirements are always considered and correctly addressed.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Services

Q28. The Service Package provides the interface definition to the Component Package. This is an appropriate deliverable from business oriented component modelling, and complete specification for the designers and developers of the appropriate middle ground, and enabling unit of reuse?

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q29. The implementation independent Service Package is a more effective and enabling unit of reuse than other forms of module or functional specification.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q30.The Service concept provides the means to define appropriate services to enable the reuse of a legacy system. While a suitable transaction interface will need to be written inside the Component, the Component is the wrapper for the legacy system. (Note: I would include traditional Cobol and object oriented systems as examples of legacy systems.)

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Specification and Model Reuse

Q31. The Use Case and related Sequence and Service Package Diagrams, plus the Service Definitions for the Service Packages, are designed to be highly reusable and implementation independent. They achieve the objective of design reuse.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q32. Design and model reuse is key to successful and economic system development. The layered approach to service definition provides a sound architectural context, and greatly facilitates redesign for deployment in multi-tier environments, for example when implementing a thick-client PC system for central offices and a Browser enabled version for remote use.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q33. The relationship from Business Requirements to Use Case, Use Case to Service Definition, and Service Definition to Component provides for full requirements traceability.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

What is needed to make this work?

Q34. Organisation and cultural are often more critical than component technology in making reuse, or new process adoption work.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q35. A good opportunity is to pilot the component development process together with a new technology project in order to stimulate interest in what some developers might see only as a ‘documentation’ exercise. The process will then achieve a similar profile as the success of the technology project (unless it fails of course!).

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q36. Assuming the pilot is a success, a policy statement is required to institute the process, and the objective of reuse, as a departmental objective. Reuse is not free use, and management commitment and the right cultural conditions are a precondition for its success.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q37. As a minimum, a Service / Service Component / Component asset catalogue is required with search facilities in order to maximise reuse opportunity.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q38. While a case tool is desirable for modelling purposes, and provides the corresponding rigour, it is feasible to achieve a workable and economic framework using basic tools. For example, a word processor (e.g. Word 6.0), or better (e.g. a Notes Client linked to a standard Notes Library Management Database) can be used for the definition of Business Requirements, Use Cases, and Service Definitions; with a diagramming tool (e.g. Visio) for the models.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q39. Appropriate training in the approach and techniques is required. Following an introduction, this is best achieved by just-in-time training, and regular consultancy support throughout the project.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

General Feedback

Q40. Do you think that there is merit in the methodology and approach described?

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree
☐	☐	☐	☐	☐
Comments/Observations/Means of Improvement:-				

Q41. Are there any other areas that you would like to have seen covered / included?

Q42. Are there any other areas that you felt were unclear, and would have liked me to expand on, or explain better?

Q43 From what you have read, would you consider a pilot to use this approach?

Q44 Are there other areas of research related to this topic that you would like to see undertaken?

Q45 Any other comments?

-- End of Feedback ---

Reviewer Details

Please complete the following details by means of identification and information:

*Reviewed By (Signature)	
*Name (Please Print)	
Job Title	
Company	
Division	
Number of Years in IT	
Number of Years in IT Consultancy, Project, or DepartmentalManagement	
Date Feedback Completed	

** If submitting feedback on diskette, just print your name*

----------------------------------- End of Report : Thank You -------------------------------

APPENDIX 5 – Consolidated Questionnaire Feedback

A5-4. Evaluation

A5-4.1 Activities in the Component Process

A5-4.1.1 Process Modelling

Q1. Business process modelling improves our understanding of the business processes, their sequence and dependency, the events that initiate them, and the outcome on completion. They provide an easily understood and visible means to communicate this understanding, and should be adopted as a standard part of the development process.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
			3	3	

Agree
- True but we should not re-invent the wheel. The business tend not to be interested in what they call ‘IT documentation’. If documentation already exists because IT is to automate an existing clerical process then why should the business pay to re-work existing papers just because it’s not quite in the required format. It may give dividends later down the line but what is required is immediate pay-back.
- Business audience often ‘turned off’ by complex methods - is there a better way?

Strongly Agree
- Modern commercial IT is all about gaining competitive edge and business advantage from employing different and/or slicker processes. BPR has been tried and in many examples has failed because it tends to be carried out in a vacuum. Integrating the IT design activity with the business process modelling activity provides the right environment for genuine BPR to happen and provides the opportunity for quantum leaps rather than incremental improvement. The biggest failing with a lot of IT projects is that the end result is simply an electronically improved version of old business processes and there has been no significant change made to those processes.

- To build any system you have to understand it. This is the basis for gaining that understanding even if at a high level. Common processes have to be identified also, as they may point to common components.

- The term ‘development process’ is usually used to mean the systems development process, whereas BPM is most definitely part of the wider process of business improvement – of which there is usually a systems element.

- There is very much a chicken-and-egg issue with BPM/BPR and (OO) development, which has not yet been fully resolved.

- We believe that the swim-lane notation is an effective means to BPR. If, however, there were to be no BPR-ing done, we would probably just opt for activity diagrams

Q2. Process models provide an effective means to communicate business processes with the user.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		1	4	1	

Not Sure
- Agree – But need to be free of terminology (which makes translation to methodology difficult). Take care of GUI presentation – as you point out, user gets misguided into viewing a system. Also beware of presentation of new / any terminology.

Agree

- This process is allocated different names in a variety of different books. The name is irrelevant. What is required is a single means of communication between business, IT managers and the various IT developers which explains not only what is done but also why it is to be done. It has been said that "A picture is worth 1000 words". True - but not if it takes 2000 words to explain it.
- If done properly. They can be built with or by the user. The actual format these days is irrelevant as long as people understand them.
- Users should be part of developing business process models, not just recipients of documentation
- The phrase 'the user' can be viewed as a) emotive b) condescending c)misleading
- Business audience often 'turned off' by complex methods - is there a better way?

Q3. Process models provide the means to enable computer and non-computer system processes to be easily identified.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
			5		1

Agree
- This is overstating the case, process models provide the means for *candidate* computer/non-computer systems processess to be identified.
- It is not necessarily easy !
- The supporting example does not 'support'. It is not *obvious* which should be not computerised. From a BPR point of view, authorising and dispatching a cheque would both be very valid candidates for computerisation.
- MM - But, this is a circular statement? i.e if used in that way then they will, easily compared to what alternative?

Comments
- Being picky here. They provide 'a means' not 'the means'

Q4. The processes to be computerised typically identify the Use Cases to be developed. Since the development of each Use Case is an activity on the project plan, this approach more closely integrates the requirements of development with those of project planning and control.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1	3		1	1

Disagree
- Again, this is only because you have predetermined that 'Processes' (Use Cases) will drive the perfect plan, it also assumes that requirements are 'process oriented', which is not the case for a lot of development (e.g. maintenance work, specific small scale enhancements etc – most work in the industry is of this nature). I am however, in favour of process driven development for New Systems along the lines of your proposal.

Not Sure
- Have all use cases been defined? How do you know when they have? When do you stop defining them? Scoping is a problem. Also, can find use-cases are driven by current processes and not re-engineered processes which may be the aim of the project.

Strongly Agree
- Only issue – 'more closely' than what ?

Comments
- Theoretically true. However, the premise may not be true for all sizes of project

A5-4.1.2 Use Cases

Q5. Use Cases are the way forward to capture business requirements. They provide a simple yet comprehensive framework to capture business requirements in user terms, and provide views of this information at appropriate levels of detail as you progress from Inception to completion of the Elaboration phase. They also provide a single source of reference to the Construction phase for purposes of design, build and test, and provide the means to plan and initiate system testing.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		3	2		1

Not Sure

- They don’t do all of the above by definition – the above seems to be a big claim.
- Not sure they cover all types of business requirement. Very good for functional / process requirements but more strategic / abstract ones may be not relevant e.g. product cross-selling from call centres.
- Agree can be used in this way. Approach seems logical and practical. What other methods are there?

Agree

- In the main, use cases capture *functional* business requirements
- NFRs can/should normally be abstracted-out into a separate document (this also helps since functional reqs go to analysis/design, NFR reqs normally go to architects etc,

Q6. The Use Case is an effective means of capturing business and system requirements from the user’s perspective.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
			5		1

Agree

- Easily understood. Simple.
- I would say that use cases document rather than capture
- Again, in the main they capture functional requirements
- But should be in context of overall high level process?

Q7. My interpretation of the contents of the use case is comprehensive?

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		2	3		1

Not Sure

- We find *intent* is better expressed as *goal*
- NFRs are initially captured here but do not end up here
- Screen Layouts should not be part of a use case
- What other requirements are to be referred to ? Use Cases plus NFRs should be it
- Error messages / what system does if processing is unsuccessful?

Agree

- I had initially thought that more needed to be recorded about the User, for example the profile of the User (skills, authority etc.), but this could just as easily form part of the User Role or even lead to a Business Rule.
- How detailed do you go with business rules? Does this form the basis of a module / program spec? If so could be very large in a complex process. What about exceptions? These can hide lots of work.

Q8. Both visual and non-visual business rules should be captured in the Use Case during analysis in the Inception and Elaboration phases.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
1		1	3		1

Strongly Disagree

- In our experience there are far too many 'business rules' (hundreds) and they are far too complicated to go into use cases without overloading the use cases
- A repository is the only way to ensure completeness, consistency and traceability.

Agree

- Gaining as complete an understanding as possible of business rules as early as possible is always key to avoiding problems later in the development cycle.
- Agree - Except, that if purely technical and irrelevant to User, then don't confuse them!

Q9. On completion of Elaboration, all rules should be allocated to services in the Service Package. The Use Case should then be updated, replacing the rule descriptions with cross-references to the Services having responsibility for implementing them

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1	1	3		1

Disagree

- There are just far too many of them and they are far too important

Not Sure

- Could end up continually updating documentation. Could take the approach that each stage is a progression and forget what has gone before.

Agree

- Could you have a 'dictionary of rules' i.e. reuse the rules / define once only.

Q10. Sequence Diagramming in the Elaboration phase restructures the general process description into the detailed steps of sequence, choice and iteration.. This provides a clear and simple, yet more formal and detailed means of describing the business requirements within the Use Case.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		2	3		1

Not Sure

- Need a fuller example than Fig 3-14 to see this.
- Requirements are 'statements' which if not 'met' by the completed systems will have a detrimental effect on the CBA/Business Case. Low-level behaviour discovered during sequence diagramming is not 'requirements' and it is dangerous to let 'users/sponsors' believe it is.
- That said, I agree with the need for Sds

Agree

- Although agreeing with the need, I'm less sure about the notation and visual side of the diagrams as presented.

Q11. Have you / your team used Use Cases to capture business requirements? If Yes, what did you like / dislike about them, if not already mentioned above?

Yes	No	Don't Know

4	1	1

Yes
- Formal and standard approach readily understood by all parties. Uses terminology the User understands rather than IT gobbledegook - once that is, they understand the term 'Use-case'. Breaks an overall process into smaller more manageable chunks.
- Users are becoming more and more computer literate - however a little knowledge can be a dangerous thing sometimes and care is required to get the right level of detail
- Needed more direction on how many to do and what should be in them. Also, how best to apply the contents of them to the next stage.
- Crucial, Pivotal, Concise, Understandable etc

Don't Know
- Ask the team! I've used something similar myself, but not as detailed.

A5-4.1.3 Domain Service Packages

Q12. The identification of the main component packages during the Inception phase is a useful means of abstracting system detail from the architectural perspective, and illustrating the dependency between components. The intended use of legacy systems can also be identified and tabled at this stage of architectural design.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
			6		

Agree
- Care however needs to be taken or precise rules need to be in place. Taking a trivial example 'verify cover' is a Claim process but probably (?) resides in the Policy DSP.
- Secondly, 'Administration Services' doesn't mean a lot really whereas Policy, Account Claim are readily understood. Thirdly, how is this step to be achieved. Fourthly, what controls are in place initially e.g. cover can mean different things to different people - is it household cover, building cover, building cover for £100K. Areas like this are always good for discussion with Business Colleagues in order to achieve a common understanding amongst the business community before trying to involve IT personnel
- Need to define here legacy systems. I would hope that the use of this methodology across all systems (legacy and green field) is viable.
- At a high level can simplify a complex system and give a good overview and clarify understanding and basic assumptions.
- We define both logical and physical sub-systems early on

Q13. Reusing this diagram in the Elaboration phase for the allocation of services to Service Packages is a good way to ensure that the architectural scope has not been exceeded.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1	3	2		

Disagree
- how does it achieve this ? (ii) see above

Not Sure
- Define architectural scope
- Where has architectural scope been constrained / defined?

- It ensures that all scope, not just architectural scope, has not been exceeded

A5-4.1.4 Allocation of Services (User, Business and Data) to Packages

Q14. Concentrating on Service provision (via Service/Control class) by the Service Package simplifies the structural definition of the system without sacrificing meaning.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		3	2		1

Not Sure
- Looks simple in the basic examples you have given, but in a complex / large system would this be workable / meaningful.

Comments
- Simplifies ? the structural definition. How ? If this is saying that you can group chunks together and show interactions between chunks rather than each sub-chunk thereby simplifying the diagram - then OK.

Q15. This is a more effective use of development resource than spending time developing domain class models that require specialised skill to design and understand, which are generally not maintained, and ultimately fall into disuse.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1	4			1

Disagree
- This work still has to be done. Being work-product oriented rather than phase-based helps.
- It is easier to get a half-decent traditional analyst to think OO and deliver a sufficiently good Analysis Object Model (it doesn't have to be perfect – it wil be changed in design), than it is to get a 'highly-skilled' OO designer to understand and analyse all the problem-domain

Not Sure
- Isn't the same true in this approach albeit possibly in different areas. "Maintenance of the whole" is certainly a key area i.e. all documentation and use-cases and code and….. Consider an issue akin to the Y2K scenario where all code is being updated by a 3rd party package. Existing documentation would be left as is. (This may be OK since there is no change of functionality but how can you be sure ? What about the Euro where external subroutines with their own style of documentation may be brought to bear ?)
- Not familiar with domain class models.
- I haven't had much experience of domain class models
- Sorry, don't really understand "domain class model"

Q16. The Service defines the interface to the component 'black box'. There is no loss of specification detail using the Service Package approach, compared to other approaches which would require the preparation of a domain class model.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
1		5			

Strongly Disagree
- It is just too complicated

Not Sure
- I would agree that interfaces need to tightly defined and openly published.
- Again, no experience here. You could do with some feedback from people using other methodologies.
- This has to be detailed, signed off, publicised interface to enable components to be independent and reusable.

A5-4.1.5 Service Definition for Service Packages

Q17. The Service Definition contains the interface specification for the Service Package, and therefore, the Component. It is defined from the detail held in one or more Use Cases. This definition provides an effective framework for the reuse of system services.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1		4		1

Disagree
- From experience, needs lot more detail in the parameter section. Data definitions alone do not provide enough information to avoid misunderstanding. The data contents have to be explained.

Agree
- I would be concerned that they may end up containing a lot of detail - I would like to see how it looks in a fuller example.
- It is the best chance we have for reuse.

Q18. The proposed structure for the definition of services is comprehensive

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1	2	2		1

Disagree
- Parameter definition is the concern. See Q17

A5-4.1.6 Layering of Service Packages

Q19 The Domain Service Package high-level diagram is reused with layering detail added. The layering of Services is a simple way to illustrate the architectural structure of the systems in terms of User, Business and Data requirements.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly	Comments
			6		

Agree
- A general comment: depending upon who the two or more parties are in a conversation then different levels of detail (or views) will be required. The diagrams and words need therefore to be automatically produced.
- Again, some reservations about how this would scale up in a large systems development.
- Useful architectural communicator

Q20. For technical design and development, teams can be assigned specific areas for development. For example, teams could be created as centres of expertise for Policy, Claims etc. The Service Package view clearly illustrates the services provided by the various packages, and is a useful means of communication.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly	Comments
			4		2

Agree
- It may be that due to the size and number of services it may be that teams become more focused, for example be centres of expertise for the Policy Data Services. Alternatively team expertise may be developed across type of package, for example, Data Services across Policy, Claims and Administration.
- Useful means of allocating Business User effort and Business Analysis expertise also.

Comments
- Yes - but so what. That can and does happen in more traditional approaches. Are you creating expertise in functions involving policy even though they are claim related ?
- Agree for development, not necessarily for high-level design

A5-4.1.7 Service Package Deployment

Q21. The deployment diagram is a useful means to illustrate the allocation of Services to nodes on a network, and highlight the impact of deployment choices involved in the chosen architecture.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly	Comments
	1		5		

Disagree
- I'm not really sure what level of detail you need here, but it seems a little high level. What is the audience for your diagram? (IT / Technical / Business)

Agree
- However, a simple diagram but some big decisions in here. Maybe a lot of work to get this agreed / defined.
- Useful architectural communicator.
- What about guidelines for deployment e.g. Tier1,2,3 in what circumstances, which characteristics at each layer / level?

A5-4.1.8 Package Class Structure and Component Design

Q22. The Service Definitions are the specifications for the detailed class design and code development. While reference is made to the Use Case for non-functional information, and to the Use Case and Layered Service Package Diagram for contextual information, no other documentation is required.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly	Comments
2	1	1			2

Strongly Disagree
- Big claim.
- Everything is too complicated – unless you intend for the designers to then do massive amounts of analysis

Disagree
- What about exceptions, error handling, and all the other 'nitty gritty' bits of detail to complete a program?

Not Sure
- Would Business requirements documents still be required at this stage for reference?

Comments
- Speak to developers!

Q23. It is at this stage that the detailed class models are designed to support the functionality of the Service / Control Classes (Façade). The allocation of class design to technical rather than business specialists is a more effective use of resources.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly	Comments
1	1	2			2

Strongly Disagree

- There is generally a problem that business specialists are not IT-cognisent and do not question or analyse – it is not that that are trad IT-ers who are OO-luddites

Not Sure
- Would prefer to see this as a co-operative joint effort.

A5-4.1.9 Others Techniques

Q24. Are there any other techniques that you would like to have questioned or seen used?

- Not sure that the level of process modelling example is sophisticated enough for serious use.
- Only a mention of process modelling techniques e.g. THD's
- Concept of 'Data Dictionary' applied to Business Processes and System Component Design.

A5-4.2 Use, Reuse and Traceability

A5-4.2.1 Use Cases

Q25. To gain the maximum advantage from Use Case documentation, it should be considered in the same way as a relational table with a number of available views. For example, a business facing view containing user role and intent at the Inception level, plus Description in the early stages of Elaboration etc. Each view would be tailored for communication with a specific audience.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly	Comments
		2	3		

Agree
- see earlier answer. All need to be aware however that their 'view' is but a subset of the whole and that further information is available if required. Problems and misunderstandings will arise if this is not the case e.g. IT will refer to 'use case' and mean the full set whereas an end-user will refer to use-case and mean only their view. Furthermore, just as there is more than one type of IT person e.g. designer, developer, DBA etc so there are different levels/roles of User personnel e.g. Head Office, customer facing ……
- Use cases need to be tailored to the person using them e.g. technical designer. Important that although a different view the interpretation / understanding is exactly the same.
- Can you have general 'use case' templates e.g Add Claim etc which can then be modified?

Q26. The ability to use one form of documentation i.e. the Use Case for complete and shareable requirements definition is a significant step forward in terms of reusable documentation.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		1	4	1	

Strongly Agree
- Strongly Agree

Q27 Continual reference to, and development of, the Use Case throughout the development lifecycle will ensure that the business requirements are always considered and correctly addressed.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		2	3		1

Not Sure
- Other business focussed documents may be required.

- May become unwieldy the deeper you go into the development lifecycle.

Agree
- Agree (In theory)
- To a certain extent, but it won’t happen on its own
- But only in respect of what is documented within the use case itself

Comments
- Terminology is often the barrier rather than the actual content. Some business users object automatically to new terms introduced by IT. If the business are happy with the term ‘use case’ then OK. If not then discuss alternatives, decide and use it throughout

A5-4.2.2 Services

Q28. The Service Package provides the interface definition to the Component Package. This is an appropriate deliverable from business oriented component modelling, and complete specification for the designers and developers of the appropriate middle ground, and enabling unit of reuse?

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1	2	2		1

Agree
- However, there will be other, more granular, levels of reuse.

Q29. The implementation independent Service Package is a more effective and enabling unit of reuse than other forms of module or functional specification.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
1		2	1		1

Strongly Disagree
- Useful starting point and useful architecturally

Not Sure
- Haven’t really seen any others aimed at component based development / reuse.

Q30.The Service concept provides the means to define appropriate services to enable the reuse of a legacy system. While a suitable transaction interface will need to be written inside the Component, the Component is the wrapper for the legacy system. (Note: I would include traditional Cobol and object oriented systems as examples of legacy systems.)

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1	1	3		1

Disagree
- Can’t necessarily reuse a legacy system as usually not objects oriented but rather they are convoluted spaghetti?

A5-4.2.3 Specification and Model Reuse

Q31. The Use Case and related Sequence and Service Package Diagrams, plus the Service Definitions for the Service Packages, are designed to be highly reusable and implementation independent. They achieve the objective of design reuse.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1	1	2		1

Disagree
- They *enable* achievement of design reuse

Agree

- In theory.

Q32. Design and model reuse is key to successful and economic system development. The layered approach to service definition provides a sound architectural context, and greatly facilitates redesign for deployment in multi-tier environments, for example when implementing a thick-client PC system for central offices and a Browser enabled version for remote use.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		2	3		1

Not Sure

- Paul your questions are quite detailed and embed a number of statements and difficult therefore to answer: "Design and model reuse is key to successful and economic system development." – Yes partly. "The layered approach to service definition provides a sound architectural context" – Yes in theory. "…for deployment in multi-tier environments" – Not sure. "…for example when implementing a thick-client PC system for central offices and a Browser enabled version for remote use." – Potentially.

Agree

- Superficially yes.

Comments

- design and model reuse is certainly a key to future economic development

Q33. The relationship from Business Requirements to Use Case, Use Case to Service Definition, and Service Definition to Component provides for full requirements traceability.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
1			4		1

Agree

- But only if maintained.
- Change control procedures? What is effort per 'change' if documentation updated compared to traditional techniques?

Strongly Disagree

- Too complicated

A5-4.3 What is needed to make this work?

Q34. Organisation and culture are often more critical than component technology in making reuse, or new process adoption work.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
			1	4	1

Strongly Agree

- To organise teams around components would be the hardest thing. Projects are usually based around business functions and costed that way. Would need a cultural shift to fully embrace this approach.

Comments

- Strong management backing is required to make any such change work which is why some prototype projects fail. Lip service is paid rather than whole-scale support

Q35. A good opportunity is to pilot the component development process together with a new technology project in order to stimulate interest in what some developers might see only as a 'documentation' exercise. The process will then achieve a similar profile as the success of the technology project (unless it fails of course!).

Strongly Disagree	Disagree	Not Sure	Agree	Strongly	Comments
	3	1	1	1	1

Disagree
- You would have to compare two similar developments one using traditional methodology one using new in order to achieve accurate results.
- "…a new technology" Does it have to be new?
- The benefits would only really come from the next project unless large level of reuse of legacy systems in this pilot project.
- Don't agree - Need to explain Benefits of this approach to those who need to be motivated to use it, rather than using it passively

Agree
- Should definitely be tried out, but I don't think it will scale.

Q36. Assuming the pilot is a success, a policy statement is required to institute the process, and the objective of reuse, as a departmental objective. Reuse is not free use, and management commitment and the right cultural conditions are a precondition for its success.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		1	4		1

Agree
- Agree. Reuse
- Definitely difficult to quantify and cost. Would need high-level management backing.

Comments
- Need to cost justify the approach so that Developers will Want to use the technique.

Q37. As a minimum, a Service / Service Component / Component asset catalogue is required with search facilities in order to maximise reuse opportunity.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		2	2	1	1

Not Sure
- Is there a min / max ? Graph of stage / benefit ?
- Depends on the level of granularity. Would you expect the catalogue to contain high-level business objects and also detailed VisualAge objects (for example) of which there are thousands.

Comments
- As a MINIMUM ?

Q38. While a case tool is desirable for modelling purposes, and provides the corresponding rigour, it is feasible to achieve a workable and economic framework using basic tools. For example, a word processor (e.g. Word 6.0), or better (e.g. a Notes Client linked to a standard Notes Library Management Database) can be used for the definition of Business Requirements, Use Cases, and Service Definitions; with a diagramming tool (e.g. Visio) for the models.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
1	1	1	1	1	1

Strongly Disagree
- Just not possible without a CASE tool.

- Too complicated.

Disagree
- Whereas basic tools could prove adequate for a pilot they could very quickly become laborious without some level of dynamic linking and automatic cross referencing were not available, especially if the documents and diagrams were subject to iterative cycles, being used as working tools for the design process. A custom built Notes based system could prove adequate as a working proof of concept

Not Sure
- Provided all have access and can share and you rigorously impose use.

Strongly Agree
- Don’t get hung up on fancy Case tools, these are a means to an end only. Attitude is key.

Comments
- The key words here are appropriate and standard. The alternatives may be feasible for small projects but are unlikely to be so for large scale developments.

Q39. Appropriate training in the approach and techniques is required. Following an introduction, this is best achieved by just-in-time training, and regular consultancy support throughout the project.

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
	1		4		

Disagree
- Experienced person is integral part of project team plus need to sell benefits.

Agree
- Education and Training requirements should be no different to any other area. People learn more by “doing”. According to the ancient saying “I hear - I forget. I see - I remember. I do - I understand”. Training is all-important as is the cost of making mistakes. I.e. the savings of fewer formal courses will be met by the increased cost of re-working when mistakes occur.
- Should the people giving the training be qualified trainers or qualified practitioners
- “consultancy support” very important. Not many people have experience of doing this.
- On-going communication and level-setting of the ‘great unwashed’ is also required

A5-4.4 General Feedback

Q40. Do you think that there is merit in the methodology and approach described?

Strongly Disagree	Disagree	Not Sure	Agree	Strongly Agree	Comments
		1	4		1

Not Sure
- Generally this approach assumes ‘waterfall’ How would it cope with iteration, and with changes
- However, I would like to see it tried, but I don’t think it will scale, so may be difficult to prove

Agree
- I think it represents a logical view of an approach.
- There is very little available which specifically focuses on the component delivery angle although componentisation receives a lot of column inches at present.

Comments
- Before this gets anywhere near the Business Community then IT management need to be certain that it will the bring the company additional benefits above and beyond what is currently in place. Primarily this will mean benefit to IT with secondary benefit to the User community. Both User and IT management are, and indeed must be, sceptical. Too often have there been claims for

improved service, improved future benefits, jam tomorrow etc. Fourth Generation Languages were once the panacea together with RAD, prototyping, relational databases, object orientation, BPR…… Having said that, we cannot remain stationary and we need to progress through evolutionary means. Revolution is difficult because of areas mentioned elsewhere e.g. existing systems, existing documentation and all the complexities and inconsistencies which have grown over the years. Whereas it's nice to start from scratch on a green-field site, most large organisations do not have this luxury. It's certainly possible in these new sites though as TIS and others bear witness.

- It's very difficult to prove additional benefit over and above the 'current' approach especially since no-one actually knows categorically what the current cost is,/ was / or would be in future. Each step through the process may be entirely logical and may get agreement but that doesn't imply that the end-result will get acceptance.
- A couple of further points:
  As mentioned the document is the contract between the various involved parties. This is a good formal way of describing it and describes the process better than 'specification'. If the use-case were to go out to a 3rd party to code then it's a contract - or at least part of a contract - so this word should be used internally as well.

  It will be interesting to see how this develops and how it copes when confronted with the normal nitty-gritty, error processing and added complexity of a real-life medium scale development rather than a simplified case study. The right tool will be a key factor

Q41. Are there any other areas that you would like to have seen covered / included?

- More detail or potential benefits of using methodology e.g. degree of reuse, potential decrease in documentation. Needs to be brought out.
- Approach across legacy applications – we now very rarely get the opportunity to go for "green field" development and hence you need to look at any potential for mapping or translation of what is there already.
- Measurements – How big could it get – how feasible to share amongst teams of developers e.g. for large system like UKRIS how long would it take?
- How would this methodology apply to the more popular development software?
- What is relative saving by using this method?
- Is there a difference between maintenance and development?
- Can user resources be used more than IT with the method?
- Platform independent business process modelling using reusable library?

Q42. Are there any other areas that you felt were unclear, and would have liked me to expand on, or explain better?

- See points above (Q41). You seem to go into the methodology quite cold.
- Section 3 needs structuring to make it clearer.
- Business Rules
- Bit too techy for the likes of me!

Q43 From what you have read, would you consider a pilot to use this approach?

YES
- Yes, if the opportunity arose and it was accepted as worthwhile by others involved.

NO
- No. You need to present more on the potential savings / benefits from this.
- No
- No. Would be interested in evaluating from other's experience, but not as "First penguin in the water".

Q44 Are there other areas of research related to this topic that you would like to see undertaken?

- The follow through into delivery, and methods for identifying and managing the next, more granular, level of reuse, those components which would be reusable across service packs, for example code subroutines.
- What level of detail can be 'got away with'
- Comments previous page.

Q45 Any other comments?

- I thought your methodology was reasonable, logical and analytical in approach. I'm sure it would represent a sound way of representing developments in terms of a "component approach" – but for me a proposal needs more about how it would work and be managed in practice and the potential benefits.
- Interesting. These views need airing and exploring - things are not all rosy at the moment. However I think that most systems are too complicated to be 'pulled-off' by this method

-------------------------------------- End of Feedback --------------------------------------

GLOSSARY

This section explains some of the terms or abbreviations used in the text.

ActiveX Microsoft's OLE API based distributed computing and component strategy to the Web. It features ActiveX Desktop and Server components using the Microsoft DCOM middleware architecture. See Appendix 4 for further details.

API Application Programming Interface

C++ Invented by Bjarne Stroustrup in 1980, it contains the entire C language, but with additions to support object-oriented programming.

COM Microsoft's component object model.

COM+ Microsoft's improved component object model with a simpler programming environment that automatically generates IDL, but not available until the millennium given its dependency on Win2000.

CORBA Common Object Request Broker Architecture. "CORBA is a commercial standard from OMG, whose mission is to define interfaces for interoperable software. Their specification, CORBA, is an industry consensus standard that defines a higher-level infrastructure for distributed computing . In general object orientation enables the development of reusable, modular software, and is moving technology towards plug-and-play software components. OMGs efforts are extending these benefits across multiple heterogeneous systems" (Mowbray & Malveau,1997).

DCOM Microsoft's distributed COM. This provides advanced services that allow you to manage the interaction between Windows environments (Linthicum,1997).

Domain Class Model In object-oriented analysis, a domain class model is prepared by the development team in order to more fully understand the main objects and their inter-relationships in the business area i.e. domain. The model is developed using terms familiar to the business experts e.g. Claim, Insured. However, a knowledge of object modelling is required in order to interpret the true meaning of the diagram.

GUI Graphical User Interface – The user interaction / presentation component in a client server system.

HTML Hypertext Markup Language

IDL Java Interface Definition Language (IDL) - enables Java components to talk to Corba compliant components.

Intranet "A private network of trusted Web servers used by a single organisation for intra-organisational applications. Intranets are essentially private infrastructure and are frequently 'firewalled' (shielded using security software or hardware) from, or run totally independently of, the Internet. The internet runs on public networks, across both private and public organisations". (Pal, Ring & Downes, 1996).

IIOP Corba's Internet Inter-Orb Protocol

Intranet Terminal Provides access to a range of applications from a Web browser, but cannot support local processing of Java applications See Appendix 2 for further details.

Java Sun Microsystems combined set of technologies for an interpreted, object oriented language. See Appendix 1 for further details.

Java Applets A "…component sized application that servers can ship to clients via ordinary HTML pages" (Orfali & Harkey, 1998). See Appendix 1 for further details.

Java Beans Components written in Java that communicate with each other, plus other types of components like ActiveX controls and OpenDoc parts. See Appendix 1 for further details.

JDBC Java Database Connectivity – a Java API to SQL relational databases.
JDK Java Development Kit
JVM Sun's Java Virtual Machine

MTS Microsoft Transaction Server - manages multiple users and load balancing across several computers. This will become part of Windows NT. (Anderson, 1998).

NC Network Computer. Runs a Web browser, supports Java, and Microsoft Windows applications via NT and X-Windows See Appendix 2 for more details
NetPC Based on a PC but smaller, and works across a network with central administration See Appendix 2 for more details

ODBC The Open Database Connectivity Standard
OLE Microsoft's standard for Object Linking and Embedding.
OMG Object Management Group – the standards committee controlling the development of CORBA

PC End User Job Function Classification (Gartner,1997)
- High Performance – performing high value, mission critical tasks with a high dependency on technology and a high cost of downtime eg stock traders, engineers, direct customer services
- Mobile – workers on the road and in the field. Often high performance workers with fragile mobile technology. Again, high dependency on technology and a high cost of downtime.
- Knowledge – the most poorly defined yet most publicised class. Definition: A worker who gathers, adds value to and communicates information in a decision support process. Cost of downtime is variable but highly visible.
- Structured Task – workers who perform the same task repetitively, typically as a link in a workflow or process. Cost of downtime varies, most workers are only partially dependent on computer availability.
- Data Entry – workers who input data into computer systems.

RMI Remote Method Invocation - for Java only environments with no database access.

Smartcard Invented in 1974 by Roland Moreno who devised a payment system based on an electronic stored value application mounted on a ring. There are currently two operating systems: Sun Microsystems (partnered with Visa and others) with the Java Virtual Machine on a card; and Maosco (Multi Application Operating System Consortium), an industry wide consortium which includes Mondex, Mastercard, Fujitsu, AmEx and others.

TCP/IP Transmission Control Protocol/Internet Protocol allow communication among a variety of independent, multivendor systems. In 1983 it became the official transport mechanism for the internet. (Sheldon, 1994).

Virtual Organization "A virtual organization is a globally distributed, loosely confederated collection of trading partners and subcontractors working with the enterprise and collaborating and communicating primarily via electronic means." (Gartner,1998)
Windows Terminal Displays Windows applications held and managed on Microsoft Windows NT servers. Workstation has no local storage or processing capability. See Appendix-2 for more details.
XML Extensible Markup Language - A universal standard for describing and exchanging data between environments that are based on different technologies and implementations.

BIBLIOGRAPHY

Allen, P. and Frost, S. (1998) *Component Based Development for Enterprise Systems*, Cambridge University Press.
Anderson, T. (1998) *Putting the Pieces Together*. Computing. 3rd September. p.42

Booch, G. (1994) *Object-Oriented Analysis and Design with Applications.* Second Edition. Benjamin Cummings.
Booth, W.C. Colomb, G.C and Williams, J.M. (1995). *The Craft of Research*. The University of Chicago.
Bright, P. (1991) *Introduction to Research Methods in Postgraduate Theses and Dissertations.* Newland Papers Number 18. The University of Hull.
Butler Group. (1998a) "Component-Based Development: Application Delivery and Integration Using Componentised Software". September. http://www.butlergroup.com [22/11/98].
Butler Group. (1998b) *Component-Based Development (CBD) Forum Newsletter.* December Issue.
Butler Group. (1999). *Component-Based Development and Integration (CBDi) Forum Newsletter,* April Issue.

Cantor, M.P. (1998) *Object-Oriented Project Management with UML*. John Wiley & Sons.
Clarke, G. (1998a) "Price is wrong for Java-Based cards." *Computing*. 3rd September. p2.
Clarke, G. (1998b), "PC's under £300 a threat to NCs" *Computing*. 1st October. p3.
Clarke, G. (1998c) "Lloyd's adopts WTS". *Computing*. 8th October. p14
Coad, P. and Yourdon, E. (1991) *Object-Oriented Design*. Prentice-Hall
Computing. (1998a) "Horses for courses' as industry divided over NT." *Computing*. 3rd September. p15.
Computing. (1998b) "Gartner hits out at ERP" *Computing*. 24th September. p4.
Computing. (1998c) "Home on the mid-range: Sun has overtaken HP Unix stakes." *Computing*. 1st October. p36
Currie, W.L. and Willcocks, L. (1995) *Reengineering and I.T. in Financial Services.* RDP95/8, Oxford Institute of Information Management, Templeton College Oxford. p12.

D'Souza, D.F. and Wills, A.C. (1998*) Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley.

Farrell, N. (1998a) "Internet to become 'dominant medium'". *Computing.* 1st October. p32.
Farrell, N. (1998b) "Dresdner pushes its objects". *Computing*. 22nd October. p2.
Fowler, M. and Kendall, S. (1997) *UML Distilled : Applying the Standard Object Modelling Language*. Addison Wesley.

Gamma, E., Helm, R., Johnson, R., and Vlissides,J. (1995). *Design Patterns: Elements of ReUsable Object-Oriented Software*. Reading, MA: Addison-Wesley.
Gane, C. and Sarson, T. (1979). *Structured Systems Analysis Tools and Techniques*. Prentice Hall.

Gartner. (1997) “TCO Analyst : A White Paper on GartnerGroup’s Next Generation Total Cost of Ownership Methodology”. Gartner Consulting. 1997
Gartner. (1998) “The Future of the Virtual Organisation*.” Gartner Group Research Note: Strategic Planning.* 11th September.
Goodwin, C. (1998) “A Place in OO Heaven”. *Computing* 24th September. p56-58.
Griffiths, J. (1998) “PC or netPC?”. *The Computer Bulletin*. British Computer Society. March. p26-28.

Hambling, B. (1996) *Managing Software Quality*. McGraw-Hill.
Hunter, P. (1998) “Living with the enemy”. *Computer Weekly*. 15th October. p46.

IBM (1998). “From e-Technology to e-Commerce” http://www.software.ibm.com/ad/cb/harkey-jul98.html. July. [02/02/1999].
IBM (1999). “IBM Component Broker : Building Business Value.” http://www.software.ibm.com/ad/cb/bbvalue.html. [31/01/1999].

Jackson, M.A. (1983). *Systems Development*. Prentice Hall.
Jacobson, I., Christerson, M., Jonsson, P.M. and Overgaard, G (1992). *Object oriented software engineering: A use case driven approach*. Addison-Wesley.
Jacobson, I., Ericcson, M., and Jacobson, A. (1994) *The Object Advantage – Business Process Engineering with Object Technology*. Addison-Wesley.
Jayaratna, N. (1994). *Understanding and Evaluating Methodologies*. McGraw Hill.

Kavanagh, J. (1998) “A Question of Foresight.” *The Computer Bulletin*. British Computer Society. September. p22-23.
Kelly, L. (1998a) “Payback time benefits Java”. *Computing*. 3rd September. p12.
Kelly, L. (1998b) “ A slip-up for Java at oil exchange.” *Computing.* 1st October. p21.
Kelly, L. (1998c) “Platform independence is the key, says analyst.” *Computing*. 1st October. p21.
Kelly, L. (1998d) “Corba supporters defy DCOM death threat.” *Computing*. 15th October. p18.

Linthicum, D. (1997) “Mutating Middleware”. *DBMS : Database and Client/Server Solutions*. Volume 3 Issue 8. October. p24-33.

Martin, J. and Finkelstein, C. (1981). *Information Engineering*. Volumes 1 and 2. Prentice Hall.
Martin, J and Odell, J,J. (1994). *Object-Oriented Methods: A Foundation*. Prentice Hall.
McGinn, J.(1998) “HP backs down over Java plan.” *Computing*. 15th October. p18.
McLeod, M. (1998) “Life firm puts Web monitoring on trial.” *Computer Weekly*. 29th October. p3.
Mannion, M. and Keepence, B (1998) “Software Jigsaws”. *The Computer Bulletin*. British Computer Society. September. p24-27.
Morris, C.R. and Ferguson H. (1993) “How Architecture Wins Technology Wars”. *Harvard Business Review*. March-April. p86-96.
Mowbray, T.J. and Malveau, R.C. (1997) *CORBA Design Patterns*. John Wiley & Sons.

Object Management Group. (1997) *Unified Modelling Language (UML) - Version 1.1*. November.
Orfali, R and Harkey, D. (1998) *Client/Server Programming with Java and Corba.* Second Edition. John Wiley & Sons.

Pal, A., Ring, K. and Downes,V. (1996). *Intranets for Business Applications: User and Supplier Opportunities*. Ovum Reports, Ovum Ltd.
Perspectives. (1993) "Will Architecture Win the Technology Wars". *Harvard Business Review*. May-June. p162-170.
Phillips, S. (1998) "Aberdeen Council launches largest smartcard project." *Computer Weekly*. 5th November. p12
Poston, T. (1998) "Lloyds TSB unveils first part of Internet banking systems." *Computer Weekly*. 5th November. p6.
Pressman, R.S. (1997). *Software Engineering: A Practioners Approach*. Fourth Edition. McGraw Hill.

Rock-Evans, R. (1992). *Data Modelling & Process Modelling*. Reed International Books.
Rumbaugh, J., Blaha, M., Premerlani, W., and Eddy, F. (1991) *Object-Oriented Modeling and Design*. Prentice Hall.

Saran, C. (1998) "Microsoft plans system to rival Java smartcard." *Computer Weekly*. 29th October. p2.
Shaw, M. and Garlan, D. (1996) *Software Architecture : Perspectives on an Emerging Discipline.* Prentice Hall.
Sheldon, T. (1994) *LAN Times Encyclopedia of Networking*. McGraw-Hill.
Shlaer,S and Mellor, S.J. (1988) *Object Oriented Systems Analysis*. Prentice Hall.

Tober, B. (1998) "Smart card pack holds the ecommerce aces." *Computing*. 29th October. p26-27.

Wallen, J. (1998) "The calm before the Unix storm". *Computing*. 3rd September. p14.
Wirfs-Brock, R., Wilkerson, B. & Wiener, L. (1990) *Designing Object-Oriented Software.* Prentice Hall.