



This work is protected by copyright and other intellectual property rights and duplication or sale of all or part is not permitted, except that material may be duplicated by you for research, private study, criticism/review or educational purposes. Electronic or print copies are for your own personal, non-commercial use and shall not be passed to any other individual. No quotation may be published without proper acknowledgement. For any other use, or to quote extensively from the work, permission must be obtained from the copyright holder/s.

# **Investigating the use of native language calls in a multi-channel business process**

Clive Jefferies

Doctor of Philosophy in Computer Science

September 2011

Keele University

## **Abstract**

### **Background**

Making system functionality available via multi-channel access (MCA) can be achieved through exposing functions and business processes as software services. When offering MCA to a business process, system performance is an important consideration due to network limitations and verbose messaging in service-oriented technologies.

### **Aims**

The first aim of this study is to investigate the potential impact on system performance and agility that may occur when an underlying business process is exposed for MCA. The second aim is to investigate if reengineering a system as a service-oriented architecture (SOA) improves agility. The work also aims to create an MCA reengineering method to transform systems from single-channel into multi-channel.

### **Methods**

A case study was used, along with an experiment, to compare the performance and agility of native language calls (NLC) and protocol based messaging (PBM) for service messaging in a business process. The case study also investigated if reengineering a system as an SOA improves agility by comparing system and code metrics. A multi-channel access reengineering method (McARM) was created and evaluated.

## **Results**

No significant difference was found between the performance of the PBM and NLC binding technologies. However, NLC bindings were found to be less agile than PBM. Reengineering a system as an SOA was found to improve the agility of a system. A method was created which was used to reengineer a system for MCA.

## **Conclusions**

Based on the results, the recommendation is that NLC should not be used instead of PBM for messaging between the services and business processes in a system reengineered for MCA. Measures should be taken to ensure that the reengineering of a system for MCA does not affect performance. Finally, an SOA can be used to improve system agility.

## **Glossary**

**Agility test case** – test case derived to measure agility in terms of time and effort.

**Binding technology** – technology used to bind a business process to a service.

**Business function** – software functionality that relates directly to a business need.

**Business object** – an object that represents a business entity.

**Business process** - related tasks which produce a business outcome according to business rules.

**Business process engine/server** – server for executing business processes.

**Business process execution language (BPEL)** – XML based language for implementing business processes.

**Case study** – scientific methodology used to investigate problems of an exploratory nature, usually having several sets of data.

**Channel** – device type used to access a system.

**Complex type** - an XML element containing other elements.

**External quality factor** – software quality attribute from an external viewpoint, such as users.

**eXtensible mark-up language (XML)** - a framework used as a notation for creating mark-up languages.

**Granularity** - the level of abstraction at which services and their functionality are aimed, focusing on the service description and what it represents.

**Interoperability** – the ease of software interacting with other software.

**Java stubs** – a Java class that can create instances of a class held on a remote server and its methods executed as though it were a local object.

**Latency** - the response time for a request in a system.

**Multi-channel access (MCA)** - a property of a system which means it has the ability to be accessed by heterogeneous devices in a consistent manner.

**Namespaces** - named elements and attributes in an XML document to avoid element name conflicts.

**Native language call (NLC)** – a service binding technology that uses the service implementation language for messaging also.

**Ontology** – representation of concepts in a domain and how they relate to each other.

**Product quality criteria** – software quality attribute from an internal viewpoint, such as developers.

**Protocol based messaging (PBM)** – a service binding that uses a messaging format which conforms to a protocol.

**Quality attribute scenarios (QAS)** – summary of a non-functional requirement.

**Semantic differential scale** - assessment by humans involving a rating scale.

**Service** - a capability that is offered for use to requesters by a provider that is *described* by the provider so that it can be *discovered* and used by requesters.

**Service description** - document that describes a service contract and use.

**Service loose coupling** - services and technologies of a service-based system are not dependent on one another, meaning a change will have little or no impact.

**Service-orientation (SO)** – a paradigm for creating service based systems.

**Service-oriented architecture (SOA)** - an architectural style for creating and managing service-oriented applications.

**Service reusability** – services can be leveraged by other systems.

**Simple object access protocol (SOAP)** – web service technology for messaging.

**Software metric** – a measure of an aspect of software.

**Software quality model** - used to describe desirable, measureable properties in software.

**Systematic Literature Review (SLR)** - a structured literature review which aims to gather literature, extract data and synthesise the data in a systematic way.

**Web services** – an XML-based platform designed for the description, discovery and messaging of services.

**Web service description language (WSDL)** – web service technology used to implement service descriptions.

**Web service invocation framework (WSIF)** - a toolkit for invoking any resource using a WSDL based description.

**Web Services Integration Framework (XWIF)** – a methodology for integrating XML and service-oriented architectures into existing systems.



**Weighted comparison table** – used to assess architectural features according to desired criteria which have been weighted by importance.

**XML schema** – a schema used in XML languages for expressing shared vocabularies.

# Contents

Abstract .....	iv
Glossary .....	vi
List of Figures .....	xvi
List of Tables .....	xvii
Acknowledgements .....	xix
Chapter 1: Introduction .....	1
1.1 Background .....	1
1.1.1 Mobile computing .....	1
1.1.2 Multi-channel Access .....	2
1.1.3 Service-orientation .....	6
1.1.3.1 Services .....	6
1.1.3.2 Service-oriented architecture (SOA) .....	8
1.1.3.3 Business processes .....	11
1.1.3.4 Service based technologies .....	13
1.2 The IBHIS project .....	15
1.3 Reengineering .....	17
1.3.1 Phases .....	18
1.3.2 Reengineering methods .....	19
1.4 Research methods .....	21
1.5 Research Objectives .....	21
1.5.1 Objective 1 .....	22
1.5.2 Objective 2 .....	23
1.5.3 Objective 3 .....	23
1.6 Contributions .....	24
1.7 Thesis outline .....	24
Chapter 2: Systematic Literature Review .....	26
2.1 Planning .....	26
2.1.1 Background .....	26
2.1.2 Review questions .....	27
2.1.3 Review methods .....	28
2.1.3.1 Search strategy and resources .....	29
2.1.3.2 Study selection .....	30
2.1.3.3 Quality Assessment .....	31
2.1.3.4 Data Extraction Strategy .....	31
2.1.3.5 Data synthesis .....	32
2.2 Results .....	32
2.2.1 SLR-RQ1 .....	33
2.2.2 SLR-RQ2 .....	33
2.2.3 SLR-RQ3 .....	34
2.3 Discussion .....	35
2.3.1 SLR-RQ1 .....	36
2.3.2 SLR-RQ2 .....	36
2.3.3 SLR-RQ3 .....	37
2.4 Conclusions .....	39
Chapter 3: Reengineering for Multi-channel access .....	40

3.1 McARM .....	40
3.1.1 Stages .....	41
3.1.2 Stage order .....	42
3.1.3 Stage details .....	43
3.1.3.1 Requirements gathering/analysis .....	43
3.1.3.2 System understanding .....	44
3.1.3.3 Service identification in legacy system.....	45
3.1.3.4 Business Process Modelling .....	46
3.1.3.5 System redesign .....	47
3.1.3.6 System recoding .....	49
3.1.3.7 Service interface implementation .....	49
3.1.3.8 Business process implementation .....	50
3.2 MCA Architecture .....	50
3.2.1 Summary of architectures .....	51
3.2.2 Suitability of architectures .....	51
3.2.3 Architectural layer types .....	53
3.2.4 Proposed architecture.....	54
Chapter 4: The performance/agility trade-off .....	57
4.1 Performance .....	57
4.1.1 Web services .....	58
4.1.2 Improving web service performance .....	59
4.1.3 Improving business process performance .....	59
4.1.4 Using WSIF to improve business processes performance .....	61
4.2 Agility .....	65
4.2.1 Information systems agility.....	66
4.2.2 SOA and agility.....	67
4.2.3 Agility characteristics .....	69
4.2.4 Agility quality model .....	72
4.2.5 Agility measures .....	75
4.2.6 Agility test cases .....	78
Chapter 5: Methodology and implementation .....	79
5.1 Planning .....	80
5.1.1 Aims.....	80
5.1.2 Case study design.....	82
5.1.3 Case selection .....	84
5.1.4 Case study roles .....	85
5.1.5 Data Collection .....	85
5.1.5.1 Data to be collected.....	85
5.1.5.1.1 CS-RQ1-P1 .....	86
5.1.5.1.2 CS-RQ1-P2 .....	86
5.1.5.1.3 CS-RQ2.....	89
5.1.5.1.4 CS-RQ3.....	89
5.1.5.1.5 CS-RQ4.....	90
5.1.5.2 How the data was collected.....	90
5.1.5.2.1 Response times.....	91
5.1.5.2.2 System metrics .....	91
5.1.5.2.3 Semantic differential measurements .....	91
5.1.5.2.4 Agility test cases data.....	92
5.1.5.2.5 Reengineering method data.....	92
5.1.5.3 Data collection plan .....	92

5.1.5.4 How data was stored .....	93
5.1.6 Analysis .....	94
5.1.6.1 CS-RQ1 .....	94
5.1.6.2 CS-RQ2 .....	96
5.1.6.3 CS-RQ3 .....	97
5.1.6.4 CS-RQ4 .....	97
5.2 Conducting .....	97
5.2.1 CS-RQ1 .....	97
5.2.1.1 Performance testing mock-up .....	97
5.2.1.2 Agility test case mock-up .....	99
5.2.1.3 Unused agility test cases .....	99
5.2.2 CS-RQ2 .....	100
5.2.3 CS-RQ3 .....	100
5.2.4 CS-RQ4 .....	100
5.2.4.1 System understanding .....	100
5.2.4.2 Service identification .....	106
5.2.4.3 Business Process Modelling .....	109
5.2.4.4 System redesign .....	111
5.2.4.5 System recoding .....	117
5.2.4.6 Service interface creation .....	117
5.2.4.7 Creating business processes .....	117
5.2.4.8 Client .....	119
5.2.4.9 Testing .....	121
5.2.5 Architecture .....	122
Chapter 6: Case study results .....	125
6.1 Results by data collected .....	125
6.1.1 Performance testing .....	125
6.1.2 Agility metrics .....	126
6.1.3 Agility test cases .....	127
6.2 Results by research question .....	127
6.2.1 CS-RQ1-P1 .....	127
6.2.2 CS-RQ1-P2 .....	128
6.2.2.1 System metrics .....	128
6.2.2.2 Agility test cases .....	130
6.2.3 CS-RQ2 .....	131
6.2.4 CS-RQ3 .....	132
6.2.5 CS-RQ4 .....	134
6.2.6 Summary .....	135
Chapter 7: Discussion .....	137
7.1 CS-RQ1 .....	137
7.1.1 CS-RQ1-P1 .....	137
7.1.2 CS-RQ1-P2 .....	138
7.1.2.1 Agility quality model .....	138
7.1.2.2 Agility test cases .....	139
7.1.3 Summary .....	141
7.2 CS-RQ2 .....	141
7.3 CS-RQ3 .....	143
7.3.1 Product quality criteria .....	143
7.3.1.1 Granularity .....	144
7.3.1.2 Complexity .....	144

7.3.1.3 Expandability .....	145
7.3.1.4 Generality.....	145
7.3.1.5 Modularity.....	146
7.3.1.6 Communication commonality.....	146
7.3.2 Summary .....	146
7.4 CS-RQ4.....	147
7.5 Agility quality model .....	150
Chapter 8: Evaluation .....	155
8.1 Systematic Literature Review .....	155
8.2 Case study .....	157
8.2.1 Checklist .....	157
8.2.1.1 Case study design.....	157
8.2.1.2 Preparation for data collection .....	158
8.2.1.3 Collecting evidence.....	159
8.2.1.4 Analysis of the evidence .....	159
8.2.1.5 Reporting.....	160
8.2.2 Validity .....	161
8.2.2.1 Construct validity.....	161
8.2.2.1.1 Construct measures .....	161
8.2.2.1.2 Mono-method bias .....	162
8.2.2.1.3 Levels of the independent variable .....	162
8.2.2.2 Internal validity.....	162
8.2.2.3 External validity.....	164
8.2.2.3.1 CS1-RQ1.....	164
8.2.2.3.2 CS1-RQ2.....	165
8.2.2.3.3 CS1-RQ3.....	165
8.2.2.3.4 CS1-RQ4.....	165
8.2.2.4 Reliability.....	165
8.3 Data collection methods.....	166
8.3.1 Response times .....	166
8.3.2 Code metrics .....	167
8.3.3 Semantic differential measurements .....	167
8.3.4 Agility test cases .....	168
8.3.5 McARM evaluation .....	169
Chapter 9: Conclusions and future work .....	170
9.1 Conclusions.....	170
9.1.1 CS-RQ1.....	170
9.1.2 CS-RQ2.....	171
9.1.3 CS-RQ3.....	172
9.1.4 CS-RQ4.....	172
9.1.5 Multi-channel Architecture .....	173
9.1.6 Agility quality model .....	173
9.1.7 Research methodologies .....	174
9.2 Future work.....	175
9.2.1 CS-RQ1.....	175
9.2.2 CS-RQ2.....	175
9.2.3 CS-RQ3.....	176
9.2.4 CS-RQ4.....	176
9.2.5 Multi-channel Architecture .....	177
9.2.6 Agility quality model .....	177

References .....	178
Appendix .....	196
Appendix A – Reengineering ordering .....	197
Appendix B – Functional testing .....	198
Appendix C - PBM and NLC response times .....	201
Appendix D – reengineered IBHIS broker (PBM) semantic differential scale – Primary reviewer.....	202
Appendix E – reengineered IBHIS broker (NLC) semantic differential scale – Primary reviewer.....	203
Appendix F – response times for CS-RQ2 .....	204
Appendix G – original IBHIS broker semantic differential scale – Primary reviewer .....	205
Appendix H – original IBHIS broker semantic differential scale - Validation .....	206
Appendix I – reengineered IBHIS broker (PBM) semantic differential scale – validation.....	207

## List of Figures

Figure 1 Flight service .....	7
Figure 2 SOA boundaries.....	9
Figure 3 Flight service at the enterprise/institutional level.....	10
Figure 4 Flight service at the external level .....	10
Figure 5 Travel booking process.....	12
Figure 6 IBHIS broker overview .....	16
Figure 7 Rosenberg general model for software reengineering .....	19
Figure 8 Horseshoe reengineering method .....	20
Figure 9 Single to Multi-Channel .....	27
Figure 10 MCA Architecture layer comparison .....	54
Figure 11 Architectural layering for reengineered IBHIS broker.....	55
Figure 12 Comparison of proposed MCA architecture with existing architectures ....	55
Figure 13 WSIF binding declarations (IBM Websphere 5.1.1 documentation) .....	62
Figure 14 SOAP binding declarations (IBM Websphere 5.1.1 documentation) .....	63
Figure 15 WSIF and SOAP bindings.....	64
Figure 16 SOAP and WSIF dependencies .....	65
Figure 17 Reliability in McCall's quality model.....	73
Figure 18 Agility quality model.....	75
Figure 19 Brokers used in case study .....	82
Figure 20 Overview of original IBHIS system .....	101
Figure 21 IBHIS broker services and data .....	102
Figure 22 Original IBHIS system GUI flowchart.....	105
Figure 23 P2Client (query) .....	106
Figure 24 IBHIS current and future use cases .....	107
Figure 25 Sequence diagram for potential services .....	107
Figure 26 IBHIS business process .....	110
Figure 27 Classes for Activation service .....	112
Figure 28 UserDetails type .....	116
Figure 29 Roles type .....	116
Figure 30 Activation business process.....	118
Figure 31 Activation input .....	120
Figure 32 Activation output .....	120
Figure 33 Original reengineering problem .....	122
Figure 34 Original IBHIS broker architecture .....	123
Figure 35 Reengineered IBHIS broker architecture .....	123
Figure 36 Direct access of reengineered IBHIS services.....	124
Figure 37 Original agility quality model .....	153
Figure 38 Revised agility quality model.....	154

## List of Tables

Table 1 Search terms.....	29
Table 2 Service-Oriented Reengineering methods .....	33
Table 3 Proposed architectures for MCA .....	34
Table 4 Service-oriented reengineering issues .....	35
Table 5 Inputs and outputs for system understanding .....	45
Table 6 Inputs and outputs for service identification.....	46
Table 7 Inputs and outputs for business process modelling .....	47
Table 8 Inputs and outputs for system redesign.....	49
Table 9 Inputs and outputs for service interface creation .....	50
Table 10 Inputs and outputs for business process implementation.....	50
Table 11 Architecture summary table.....	51
Table 12 Weighted comparison table for multi-channel architecture.....	53
Table 13 Agility characteristic comparison .....	71
Table 14 Agility quality factors .....	73
Table 15 Research questions and measures .....	83
Table 16 Message sizes.....	86
Table 17 CS-RQ1-P1 QAS .....	86
Table 18 CS-RQ1-P2 QAS .....	87
Table 19 Agility test case 1 QAS.....	87
Table 20 Agility test case 2 QAS.....	88
Table 21 Agility test case 3 QAS.....	88
Table 22 Agility test case 4 QAS.....	89
Table 23 CS-RQ2 QAS.....	89
Table 24 CS-RQ3 QAS.....	90
Table 25 CS-RQ4 QAS.....	90
Table 26 Summary of outcomes .....	96
Table 27 Component table for original IBHIS system .....	104
Table 28 Service table.....	108
Table 29 Interaction Scenarios.....	115
Table 30 Service messages .....	115
Table 31 Complex types .....	116
Table 32 Results for performance testing .....	126
Table 33 Agility metrics .....	126
Table 34 Time and effort for a new service .....	127
Table 35 Table of means.....	127
Table 36 ANOVA summary table for performance investigation.....	128
Table 37 Semantic differential measurements for the bindings.....	129
Table 38 PBM/NLC agility comparison .....	129
Table 39 Time and effort for a new service .....	130
Table 40 Time and effort to change a service.....	131
Table 41 Layer comparison table.....	131
Table 42 ANOVA for CS-RQ2.....	132
Table 43 Code metrics for original/PBM.....	133
Table 44 Semantic differential measurements for original/reengineered IBHIS.....	133
Table 45 Original/Reengineered IBHIS (PBM) comparison.....	134
Table 46 Assessment of reengineering method .....	135
Table 47 McARM reengineering phases .....	149



Table 48 McARM coverage of reengineering lifecycle .....	149
Table 49 Case study design evaluation .....	158
Table 50 Preparation for data collection evaluation .....	158
Table 51 Collecting evidence evaluation .....	159
Table 52 Analysis of the evidence evaluation .....	160
Table 53 Reporting evaluation.....	160
Table 54 Threats and controls for performance measurements .....	163
Table 55 Threats and controls for agility measurements .....	164

## **Acknowledgements**

First, I would like to thank my family for their support and encouragement. I would also like to thank my supervisors Pearl Brereton and Mark Turner for their time and expertise. Pearl for believing in me and Mark for his patience in understanding the technical aspects of the work. I have to thank all of the friends that I have had the pleasure of sharing the attic with, and who I have the greatest respect; Phil Woodall, Ryad Soobhany, Kapila Ponnampuruma, Siffat Ullah Khan, Adam Pountney, James Rooney, Rob Emery and especially John Butcher. I must also thank all of the members of the Software Engineering group at Keele; Barbara Kitchenham, Steve Linkman, Thomas Neligwa, Mahmood Niazi and Zhi Li. Finally, I wish to thank the technical and secretarial staff in the School Computing and Mathematics at Keele.

## Chapter 1: Introduction

Enabling software systems to be accessed by heterogeneous devices is known as multi-channel access (MCA). In this research, the wider implications of MCA are investigated by exploring potential architectures and reengineering methods. System performance is also investigated, which can be a problem when accessing large data sets from devices with low processing power, on networks that may be unreliable or slow. While there are potential solutions to improve the system performance of business processes, for example through the use of native language calls (NLC) for messaging, such solutions may have an adverse affect on the agility of the system.

### 1.1 Background

This section outlines the main concepts used in the thesis.

#### 1.1.1 Mobile computing

The increase in both processing and network speed on mobile devices has resulted in an increased demand for more complex, feature rich applications. This can be useful for companies or institutions who wish to make their systems available outside of their offices. Examples of systems that companies may wish to expose are; *supply chain management* (SCM) e.g. inventory, order entry and purchasing and *customer relationship management* (CRM) e.g. sales and marketing.

There are institutions which may wish to be able to access their systems from mobile devices, such as:

- Healthcare e.g. patient healthcare records
- Police e.g. accessing crime details
- Local government e.g. local statistics and information
- Charities e.g. when working in other countries with internet access

Mobile devices allow workers the freedom to access information instantly and continuously. This would make workers more productive and able to respond quickly to queries. This has resulted in the need to reengineer existing systems in order to enable them to be accessed from multiple, heterogeneous devices – or multi-channel access.

### **1.1.2 Multi-channel Access**

A system that has multi-channel access (MCA) is one that can be accessed by heterogeneous devices in a consistent manner (Newcomer & Lomow, 2005). More than one device can access the functionality of the system without having to create new solutions specifically for that channel. Examples of types of channels are: mobile devices, web site, a desktop application, ATM, television. A channel can be classified by the following (Marchetti et al. 2003):

- *Device* e.g. PDA, PC, smart phone
- *Application protocol* e.g. HTTP, SMTP, SOAP
- *Network* e.g. UMTS, GPRS

MCA is different from a system that has multi-modal access (MMA). MMA enables different input types to be used in a single transaction e.g. voice, data, SMS (Bisdikian et al, 2002).

The business services in a system change less frequently than the delivery channels that are used to access the services (Newcomer & Lomow, 2005). Therefore, a system that it is accessible from current and future channels is extracting more value from these systems. There are other advantages of MCA (Ganesh et al., 2004), such as:

- Ability to integrate with partners (e.g. credit card firms, courier companies)
- Ability to provide a single view of customers and transactions across multiple channels
- Increased efficiency by leveraging existing legacy assets

The availability of a system over multiple channels creates new business opportunities (Ganesh et al., 2004), such as:

- Loyalty management over multiple channels e.g. bonus points
- Point of sales technology could be integrated with other systems, giving the sales person pertinent information about a customer
- Promotions to multiple channels using personalisation based on user data

There are important considerations that need to be made when designing a multi-channel system. From the user point of view, an important consideration is the cost of data transmission (Park et al., 2006). Currently, data on mobile devices is expensive which means that users want the quantity of data being sent to be as small as possible. Another consideration for multi-channel systems is the characteristics of devices that

will potentially be accessing the system. Below are some of these characteristics (Marchetti et al. 2003):

- Screen and keyboard sizes - Mobile users are less able to spend long periods of time looking at small screens than desktop users, who generally have more screen real estate. Mobile devices may also have input mechanisms (e.g. stylus) which make entering a large quantity of information time consuming and awkward.
- Input types - These can vary e.g. a mobile phone could have touch screen input, numbered keys, stylus, speech input or even all of these methods.
- Dynamic spatial position – This introduces interesting opportunities such as location based services (LBS), which are services or functionality that is contextual, based on the location of the user e.g. a web page or an application would display a listing of nearby points of interest.
- Communication technology - Each of the devices may have different communications technology e.g. HTTP, sockets, RMI.
- Networks - Current mobile networks, such as Global System for Mobile Communications (GSM), and third generation (3G), have limited bandwidth (Park et al., 2006). In a multi-channel system it may have to be assumed that the system is being accessed using *any* of the mobile networks, as well as wired networks. The mobile networks can also have less connection stability. Not all mobile networks have full coverage of the country and there may also be certain places where the signal is unobtainable such as tunnels or buildings with thick walls. This means that multi-channel systems should be able to deal with a connection loss gracefully.

- Security - The devices may potentially be accessing systems from unsecure networks for example GSM or an unsecured wireless local area network. There is also the risk of users losing devices or the devices being stolen and then used inappropriately to access sensitive information.

Quality of service (QoS) requirements, need to be considered also, not just from the user perspective but also the channel itself (Comerio et al., 2004). For example, a user of a streaming video service who wishes to watch a short video will have a different tolerance to slow delivery than of an entire film. In terms of the channel itself, the acceptable levels of quality between a mobile device and a desktop device will be different e.g. users may be more likely to tolerate interruptions when accessing from a mobile device. System performance is an important QoS requirement for a system with MCA, which can be prohibited by the networks used to access the system. In this thesis the main consideration is the system performance.

One way of implementing MCA is by using a service-oriented architecture (SOA) (Krafzig et al., 2004). However, an SOA is not the only way of implementing MCA (Stroulia & Kapoor, 2001; Menkhaus 2002; Bertini & Santucci 2004, Eisenstein et al. 2001). Menkhaus, (2002), proposed a multiple user interface, single application (MUSA) architecture. This is a layered architecture where the service logic is passed into the core system which then adapts content to different device types. In the work proposed by Bertini & Santucci, (2004) and Eisenstein et al., (2001), the focus is on adapting the user interfaces according to the device types, using abstract interaction objects (AIOs). The AIOs can be used to create user interfaces which are mapped onto concrete implementations per device. These are then mapped onto the

characteristics of the device accessing the system e.g. scrolling ability and Java enabled. In the work proposed by Stroulia & Kapoor, (2001), a wrapper is used on legacy-systems with text based user interfaces to enable them to be accessed from the web using desktop and mobile web technologies.

This work will concentrate on the service-oriented method of achieving MCA, as the use of an SOA means that data can be accessed by a presentation client without being tied to the presentation layer.

### **1.1.3 Service-orientation**

An SOA is an aspect of the service-oriented (SO) paradigm. SO, like object-orientation (OO), is an approach to system analysis, design, implementation and management. The emphasis in SO is on the services provided by a system rather than the objects identified. As well as MCA there are also other benefits that come with SO, e.g. agility (Krafzig et al. 2004; Allen 2006; Erl 2007; Newcomer & Lomow 2005).

#### **1.1.3.1 Services**

A software system architecture is made up of components and their interconnections (Pfleeger, 2001). In SO the primary components are *services*. A service is a capability that is offered for use to requesters by a provider (OASIS RM-CS, 2006). The services are *described* by the service provider, so that they can be *discovered* and used by requesters (Allen, 2006). A service should be self contained and its implementation



opaque to the requester (Kreger & Estefan, 2009). Figure 1 is an example of a flight service that might be offered by an airline.

<b>FlightService</b>
GetAvailability
GetCost
BookFlight
GetBooking

**Figure 1 Flight service**

The flight service has four operations: `GetAvailability`, which checks for available flights; `GetCost`, which retrieves the cost of a flight; `BookFlight`, which allows a flight to be booked and `GetBooking`, which gets the details of a booking.

An important concept when designing services is their *granularity*. Granularity is the level of abstraction at which the services and their functionality are aimed, focusing on the service description and what it represents (Erl, 2007). A service should be aimed at fulfilling business requirements. Therefore the functions and services should reflect business functions (Newcomer & Lomow, 2005). At the extremes, granularity can be considered as being *coarse grained* or *fine grained*. A coarse grained system has just a few higher level services that represent business functions. A fine grained system exposes a large number of low level functions as services. An example of a coarse grained business service would be one that has a function to add a business customer address. This service represents a distinct business function and would be useful as it can be reused by other applications. Ensuring that a business customer address is valid can be classed as a fine grained function. This is only useful within the context of adding a business customer address, which means that it would not be

called outside of the service in which it resides. Granularity can be measured in the context of the system that the service is going to be used. For example, in some cases having a service that validates an address may be useful if there are other systems that require this functionality. The granularity of the functionality within the service can also be measured, which is independent of the system. For example, if the business address validation works on the entire address at once, then this would be coarse grained, if the validation could be performed on individual lines of the address, then this would be classed as fine grained.

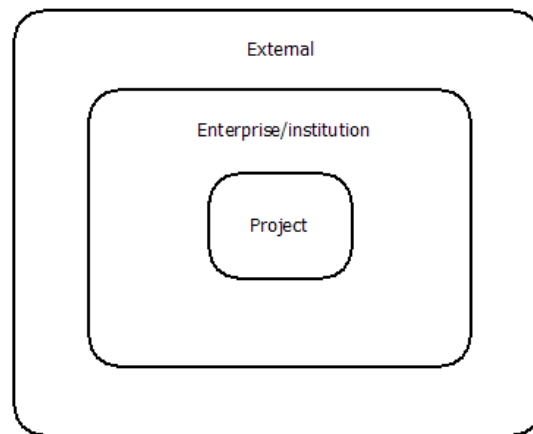
### **1.1.3.2 Service-oriented architecture (SOA)**

There are many definitions and interpretations of an SOA. According to the OASIS Reference Model for Service Oriented Architecture 1.0 (OASIS RM-CS, 2006) it is ‘a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains’. The Open Group states ‘An Service-oriented Architecture (SOA) facilitates the creation of flexible, reusable assets for enabling end-to-end SOA-based business solutions’ (Open Group, 2009). In a paper that examines the current standards for SOA, these two definitions are combined and an SOA is described as ‘architectures that support thinking and organizing in terms of services with distributed capabilities which may be under the control of different ownership domains, and is an architectural style as well as a paradigm for business and IT architecture.’ (Kreger & Estefan, 2009). Although, these definitions describe the main aspects of an SOA, an SOA does not necessarily have to be in a business context. Any system that requires a provider/requester style architecture, using services as components, could be categorised as an SOA.

The architectural aspect of an SOA views a system in terms of the services that it provides to fulfil a requirement and its interactions with potential requesters (Newcomer & Lomow, 2005). Potential requesters of a service could be:

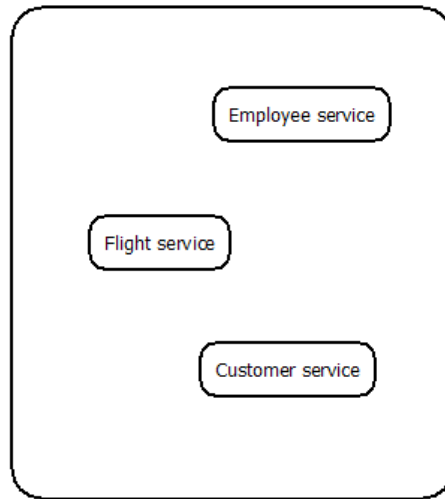
- Internal systems (project level)
- Systems within the enterprise/institution
- External systems

Figure 2 shows the boundaries of the potential requesters.



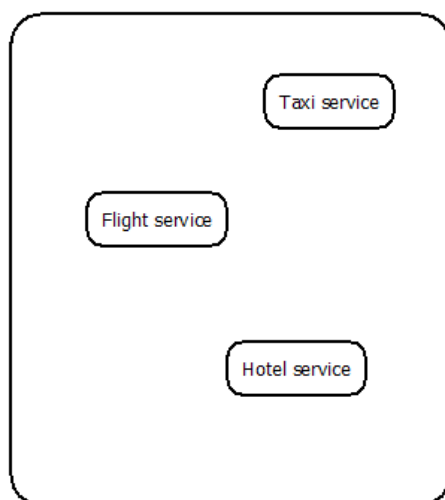
**Figure 2 SOA boundaries**

In figure 2 the *enterprise/institutional* level the services can be used by other departments or projects. Figure 3 shows some services that might be used at the enterprise/institutional level for an airline.



**Figure 3 Flight service at the enterprise/institutional level**

Figure 3 shows that there might be an employee service for managing employees, a customer service for managing customers and the flight service for booking customer flights. In this example, the `FlightService` detailed in section 1.1.3.1, may not seem like it will be of use to other departments in the company, but the `GetBooking` operation could be used by the billing department in order to get the booking details for customer billing. At the *external* level the services could be used by business partners (see Figure 4).



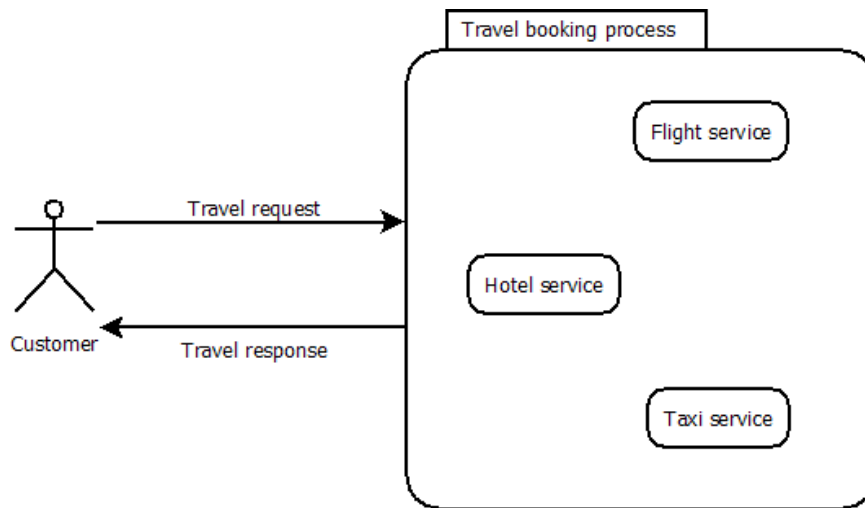
**Figure 4 Flight service at the external level**

Figure 4 shows some of the services for a travel booking company. The travel booking company uses its own internal `Hotel` service for booking hotels, an external `Taxi` service for booking taxis and the external `Flight Service` offered by the airline for booking flights.

These boundaries cause a lot of misunderstanding with the definition of an SOA. When creating services at the project level, it is recommended that they are also considered at the enterprise level and ideally, at the external level (Erl, 2004). However, it should be noted that even if a service is not considered at these levels it does not mean it is not part of an SOA.

#### **1.1.3.3 Business processes**

According to (Newcomer & Lomow, 2005), a business process is ‘a real-world activity consisting of a set of logically related tasks that, when performed in the appropriate sequence, and according to the correct business rules produces a business outcome’. In terms of service orientation, an orchestrated business process is one that calls a service or number of services, in order to fulfil a business transaction. This could be a process that calls a single service to carry out a transaction or it could call multiple services, the responses of which could then be aggregated and returned to the client. For example, the travel booking company may have a travel booking process that consumes multiple services, internal as well as external, in order to make travel arrangements (see Figure 5).



**Figure 5 Travel booking process**

In Figure 5 the customer sends a destination and the travel dates in a travel request to the travel booking process. The travel booking process may then query the hotel service, taxi service and the flight booking service. It may be the case that the travel booking process queries multiple services for each of these aspects and then compares the responses from each of these and then makes decisions using a set of business rules. For example the process may compare the prices of a number of flight services in order to offer the least expensive travel arrangements.

The travel booking process could also compare the hotels in terms of the distance from the airport. In this case, the process may query all of the services in parallel. Alternatively, the process may perform the calls to services in a strict order so that it can use the results from one service to determine the inputs of another service. For instance, if there are no taxi services available between an airport and a hotel another hotel may have to be chosen. However, in all cases the business process and calling of the sub-services (flight, hotel and taxi services) are invisible to the customer.

The travel process can also be accessed as a service, for example it may be used in a business travel process. The business travel process could use other services that are called before or after the travel process. For example, before the travel process is called the business travel process may call a service that queries the company's database to see if the employee is entitled to business or economy class seating.

#### **1.1.3.4 Service based technologies**

One technology that specifically aims to fulfil the needs of SO, is web services. Web services are used to implement services in an SOA. Web services are a platform designed for the description, discovery and messaging of services. All of these technologies are implemented using the eXtensible mark-up language (XML). XML is a framework that can be used as a notation for creating mark-up languages (Moller & Schwartzbach, 2006). The advantage of using XML in an SOA is that it is implemented by a number of programming languages and platforms, thereby improving interoperability.

The description of services is implemented using the web service description language (WSDL). A WSDL document describes a service and the functionality it can perform, as well as a physical resource to access the service. The discovery aspect of a service is implemented using universal description, discovery and integration (UDDI). UDDI is a repository for service descriptions, which allows requesters to search and find the service that meets their needs. Web service messaging is implemented using the simple object access protocol (SOAP). SOAP messages are documents which contain data concerning the message (header) and the message itself (body).

There have been other efforts to improve the interoperability of software systems such as the common object request broker architecture (CORBA) but these have been largely unsuccessful (Newcomer & Lomow, 2005). One of the reasons that CORBA has been considered unsuccessful was its lack of interoperability between the products (Hohpe & Woolf, 2003). Also, CORBA uses a custom protocol called the internet inter-orb protocol (IIOP) which means that this new protocol must be used. SOAP however, uses common web protocols, HTTP and XML. Another reason is that no port for communication was chosen as part of the standard, so a new port had to be opened for anyone wishing to communicate using CORBA (Jong, 2002). SOAP on the other hand, uses the internet port which means that no configuration is required. CORBA is also a complex standard, which makes it difficult to use (Cohen, 2001).

Business processes can be written in the business process execution language (BPEL), which is based on web services and XML. This makes BPEL an ideal candidate for making composite applications that consume web services. Like web services, BPEL uses WSDL for description and SOAP for messaging. Web services and BPEL both use XML as their data format making them interoperable. BPEL also uses some of the advanced XML functionality such as XQuery for querying documents and extensible stylesheet language transformations (XSLT) for transforming documents. In a BPEL process document there are typically the following:

- Partnerlinks - the services to be consumed
- Variables - for maintaining persistence throughout the process
- Fault handlers – for handling faults gracefully
- Process logic – the sequence in which partnerlinks are invoked, variables assigned.



Examples of the constructs used in the process logic are:

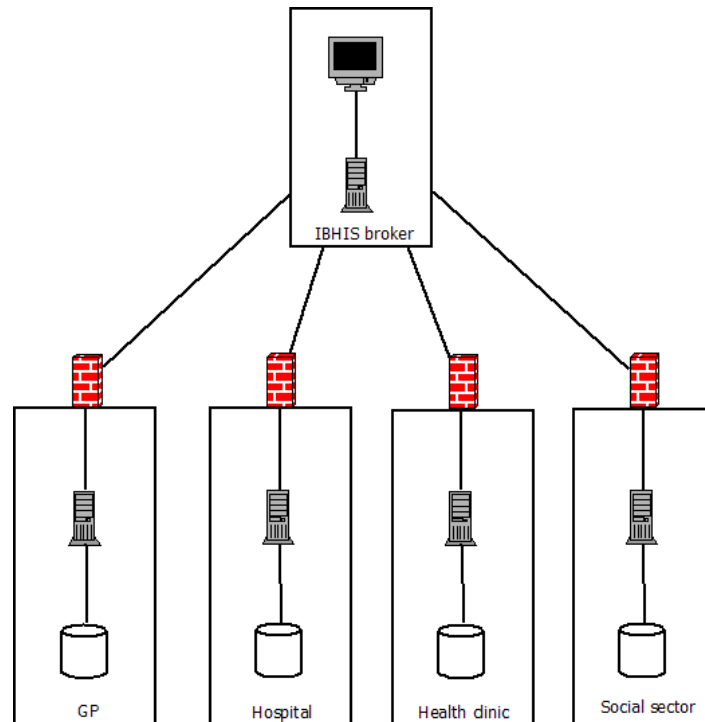
- flow – for concurrent activities
- receive – receives messages from client
- invoke – calls a service for consumption
- reply – sends response to client
- assign – updates a variable
- if – conditional logic
- throw – for throwing a fault inside the process

One of the main reasons for using an SOA for MCA is the SOAP protocol used for messaging. As long as a device supports web services or has the ability to parse SOAP messages, then it should be able to access a service or business process.

## **1.2 The IBHIS project**

An example of a system that could benefit from having MCA is the Integration Broker for Heterogeneous Information Source (IBHIS) broker (Turner et al, 2004). The IBHIS broker was part of the IBHIS project that investigated the integration of data from heterogeneous data sources using service-based systems.

The domain for this project was healthcare and examples of the data sources are patient records from primary care doctors, hospitals and other health agencies. Figure 6 shows an overview of the IBHIS broker (Kotsiopoulos et al., 2003).



**Figure 6 IBHIS broker overview**

The data sources shown in Figure 6 could be from machines using different operating systems (e.g. Windows or Linux) and stored using different database technologies e.g. MySQL or Oracle.

The IBHIS broker receives a query, which could be from a general practitioner or a hospital doctor. This query is then translated into the local language of each data source with the help of an ontology (Turner et al, 2004). The ontology is used to describe domain concepts and their relationships and is used like a dictionary to ensure that terms in a domain are used consistently. This means that the domain language used by the different data sources is consistent and that a global query could be translated into local queries for each data source. The IBHIS broker then makes calls to each data source using web service technologies. The results returned from the data source are aggregated and returned to the user.

The IBHIS broker would benefit from being accessible from multiple channels so that patient data can be accessed by mobile workers. These workers, such as on-call doctors or ambulance drivers, could access patient data from multiple sources on demand, at the time and place in which it was needed. However, when the IBHIS broker was created there was only one delivery channel in mind – that of a web application. For this study the IBHIS broker will be reengineered for MCA using service-based technologies and the issues of performance and agility associated with such reengineering will be investigated.

### **1.3 Reengineering**

Reengineering is an aspect of software rejuvenation, which is to ‘increase the overall quality of an existing system’ (Pfleeger, 2001). Listed below are the three scenarios for reengineering (Jacobson & Lindstrom, 1991).

1. A complete change of implementation technique and no change in the functionality.
2. A partial change in implementation technique and no change in functionality.
3. A change in functionality

There are two more scenarios which could be added to this list:

- A complete change in the implementation technique and a partial change in the functionality.
- A partial change in the implementation technique and a partial change in the functionality.

The second of these new scenarios describes the reengineering of IBHIS towards MCA using service-based technologies.

### 1.3.1 Phases

The reengineering process has three main phases. These are: *reverse engineering*, *transformation* and *forward engineering* (Pfleeger, 2001). The reverse engineering phase examines the source code and documentation of the original system, in order to create documentation of the current state of the system e.g. use cases, class diagrams and flow documents. The transformation phase involves stating how the original system will need to change in order to reach the target system. The forward engineering phase involves making changes to the original system to reflect the target system, based on the transformation phase.

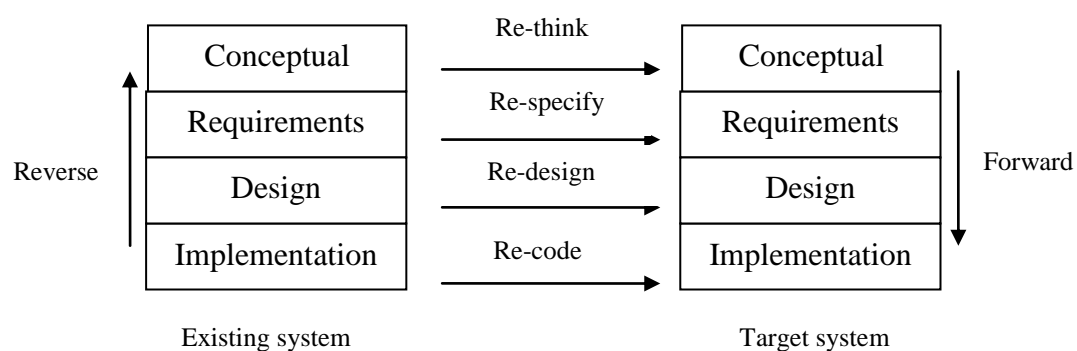
One of the main activities of reverse engineering is re-documenting (Arnold, 1993). Documentation about the system is constructed from the source code which shows an accurate picture of the current state of the system. The reason for using system code is that design documents may be out of date and not representative of the system. Re-documenting is performed using various tools e.g. static analysers, depending on the language that the system is written in and the type of documentation that is required (Pfleeger, 2001).

The main activity of forward engineering is recoding the system (Pfleeger, 2001; Arnold, 1993). In order to change the system code there are two approaches that can be used: *black box* – where no knowledge of the code is needed (e.g. wrapping) and *white box* – where the source code is changed. Black box methods are usually seen as

an interim solution rather than a permanent one (Anand et al., 2005). Black box methods can also be difficult to apply in some cases, for example, business logic may be tied up with presentation logic (Koutsoukos et al., 2006). White box reengineering is potentially a more permanent solution but does require extra time to understand the system and make the required changes (Hasselbring, 2004). As with re-documenting, there are a number of tools that can be used for recoding depending on the languages used e.g. text editors, integrated development environments (IDE) and program restructuring systems.

### 1.3.2 Reengineering methods

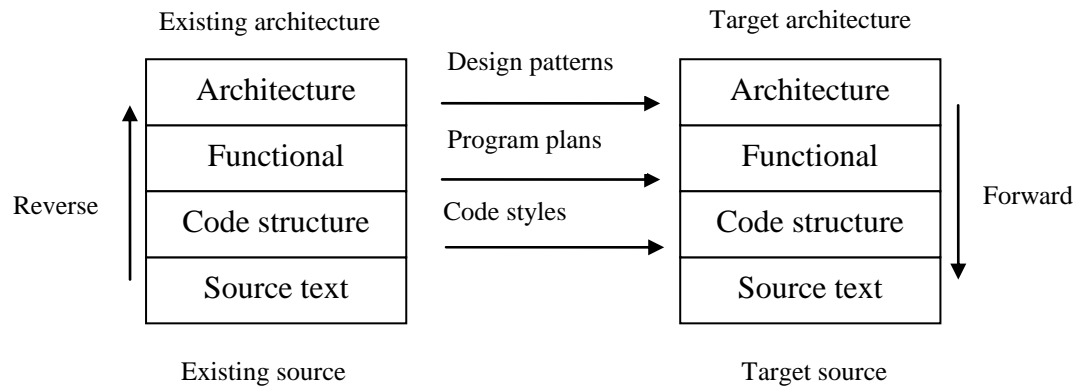
Reengineering methods can be grouped into two types, *structured* methods and *formal* methods (Arnold, 1993). Structured methods are those that have a set of steps but do not have mathematical underpinnings. Formal methods are based on mathematical underpinnings. Rosenberg, (1996) proposes the following general model for reengineering (See figure 7):



**Figure 7 Rosenberg general model for software reengineering**

The reverse engineering direction is on the left, the forward engineering direction on the right and the transformation steps in the middle. Changes start at the required level of abstraction and then move downward toward the implementation of the target

system (Rosenberg, 1996). This model is an extension to the model proposed by Arnold (1993), which does not include a conceptual level. This extra level of abstraction describes the functional characteristic in general terms whereas the requirements level describes it in detail. Another generic reengineering method is the Horseshoe method (Bergey et al., 1999) shown in Figure 8:



**Figure 8 Horseshoe reengineering method**

Figure 8 shows that the Horseshoe method is similar to the Rosenberg model. The difference being that the Horseshoe method views the system as four levels of abstraction whereas the Rosenberg model only sees as two, but includes requirements and the conceptual level. Using these generic models the following levels can be identified:

- Code
- Functional
- Design
- Architectural
- Requirements
- Conceptual

Reverse engineering, transformation and forward engineering can happen on all of these levels. There are reengineering methods that focus on specific levels on the reengineering process e.g. at an architectural level (Hunold et al. 2008; Lung, 1998.). There are also methods that look at transforming a system from one paradigm to another e.g. an existing system to an object-oriented system (Jacobson & Lindstrom, 1991; Berzins et al, 2000) or component based system (Alvaro et al., 2003). There are also reengineering methods that look at achieving a certain goal such as reusability (Jarzabe, 1993) and integration (Sneed, 2005; Liem et al., 2006.) or good design practices (Chu et al., 2000).

## **1.4 Research methods**

Before attempting to reengineer the IBHIS broker for MCA, it was necessary to investigate the different service-oriented and multi-channel reengineering methods and architectures that are available, as well as the challenges that are likely to be encountered. In order to determine this information, the Systematic Literature Review (SLR) method was used. An SLR enables information to be gathered and synthesised to make it meaningful. Alongside the SLR, a case study (CS) methodology was used to investigate the specific research objectives (outlined in section 1.5). A CS was used due to the exploratory nature of the research questions and the fact that multiple units of measurement were gathered for the study.

## **1.5 Research Objectives**

The aims of the study were:

1. To investigate the potential impact on system performance and agility that may occur when an underlying business process is exposed as a service to enable MCA.
2. To investigate if reengineering a system as an SOA improves agility.
3. To create an MCA reengineering method to transform single-channel systems into multi-channel.

In order to fulfil these aims a set of objectives were needed.

### **1.5.1 Objective 1**

The first objective, which addresses the first aim, was to investigate the challenges of reengineering a system for MCA. This was investigated in SLR research question three:

- SLR-RQ3 - What are the issues that need to be addressed when attempting to reengineer a system as a service?

As a result of the SLR, system performance was identified as an important challenge for multi-channel systems. This was investigated in case study research question two:

- CS-RQ2 - Does the inclusion of extra layers required for MCA reduce performance?

A potential trade off with performance and agility was identified when investigating potential technologies to improve the performance of business processes used for MCA. This was investigated in case study research question one.



- CS-RQ1 – Does the use of native language calls to improve the performance of a system reengineered for MCA, lead to a reduction in agility?

### 1.5.2 Objective 2

The second objective is to investigate the claim that an SOA improves the agility of a system, which is the second aim. This is investigated by case study research question three:

- CS-RQ3 - Does reengineering a system as an SOA improve agility?

### 1.5.3 Objective 3

In order to achieve the third aim, the objective was to identify the architectures and methods to enable the IBHIS broker to be reengineered for MCA. This objective was addressed by SLR research questions one and two:

- SLR-RQ1 - Have there been any proposed frameworks for reengineering a system as a service?
- SLR-RQ2 - Are there any service-oriented reengineering methods or architectures that focus specifically on MCA?

The Multi-channel Access Reengineering Method (McARM) and MCA architecture (McArc) were proposed based on the results of the SLR. These were investigated in case study research question four: CS-RQ4 - Is McARM an effective method for reengineering a system for MCA?

## 1.6 Contributions

The first contribution of this work is the proposal of a method that can be used to reengineer a software system for access from multiple devices. This method, McARM was developed as a result of analysing and integrating several aspects of existing reengineering methods and then applying considerations that are specific to MCA.

The second contribution is a set of multi-channel architectures, which can be used for creating a multi-channel system. The third contribution is a method for measuring the agility of a system. Agility was compared through the creation of a quality model, which was applied using software metrics and system assessment. Agility was also measured using agility test cases. A method was created for creating the agility test cases.

For a multi-channel business process, the research found that native language calls (NLC) should not be used instead of protocol based messaging (PBM) for messaging between the services and business processes to improve performance. NLC should be used if minimal performance gains are important and agility is not a strong concern. It was also found with each additional layer required for MCA there was a decrease in system performance. It was also found that reengineering a system as an SOA does improve the agility of a system.

## 1.7 Thesis outline

This thesis is organised as follows:

- Chapter 2 – This chapter details the planning and results of the SLR carried out in order to investigate reengineering a system for multi-channel access.

- Chapter 3 – This chapter proposes the Multi-channel Access Reengineering Method (McARM) and an architecture (McArc), for exposing the IBHIS broker to multiple channels. This chapter also details the performance/agility trade-off which includes a method for measuring agility.
- Chapter 4 – This chapter details the case study carried out to investigate the performance/agility trade-off, the impact of the extra layers required for MCA on performance, the claim that an SOA can improve agility and finally the McARM method.
- Chapter 5 – This chapter presents the results of the case study.
- Chapter 6 – This chapter discusses the results of the case study
- Chapter 7 – This chapter evaluates the research methodologies used.
- Chapter 8 – This chapter presents the conclusions and suggests future work and suggests changes to the agility model.

## Chapter 2: Systematic Literature Review

In order to investigate the various methods that could be used to reengineer the IBHIS broker for MCA, a systematic literature review (SLR) was performed. An SLR is a structured literature review that aims to gather literature, extract data and synthesise the data in a systematic way. The benefit of having a clearly defined process means that the review will be more thorough and less biased than an unstructured literature review, increasing its scientific value (Kitchenham, 2004). There are three main phases for an SLR, these are: *planning*, *conducting* and *reporting*. This chapter reports the planning and conducting phases.

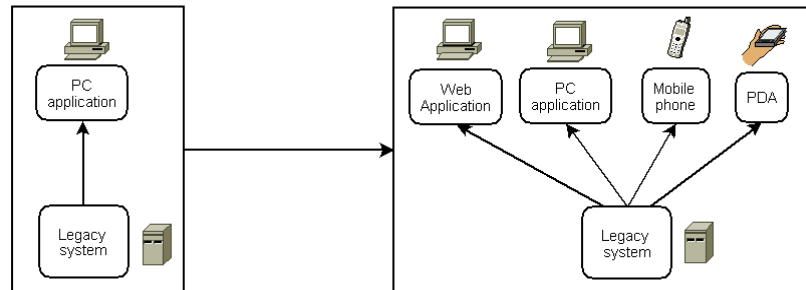
### 2.1 Planning

A protocol was created for the planning phase, which outlines how the SLR was conducted (Kitchenham et al., 2007). This section is based on this protocol (Jefferies et al. 2008) and outlines the following: the *background of the study*, the *review questions* and the *review methods*.

#### 2.1.1 Background

Many systems are built with one delivery channel in mind (Newcomer & Lomow, 2005). In order to offer their functionality to multiple channels an existing system could be completely replaced, either by creating a new system or buying a system off-the-shelf (Koutsoukos et al., 2006). However, this may be expensive and time

consuming in terms of costs and the potential downtime (Newcomer & Lomow 2005; Hasselbring 2004; Sneed 2007). Alternatively, it is possible to offer the same functionality to multiple channels by reengineering the original system (see Figure 9).



**Figure 9 Single to Multi-Channel**

As covered in chapter one, one way to offer a system to multiple channels – and therefore enable MCA - is to implement an SOA. Therefore the focus of this review was to investigate reengineering as an SOA for MCA.

A previous SLR on service-based systems was conducted by Brereton et al. (2006), which aimed to summarise the evidence concerning service-based systems. Two of the issues highlighted by Brereton et al., (2006) were ‘migration’ and ‘heterogeneity of portable access devices’. These were investigated as part of the current study.

### 2.1.2 Review questions

The four main areas investigated were:

- Service-oriented reengineering
- MCA reengineering methods
- Architectures for achieving MCA
- Issues relating to reengineering a system as a service.

This information was used to discover a suitable reengineering method and architecture for reengineering the IBHIS broker for MCA, as well as understanding any problems that may arise. The research questions for the study were:

- SLR-RQ1 – Are there any proposed frameworks for reengineering a system as a service?
- SLR-RQ2 - Are there any service-oriented reengineering methods or architectures that focus specifically on MCA?
- SLR-RQ3 - What are the issues that need to be addressed when attempting to reengineer a system as a service?

A preliminary search of the literature had determined that few methods existed that were specific to reengineering for MCA. Therefore the SLR was extended to investigate service-oriented reengineering methods.

### **2.1.3 Review methods**

The review methods section outlines the following aspects of the SLR:

- The search terms and resources
- Study selection criteria
- Quality assessment
- Data extraction strategy
- How the data will be synthesised

The following sections detail these aspects further.

### 2.1.3.1 Search strategy and resources

In order to search electronic sources, such as online journal databases, it is necessary to construct a search string that is comprised of suitable search terms. In order to determine the search terms that would be used, a trial search was conducted using the Google search engine with the following string: '*white box reengineering for SOA and multi-channel access*'. After reading the papers found, relevant search terms were listed along with synonyms, abbreviations and alternative spellings (see Table 1).

Term	Synonym	Abbreviation	Alternatives spellings
Service-oriented reengineering	service enablement, service oriented re-design, service oriented migration	-	re-engineering
Multi-channel Access	mobile web services	-	-

**Table 1 Search terms**

Combining the synonyms and alternative spellings in Table 1 resulted in the following search strings:

- **String 1:** service oriented reengineering OR service oriented re-engineering OR service enablement OR service oriented re-design OR service oriented migration
- **String 2:** multi channel access OR mobile web services

The following sources were used for the SLR:

- IEEExplore
- ACM digital library
- Google search engine
- Keele Library

The first two sources were chosen as they index well known journals in the computing field. The Google search engine was used to find more recent papers that may not have been published in journals. Keele Library was chosen as it contained books with information relevant to the search.

### 2.1.3.2 Study selection

Once the searches are conducted the next stage of the review process is to assess the results from each source to discover which are relevant to this study. To achieve this, a set of *inclusion/exclusion* criteria were applied to remove papers that were not suitable. To be *included* they had to be full papers including one or more of the following:

- Methods for service-oriented reengineering
- Reengineering methods and architectures for MCA
- Issues found when reengineering a system as a service and MCA

The criteria for being *excluded* from the review were any papers that covered:

- Reengineering methods that were theoretical (not tested with a real system)
- Reengineering methods that used a black box approach

Listed below is a summary of the process used to select primary sources:

1. Selection was made according to title, keywords, abstract and perceived relevance to the problem.
2. The papers found were then filtered further according to the *inclusion/exclusion* criteria.



### **2.1.3.3 Quality Assessment**

In order to maintain the integrity of the SLR the papers should adhere to measures of quality set out by the reviewer(s). This assessment looks at the methodological soundness of the experiments reported. For this review, quality assessment was not carried out as the aim of the literature review was to map out the area rather than directly compare techniques. Quality was not as critical since no aggregation was carried out as it was a classification activity.

### **2.1.3.4 Data Extraction Strategy**

The data extraction process involves reading the papers found and documenting pertinent information using an extraction form. For this SLR, two types of data were extracted; information about the paper and information that answered one or more of the research questions. The information gathered about each paper is listed below:

- Date
- Title
- Authors
- Reference
- Database source
- References found
- Author contacted for further information/papers

For the research questions, the following was also extracted from each paper:

1. Proposed methods for service-oriented reengineering of a system.
2. Reengineering methods and architectures for MCA.

### 3. Technical issues found when reengineering a system as a service and MCA

The information for the research questions was a summary of that found in the original paper. The reason for summarising the information was for clarity and to keep the terminology as consistent as possible. Relevant references found in a paper were also noted and searched for. If obtained, they were also subject to inclusion/exclusion and data extraction. Authors were also contacted if it was felt that further information or papers were required.

#### **2.1.3.5 Data synthesis**

Once the data has been gathered the next stage is to analyse and synthesise it in order to address the research questions of the study. The data was synthesised for each research question as follows:

- SLR-RQ1 - The service-oriented reengineering methods were listed along with their steps.
- SLR-RQ2 - The MCA architectures were listed and the MCA reengineering methods were listed along with their steps.
- SLR-RQ3 - The service-oriented reengineering issues were grouped.

## **2.2 Results**

A total of 92 papers were found, 28 of which were used for data extraction after inclusion/exclusion criteria had been applied. Each research question will now be examined separately.

### 2.2.1 SLR-RQ1

Sixteen service-oriented reengineering methods were found (see Table 2):

Method	Reference
Wrap and service bus	Zhang et al. (2006)
Service Oriented Analysis and Design (SOAD)	Zimmermann et al. (2004a)
4 stage method	Krafzig et al. (2004)
Smooth migration	Hasselbring et al. (2004)
XML → legacy gateway	Newcomer & Lomow (2005)
XML and Web services Integration Framework (XWIF)	Erl (2004)
The Service-Oriented Migration and Reuse Technique (SMART)	Lewis et al. (2005)
Architecture based service-oriented reengineering approach	Zhang et al. (2003)
Feature analysis method	Chen et al. (2005)
Service-Oriented Reengineering (SOR)	Zhang et al. (2006)
Sensoria reengineering approach	Koutsoukos et al. (2006)
Code wrapping for SOA reuse	Sneed (2006)
Service-Oriented Software Reengineering (SoSR)	Chung et al. (2007)
Incubating legacy systems for service migration	Zhang & Yang (2004)
SoftLink	Sneed & Sneed (2003)
Migrating to Web services	Sneed (2007)

**Table 2 Service-Oriented Reengineering methods**

### 2.2.2 SLR-RQ2

The first part of this research question was to discover methods for reengineering a system for MCA. There were two methods found:

- Reengineering toward a channel agnostic, conversational system (Zimmermann et al., 2005)
- Multi-Channel Adaptive Information Systems (MAIS) (Comerio et al., 2004).

The second part of the research question was to find architectures for MCA. There were seven found in total (see Table 3).

Architecture	Reference
Service gateway + service integration bus	Ganesh et al. (2004)
Multi Channel adaptive	Marchetti et al. (2004)
Fundamental SOA	Krafzig et al. (2004)
Service Façade	Krafzig et al. (2004)
Process-enabled SOA	Krafzig et al. (2004)
Service Bus	Newcomer & Lomow (2005)
Reengineering toward a channel agnostic, conversational system	Zimmermann et al. (2005)

**Table 3 Proposed architectures for MCA**

Some of the architectures in Table 3 were from the same source. All of these architectures were for MCA, but some had extra layers for more complex systems.

### 2.2.3 SLR-RQ3

Four main categories of issue were found:

- Project management
- The existing system
- Architectural
- Web service technologies

Table 4 shows the issues found in each of these categories.

Category	Issue
Project Management	<ul style="list-style-type: none"> <li>• Risk</li> <li>• Staff</li> </ul>
Existing system	<ul style="list-style-type: none"> <li>• Code duplication</li> <li>• Understanding</li> <li>• Dependencies</li> <li>• Validation</li> <li>• Poor documentation</li> </ul>
Architectural	<ul style="list-style-type: none"> <li>• Granularity</li> <li>• Relationships between services</li> <li>• Complexity</li> <li>• Compensation (transaction rollback)</li> <li>• Naming conventions</li> <li>• Non-functional requirements</li> </ul>
Web services	<ul style="list-style-type: none"> <li>• Performance</li> <li>• State management</li> <li>• Interoperability</li> <li>• WSDL</li> <li>• Security</li> <li>• Using between layers</li> <li>• Data translation</li> <li>• Tool support</li> <li>• BPEL</li> <li>• Testing</li> </ul>

**Table 4 Service-oriented reengineering issues**

In Table 4 the project management and the existing system categories are similar to what would be found in any other reengineering project. The more service-oriented specific problems were found with the architecture and the technologies used (e.g. web services). Some of the categories were grouped under a generic title, for example the category ‘WSDL’ related to any issues regarding the limitations the WSDL language and documents.

## 2.3 Discussion

This section examines the findings of the SLR and their implications.

### **2.3.1 SLR-RQ1**

Sixteen service-oriented reengineering methods were found with differing coverages of the software lifecycle. Some had almost full coverage of the software lifecycle e.g. SOR (Zhang et al., 2006), while others focused on specific stages e.g. SMART (Lewis et al., 2005) which focussed on analysis and design. The ordering of the reengineering activities also differed between the methods. Ordering can be performed in one of three ways (Koutsoukos et al., 2006);

1. Top down - starts with the business processes of the target system and works down to the code of the existing system.
2. Bottom up - starts with the existing system code, creating services from the code and then composing them as business processes.
3. Meet-in-the-middle - starts at both ends of the system (business processes and system code) with the aim of using services found in the system code to fulfil the target business processes.

A bottom-up method can create an architecture that is too finely grained with many low level functions that may not be required as services, but will include all of the system functionality. A top-down method may result in business processes that do not reflect the current system functionality, but will address the business needs. A meet-in-the-middle method is a compromise between these two as it compares the potential services in the system code with business processes required.

### **2.3.2 SLR-RQ2**

Two methods for reengineering existing systems for MCA were found. In the Zimmermann et al. (2005) paper, a meet-in-the-middle MCA reengineering method

was proposed. The method exposed the system to two channels; a web-browser channel for small to medium sized companies and a web-services channel for business integration with larger companies. Although the system was not exposed for heterogeneous devices it could be argued that the web services channel could be used outside of the integration context. The second reengineering method found was created as part of the MAIS project (Comerio et al., 2004). This meet-in-the-middle reengineering method aimed to adapt existing services so that they can be accessed by different types of networks and devices.

There were seven different MCA architectures found. Each of these architectures has different layer types and numbers of layers. These architectures will need to be examined for any similarities or differences to see if a generic MCA architecture can be created from them.

### **2.3.3 SLR-RQ3**

The problems found when reengineering a system towards an SOA will be discussed in the context of reengineering the IBHIS broker for MCA.

The state of the existing system could have an affect on the level of effort required for the reengineering project e.g. separation of concerns (Koutsoukos et al. 2006). If the system has layers that are overlapping it will take more effort to create services required for an SOA. For example, if the business logic is tied up with the presentation logic, the layers would need to be untangled and service interfaces created. This was known to be a problem with the IBHIS broker.

In terms of the architecture, when designing a system that can be accessed by heterogeneous devices the number of interactions with the system should be minimised as the devices may be using unreliable networks. Therefore, creating coarse grained services that promote delegation is important. Another architectural issue is the non-functional requirements of the system e.g. system performance and architectural extensibility (Hasselbring, 2004). *Performance* is an issue when reengineering a system for MCA (Zimmermann et al. 2004b; Hasselbring et al, 2004; Krafzig et al, 2004; Zhang et al., 2006; Brown & Reinitz, 2003), as the devices that are accessing the system are unknown and may have limited system resources.

One of the issues found with using web service technologies such as SOAP and WSDL was the effect on performance. These technologies use XML messages which can be verbose due to the additional tags required. The system performance can be decreased by these large messages especially if there are many messages (Zimmermann et al., 2004). This issue of the verbose nature of web service technologies, coupled with the performance concerns of MCA could mean that the response times are unacceptable for the user. The *extensibility* of the system is also an important consideration. The system should be able to accommodate change easily with minimum effort.

In relation to the previous SLR conducted by Brereton et al. (2006), this review found that there is now a larger body of evidence that looks at supporting reengineering and understanding of existing service-based software. In terms of the technical issues found by the current SLR, there were some similar to the previous SLR. These were the comprehension/understanding of service-oriented software, quality of service,



testing and security. The main difference between the issues found in the two reviews is that when exposing a system for MCA, rather than business reasons, the business considerations are less likely to apply e.g. contract negotiation.

### 2.4 Conclusions

For SLR-RQ1 *‘have there been any proposed frameworks for reengineering a system as a service?’* sixteen reengineering methodologies were found for service-oriented reengineering. The methods had varying coverage of the software lifecycle and also different orders in which the stages of the methods are performed. This highlighted that it would be beneficial to perform a comparison of these methods as this has not been conducted and also none of these methods refer to one another.

For SLR-RQ2 *‘are there any service-oriented reengineering methods or architectures that focus specifically on MCA’* two reengineering methods and seven architectures were found. Neither of the reengineering methods is comprehensive in their coverage of the reengineering lifecycle, which needs to be investigated further. The seven architectures for MCA found need to be compared and contrasted to see which are better suited to the original problem of service-enabling the IBHIS broker so that it can be accessed by heterogeneous devices.

For SLR-RQ3 *‘what are the issues that need to be addressed when attempting to reengineer a system as a service?’* the problem of verbose messages used by web services and the potential performance reduction needs to be investigated further.

## Chapter 3: Reengineering for Multi-channel access

This chapter builds on the findings of the SLR. Firstly, an MCA reengineering method (McARM) is proposed. This addresses the need for an MCA reengineering method highlighted by SLR-RQ1 and SLR-RQ2. The second part of the chapter proposes an MCA architecture for the reengineering of the IBHIS broker, based on a comparison of the architectures found by SLR-RQ2.

### 3.1 McARM

After analysing the reengineering methods found by the SLR (see sections 2.3.1 and 2.3.2), it was found that none were fully comprehensive (see Appendix A). There were two methods specifically for reengineering for multiple channels found, both of these methods covered most of the reengineering lifecycle, however, neither of the methods were comprehensive. The MAIS method (Comerio et al., 2004) did not detail code understanding or the design of web service interfaces. The methodology proposed Zimmermann et al. (2005) did not detail service identification in the existing system or recoding the existing system. Also, this method only exposed the system to two channels; a web-browser channel for small to medium sized companies and a web-services channel for business integration with larger companies. It did not focus on creating a system for heterogeneous devices. The majority of the service-oriented reengineering methods covered around half of the software lifecycle, with the most comprehensive having two stages that were not detailed. It was also found that these methods were mainly aimed at commercial software systems where performance was not an issue.

Therefore, it would be difficult to apply any of these reengineering methods when the entire reengineering lifecycle is required and also, in some cases specialist tools/knowledge would be required which may not be obtainable or stable enough to use. Thus, a new method was created by combining the stages of several of the existing reengineering methods found, **using the commonly used approaches to the stages that were appropriate for reengineering for MCA**. To make the method simple to use, the unified modelling language (UML) was chosen as the analysis and design notion. The reason for choosing UML is that it has a suitable range of diagrammatical representations and is also widely used in industry and academia. McARM is created specifically for reengineering a system for MCA. McARM is different to traditional reengineering methods as it reengineers towards very coarse granularity and delegation in order to keep the number of calls over the network as low as possible. The reengineering methods described in sections 2.3.1 and 2.3.2 do not specifically have a very coarse grained system as a target. McARM also makes recommendations specific to MCA during requirements gathering, analysis and design.

The rest of this section outlines the multi-channel access reengineering method (McARM) in terms of the reengineering stages, the ordering of these stages and how each stage should be performed.

### 3.1.1 Stages

The following lists the stages of the McARM method, which were derived from examining the methods found by SLR-RQ1 and SLR-RQ2 (See Appendix A):

- Requirements gathering/analysis
- Business process modelling

- Business process implementation
- Service interface implementation
- System redesign
- System recoding
- Service identification in existing system
- System understanding

Business process implementation and modelling were originally grouped as one task called business process management. However, these two activities may not happen at the same time and so were separated. Each of the stages for all of the reengineering methods found by the SLR were compared and contrasted in order to decide which stages and approaches to implementing these stages were the most appropriate for McARM. The use of UML and not requiring specialist tools were important factors in this decision. In some cases, a stage would combine approaches of more than one method.

### **3.1.2 Stage order**

McARM is a ‘meet-in-the-middle’ method, which means it will result in a system that has services that matches the use cases with the functionality that already exists in the system. The stages in the reengineered method outlined in 3.1.1 need to be reordered according to the meet-in-the-middle method and are now as follows:

1. Requirements gathering/analysis
2. System understanding
3. Service identification in legacy system
4. Business process modelling

5. System redesign
6. System recoding
7. Service interface implementation
8. Business process implementation

This ordering indicates that firstly, the problem domain is understood in stages 1 and 2. Next, the transformation of the system is carried out in stages 3, 4 and 5. The services and business processes are identified and then the system is redesigned based on the services. Finally, the implementation stages (6, 7 and 8) are performed. The system is recoded and the service interfaces are created from the system classes. The service interfaces are then used by the business processes.

### **3.1.3 Stage details**

The following subsections explain each of the stages for McARM, outlined in section 3.1.3.

#### **3.1.3.1 Requirements gathering/analysis**

The choice of a requirements engineering approach depends on the nature of the project being undertaken. For example, if a project will involve a lot of change, then an agile method of requirements engineering may be suitable (Beck, 2000). If a project is less likely to involve change and the system is going to be more complex, then a more traditional method, such as that proposed by (Sommerville & Sawyer, 1997) may be more appropriate.

However, the limitations of heterogeneous devices must be kept in mind. An example of a consideration would be the size of any data sets that are used. It is recommended that if the returned data set of a function is very large, this may not be suitable for being exposed for a multi-channel system due to the limited processing power of mobile devices and limited network speed of mobile networks. This limitation is less likely in a service-oriented reengineering method that does not focus on MCA.

### 3.1.3.2 System understanding

There are two sources that can be used to help in understanding the existing system. These are the *design documentation* and the *existing code*. Examples of design documentation are class diagrams and architectural diagrams. The outputs from examining the design documentation are:

- *Architecture* - an overview of the system.
- *Component table* – information concerning the main components of the system such as: function name, size, level of documentation etc. (Lewis et al., 2005).

Using the existing code, a model of the system can be created. The outputs suggested by Zhang & Yang, (2004) are:

- *Component class diagrams* – with their respective public interfaces (Erl, 2004).
- *Sequence diagrams* – derived from Graphical User Interfaces (GUIs).

Table 5 is a summary of the inputs and outputs for the system understanding.

Inputs	Outputs
<ul style="list-style-type: none"> <li>• Design documentation</li> <li>• Information from technical personnel</li> <li>• Source code</li> <li>• GUIs</li> </ul>	<ul style="list-style-type: none"> <li>• Architectural diagram</li> <li>• Class diagram</li> <li>• Component table</li> <li>• Sequence diagram</li> </ul>

Table 5 Inputs and outputs for system understanding

### 3.1.3.3 Service identification in legacy system

The objective is to identify the services that are needed for the system, based on the required business functions (Hasselbring et al., 2004). There are two sources that can be used to identify services in the legacy system. These are *stakeholders* and *documentation*. Examples of stakeholders that can be used to identify the services in the system are users, corporate architects and domain groups. The outcome of this analysis is a *service table* which identifies services from components and those by the organisation and information regarding those services. For the documentation, Comerio et al (2004), suggest either verifying the current service model if one exists or defining a new service model from the current system based on documentation, interface descriptions and specifications. Reference models can also be used for identifying services required for use in a wider context (Lewis et al., 2005). The deliverable documents are use cases, class diagrams and sequence diagrams.

Once the candidate services have been identified, the services should then be evaluated. An important consideration is the granularity of the services. The system needs to be as coarse-grained as possible. The reason for this is that an MCA system should have as fewer interactions between the client and the system as possible. The reason for this is due to the fact the unreliable mobile networks are potentially being used. Reducing the number of service interactions means that the number of points of

failure will also be reduced. Where possible, services should be combined into one service which can then delegate. This consideration is less likely in a service-oriented reengineering method that does not focus on MCA.

Examples of questions that should be asked in order to evaluate the services are:

- Which services are the better match for the goals and expectations of the migration effort?
- What are the services with greater potential for use by service consumers?
- What are services with a better match to existing capabilities?
- What are the interfaces for these services in terms of inputs and outputs?
- For each service, what are the specific legacy components that contain the functionality required by the services?
- What new code will have to be written to fully satisfy service requirements?

These are a subset of those suggested by Lewis et al. (2008). Table 6 provides a summary of the inputs and outputs for this stage.

Inputs	Outputs
<ul style="list-style-type: none"> <li>• Design documentation</li> <li>• Information from technical personnel</li> <li>• Reference models</li> <li>• Interface descriptions</li> <li>• Specifications</li> </ul>	<ul style="list-style-type: none"> <li>• Service table</li> <li>• Use cases</li> <li>• Class diagrams</li> <li>• Sequence diagrams</li> <li>• Evaluation table</li> </ul>

**Table 6 Inputs and outputs for service identification**

### 3.1.3.4 Business Process Modelling

When creating business processes, the requirements and the services identified are used as the inputs. The inputs and outputs of each service identified should be used as



a guide for the flow of the system. The business processes are represented using UML activity diagrams (Chung et al., 2007; Comerio et al., 2004). The activity diagrams should show:

- Inputs and outputs for the business processes
- Services that will be used by the business processes
- Data flow
- Service calls

The requirements document should be used to check that the business process satisfies all of the functionality required. Table 7 is a summary of the inputs and outputs for this stage.

Inputs	Outputs
<ul style="list-style-type: none"> <li>• Requirements</li> <li>• Documents from Service Identification</li> </ul>	<ul style="list-style-type: none"> <li>• Activity diagram</li> </ul>

**Table 7 Inputs and outputs for business process modelling**

### 3.1.3.5 System redesign

The system redesign stage aims to show the change of the system from its original state to a new state. This is based on the original system and the services identified for the new system. The redesign of the code and the service interfaces can be performed separately, but if redesigned at the same time it should ensure that the code and service interfaces are created in close alignment with each other. Therefore the McARM method recommends that the redesign of the code and service interfaces is done at the same time.

*Classes* – for redesigning the classes the stages from the XWIF method (Erl, 2004) were chosen. The reason XWIF was used is because it defines a comprehensive, manual, object-oriented design approach that requires no specialist tools. After determining the classes that will be required for the system, the following XWIF steps should be followed (Erl, 2004):

1. Create granular, task-oriented service classes by removing any methods that may need to have their own class
2. Group methods into finer grained classes
3. Identify service candidates from classes (check for reuse)
4. Review non-services classes (can they be put together or left granular?)
5. Identify cross consolidation opportunities (methods with compatible requirements may be put in generic classes)

*Service interface modelling* - XWIF can also be used to model the service interfaces. Based on the services identified, use cases and the activity diagrams the interfaces are modelled by using the following the steps:

1. Choose service model
2. Establish scope of business function
3. Identify requestors (other potential uses)
4. Identify data (resources should be separated from parameter data)
5. Explore application paths (is it optimal solution?)
6. Encapsulation boundary (what parts of application will service interact with?)
7. Model the interface
8. Map interaction scenarios (to understand dynamic binding situations)
9. Design message payload (model XML documents)

## 10. Refine service model using best practice strategies (Erl, 2004)

It may not be necessary to perform all the stages outlined but these are a useful guide of tasks that should be performed.

The two main ways in which an MCA reengineering effort differs from a service-oriented reengineering effort are the granularity of the services and heterogeneity of clients. Therefore, when modelling the service interfaces, these must be considered at each step. Table 8 is a summary of the inputs and outputs for system redesign.

Inputs	Outputs
<ul style="list-style-type: none"> <li>• Class diagrams</li> <li>• Service table</li> <li>• Activity diagrams</li> <li>• Documents from Service Identification</li> </ul>	<ul style="list-style-type: none"> <li>• Class diagrams</li> <li>• Service option table</li> <li>• Service interface descriptions</li> </ul>

**Table 8 Inputs and outputs for system redesign**

### 3.1.3.6 System recoding

The SLR did not find any generic methods for recoding. This is due to the many different programming languages and the fact that every project is different. However, the SLR did highlight useful advice related to recoding, for example Krafzig et al. (2004) recommended decoupling visual code from non visual code if required.

### 3.1.3.7 Service interface implementation

This stage looks at how the service interfaces are implemented. This is usually done by wrapping the legacy business logic with a service description document (Hasselbring et al., 2004; Krafzig et al., 2004; Chen et al., 2005; Zhang & Yang,

2004; Sneed, 2007). The service description documents can normally be created automatically by the development environment. Table 9 is a summary of the inputs and outputs of the service interface creation stage.

Inputs	Outputs
<ul style="list-style-type: none"> <li>Service interface descriptions</li> </ul>	<ul style="list-style-type: none"> <li>Service interfaces</li> </ul>

**Table 9 Inputs and outputs for service interface creation**

### 3.1.3.8 Business process implementation

The business processes should be based on the activity diagrams from the modelling stage. The business processes are created using an integrated development environment (IDE). Table 10 is a summary of the inputs and outputs of the business process creation stage.

Inputs	Outputs
<ul style="list-style-type: none"> <li>Business process activity diagrams</li> </ul>	<ul style="list-style-type: none"> <li>Business processes</li> </ul>

**Table 10 Inputs and outputs for business process implementation**

## 3.2 MCA Architecture

In order to determine the most appropriate architecture for the problem of reengineering the IBHIS broker, the architectures that were found by the SLR were evaluated against the following aspects:

- The number of layers in the architecture – if this is high there could be a negative affect on performance.
- The suitability of the architecture to the problem – does the architecture reflect the study goals?

- The types of layers in the architecture – do the layers in the architecture reflect those required for reengineering the IBHIS broker?

### 3.2.1 Summary of architectures

The architectures identified by the SLR and the number of layers for each is shown in Table 11.

Number	Source	Name	Layers
1	Zimmermann et al. (2005)	Channel agnostic architectural pattern	6
2	Ganesh et al. (2004)	Service gateway + Service Integration Bus	4
3	Krafzig et al. (2004)	Fundamental SOA	2
4	Krafzig et al. (2004)	Service Façade	3
5	Krafzig et al. (2004)	Process-enabled SOA	4
6	Newcomer & Lomow (2005)	Service Bus	6

**Table 11 Architecture summary table**

Table 11 shows that the number of layers used is between two and six. The fundamental SOA (Krafzig et al., 2004) has the least number of layers but further analysis is required to find out which layers are suitable to enable successful reengineering of the IBHIS broker (Section 3.2.2).

### 3.2.2 Suitability of architectures

The suitability of the architectures was assessed using a *weighted comparison table*, proposed by Shaw and Garlan (Pfleeger, 2001). This table allows the architectural features to be assessed according to a set of desired criteria, which are weighted by importance and finally, a rating is multiplied by this weighting. When choosing the comparison criteria for the weighted comparison table, quality factors should be used

(Bianco et al. 2007). Based on the objectives of the research, the quality factors for this study were:

- *Portability* – the system should be accessible by as many devices (current and future) as possible.
- *Performance* – the system may be accessed by devices with limited resources, which means that performance is a central issue. The fewer layers that the architecture has, the less impact there will be on the system performance (Hasselbring et al. 2004, Krafzig et al 2004).
- *Agility* – the architecture should be flexible so that it can be changed as needed.
- *Correctness (depth of architecture)* – although the architecture must have as few layers as possible it is also important not to have too few layers. For example, it may be undesirable to make low level functions available for access by external requesters or the system may be too fine grained, making the system difficult to understand for requesters.
- *Correctness (suitability of layer types)* – Do the layer types in the architecture represent those needed by the system?

The priority of each architectural property was rated between one and five (one being low) based on their relevance to the study. The architectures were assessed using the information available in the literature found by the SLR, the results were then entered into the weighted comparison table (see Table 12).

Architectural Properties	Priority	Architecture					
		1 Zimmermann et al. (2005)	2 Ganesh et al. (2004)	3 Krafzig et al. (2004)	4 Krafzig et al. (2004)	5 Krafzig et al. (2004)	6 Newcomer & Lomow (2005)
Portability	5	4 (20)	5 (25)	5 (25)	5 (25)	5 (25)	5 (25)
Performance	4	2 (8)	3 (12)	5 (20)	4 (16)	3 (12)	2 (8)
Agility	4	4 (16)	3 (12)	4 (16)	4 (16)	4 (16)	4 (16)
Depth	3	5 (15)	5 (15)	1 (3)	2 (6)	4 (12)	5 (15)
Suitability	3	4 (12)	3 (9)	1 (3)	2 (6)	3 (9)	4 (12)
<b>Total</b>		<b>71</b>	<b>73</b>	<b>67</b>	<b>69</b>	<b>74</b>	<b>76</b>

Table 12 Weighted comparison table for multi-channel architecture

The architecture with the highest total in Table 12 represents the architecture that most meets the required criteria. Therefore, architecture 6 (Newcomer & Lomow, 2005) is rated the most suitable for this study. However, it is shown in Table 11 that this architecture has the most number of layers and may cause performance problems.

### 3.2.3 Architectural layer types

The potential layer types of the proposed MCA architectures are shown in Figure 8. This can be used to analyse the existing architectures in order to create an architecture based on the needs of the target system, in this case the IBHIS broker.

L a y e r s	Device		Enterprise	Enterprise	Enterprise	Client/ presentation
	Channel controller	Service gateway	Process- centric			Channel access
	Integration			-		Communication infrastructure
	Business process	Integration	Process- centric			Business service access
	Process activity					
	Intermediary				Intermediary	
	Business services	Business services	Basic	Basic	Basic	Business service
	Application services	Business services	Basic	Basic	Basic	
	Core systems					Business data
Arc	1 Zimmermann et al (2005)	2 Ganesh et al (2004)	3 Krafzig et al (2004)	4 Krafzig et al. (2004)	5 Krafzig et al. (2004)	6 Newcomer & Lomow (2005)

**Figure 10 MCA Architecture layer comparison**

Figure 10 shows nine possible layers in a multi-channel architecture. Although some of the layer names on the same level are different, the description of each of the layers is the same. The most appropriate architecture from the weighted comparison table was architecture 6, but this has layers that were not needed for the current project so a simplified architecture, based on analysis of all of the architectures is proposed in section 3.2.4.

### 3.2.4 Proposed architecture

A simplified architectural layering was proposed for the multi-channel architecture (McArc), used for reengineering the IBHIS broker (see Figure 11).



Channel
Business process
Service

Figure 11 Architectural layering for reengineered IBHIS broker

The layers included in this architecture are as follows:

- *Service*: These are individual services that represent the services in the existing system.
- *Business process*: This layer represents the business processes that exist in the system.
- *Channel*: This is the layer that accesses the system e.g. program or web page on mobile or desktop.

Figure 12 compares McArc with those found by the SLR.

L a y e r s	9	Device	Enterprise	Enterprise	Enterprise	Client/ presentation	Channel
	8	Channel controller	Service gateway		Process centric	Channel Access	
	7	Integration			-	Communication infrastructure	
	6	Business process	Integration		Process centric	Business service access	Business Process
	5	Process activity					
	4	Intermediary			Intermediary		
	3	Business services	Business services	Basic	Basic	Basic	Business Service
	2	Application services	Business services	Basic	Basic	Basic	Service
	1	Core systems				Business Data	
Arc		Zimmermann et al. (2005)	Ganesh et al. (2004)	Krafzig et al. (2004)	Krafzig et al. (2004)	Krafzig et al. (2004)	McArc

Figure 12 Comparison of proposed MCA architecture with existing architectures

Figure 12 shows that McArc has one more layer than the architecture with the fewest layers – the Fundamental SOA (Krafzig et al., 2004), which meant that the architecture did not have a large number of layers. The *service layer* encompasses the bottom three layers, containing the core systems, the application services (utility services) and the business services. Layer one would be required however if the relevant code could not be extracted from the system. Layer four is a service facade and for unifying many services but as the authors themselves (Krafzig et al., 2004) state, this has limited use for a multi-channel application and is not required for McArc. The *business process layer* encompasses layers five and six and potentially seven. Layer eight is a channel access layer which is responsible for routing the request based on the channel type. This layer was not required for McArc as it was the data that was being exposed only. The *channel layer* is the client that accesses the system. With these layers, McArc should enable the IBHIS system to be accessed from multiple channels and improve the agility of the system without having a high impact on the performance.

## Chapter 4: The performance/agility trade-off

This chapter looks at the performance challenge when reengineering a system for MCA, identified by the analysis of the results of SLR-RQ3. A potential way of improving the performance of systems reengineered for MCA is identified, which uses native language calls (NLC) instead of the protocol based messaging (PBM) for calling web services from business processes. However, unlike PBM, NLC are not usually designed with interoperability in mind and therefore could have an affect on agility. This leads to the primary research question for this study: does using NLC to improve the performance of a software system reengineered for multi-channel access, lead to a reduction in agility?

### 4.1 Performance

After conducting the SLR it was found that a problem that required further investigation was the *performance* of a system reengineered for MCA, with regard to the verbose nature of web services messages and the limited communication bandwidth of mobile devices (Zimmermann et al. 2004, Zhang et al. 2006, Brown & Reinitz 2003). As multi-channel systems can potentially be called from mobile devices which have low processing power, through limited networks, performance is important in multi-channel systems. However, the extra layers required for an SOA can have a negative effect on performance (Hasselbring et al. 2004, Krafzig et al. 2004). In terms of usability an acceptable response time of a desktop application is ten seconds, after this a user's attention will shift to other tasks (Neilson, 1993). In a mobile context the acceptable time is reduced to between 4-8 seconds depending on

the task being performed and the location of the task being performed (Roto & Oulasvirta, 2005). However, users may be willing to accept slower response times for the convenience of having the data on their mobile devices. The rest of this section looks at the performance problems that arise from using web service technologies, such as web services and BPEL, and ways to address these challenges.

#### 4.1.1 Web services

The main areas that web services can cause performance reduction as identified by the SLR-RQ3 are:

- *Additional layers* - Adding the SOA layers required for MCA can reduce performance e.g. the web services and business processes (Hasselbring et al. 2004, Krafzig et al 2004). Each additional layer can be expected to have an impact on the response times due to the extra messages required to implement the layer.
- *Data conversion* - This is the process of converting XML data contained within the SOAP message to the native language and vice versa, using serialisation/de-serialisation. This usually requires the addition of XML tags onto the data and insertion into an XML document structure. Sneed (2007) reported that such data conversion reduces response times and also consumes system resources.
- *Message verbosity of SOAP* - Using web services can reduce performance as the SOAP protocol used for messaging is verbose (Zimmermann et al. 2004, Park et al. 2006).
- *Inappropriate use* – Web services are not appropriate for all situations, such as systems that are tightly coupled and not subjected to frequent change. (Brown & Reinitz, 2003).

#### **4.1.2 Improving web service performance**

The following list shows the areas in which web services can affect performance (Machado & Ferraz, 2005):

1. Message size
2. Calculation of the message size
3. XML parsing
4. Serialisation and de-serialisation
5. Connection establishment (HTTP 1.0 requires three way handshake algorithm)
6. Network level

Previous research has attempted to address some of these areas in a resource restricted context. Compression and binary XML representation have been used to address the problem of large web service message size (Kohlhoff and Steele 2003, Tian et al. 2003, Sandoz et al. 2003). In order to address XML parsing, one approach is to use a wireless portal architecture (Adacal & Bener, 2006). In this solution, XML processing (SOAP) is performed by a gateway that then returns the data in a form that can be handled by a mobile device. Another method to reduce serializing and parsing on the mobile device is to separate message content from the syntax and then stream the messages (Oh & Fox, 2007). The issue of performance at the network level was investigated by using alternative transport protocols such as UDP instead of HTTP for mobile data transfer (Lai et al., 2005).

#### **4.1.3 Improving business process performance**

A problem with using BPEL business processes as well as web services is the further reduction of performance caused by the introduction of another layer which is based

on XML. BPEL uses SOAP messages to call the services it consumes as well as for communicating with clients. A system that has many of these XML messages as the communication mechanism could suffer in terms of performance as outlined in the previous section. A BPEL process can be accessed by a mobile device in the same way as a web service, it is therefore important to examine the ways in which system performance can be improved. There are four areas that can be addressed to potentially improve the performance of business processes: hardware, middleware, business process logic and the service bindings used.

*Hardware* - a business process can be improved by simply changing the hardware. A server machine or network that is too slow will create a performance bottle neck which could be improved with machine/network upgrades.

*Middleware* - Another way to improve the performance of a business process would be to improve the business process server. There are many business process servers available such as Oracle BPEL Process Manager, ActiveBPEL and JBoss. These servers could be tested to see which is best in terms of performance. If accessing a business process via a mobile device an alternative would be to install a BPEL server on the device itself. In the Sliver project (Hackmann et al., 2006) this was made possible by creating a lightweight BPEL server for mobile devices. In terms of the performance, the BPEL server on the mobile device took, on average, twice as long as the desktop server (Hackmann et al., 2006). Despite this performance decrease, the performance was at an acceptable level (Hackmann et al., 2006). The limitation of this technology, however, is that the BPEL server can only be installed on devices that support the language in which it is implemented.

*Bindings* - Alternative bindings to SOAP for the services can be used to improve performance e.g. REST. REST (Pautasso et al. 2008) uses a HTTP Universal Resource Identifier (URI) to point to a specific resource which also allows `put`, `get`, `post` and `delete` operations from the HTTP protocol to perform create, read, update and delete (CRUD) operations. REST does not require any of the additional XML mark-up associated with the SOAP protocol in the messages or any processing on the server side which means it is an option when performance is an important factor. However, REST does not support the advanced messaging protocols that SOAP does (reliable messaging, message level security etc.) which may be needed in an enterprise integration scenario (Pautasso et al., 2008). Another type of service binding is the web service invocation framework (WSIF) (see section 4.1.4).

*Business process logic* – It is also possible to improve performance by examining the business processes themselves. This can mean either improving the efficiency of the code or the logic within the code (Chen et al. 2008).

#### **4.1.4 Using WSIF to improve business processes performance**

WSIF is a toolkit that provides a simple API to invoke native resources (e.g. Enterprise Java Bean or Java class) using a WSDL based description. It is possible to implement a binding to any language by creating a binding for the BPEL server. The two main requirements are that the service is described by the WSDL document and that the binding is plugged into the framework (Duftler et al, 2001). WSIF is implemented in the binding part of the WSDL document and uses the abstract description to perform NLCs rather than a particular protocol (PBM) such as target

URIs and encoding styles in SOAP (Mukhi, 2001). Figures 13 and 14 show the difference between a WSIF binding and a SOAP binding in the WSDL document.

```
<binding name="JavaBinding" type="tns:AddressBookPT">
  <java:binding/>
  <format:typeMapping encoding="Java" style="Java">
    <format:typeMap typeName="typens:address"
formatType="wsif.types.WSIFAddress"/>
    <format:typeMap typeName="xsd:string"
formatType="java.lang.String"/>
  </format:typeMapping>
  <operation name="addEntry">
    <java:operation
      methodName="addEntry"
      parameterOrder="firstName lastName address"
      methodType="instance"/>
    <input name="AddEntryFirstAndLastNamesRequest"/>
  </operation>
    <input name="GetAddressFromNameRequest"/>
    <output name="GetAddressFromNameResponse"/>
  </operation>
</binding>
```

**Figure 13 WSIF binding declarations (IBM Websphere 5.1.1 documentation)**

Figure 13, taken from the IBM Websphere 5.1.1 documentation, shows the binding declarations in a WSDL document for an address book service. The typeMap elements state how the XML data types map to the Java data types which can either be a class or a native data type. The operation `AddEntry` maps to a method of the same name. The parameter order defines the data that is passed to the method calls. In this case `firstName` and `lastName` which are both strings and `address` which is an object of the `WSIFAddress` class. Figure 14 shows the equivalent binding declarations for SOAP.



```

<message name="AddEntryFirstAndLastNamesRequestMessage">
  <part name="firstName" type="xsd:string"/>
  <part name="lastName" type="xsd:string"/>
  <part name="address" type="typens:address"/>
</message>

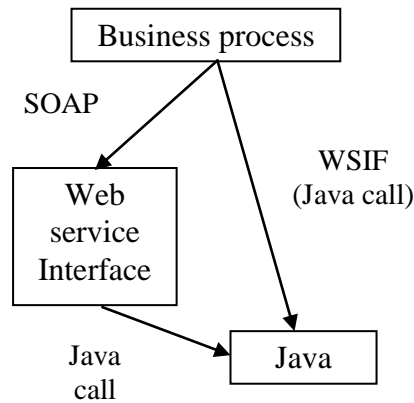
<binding name="SOAPHttpBinding" type="tns:AddressBookPT">
  <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="addEntry">
    <soap:operation soapAction=""/>
    <input name="AddEntryFirstAndLastNamesRequest">
      <soap:body use="encoded"
namespace="http://www.sample.com/namespace/wsif/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  </operation>
</binding>

```

**Figure 14 SOAP binding declarations (IBM Websphere 5.1.1 documentation)**

The first five lines in Figure 14 show the details of the messages used for the SOAP binding (the complex type address is not shown). These types and complex types are used rather than the Java objects and native data types in WSIF.

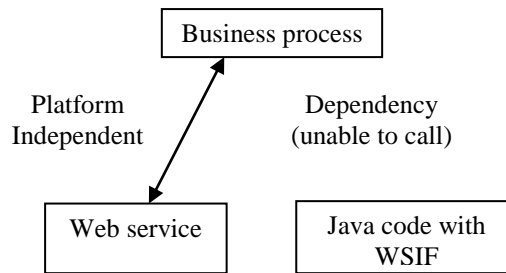
WSIF has been found to be a more efficient binding than using PBM such as SOAP with regard to overall performance (Blanvalet et al. 2006, Mukhi and Slominski, 2001). The reason that WSIF is thought to improve performance over SOAP messaging is that it uses NLC instead of PBM (Mukhi, 2001), allowing the business process to call the underlying code directly, rather than having to interact through web services (Figure 15).



**Figure 15 WSIF and SOAP bindings**

Figure 15 shows how a web service binding has an extra layer of processing whereas a WSIF call is in the native language e.g. Java. This means that no serialisation or de-serialisation is needed for the services used by a business process. Also, the data is not surrounded with XML tags, meaning that the message size will be smaller (Mukhi, 2001). The performance saving was shown in work by Migliardi & Podesta, (2004) where WSIF was used in a grid application scenario and it was found that a Java binding in WSIF was promising for improving performance. They found that although WSIF was leaner than SOAP in terms of message size, the required parsing of the WSDL files at run-time meant that this saving was reduced (Migliardi & Podesta, 2004).

Although the use of WSIF in a system reengineered for MCA is likely to have a positive impact on performance it may negatively impact upon the agility of the system (see Figure 16).



**Figure 16 SOAP and WSIF dependencies**

If the calling business process technology does not support WSIF the classes would not be callable, reducing the agility of the system. However, if a PBM binding is used such as SOAP, integration with business processes on other servers is possible as they communicate by using a shared protocol. The dependent nature of NLC may mean that invocations of the service will not work if the calling technology does not support the particular NLC technology being used (i.e. WSIF). This leads to the research question: CS-RQ1 - *Does using native language calls to improve the performance of a software system, reengineered for multi-channel access, lead to a reduction in agility?* For this study, WSIF is used to implement NLCs and SOAP is used to implement PBM.

## 4.2 Agility

In order to determine if WSIF does have an affect on the agility of the system there needs to be a definition of what is meant by the term agility, so that a suitable measure can be derived. External factors can affect agility, such as consumer demand, are factors that the organisation has little control over – these are the *drivers of change* (Artet & Giachetti, 2004). Internal factors are the things that an organisation does have control over and are likely to be part of the enterprise (including information systems, manufacturing and employees), these are the *facilitators for change* (Artet &

Giachetti, 2004). An internal factor allows an organisation to respond to an external factor.

For this investigation, the external factors will not be considered and the focus will be on the internal factors. In manufacturing, the ‘quick movement (change) of the whole enterprise in a certain direction’ is known as *enterprise agility* (Tsourveloudis et al., 1999). One of the enablers of enterprise agility is the information systems.

#### **4.2.1 Information systems agility**

There are two forms of information systems agility (Krafzig et al., 2004), these are: *the ability to react to changing business requirements* and *the ability to create new business processes*. Changing business requirements could mean either business process *extension* or process *refinement*. An example of business process *extension* is when a service is added that performs pre or post processing on data. This would be simply plugging a new or existing service into the business process. An example of business process refinement would be to change an individual service for economic reasons. There may be a service in the business process that is provided by an external source and an alternative service is found that performs the same functionality but is less expensive or quicker. If the new service provides an interface that is similar to the previous service used, little change in the business process would be required. Even if the interfaces were not similar, the fact that a standards based technology (web services) is being used should mean that the process can be reconfigured to use the new service relatively simply. Another form of business process refinement is to streamline it to improve efficiency e.g. performance or stability. An example of this

maybe a service in the business process that performs some data validation is no longer required and therefore removed.

The other form of information systems agility is the capacity to create new applications/business processes from a set of reusable services. In this case the enterprise is said to have a 'good' level of agility if it can fulfill a set of business requirements by rapidly creating a system from a set of existing services (Krafzig et al., 2004). This means that the organization must have a reasonable number of services readily available in order to rapidly deploy a new system. These services can range from simple utility services such as currency conversion to business processes e.g. creating a new customer process. For the rest of the thesis information systems agility will be referred to as 'agility'.

#### **4.2.2 SOA and agility**

An SOA has properties and technologies that aim to facilitate a high level of agility (Newcomer & Lomow, 2005). The following is a list of design principles for an SOA and how these principles can improve the agility of a system (Erl, 2007):

- *Standardized service contract* – data and functionality are described consistently, making them easier to use.
- *Service loose coupling* – service consumers are able to change the services that they are using with minimal disruption.
- *Service abstraction* – implementation hiding means that service implementation can be changed easily without affecting service consumers.
- *Service reusability* – existing services can be leveraged, reducing the time and effort to change and create new business processes.

- *Service autonomy (modularity)* – greater reliability and predictability created by the autonomous nature of services supports changing business requirements.
- *Service statelessness* – scalability is improved as no data needs to be stored making services more efficient for re-use opportunities.
- *Service discoverability* – services can be located easily for re-use.
- *Service composition* – services that can be used easily within business processes means that integration is simpler.

Another reason that an SOA facilitates a high level of agility is that services and business processes are designed and created in alignment with business requirements. This means that instead of having disparate software systems within an organization, a well built SOA is designed so that it reflects the organisation in terms of the services that are provided and consumed. This makes a system easier to understand and promotes re-use. The primary web service technology that enables the re-use of services in this way is the WSDL document. Using standardized service description documents facilitates business integration as they allow parts of the system to communicate without having to create custom messaging solutions. Because the language is based on standards it makes it interoperable with other systems regardless of the language in which it is written.

*Service coupling* is an important concept that relates to agility. Service coupling refers to the extent that the modules of a system are dependent on another and that a change in the implementation will have little or no impact on the other (Erl, 2007). In an SOA there are two types of service coupling.

The first is between the service description and the technical aspects of the service itself (service logic, implementation, technology used etc.). Ideally, the service description should be independent of the technical aspects of the service. If the service is designed from the service description, in theory it should be possible to completely replace the implementation of the service without disruption to calling services. If the implementation of the service is tied to the description, then this will not be possible and the coupling of the service is said to be tighter. This can be caused by creating the service descriptions from the solution logic.

The other type of service coupling is between the service consumer and the service (Erl, 2007). For example, a consumer does not want their system to be over reliant on the particular service they are using. The main reason for this is that the service may fail. If this happens, the consumer should be able to replace the particular service with another.

### **4.2.3 Agility characteristics**

This section looks at defining the system characteristics that constitute agility in software by comparing two definitions found in the current literature (Krafzig et al., 2004; Allen, 2006). This definition will then form the basis of a quality model, which can be measured using system metrics.

According to Krafzig et al. (2004), to improve agility a system must be:

- *Granular* – the appropriate number of services. It is recommended that the system should be as coarse-grained as possible (a few simple business

functions). However, having fine granularity means that more configurations of the system are possible.

- *Simple (complexity)* – an architecture that is simple to follow for people who will be working on/with the system. This could be internal (integration) or external service users.
- *Flexible and maintainable* – having distinct components that can be rearranged and reconfigured with relative ease and are easy to add/modify.
- *Reusable* – re-use software assets as much as possible by creating an inventory of useful building blocks. Functionality and data should be shared across projects/departments. However, this can be difficult as it involves many departments, shared ownership etc. and requires effective governance (Erl, 2007).
- *Functionally and technologically decoupled (interoperability)* – architecture must tolerate heterogeneity and change to its technical infrastructure. Business functionality must be decoupled from underlying technology.
- *Comprehensibility of SOA* – good documentation facilitated by Service Level Agreements (SLA), which must be simple so that anyone that wishes to use the SOA will be able to understand how it works.

Allen (2006) proposes the following quality factors for a measure of agility:

- *Reusability* – the number of different invocations for components.
- *Replaceability* – the ease of replacing an implementation with another (using the same interfaces).
- *Interoperability* – the ease of software interacting with other software.
- *Flexibility* – the ability of service to perform outside given context.



- *Adaptability* – extensibility to meet new requirements and portability across implementation environments.

Allen's factors are similar to the recommendations by Krafzig et al. Table 13 compares the factors proposed by Allen against the recommendations by Krafzig et al.

<b>Krafzig et al (2004)</b>	<b>Allen (2006)</b>
Granularity	-
Complexity	-
Flexibility	Adaptability
Maintainability	-
Reusability	Reusability, Flexibility
Interoperability	Interoperability, Replaceability
Comprehensibility	-

**Table 13 Agility characteristic comparison**

The main difference is that the notions of service granularity, complexity, maintainability and comprehensibility are not included in Allen's list. An interesting point is that Allen states that the factors should be measured in terms of cost and time.

Krafzig et al.'s characteristics were chosen as they included all of Allen's characteristics. The agility characteristics not included in the quality model, and their reason for exclusion, are:

- *Maintainability* – this was excluded as the focus of this study was on extracting class level code rather than making significant changes.
- *Complexity* – this is regarded as a lower level attribute in the proposed quality model for this study.
- *Comprehensibility of SOA* - There are no Service Level Agreements in the IBHIS broker.

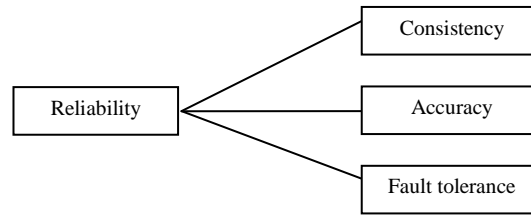
The final list of agility characteristics that will be used for this work is:

- Granularity
- Flexibility
- Reusability
- Interoperability

These characteristics will be used as a basis for an agility quality model. This will be detailed further in section 3.3.5.4.

#### **4.2.4 Agility quality model**

In order to measure the agility of a system an agility quality model was created. There are a number of existing quality models such as McCall's quality model (McCall et al. 1997), Boehm's quality model (Boehm et al. 1978), ISO 9126 (International Organization for Standardization, 1991), Goal Question Metric (GQM) (Basilli et al., 1996) and Dromey's model (Dromey, R. G., 1995). However, these models tend to describe *all* of the desirable properties in software over and above agility, so a subset was required to measure agility only. McCall's model was used as it most matches the characteristics determined for agility. In McCall's quality model, the aspect of quality that is being assessed is known as an *external quality factor*. This is the software quality attribute from an external viewpoint, as seen by the users. There are a number of external quality factors in McCall's model e.g. *reliability* (see Figure 17).



**Figure 17 Reliability in McCall's quality model**

Figure 17 shows the *external quality factor* (reliability) and the *product quality criteria* it is composed of (consistency, accuracy and fault tolerance). The *product quality criteria* are the software quality attributes from an internal viewpoint, as seen by the developer. In a quality model, a product quality criterion can also belong to more than one external quality factor. Table 14 shows the *external quality factors* for agility which are the characteristics identified in section 3.3.5.3. It also shows the corresponding *product quality criteria* from McCall's model for each of the external quality factors.

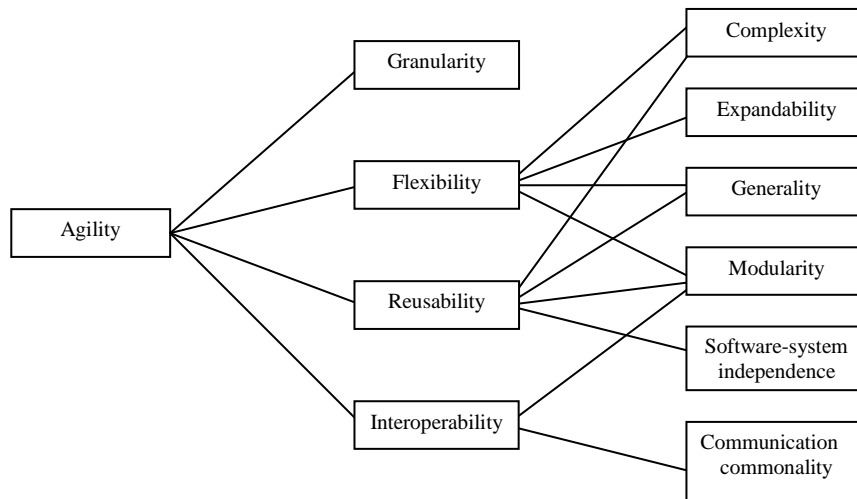
External quality factor	Product quality criteria
Granularity	-
Flexibility	<ul style="list-style-type: none"> <li>• Complexity</li> <li>• Expandability</li> <li>• Generality</li> <li>• Modularity</li> </ul>
Reusability	<ul style="list-style-type: none"> <li>• Complexity</li> <li>• Generality</li> <li>• Modularity</li> <li>• Software-system independence</li> </ul>
Interoperability	<ul style="list-style-type: none"> <li>• Modularity</li> <li>• Communication commonality</li> </ul>

**Table 14 Agility quality factors**

Listed below are the definitions for *product quality criteria* from Table 14 (McCall et al. 1997).

- *Granularity* – the level of abstraction of the service interfaces that represent the system.
- *Complexity* - the complexity of the system.
- *Expandability* - the degree to which architectural, data or procedural design can be extended.
- *Generality* - the breadth of potential applications of program components.
- *Modularity* - the functional independence of program components.
- *Software system independence* - the degree to which the program is independent of non-standard programming language features, operating system characteristics and other environmental constraints.
- *Communication commonality* - the degree to which standard interfaces, protocols and bandwidth are used.

Two *product quality criteria* were not used in the agility quality model for this thesis. There were *data commonality (from interoperability)* and *machine independence (from reusability)*. These criteria were not included as the integration of data sources from potentially different sources was one of the main goals of the IBHIS project. Figure 18 illustrates the proposed quality model for agility.



**Figure 18 Agility quality model**

Figure 18 shows the relationships between the *external quality factors* and the *product quality criteria* for the agility quality model.

#### 4.2.5 Agility measures

Once a quality model has been chosen or created it is important to define how to measure the *product quality criteria*. In some cases, objective metrics cannot be used and a subjective measurement is required. When a subjective rating is required, the rating was made by a human using a semantic differential scale (Coolican, 1999). In the semantic differential scale the assessor marked the aspect on a scale of 1-10 (1 being low, 10 being high). The following describes the methods used to measure each of the agility product quality criteria.

*Granularity* – this is a subjective assessment of the suitability of the number of service interfaces in relationship to the number of services offered by the system.

*Complexity* – This is a measurement of the complexity of the system code. The complexity metrics identified are based on Lorenz object-oriented metrics and Chidamber & Kemerer metrics, (Kan, 2003), and are as follows:

- The number of classes (not including interfaces)
- Average method size
- Average number of methods per class
- Average number of instance variables per class
- Number of children (immediate subclasses)

The *number of children* metric can be viewed as both positive and negative depending on the system context. For example, if reuse is important, then a larger number of immediate subclasses could be a positive factor. However, a larger number of children can increase the complexity as developers would have to understand the implication of changing higher level classes on their subclasses.

*Expandability* – this measure this is a subjective assessment that examines how easily the architecture can be expanded. For example, if there is a poor separation of concerns the rating would be lower.

*Generality* – this subjective measure examines the extent to which the services can be used outside of their normal business applications and how specific are they to a given context.

*Modularity* - modularity is a well known property of software that can have both desirable and undesirable forms (Budgen, 2003). The two most widely used measures

of modularity are *coupling* and *cohesion*. For this study, *coupling between objects* and *lack of cohesion on methods* will be used. *Coupling between object classes (CBO)* is the number of times a class invokes functions/instance variables of another class plus the number of classes that referenced the class. If a class appears in both the referenced and the referred classes it is only counted once. If an object invokes a function of another class it is known as control coupling (Budgen, 2003). If one class calls a procedure on another class this is said to be 'necessary'. However, if the actions of the class being called are determined by a parameter from the calling class this is said to be 'undesirable'. If the classes communicate with parameters only and these do not have any control element this is known as data coupling and is said to be desirable (Budgen, 2003). *Cohesion* examines the similarity of the components of a class with the view of fulfilling a single purpose (Budgen, 2003). A lack of cohesion on methods (LCOM) means that a class that is not cohesive and would probably benefit from subdivision. For this metric the aim is to see how closely the local methods relate to local instance variables in the class. If a high number of methods use an instance variable, it is thought to be cohesive. The range of LCOM defined in this way can range between 0-2, anything above one is considered to have poor design.

*Software system independence* – this subjective measure examines the level of the use of anything that is non-standard in the system e.g. programming language, operating system. An example restriction caused by using non-standard languages and operating systems may be that the system could not be moved into another environment.

*Communication commonality* – this subjective measurement examines the extent to which standards and common protocols are used in the system e.g. (SOAP, HTTP).

#### **4.2.6 Agility test cases**

Agility test cases are agility-focused use case tests that examine how well the system can respond to future change by measuring the time and effort required to make the change. When reengineering a system a lot of time can be spent understanding the code of the existing system, in some cases this can be around fifty percent (Mayhauser & Vans, 1995). However, for this study it is assumed that the reverse engineering has already taken place and the subsequent activities are the recoding aspect of the forward engineering effort.



## Chapter 5: Methodology and implementation

This chapter describes the planning and conduct of a case study (Jefferies et al., 2009) to explore the following research questions proposed in section 1.4:

- CS-RQ1 - Does using native language calls to improve the performance of a software system reengineered for multi-channel access, lead to a reduction in agility?
- CS-RQ2 - Does the inclusion of extra layers required for MCA reduce performance?
- CS-RQ3 - Does reengineering a system as an SOA improve agility?
- CS-RQ4 - Is McARM an effective method for reengineering a system for MCA?

A case study can be used to investigate problems that have several sets of data, in order to understand insights and ideas (Runeson & Höst, 2009). This ability to understand insights and ideas is the main strength of the case study and the reason it is used in this instance. However, a weakness of a case study method is that it does not allow generalisation beyond similar cases. As only one system (IBHIS broker) was used, this is classed a *single case study*. The reason for only having one case is due to the lack of available service-based systems of a reasonable size.

## 5.1 Planning

This planning of this case study was based on a set of guidelines (Brereton et al. 2008, Yin, 2003). This section outlines the case study design and preparation for data collection phases.

### 5.1.1 Aims

Listed below are the research aims for the case study:

**CS-RQ1** - *Does using native language calls to improve the performance of a software system reengineered for multi-channel access, lead to a reduction in agility?* This question addresses the potential performance/agility trade-off of a system reengineered for MCA, using native language based method calls (NLC) or protocol based messaging (PBM) binding technologies. This can be broken down into two propositions:

- CS-RQ1-P1 - NLC used in a system reengineered for MCA will result in *improved performance* when compared to PBM for a mobile service-based system.
- CS-RQ1-P2 - A system reengineered for MCA using NLC will be *less agile* than one using PBM.

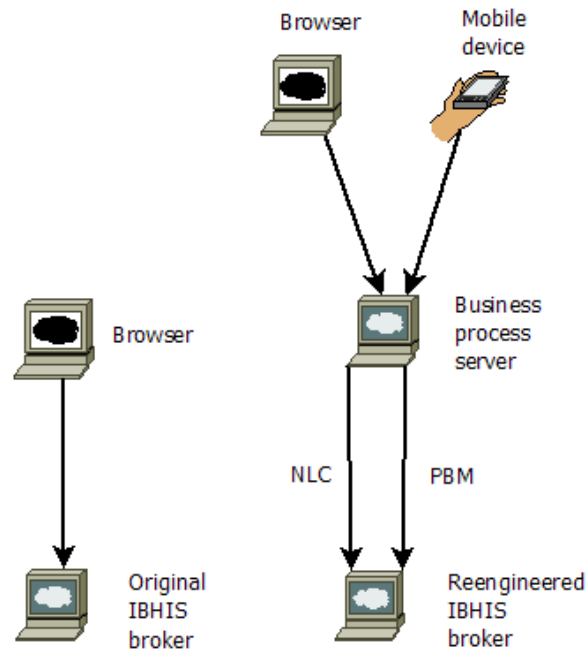
**CS-RQ2** - *Does the inclusion of extra layers required for MCA reduce performance?* The aim of this research question is to investigate a further issue that that may lead to a reduction in performance when reengineering a system for MCA

– that of the additional layers required when using service-based technologies as the basis for reengineering. The proposition for this research question is: the extra layers required for MCA will reduce performance.

**CS-RQ3** - *Does reengineering a system as an SOA improve agility?* This question investigates whether or not a system reengineered as an SOA has improved agility. The proposition for this research question is that reengineering a system as an SOA *will* increase its agility.

**CS-RQ4** - *Is McARM an effective method for reengineering a system for MCA?* This research question looks at the McARM reengineering method that was created and assesses it in terms of effectiveness (the quality of the output) and efficiency.

In order to investigate the research questions, the original IBHIS broker was reengineered toward the proposed MCA architecture (McArc) using McARM. There are thus two versions of the IBHIS broker used in the case study; the *original IBHIS broker* and the *reengineered IBHIS broker* (see Figure 19).



**Figure 19 Brokers used in case study**

The original IBHIS broker is called from standard web browser technology. The reengineered IBHIS broker will be accessed by business processes in the business process server. The reason for having a business process layer is explained in section 3.2.4. The business processes will bind to the services in the reengineered IBHIS broker using either NLC or PBM. The business processes can be called from a number of different channels e.g. browser, mobile device. Both of the bindings (NLC and PBM) will be evaluated for the reengineered IBHIS broker for this study, as well as the original IBHIS broker.

### 5.1.2 Case study design

The objects of the study are the binding technologies (NLC and PBM) and the McARM reengineering method. As there are multiple units of analysis this is classed

as an *embedded* case study (Runeson & Höst, 2009). For this investigation the units of analysis measured were:

- Performance of the systems
- Agility of the systems
- Reengineering method ease of use

The data collected for the units of analysis are:

- Response times
- Code metrics
- Semantic differential scale evaluation of the system
- Agility test case data (time and effort to make changes)
- Reengineering method data

Table 15 shows each research question, units of analysis and data collected.

Research question	Unit of analysis	Data
CS-RQ1	P1 – Performance	Response times
	P2 – Agility	Code metrics
		Semantic differential scale evaluation of the system
		Agility test case data
CS-RQ2	Performance	Response times
CS-RQ3	Agility	Code metrics
		Semantic differential scale evaluation of the system
CS-RQ4	Reengineering method ease of use	Efficiency and effectiveness data

**Table 15 Research questions and measures**

For CS-RQ1-P1, an experiment was conducted to compare the two service bindings. The services in the reengineered IBHIS broker were bound to business processes using the two different binding technologies (NLC and PBM). The between-subjects variable was the binding technology and the within-subjects variable was the message size.

For CS-RQ1-P2 system metrics and test cases were used to compare the two bindings (NLC and PBM) in terms of agility. The agility measure uses three data collection methods - metrics, semantic differential rating and agility test cases. This provides methodological triangulation (Runeson & Höst, 2009). In order to address CS-RQ2, the response times of the reengineered IBHIS broker (PBM bindings) were compared at three points in the architecture; the *service*, the *business processes* and the *mobile client*. The service represents the basic system functionality, and the business process and mobile client are the additional layers required for MCA.

For CS-RQ3, a direct comparison was made between the agility metrics for the original IBHIS broker and the reengineered IBHIS broker (PBM binding).

For CS-RQ4, self-reporting measures were used in the form of semantic differential and qualitative measures data to evaluate the effectiveness and efficiency of the McARM reengineering method.

### **5.1.3 Case selection**

The case that was used in this study was the exercise of reengineering the original IBHIS system towards McArc using McARM. The reason for using this as the case

was that there are few large scale service-based systems available. Getting access to a medium/large system that could be reengineered for MCA would have been difficult.

#### **5.1.4 Case study roles**

This section defines roles of the people conducting and participating in the case study.

The roles of the study were:

- Case study participant and observer: Clive Jefferies
- Evaluation: Clive Jefferies and Mark Turner
- Code assessment: Mark Turner

There was also a divergence table, which was updated during the study if there were any deviations from the protocol, which was created and followed for this case study (Jefferies et al., 2009).

#### **5.1.5 Data Collection**

The following section outlines the data to be collected, how the data was collected, a data collection plan and how the data was stored.

##### **5.1.5.1 Data to be collected**

The following section describes the data collected for each research question. Where possible, *quality attribute scenarios* (QAS) were developed to summarise each non-functional requirement for the research question (Bianco et al., 2007).

#### 5.1.5.1.1 CS-RQ1-P1

The performance measure was the response time for a service which is the time that it takes to send and then receive a message (Machado & Ferraz, 2005). There are other performance measures that could have been used, for example *throughput* and *scalability* (Pfleeger, 2001), but these were not used due to time constraints. In order to reduce bias due to the size of the query, a range of message sizes was required for the response time measure. Three sizes were used from the original IBHIS system (See Table 16)

Message size	Fields	Request	Response	Total bytes
Small	1	1155	597	1752
Medium	7	1429	884	2313
Large	15	1763	1212	2975

**Table 16 Message sizes**

Table 17 illustrates the QAS for CS-RQ1-P1

Stimulus	Perform patient query (small, medium, large)
Artefact	IBHIS system
Environment	Normal operation
Response	Patient query data is returned to the user
Response measure	Response time (milliseconds).

**Table 17 CS-RQ1-P1 QAS**

#### 5.1.5.1.2 CS-RQ1-P2

There were two data collection methods, *system metrics* and *agility test cases*. The system metrics for agility are those proposed in chapter 3. There were two types of system measurements gathered, the *code metrics* and *semantic differential measurements*. The code metrics were derived from direct measurements of the



software, whereas the semantic differential measurements were a human assessment of system properties. Table 18 presents the QAS for the CS-RQ1-P2.

Stimulus	Evaluate system agility
Artefact	System code
Environment	Normal operation
Response	None
Response measure	Code metrics, Semantic differential measurements

**Table 18 CS-RQ1-P2 QAS**

There were four *agility test cases* used as a basis for measuring agility in terms of the ease of change. These test cases represent potential changes that can occur in the system. Each test case measured the time and effort required to implement a change that can occur within the system.

*Agility test case 1: Create a new service for a business process.*

The aim of this test case was to show the time and effort required to create a new service and to add it to a business process in the reengineered IBHIS system. The agility is evaluated based on the time and effort to make the change. Table 19 shows the QAS for agility test case 1.

Stimulus	Creating a new service for a business process
Artefact	Web service, WSIF service, business process
Environment	Normal operation
Response	Business process is using new service
Response measure	Time (mins) and effort (NOS, number of artefacts)

**Table 19 Agility test case 1 QAS**

*Agility test case 2: Amending an existing service in a business process.*

This test case involved amending a service that is being used by a business process in the reengineered IBHIS broker. A service was changed to incorporate a new field and

the business process in the reengineered IBHIS broker was updated to enable it to work with the updated service. The agility was then evaluated based on the time and effort required to implement the change. Table 20 shows the QAS for agility test case 2.

Stimulus	Amending a service with a new field
Artefact	Business process, web service, WSIF service
Environment	Normal operation
Response	Business process is using amended services
Response measure	Time (mins) and effort (NOS and number of artefacts)

**Table 20 Agility test case 2 QAS**

*Agility test case 3: Replacing a service in a business process*

The next test case was reconfiguring a business process in the reengineered IBHIS broker to use an alternative service that performs the same functionality. This is to discover the level of difficulty involved in replacing a service in the reengineered IBHIS broker, in both the NLC and PBM binding versions. Table 21 shows the QAS for agility test case 3.

Stimulus	A service is replaced in a business process
Artefact	Business process, web service, WSIF service
Environment	Normal operation
Response	Business process is working with replacement service
Response measure	Time (mins) and effort (NOS and number of artefacts)

**Table 21 Agility test case 3 QAS**

*Agility test case 4: Access a WSIF service from another server*

The aim of this test case was to determine if a service from the reengineered IBHIS broker can be accessed by a business process using a different business process server. Table 22 shows the QAS for agility test case 4.

Stimulus	A business process that is run on a server wants to use a WSIF service
Artefact	Business process, WSIF service
Environment	Normal operation
Response	Business process can access WSIF service
Response measure	Time (mins) and effort (NOS and number of artefacts)

**Table 22 Agility test case 4 QAS****5.1.5.1.3 CS-RQ2**

To investigate the additional response time caused by the additional layers required for MCA, the reengineered IBHIS broker (PBM binding) was measured at three points. For CS-RQ2, only the PBM bindings were used as both bindings had the same number of layers. Table 23 presents the QAS for the CS-RQ2.

Stimulus	Perform patient query on the IBHIS system.
Artefact	Reengineered IBHIS broker (service, business process and mobile client)
Environment	Normal operation
Response	Patient query data is returned to the user
Response measure	Response time (milliseconds)

**Table 23 CS-RQ2 QAS****5.1.5.1.4 CS-RQ3**

The system measurements (code metrics and semantic differential measurements) of the original IBHIS broker were compared with code metrics of the reengineered IBHIS broker (PBM binding). This was done to determine the effect that reengineering a system as an SOA may have on agility. Table 24 presents the QAS for the CS-RQ3.

Stimulus	System is reengineered as an SOA
Artefact	Original IBHIS broker/ reengineered IBHIS broker
Environment	Normal operation
Response	-
Response measure	Code metrics, Semantic differential measurements

**Table 24 CS-RQ3 QAS**

Agility test cases were not used for this part of the investigation as the agility test cases are contextual. For this research question, the difference between the architectures was the focus, therefore the time and effort to make future changes was not examined.

#### 5.1.5.1.5 CS-RQ4

Table 25 defines the measurements for the reengineering process and how they were measured for CS-RQ4:

Area	Definition	Measurement
Efficiency	Time to perform each stage	<ul style="list-style-type: none"> <li>Hours</li> </ul>
Effectiveness	Was it easy to follow? Were there any problems?	<ul style="list-style-type: none"> <li>Ease (Semantic differential)</li> <li>Problems (qualitative)</li> </ul>

**Table 25 CS-RQ4 QAS**

#### 5.1.5.2 How the data was collected

Five sets of data were collected during the case study:

- Response times
- Code metrics
- Semantic differential scale evaluation of the system
- Agility test case data
- Reengineering method data

#### **5.1.5.2.1 Response times**

For CS-RQ1-P1, the response times were taken using the BPEL console, which is part of the BPEL server. Two business processes were created for the reengineered IBHIS broker (one for each binding technology), and both were called twenty times for each message size and the time for the response was recorded.

For CS-RQ2, the response times for the service were recorded twenty times for each message size, using the HTTP analyser in Oracle JDeveloper 10.1.3.3.0. The business process response time was recorded twenty times for each message size, and the time for the response was recorded from the Oracle BPEL Process Manager console. The mobile client response was recorded twenty times for each message size and the time for the response was taken by the network monitor in the Sun Wireless Toolkit 2.5.2.

#### **5.1.5.2.2 System metrics**

In order to address CS-RQ1-P2 and CS-RQ3, the original IBHIS broker and reengineered IBHIS broker metrics were measured using JHawk metrics software.

#### **5.1.5.2.3 Semantic differential measurements**

The semantic differential measurements for CS-RQ1-P2 and CS-RQ3 were recorded on an *agility measurement form*. On the agility measurement form, there was also a comments section to justify why each score was given. The semantic differential measurements were validated by a member of the supervisory team. This person was responsible for coding the original system. It was necessary to use a person with detailed knowledge of the IBHIS systems in order to provide useful measurement.

#### **5.1.5.2.4 Agility test cases data**

The time and effort to make the changes was entered into an *agility test case data form* for CS-RQ1-P2.

#### **5.1.5.2.5 Reengineering method data**

A log was kept to collect the data needed for CS-RQ4. For each stage, the time taken was noted as well as any problems found.

#### **5.1.5.3 Data collection plan**

The procedure for the case study was:

1. The original IBHIS broker was evaluated in one way:
  - The agility of the original IBHIS broker was measured in order to compare it with the reengineered IBHIS broker (PBM) to investigate if reengineering a system as an SOA improves agility for CS-RQ3.
2. The original IBHIS system was reengineered to reflect the architecture described in chapter three using McARM to address CS-RQ4.
3. The reengineered IBHIS system was tested for correctness in order to ensure that the system had been implemented to a standard equal to that of the original IBHIS broker. This was assessed by a member of the original technical team to ensure the system fulfils the original functional requirements and was correctly documented.
4. The reengineered IBHIS broker was assessed to decide if it is functionally the equivalent to the original IBHIS broker. The implementation of the two bindings

was also assessed to ensure that they were implemented to an equivalent standard for fairness. This was conducted by a member of the supervisory team.

5. Evaluation of calling the reengineered IBHIS broker services from business processes using both binding technologies (NLC and PBM) was conducted in the following ways:

- The performance was measured for CS-RQ1-P1 and CS-RQ2 (PBM only).
- The agility was measured for CSRQ1-P2 and CS-RQ3.

The machine being used for the case study was: Mesh, Microsoft Windows XP professional 2002 SP3, AMD Athlon 64 processor, 3200+, 2.01GHz 2.00GB RAM

The application server used for the existing IBHIS system and reengineered IBHIS system was IBM Websphere 5.0.1. The business process server used was Oracle BPEL process manager version. 10.1.3.1. The mobile client was created and tested using the Sun Java Wireless Toolkit: J2ME WTK 2.5.2. The system metrics were gathered using JHawk.

### **5.1.5.4 How data was stored**

Each data type was stored as follows:

- Response times – Spreadsheet document
- Code metrics – Spreadsheet document
- Semantic differential measurements – Word document
- Agility test case data - Word document
- Reengineering process evaluation - Word document

These were stored on the Keele University network drive.

### 5.1.6 Analysis

Each research question will be examined individually.

#### 5.1.6.1 CS-RQ1

There are three sources of data for CS-RQ1, these are:

- Performance measurements
- Agility metrics
- Agility test cases

The *performance measurements* for each of the bindings, for each of the message sizes, were analysed using inferential statistics with an Analysis of Variance (ANOVA). The ANOVA was used rather than multiple t-tests as these can result in a Type 1 error (capitalising on chance) (Coolican, 1999). The *agility metrics* of the two bindings were compared and the binding with the higher values was considered the more agile. The *agility test cases* were analysed in the same way as the metrics; the binding that had the higher values was considered more agile. The possible outcomes from the results were:

- 1 The NLC binding is found to be *faster* than the PBM binding, the agility metrics indicate that the NLC is *less* agile than PBM and the test cases indicate the NLC is *less* agile than PBM.



- 2 The NLC binding is found to be *faster* than the PBM binding, the agility metrics indicate that the NLC is *less* agile than PBM and the test cases indicate the NLC is *more* agile than PBM.
- 3 The NLC binding is found to be *faster* than the PBM binding, the agility metrics indicate that the NLC is *more* agile than PBM and the test cases indicate the NLC is *less* agile than PBM.
- 4 The NLC binding is found to be *faster* than the PBM binding, the agility metrics indicate that the NLC is *more* agile than PBM and the test cases indicate the NLC is *more* agile than PBM.
- 5 The NLC binding is found to be *slower* than the PBM binding, the agility metrics indicate that the NLC is *less* agile than PBM and the test cases indicate the NLC is *less* agile than PBM.
- 6 The NLC binding is found to be *slower* than the PBM binding, the agility metrics indicate that the NLC is *less* agile than PBM and the test cases indicate the NLC is *more* agile than PBM.
- 7 The NLC binding is found to be *slower* than the PBM binding, the agility metrics indicate that the NLC is *more* agile than PBM and the test cases indicate the NLC is *less* agile than PBM.
- 8 The NLC binding is found to be *slower* than the PBM binding, the agility metrics indicate that the NLC is *more* agile than PBM and the test cases indicate the NLC is *more* agile than PBM.

Table 26 summarises these outcomes.

Number	Performance	Metrics	Test Cases
1	NLC	PBM	PBM
2	NLC	PBM	NLC
3	NLC	NLC	PBM
4	NLC	NLC	NLC
5	PBM	PBM	PBM
6	PBM	PBM	NLC
7	PBM	NLC	PBM
8	PBM	NLC	NLC

**Table 26 Summary of outcomes**

Outcome 1 would suggest that NLC should be used to improve the performance of a business process if agility is not an essential feature of the system. Outcome 2 suggests that NLC should not be used if code agility is important. Outcome 3 suggests NLC should not be used if the time and effort involved in changing the system are important. Outcome 4 suggests that NLC should be used in a business process as it improves both performance and agility. For outcome 5, NLC should not be used in a business process as neither performance nor agility is improved. Outcome 6 suggests that NLC should be used if only the time and effort involved in changing the system are important. Outcome 7 suggests that NLC should only be used if code agility is important. For outcome 8, NLC should be used to improve agility but not performance.

#### **5.1.6.2 CS-RQ2**

The *performance measurements* for each of the layers, for each of the message sizes, were analysed using an Analysis of Variance (ANOVA).

### **5.1.6.3 CS-RQ3**

The system metrics of the original IBHIS system and the reengineered IBHIS system were compared in the same way as the agility metrics in CS-RQ1, to determine if reengineering as an SOA improves agility.

### **5.1.6.4 CS-RQ4**

McARM was analysed based on the feedback and experience of using the method. This was used to determine if this method is effective and efficient and could be used for reengineering a system for MCA.

## **5.2 Conducting**

This section details the divergences for CS-RQ1, CS-RQ2 and CS-RQ3. The outputs for CS-RQ4 and the architecture are also presented.

### **5.2.1 CS-RQ1**

This section highlights the divergences for CS-RQ1.

#### **5.2.1.1 Performance testing mock-up**

The performance testing was conducted using a recreated version of the reengineered IBHIS broker. This was necessary as the original IBHIS broker was running within the application server included in IBM Websphere Application Developer 5.1.1, which was found to have limited support of WSIF bindings. The enterprise application client (EAC) technology for calling WSIF was unable to access drivers for

a database used by one of the services. The EAC is able to access database drivers for its own classes but not those from another project. An attempt was made to move the system code into the EAC project but certain libraries could not be used with this type of project. Even if this had been achieved, the EAC can only be accessed by systems that support J2EE. Therefore, in order to undertake performance tests, a mock-up of the reengineered IBHIS broker (reengineered IBHIS performance mock-up) was made in an environment that would enable a system to call a WSIF service from a business process. Using a mock-up for the performance testing may have had an effect on the results. The reason for this is that the difference in performance between SOAP and WSIF on Oracle BPEL Process Manager may not be the same as in IBM Websphere Application Developer 5.1.1.

The mock up was a simulation created on Oracle BPEL Process Manager using the same message sizes as in the reengineered IBHIS broker. A member of the supervising team was asked to compare the code from the two binding implementations to assess if they were coded equivalently. This additional verification was needed because creating WSIF services is different between Oracle BPEL Process Manager and IBM Websphere. The code was functionally the same, but the way in which the classes were created in Oracle BPEL Process Manager was slightly different. The difference was that when using WSIF in Oracle BPEL Process Manager, system classes are not instantiated directly, instead they are created by a factory class. Creating the classes from a factory would mean the response time would be slightly slower due to the extra step needed. In IBM Websphere Application Developer 5.1.1 the classes are called directly rather than via a factory class.

#### **5.2.1.2 Agility test case mock-up**

The agility test cases were performed on a different mock-up of the reengineered IBHIS broker (reengineered IBHIS agility mock-up) using Oracle BPEL Process Manager. The reason for using a mock-up for the test cases was the inability to use the services in an external business process (see section 4.2.1.1). As the WSIF services in IBM Websphere Application Developer 5.1.1 could not be called externally, this had to be tested in Oracle BPEL process manager. If it had have been possible to use a more recent version of IBM Websphere Application Developer, which had a business process server e.g. IBM Integration Developer, this may not have been a problem. However, this was not possible due to cost and the effort that would be required to port the original IBHIS broker to a different version.

#### **5.2.1.3 Unused agility test cases**

*Agility test case 3: Replacing a service in a business process* was not carried out as it was realised that if a service was to be replaced by another service (WSIF or SOAP) that performed the same functionality, the new service would be called in the same way as the old service regardless of the binding. For example, if the `queryIBHIS` operation is called, the business process only *interacts* with this call and is not concerned with the binding that is used to perform the call.

*Agility test case 4: Access a service from another server* was not carried out either. An attempt was made to access the services from a business process server (Oracle BPEL Process Manager). However, it was found that a WSIF service could not be accessed from a different virtual machine. This is, however, possible using SOAP.

### **5.2.2 CS-RQ2**

For CS-RQ2 the only divergence was that the response time measurements were carried out using the reengineered IBHIS performance mock-up for the reasons highlighted in section 4.2.1.1.

### **5.2.3 CS-RQ3**

There were no divergences however, it should be noted that for the original system the web pages (.jsp) that contained the business logic of the system were also included in the metrics as these are considered to be classes. If the page contained presentation code only, it was not counted.

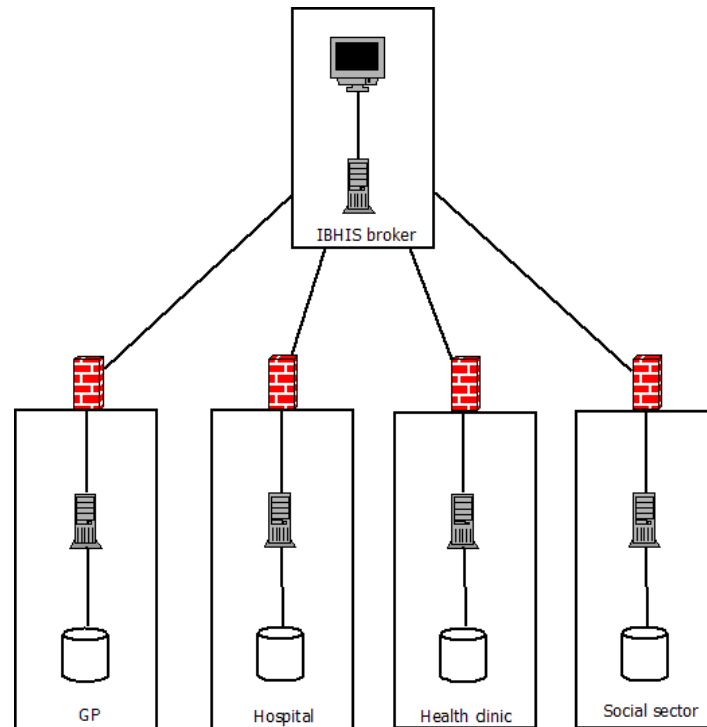
### **5.2.4 CS-RQ4**

For this research question, the use of the McARM reengineering method for reengineering the original IBHIS broker was followed. The results of using the method are outlined in this section. There were no divergences.

#### **5.2.4.1 System understanding**

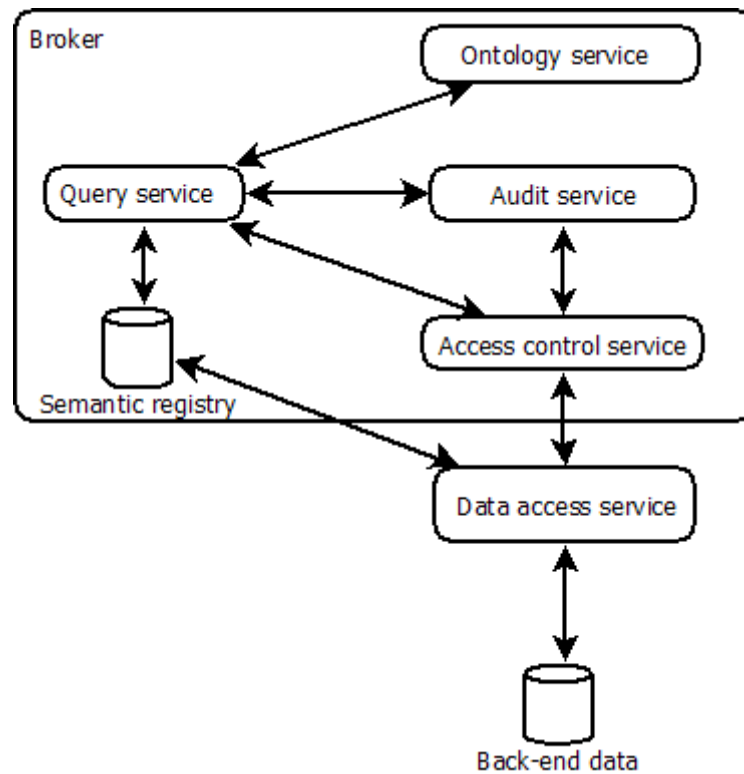
There are two sources that can be used to understand the existing system; the documentation and the code.

*Documentation* - The following figures outline the current architecture of the system based upon the available documentation.



**Figure 20 Overview of original IBHIS system**

Figure 20 (Kotsiopoulos et al., 2003) gives a high level overview of the IBHIS broker, showing the broker and the heterogeneous data sources that it queries. The services and databases that the original IBHIS broker uses are shown in Figure 21 which is adapted from (Budgen et al. 2007).



**Figure 21 IBHIS broker services and data**

In figure 21, it should be noted that although this diagram labels certain aspects of the system as services, these are not all exposed as web services. Rather, they are internal components that the system uses e.g. the audit service. Once the overview of the system was gained from reading the documentation, the next task in McARM was to outline the main components of the system. The main components of the original IBHIS broker are (Kotsiopoulos et al., 2003):

- *Access Control Service (ACS)* - for user authentication and authorisation.
- *Ontology Service (OS)* - defines the terms used in the original IBHIS system for mapping queries to local data sources. Consulted during the query decomposition.



- *Data Access Service (DAS)* - the operational core, constructed using web services. For each enquiry the QS decomposes the enquiry into a set of sub-queries and also binds with the data services.
- *Query Service (QS)* - comprises two sub-modules: the *query decomposer* and the *query integrator*. The *query decomposer* decomposes the query into a set of local queries in consultation with the matchmaker and the semantic registry, which holds the semantic descriptions of the export schemas. The *query integrator* receives and integrates the individual results from the DASs.

After determining the components, it was necessary to gather details about these components using a component table (see Table 27). The information gathered was a subset of those outlined by Lewis et al. (2005):

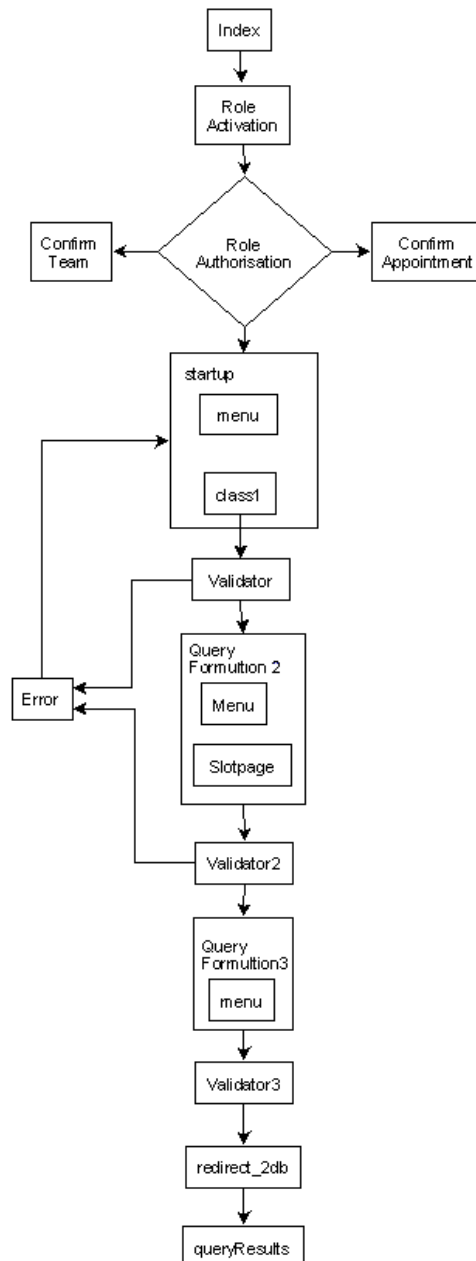
- Component name
- Component size (LOC)
- Level of documentation
- Number of base classes
- Programming standards compliance
- Black box v white box suitability
- Scale of change required

<b>Component Name</b>	<b>Size</b>	<b>Doc</b>	<b>Base classes</b>	<b>Standards</b>	<b>Bb v wb</b>	<b>Scale of change</b>
ACS	3678	None	6	None	Bb	small
OS	318	None	2	None	Bb	small
QS	1287	None	12	None	Wb	small
DAS	1423	None	4	WS	Wb	small

**Table 27 Component table for original IBHIS system**

The information for the component table should have been based on the system documentation. However, no documentation was available for the original IBHIS broker and, therefore, it was necessary to determine the required information from the system code.

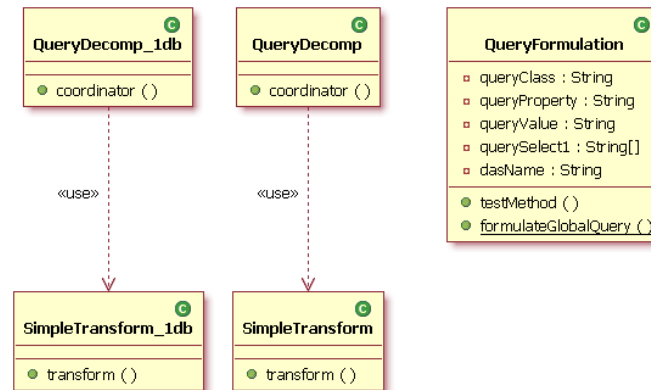
*Existing code* - The other source used for system understanding is the existing code. In order to document the flow of the system, a UML sequence diagram was needed from the graphical user interfaces (GUI). Before the sequence diagram was created, a flowchart was created, which showed a higher level view of the system, making it easier to create the sequence diagram (see Figure 22).



**Figure 22 Original IBHIS system GUI flowchart**

In Figure 22, the earlier stages are for the access control, followed by choosing a category from the ontology and finally performing the query itself. The flowchart also shows validation during the query composition that was used for demonstration purposes. The sequence diagram details the system further, not only looking at the flow of the web pages but also the components being called. The sequence diagram is

not included in the thesis due to its size. Another output for system understanding is a reconstruction of the system classes from the code.



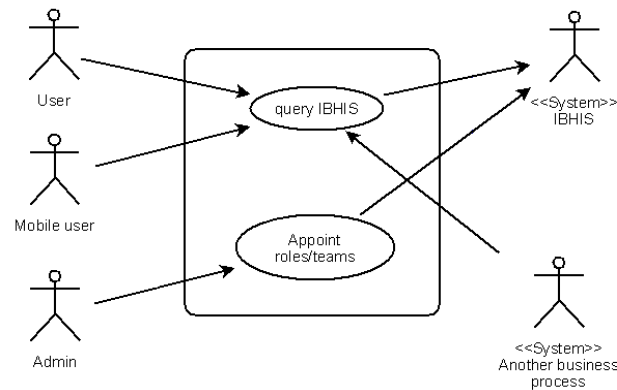
**Figure 23 P2Client (query)**

Figure 21 shows an example of a subset of the classes used for formulating a query in the original IBHIS broker. These class diagrams were created using the IDE (IBM Websphere) in which the original IBHIS broker had been created. In total there were 106 classes and managing these classes was found to be difficult, especially if the class was large or there were many interdependencies.

#### 5.2.4.2 Service identification

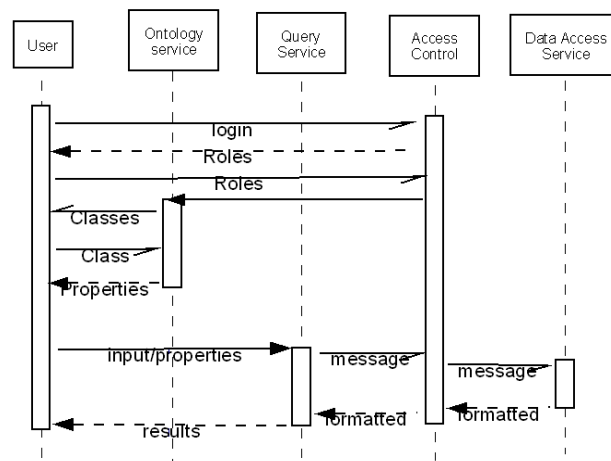
The stakeholder that was available for information was a member of the supervisory team who coded part of the original IBHIS broker who was consulted when trying to identify the services in the original IBHIS broker. The other source for identifying the current services in the system is the current documentation. Class diagrams can be useful for identifying potential services, so those created in the system understanding stage were examined. The system code was found to be another way of identifying services. For example the appoint roles/teams function in Figure 24, which had not

been documented, was only found as a result of a detailed analysis of the code. The starting point was to identify the current and new use cases for the system (Figure 24).



**Figure 24 IBHIS current and future use cases**

Figure 22 shows that the use cases remain the same but the actors that perform these cases are different. The ‘mobile user’ and ‘another business process’ are the new actors that will perform the ‘query IBHIS’ use case. The next task in McARM is to show how the potential services in the system interact with each other (Figure 25).



**Figure 25 Sequence diagram for potential services**

The sequence diagram in Figure 25 shows how the core services in the original IBHIS broker interact with each other and the user. The user logs into the system and is

shown the roles and teams available. They then choose the roles and teams they wish to use. They are then shown the ontology and they must choose the elements of the ontology they would like to query. The properties for an element are then returned to the user who must then choose which elements they wish to query. The query is then made via the access control service, which sends the query to the data access service, which performs the local queries. This is then returned to the user via the access control service, which removes any data that the user is not entitled to see.

The next task was to summarise the potential services in order to evaluate them (see Table 28).

Service identified	Information regarding service
Access control service	Responsible for user authentication and authorisation. This also takes the roles that a person has and only lets them see what they are allowed to see.
Ontology service	Holds relevant ontologies for queries.
Query service	For performing the query.
Data access service	Acts as an entry point for the data sources.
Appointment service	Allows a user to appoint role and team access rights to another user.

**Table 28 Service table**

It should be noted that the services are the same as the components identified in the system understanding. The reason for this is that the original IBHIS broker was based on service-oriented principles. However, these were not always implemented as services within the final system.

In order to evaluate the potential services for the new system, a service evaluation table was compiled. This information identified that the *ontology service* would not be required as the users were querying patient data only and therefore the ontology information could be hardcoded as the user would not need to select the ontology

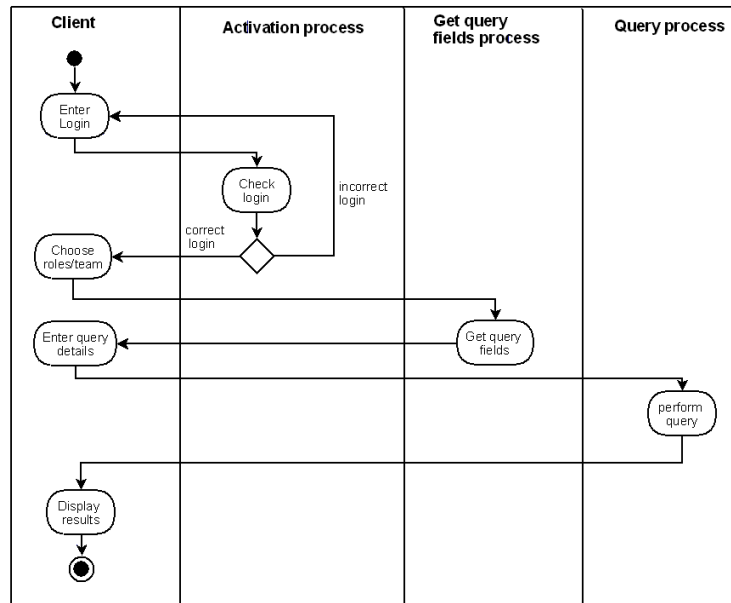
information. Although the ontology could be re-used in another health care system, the focus of the study was multi-channel access to the patient data. The *data access service* is functionality that is running in the background (utility) and is not something with which the user is aware of and there is no direct interaction with the user. The *appointment service* was discounted as the system was for patient data queries only and would not focus on administrative aspects. It could be argued that these services should be used for alternative or new service configurations but a coarse grained approach was preferred to limit the number of interactions with the system.

For the IBHIS system to be accessed by multiple channels the candidates for services were the *access control service* (Activation and Authorisation) and the *query service*. The activation part of the access control retrieves the roles and teams available to the user. The authorisation part of the access control adds authorisation rules to the chosen roles and teams. The authorisation aspect of the access control was combined with the query fields available, so that the user only has to call the system once to get all of the information they need to perform a query. This service was named ‘get query fields’. Finally, the user can make the actual query using the query service. The final list of services was:

- Activation service
- Get query fields service
- Query service

### 5.2.4.3 Business Process Modelling

Figure 26 shows the UML activity diagram for the business processes for the reengineered IBHIS broker.



**Figure 26 IBHIS business process**

The activity diagram in Figure 26 outlines how the client will interact with each business process. Firstly, the client allows the user to enter their login details, which are sent to the *activation process*, which in turn calls the *activation service*. If the username and password are valid, the roles and teams for that user are made available to the user. This allows the user to activate those that are relevant to that particular session, otherwise they have to re-enter their user name and password. Once the user has chosen the roles and teams, these are submitted to the *get query fields process* from the client. The *get query fields process* calls the *query fields service*, which performs authorisation on the roles and teams submitted and also returns the query fields for querying patient records. The roles and teams are also authorised to ensure that the user is allowed to activate the particular role(s) and team(s) chosen e.g. based upon the time of day. Finally, the user submits their username, the authorised roles and teams, and the query data. The client then calls the *query process*, which calls the *query service*. The query data is the query criteria (e.g. 'Surname'), the query value (e.g. 'King') and the fields that the user wishes to have returned to them (e.g. first



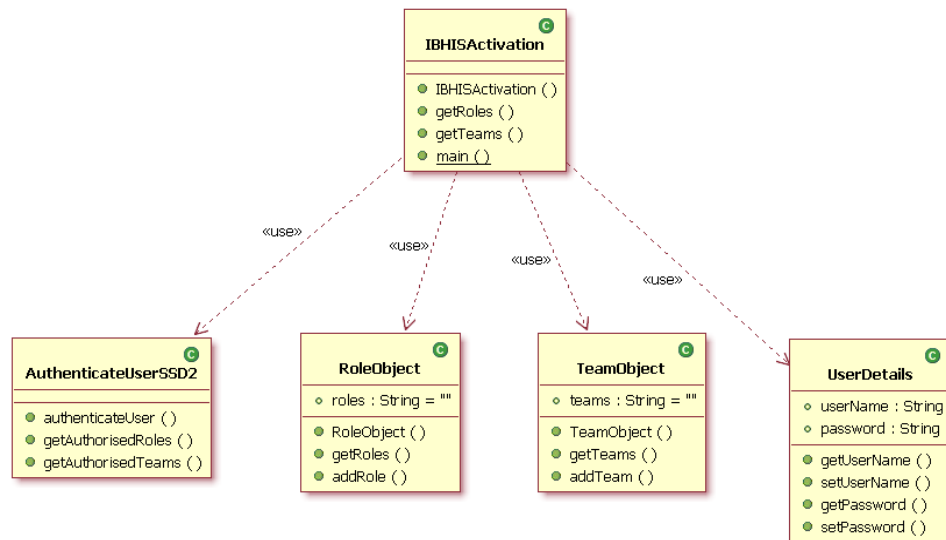
name, address, marital status, date of birth and health information). The results are then returned to the client for the user to view.

### **5.2.4.4 System redesign**

The system redesign involved two stages; deriving the system classes and modelling the service interfaces. The Activation service will be used as an example.

#### **System classes**

Originally it was intended to use the XWIF method (Erl, 2004) for redesigning the classes. However, this method requires redesigning the business logic classes and the original IBHIS broker had no business logic classes as the logic was in the original IBHIS broker GUI code. This meant that the relevant code needed to be extracted from the web pages of the GUI and then moved into business logic classes. In order to decide on the method signatures for the business logic classes it was necessary to examine each of the web pages and determine which system classes (Java beans) were being instantiated and which methods were being called. The instantiations and method calls, along with any other logic associated with the calls were then added to a business logic class. The aim was to expose each business logic class as a web service. If more than one method was called for a service, all of the parameters needed were passed to a single method signature and the class delegated each parameter to the correct method.



**Figure 27 Classes for Activation service**

Figure 27 shows the class diagram for the Activation service. The class labelled `IBHISActivation` is the business logic class which contains the code extracted from the GUI. The class named `AuthenticateUserSSD2` was extracted from the original system and is called from the business logic class (`IBHISActivation`) instead of the GUI. The remaining classes (`RoleObject`, `TeamObject` and `UserDetails`) are the business objects (implemented as Java beans). These were used as inputs and outputs for the `IBHISActivation` class and map to complex types found in the service description file (WSDL). It was necessary to use complex types as WSIF return types must be complex types rather than simple types.

### Service interfaces

In order to create the service interfaces, the XWIF (Erl, 2004) method was used.

### *Step 1 - Choose a service model*

The first step in the XWIF method involves stating the type of service in order to understand the restrictions and implications of the service. For the Activation service the details are as follows:

- **Service name:** Activation service
- **Service type:** Business service

The service type ‘business service’ (Erl, 2004) means that its functionality is used for a unique business context.

### *Step 2 - Establish scope of business function*

This step states the ‘applications’ in which the service will be used. For the Activation service the details are as follows:

- **Service name:** Activation service
- **Scope:** This service can only be used for access control

### *Step 3 - Identify potential requestors*

The third step identifies the future usage of the system, not only by other service requesters but also changes in the current business processes. For the Activation service the details are as follows:

- **Service name:** Activation service
- **Potential services:** N/A
- **Other applications:** N/A
- **Other business processes:** The activation service could be used within another health related business process.

- **External organizations:** Another healthcare organisation may wish to utilise role/team access control.

### *Step 4 - Identify data*

This stage of the XWIF method involves separating the resources that are needed by the service from the parameter data. For the Activation service the details are as follows:

- **Service name:** Activation service
- **Parameter data:** Username, password
- **Resource data:** None

### *Step 5 - Explore application paths*

The fifth step involves determining if there are any other ways that the functionality could be performed at the class level. For the Activation service the details are as follows:

- **Service name:** Activation service
- **Other paths:** A system class containing the desired functionality is called from the business logic class. The only way to make this more efficient would be to move the code from the class into the business logic class. However, it is desirable to keep these layers distinct to ensure that the system is loosely coupled.

### *Step 6 - Encapsulation boundary*

This stage examines the parts of the application with which the service interacts. For the Activation service, the details are as follows:

- **Service name:** Activation Service
- **Service class:** IBHISActivation
- **Classes used:** AuthenticateUserSSD2, UserDetails, RoleObject, TeamObject

#### *Step 7 - Model interface*

This stage defines the service interface and its operations. For the Activation service the details are as follows:

- **Service name:** Activation service
- **Operations:** get roles, get teams

#### *Step 8 - Map interaction scenarios*

This stage involves stating the methods in the business logic class that will be called by the service interface operations. See Table 29 for the interaction details for Activation service.

Operation	Methods
getRoles	getRoles ()
getTeams	getTeams ()

**Table 29 Interaction Scenarios**

#### *Step 9 - Design message payload*

This stage involves modeling the XML documents that will be passed to and from the service. See Table 30 for the operations for the Activation service with their inputs and outputs.

Operation	Input	Output
getRoles	userDetails	Roles
getTeams	userDetails	Teams

**Table 30 Service messages**

In Table 30 all of the inputs and outputs are complex types. For example, the input for the operation `getRoles` is a complex type called `userDetails` and it returns the `roles` complex type. Table 31 shows how the complex types are structured.

Name	Element	Type
userDetails	username	String
	password	String
Roles	Roles	String
Teams	teams	String

**Table 31 Complex types**

The elements for `userDetails` are string values containing the user name and password submitted by the user. Figure 28 and Figure 29 illustrate the XML types for `userDetails` and `roles`.

```
<complexType = "userDetails">
  <sequence>
    <element name ="username" type="string" />
    <element name ="Password" type="string"/>
  </sequence>
</complexType>
```

**Figure 28 UserDetails type**

```
<complexType = "roles">
  <sequence>
    <element name ="role" type="string"/>
  </sequence>
</complexType>
```

**Figure 29 Roles type**

Figures 28 and 29 correspond to the business object classes (`UserDetails`, `RoleObject`) that were outlined in the class creation. The data types for the `roles/teams` elements are also strings values. These are comma separated strings containing the `roles/teams` in the database that are available to the user. The reasons for using a comma separated string rather than separate elements for each role or team

are that searching through a string would be faster than parsing a complex type with many elements and also the message size would be smaller than an XML document with complex types having multiple elements.

### **5.2.4.5 System recoding**

The system recoding consisted of the following steps:

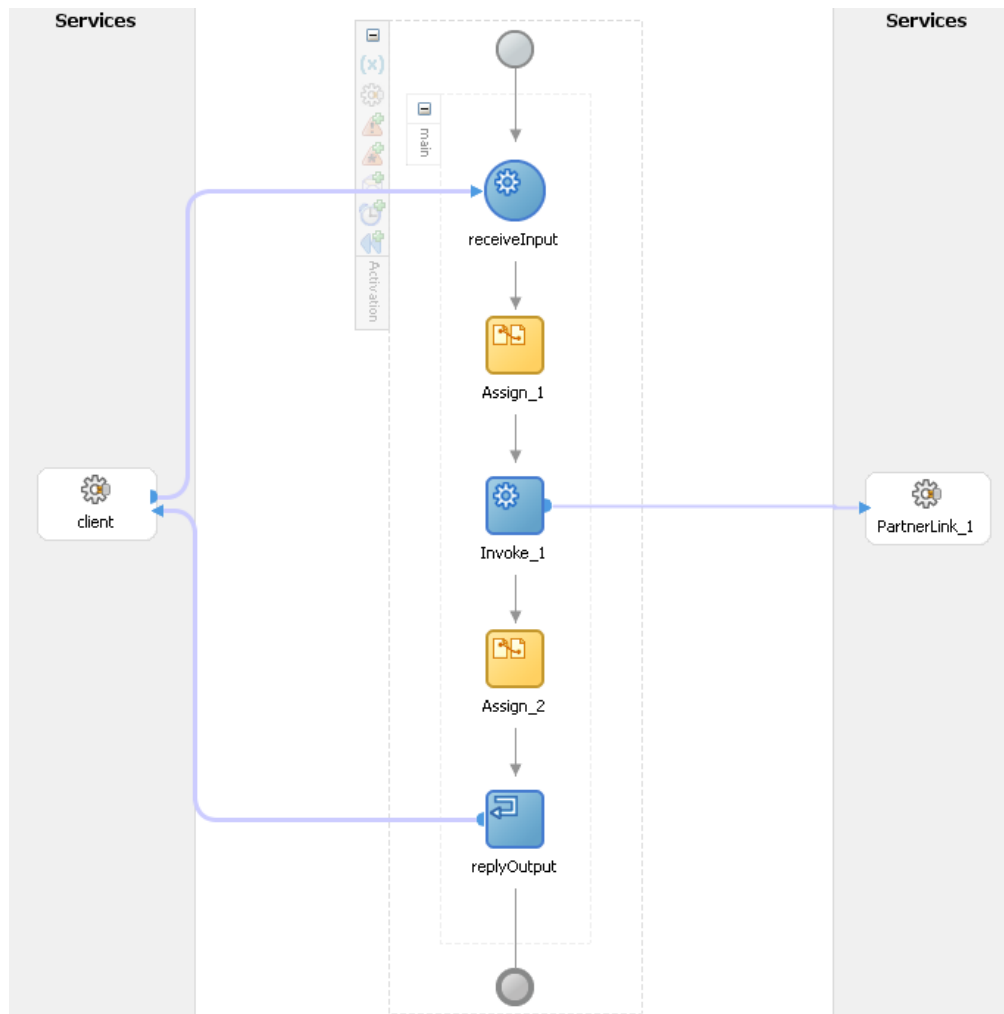
1. Create a project for the service
2. Create the business logic class
3. Create the business object classes
4. Move the classes and required libraries from the original IBHIS broker to the reengineered IBHIS broker.
5. Move the business logic code from the web pages into the business logic class
6. Expose the business logic class as a web service

### **5.2.4.6 Service interface creation**

The service interfaces were created automatically from the business logic classes. These were then checked to see if they matched the service interfaces designed in the modeling stage.

### **5.2.4.7 Creating business processes**

Figure 30 shows the Activation business process which calls the Activation service. This process represents the Activation process swim lane in Figure 26.



**Figure 30 Activation business process**

Figure 30 shows the Activation business process for role activation. In the left hand column labeled ‘services’, there is the client that initiates the services by sending the `userDetails` complex type containing the login details. This complex type is then assigned to the input type for the Activation service call. When the call to the `getRoles` operation is made to the Activation service the role complex type is returned. The Activation service is labeled `PartnerLink_1` in the right hand services column. The returned roles complex type is then assigned to the output variable for the process which is finally returned back to the client.



Figure 30 can also be used to show the flexibility of using a business process. Functionality could be added or removed from the business process with relative ease by either using other services or BPEL language functions before or after the Activation service is called.

One problem found with generating the web service interfaces automatically from the business logic code, was that they were not compatible with the business processes. The result was that the web services had to be manually edited in order to be consumed by the business processes.

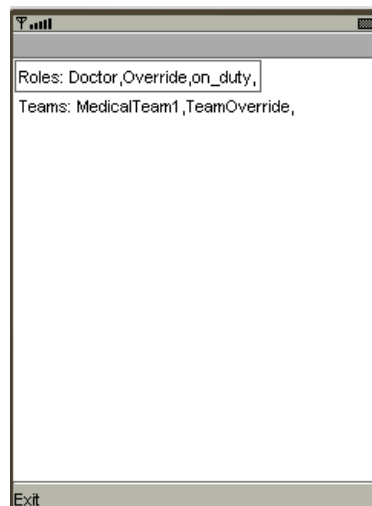
### **5.2.4.8 Client**

In order to test the reengineered IBHIS broker, it was necessary to create a mobile client for the business process. This was done using the Java Wireless Toolkit with additional web services (JSR172) support. The method for using web services with JSR172 is that Java stubs are created from the description document (WSDL). Each service has a Java class representing the service and also classes representing the types. This then enables the service to be used programmatically. As BPEL processes are exposed as web services and have a description document (WSDL) the same method was used to create stubs to call the business process from the mobile application code. Figure 31 and Figure 32 show the input and output forms for the Activation process on a mobile emulator.



**Figure 31 Activation input**

Figure 31 shows the entry point for the system, which involves the user entering their login details. When the user clicks ‘Next’, the application makes two calls to the Activation process. The first call is to retrieve the roles and the second call is to retrieve the teams. Figure 32 shows the roles and teams that are available to the user, including overrides.



**Figure 32 Activation output**

There were some problems when coding the client. The first was related to the messages that were returned. When a business process calls a service, the types that

are returned use the namespaces (URI) from the service and these do not change when the business process returns that data. However, when the Java stubs are created they cannot interpret these namespaces as the Java stubs use the namespaces that are declared in the business process description document. The result is that when the data is returned, the Java stub does not recognize the namespace with which it is associated. This was overcome by manually editing the Java stubs so that they would recognize the namespaces. Another problem was with deploying this on a device. When the mobile application creates the stubs they are based on the web services standard. BPEL processes are also based on the web service standards. However, the Oracle BPEL Process Manager implementation of the standard returned a response with empty header values as shown below:

```
<header/>
```

Whereas the implementation of web services on the Nokia phones tested however, was different and required matching tags for empty values as shown below:

```
<header></header>
```

This conflict meant that the mobile application would throw an exception due to the fact that there was no matching tag. This is known as the ‘empty header problem’ (Pellerin, 2007). Unfortunately, this problem could not be worked around, which meant that the mobile business processes could only be accessed from the emulator and not the physical device.

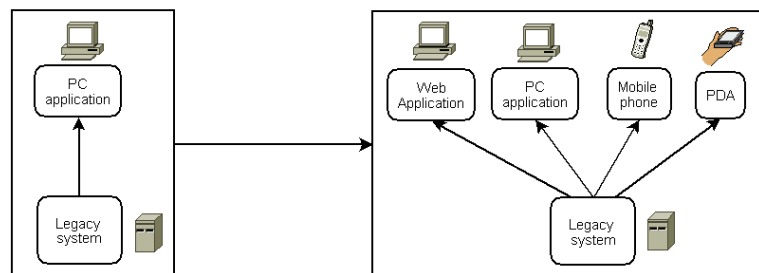
### 5.2.4.9 Testing

Functional testing was performed on the reengineered IBHIS broker (See appendix B). It was intended that a test specification from the original project be used. Unfortunately this could not be found in the documentation. There were however,

three examples of usage found in a demonstration document. These test cases were used to ensure that the items returned were the same and the reengineered IBHIS system returned the same items as the original IBHIS broker. Also, the services provided by the reengineered IBHIS broker, were assessed to see how well they fulfilled the requirements and offered the same functionality of the original IBHIS broker. This was performed by the researcher and one of the supervising team, who had worked on coding the original IBHIS broker. The service interfaces of the reengineered IBHIS broker were rated on a semantic differential scale. Out of a possible score of ten, the reengineered IBHIS broker was rated nine. In terms of the channels, only the mobile channel and web channel were tested.

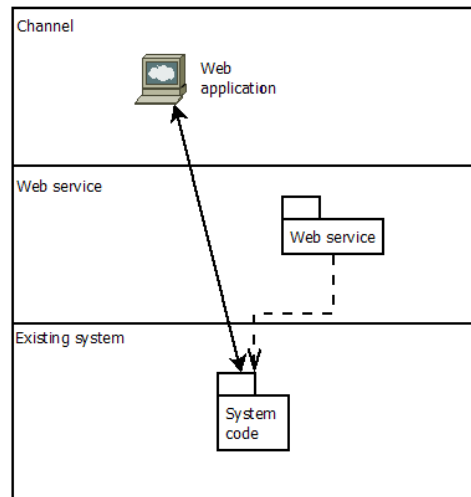
### 5.2.5 Architecture

The original problem of reengineering a system for MCA is shown in Figure 33.



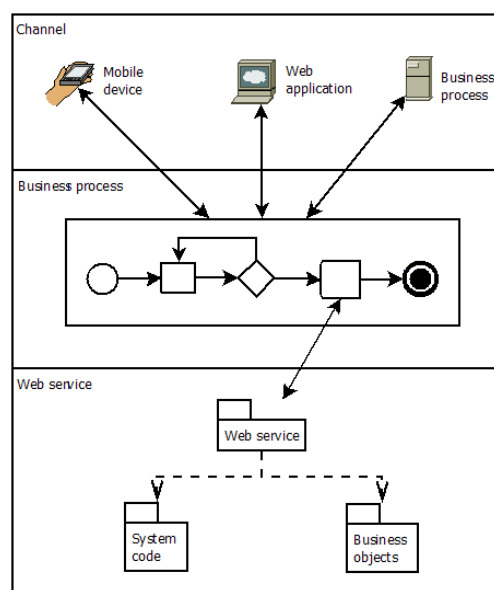
**Figure 33 Original reengineering problem**

Figure 33 shows the problem of a single channel system that needs to be accessed from multiple channels. Figure 34 shows the original IBHIS broker, which represents the single channel system in Figure 33.



**Figure 34 Original IBHIS broker architecture**

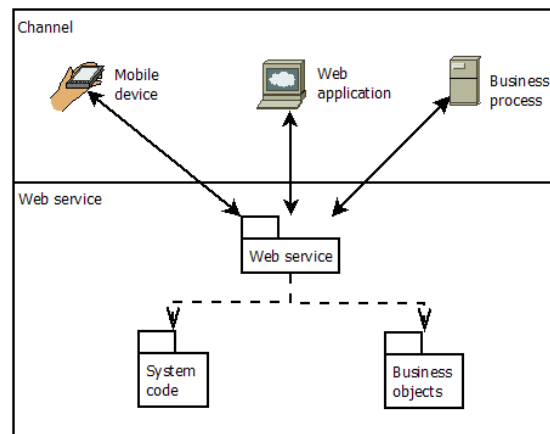
Although there are a number of web service interfaces in the original IBHIS broker, they do not represent coarse grained business functions, and instead they expose low level classes. In order to improve the granularity and make the system accessible from multiple channels, McArc was proposed, which comprised of the Channel layer, Business process layer and Web service layer. Figure 35 shows the architectural layers of the reengineered IBHIS broker.



**Figure 35 Reengineered IBHIS broker architecture**

Figure 35 shows the architectural layers for the reengineered IBHIS broker. The web service layer contains code that was extracted from the original IBHIS broker as well as the business objects (types) used and the service classes. The business process layer calls the services from the web service layer. The business process layer is called from the channel layer (mobile device, web app, another business process).

The web service layer could also be called directly from the channel layer, allowing the business process layer to be bypassed completely, as shown in Figure 36.



**Figure 36 Direct access of reengineered IBHIS services**

In this case if the business processes were not required the services could be accessed directly due to the loosely coupled nature of business processes and services.

## Chapter 6: Case study results

This chapter presents the results of the case study. First the results are presented by the data collected. These are then grouped by research question. The performance measurements are explained in relation to research questions CS-RQ1-P1 and CS-RQ2. The agility metrics are explained in relation to research questions CS-RQ1-P2 and CS-RQ3. The agility test cases are explained in relation to research question CS-RQ1-P2. Finally the results of the evaluation of McARM are presented for CS-RQ4.

### 6.1 Results by data collected

The results for the performance testing, agility metrics and agility test cases are presented in this section.

#### 6.1.1 Performance testing

Table 32 presents the results of the performance testing for each technology, for the three different message sizes (see chapter 5) in:

- the performance mock-up IBHIS broker
- the original IBHIS broker
- the reengineered IBHIS broker

RQ	Point of measurement	Message size		
		Small	Medium	Large
1a	WSIF BP (reengineered IBHIS performance mock-up)	213.25	196	180.55
	SOAP BP (reengineered IBHIS performance mock-up)	196	175.8	213.4
2	Service layer	16.5	17.25	16.5
	Business process layer	231.2	228.85	238.15
	Channel layer	705.15	713	722.25

Table 32 Results for performance testing

### 6.1.2 Agility metrics

Table 33 shows the agility metrics for the original IBHIS broker and the reengineered IBHIS broker with respective bindings.

Product Quality criteria	Metric	Original IBHIS broker	Reengineered IBHIS broker – SOAP (PBM)	Reengineered IBHIS broker- WSIF (NLC)
Complexity	No. of service interfaces (total)	22	5	5
	Number of classes (total)	106	66	66
	Average number of methods per class	3.041	3.506	3.506
	Average method size	17.090	8.292	8.292
	Average number of instance variables	3.242	1.48	1.48
	Number of children	0	0	0
Modularity	Coupling between object classes	1.170	1.015	1.015
	Lack of cohesion on methods	0.57	0.758	0.758
Granularity	-	4	8	8
Expandability	-	5	9	4
Generality	-	5	5	5
Software system independence	-	6	6	6
Communication commonality	-	7	8	3

Table 33 Agility metrics

Table 33 shows that code metrics for the reengineered IBHIS broker (SOAP) and reengineered IBHIS broker (WSIF) are the same. This is because they use the same system code, only the WSDL bindings are different.



### 6.1.3 Agility test cases

Table 34 shows the results of the agility test cases using the agility mock-up IBHIS broker.

Test case	Binding	Time (min)	Number of statements	Artefacts
New service	SOAP	10	48	3
	WSIF	55	48	2
Amend service	SOAP	10	6	2
	WSIF	5	2	2

**Table 34 Time and effort for a new service**

## 6.2 Results by research question

In this section the data collected will be presented by research question.

### 6.2.1 CS-RQ1-P1

A 2x3 mixed factorial ANOVA was used to analyse the differences between the binding used (NLC or PBM) in the three conditions (message size) using the reengineered IBHIS performance mock-up. The mixed factorial was used as the message sizes were not equal for each binding. Due to the structure of the data being sent a small message size for WSIF may be 10 bytes whereas a small message for the SOAP binding may be 100 bytes. Table 35 shows the table of means for the response times (ms) of the two binding technologies and the three different message sizes (See Appendix C).

Binding	Response time (ms)		
	Small	Medium	Large
WSIF (NLC)	213.25	196	180.55
SOAP (PBM)	196	175.8	213.4

**Table 35 Table of means**

Table 35 shows that WSIF (NLC) performs worse than SOAP (PBM) in terms of time for the small and medium sized messages but not for the larger messages. Table 36 shows the ANOVA summary table:

Source		Type III Sum of Squares	df	Mean Square	F	Sig value
Binding	Hypothesis	72.075	1	72.075	0.012	Binding
	Error	24900.767	4	6225.192		
Message size(Binding)	Hypothesis	24900.767	4	6225.192	0.349	Message size(Binding)
	Error	2035777.150	114	17857.694		

**Table 36 ANOVA summary table for performance investigation**

Table 36 shows the value of F (0.012) for the binding is less than the significant value (7.71). Therefore there was no significant difference found between the two bindings ( $F[1,4] = 0.012$ , MS error=6225.192,  $p > 0.05$ ). The value of F for the message size (0.349) was less than the significant value (2.45). This means that there was no significant difference for the message sizes ( $F[4,114] = 0.349$ , MS error=17857.694,  $p > 0.05$ ).

### 6.2.2 CS-RQ1-P2

For agility there were two measures: *system metrics* and *agility test cases*.

#### 6.2.2.1 System metrics

The system metrics were the code metrics and the semantic differential measurements. The code metrics for the SOAP (PBM) and WSIF (NLC) implementations were the same as the same system code was used in the reengineered

IBHIS broker. The only difference is the WSIF binding declarations in the service description (WSDL) to access the system. Table 37 shows the semantic differential measurements for the PBM (See appendix D) and NLC (See appendix E):

Area	SOAP (PBM)	WSIF (NLC)
Granularity	8	8
Expandability	9	4
Generality	5	5
Software system independence	6	6
Communication commonality	8	3

**Table 37 Semantic differential measurements for the bindings**

Table 37 shows that the areas where SOAP (PBM) and WSIF (NLC) differ are *expandability* and *communication commonality*. In both cases the PBM implementation is rated higher. The *code metrics* and the *semantic differential measurements* from Table 31 are compared in terms of the quality model (see Table 38). These are summarised using a Boolean value (1 or 0) which indicates which technology scored higher. If they are the same then both score 0.

External quality Factor	Product quality Criteria	SOAP (PBM)	WSIF (NLC)
Granularity	-	0	0
	<b>Total</b>	<b>0</b>	<b>0</b>
Flexibility	Complexity	0	0
	Expandability	1	0
	Generality	0	0
	Modularity	0	0
	<b>Total</b>	<b>1</b>	<b>0</b>
Reusability	Complexity	0	0
	Generality	0	0
	Modularity	0	0
	Software-system Independence	0	0
	<b>Total</b>	<b>0</b>	<b>0</b>
Interoperability	Modularity	0	0
	Communication commonality	1	0
	<b>Total</b>	<b>1</b>	<b>0</b>

**Table 38 PBM/NLC agility comparison**

Table 38 shows that SOAP (PBM) is rated higher than WSIF (NLC) in terms of flexibility and interoperability. The lower flexibility of NLC is due to the lack of expandability of the architecture. The lower interoperability of NLC is due to the lack of communication commonality. As the code metrics were the same for both bindings, they scored 0 in these cases e.g. complexity (see Table 33).

#### 6.2.2.2 Agility test cases

Along with the system metrics, the *agility test cases* were used to assess the agility of the PBM and NLC bindings. These will be examined individually.

*Agility test case 1: Create a new service for a business process.*

Table 39 shows the time to create a new service, the number of statements and the number of artefacts created (WSDL documents and classes).

	Time (ms)	NOS	Artefacts
SOAP (PBM)	10	48	3
WSIF (NLC)	55	48	2

**Table 39 Time and effort for a new service**

The main difference shown in Table 39 is the extra time needed to implement the WSIF description document.

*Agility test case 2: Amending an existing service in a business process.*

The time needed to change the service, the number of statements created and the number of artefacts changed (WSDL documents and classes), were recorded for this test case (see Table 40).

	Time (ms)	NOS	Artefacts
SOAP (PBM)	10	6	3
WSIF (NLC)	5	2	3

Table 40 Time and effort to change a service

Table 40 shows that it requires less statements to be created and takes less time to change a service implemented in WSIF. It should be noted that the number of statements also includes changes to the business process.

### 6.2.3 CS-RQ2

To investigate the potential affect of the additional layers required for MCA on performance, a comparison was made between layers (see figure 11) of the reengineered IBHIS broker (PBM):

- Service layer
- Business process layer
- Channel layer

The *service layer* response times were measured in the development environment, the *business process layer* response times were measured in the business process server and the *channel layer* response times were tested in a mobile emulator. Table 41 shows the average response time (milliseconds) for each technology (See appendix F for more details).

Calling technology	Response time (ms)			
	small	medium	large	
Service	16.5	17.25	16.5	16.75
BP	231.2	228.85	238.15	232.73
BP (Mobile)	705.15	713	722.25	713.47

Table 41 Layer comparison table

The results show that adding a business process layer does reduce performance by around 216ms and that adding the mobile client layer reduces performance by around 480ms. Table 42 shows the ANOVA summary table:

Source		Type III Sum of Squares	df	Mean Square	F	Sig value
POINT	Hypothesis	15263349.033	2	7631674.517	11821.159	5.14
	Error	3873.567	6	645.594		
SIZE(POINT)	Hypothesis	3873.567	6	645.594	.018	2.10
	Error	6182676.350	171	36156.002		

**Table 42 ANOVA for CS-RQ2**

Table 42 shows the value of F (11821.159) for the point is higher than the significant value (5.14). Therefore, there is a significant difference found between the three layers ( $F[2,6] = 11821.159$ , MS error=645.594,  $p > 0.05$ ) between the points. The value of F for the message size (0.18) was lower than the significant value (2.10). This means that there was not a significant difference for the message sizes ( $F[6,171] = 0.18$ , MS error=36156.002,  $p > 0.05$ ).

#### 6.2.4 CS-RQ3

In order to investigate if reengineering a system as an SOA improves agility, the original IBHIS system and the reengineered IBHIS system (PBM) were compared. Table 43 shows the code metrics for the two systems.

Product quality criteria	Metric	Original IBHIS broker	Reengineered IBHIS broker (PBM)
Complexity	No. of service interfaces (total)	22	5
	Number of classes (total)	106	66
	Average number of methods per class	3.041	3.506
	Average method size	17.090	8.292
	Average number of instance variables	3.242	1.48
	Number of children	0	0
Modularity	Coupling between object classes	1.170	1.015
	Lack of cohesion on methods	0.57	0.758

Table 43 Code metrics for original/PBM

The code metrics show that the reengineered IBHIS broker improved on the original IBHIS broker in all of the metrics except *number of methods*, *lack of cohesion* on methods and *number of children* (subclasses) which stayed the same. This means that in terms of the complexity, the reengineered IBHIS broker was found to be less complex. In terms of modularity (CBO and LCOM), the systems were rated the same as each system scored higher in one aspect of modularity. Table 44 shows the semantic differential measurements for the original IBHIS broker (See appendix G) and the reengineered IBHIS broker (See appendix D).

Area	Original IBHIS broker		Reengineered IBHIS broker (PBM)	
	Researcher	Validation	Researcher	Validation
Granularity	4	5	8	7
Expandability	5	4	9	6
Generality	5	7	5	7
Software system independence	6	3	6	3
Communication commonality	7	7	8	8

Table 44 Semantic differential measurements for original/reengineered IBHIS

For the semantic differential measurements, the reengineered IBHIS broker (PBM) scored higher than the original IBHIS broker on all aspects, except *generality* and *software system independence* which remained the same. Table 45 compares the original system against the reengineered IBHIS broker (PBM) in terms of the quality

model. These are summarised using a Boolean value (1 or 0) which indicates which technology scored higher. If they are the same then both score 0.

External quality factor	Product quality Criteria	Original IBHIS broker	Reengineered IBHIS broker (PBM)
Granularity	-	0	1
	<b>Total</b>	<b>0</b>	<b>1</b>
Flexibility	Complexity	0	1
	Expandability	0	1
	Generality	0	0
	Modularity	0	0
	<b>Total</b>	<b>0</b>	<b>2</b>
Reusability	Complexity	0	1
	Generality	0	0
	Modularity	0	0
	Software-system Independence	0	0
	<b>Total</b>	<b>0</b>	<b>1</b>
Interoperability	Modularity	0	0
	Communication commonality	0	1
	<b>Total</b>	<b>0</b>	<b>1</b>

**Table 45 Original/Reengineered IBHIS (PBM) comparison**

Table 45 shows that the reengineered IBHIS broker (PBM) improved all aspects of agility when compared the original IBHIS broker. The key areas were; granularity, complexity, expandability and communication commonality.

#### 6.2.5 CS-RQ4

The McARM reengineering method was evaluated in terms of *efficiency* and *effectiveness* (Pfleeger, 2001). The efficiency examined the time taken to carry out each stage of the method. The effectiveness reported the following:

- Any problems encountered
- Rating of perceived level of difficulty (semantic differential scale).



Table 46 shows the ratings of the method stages

Stage	Time (hours)	Problems	Ease of use
System understanding	26.25	The size of the system	7
Service identification	8.5	None	7
Business process modelling	5	None	8
System Redesign	27	None	7
Recoding	130.5	Problems with technology, not enough detail	2
Business Process creation	66	Problems with technology, no examples/tutorials for WSIF	4

**Table 46 Assessment of reengineering method**

Table 46 shows that whenever there was a problem with a stage there would be an increase in the amount of time performing the stage and a perceived increase in difficulty. In terms of the time taken to perform the stage, *business process creation* and *recoding* were those that took longer than the other stages. These stages also had the most problems.

### 6.2.6 Summary

For CS-RQ1-P1, the results show there was no significant difference found between the two bindings. There was also no significant difference for the message sizes. For CS-RQ1-P2, the results show that SOAP (PBM) is rated higher than WSIF (NLC) in terms of flexibility and interoperability. For CS-RQ2, there was a significant difference found between the three layers but there was not a significant difference between the message sizes. For CS-RQ3, the results show that reengineering the IBHIS broker as an SOA improved all aspects of agility. The key areas were; granularity, complexity, expandability and communication commonality. The stages

found with problems were those where new technologies needed to be learnt. These were also the stages that took the most time to complete.

## Chapter 7: Discussion

This chapter will discuss the results of the case study by research question. Changes to the agility quality model are also proposed.

### 7.1 CS-RQ1

Case study research question 1 (CS-RQ1) is divided into two propositions:

- CS-RQ1-P1 - NLC used in a system reengineered for MCA will result in *improved performance* when compared to PBM for a mobile service-based system.
- CS-RQ1-P2 - A system reengineered for MCA using NLC will be *less agile* than one using PBM.

These will be first discussed separately and then together in a summary.

#### 7.1.1 CS-RQ1-P1

This research question investigates the performance of NLC (WSIF) and PBM (SOAP) in multi-channel business processes. The results in Table 36 in section 5.2.1 show that there is not a decrease in response times when using WSIF instead of SOAP in the small message medium sized messages, but there was for the large message. However, neither message size nor binding was found not to be at a significant level (Table 37 in section 6.2.1). The working hypothesis is not supported and therefore

NLC used in a system reengineered for MCA will *not* result in improved performance when compared to PBM. This means that NLC should not be used instead of PBM for improving the performance of a multi-channel business process. This result was not expected as previous work had found that systems using NLC technologies perform better than PBM, specifically WSIF has been found to be faster than using SOAP (Migliardi & Podesta, 2004). One reason that this could be the case is that the message sizes were not representative of the possible range of message size for SOAP and the range offered by querying the IBHIS broker were too similar in size.

### **7.1.2 CS-RQ1-P2**

The agility of NLC (WSIF) and PBM (SOAP) bindings were compared using the agility quality model and the agility test cases.

#### **7.1.2.1 Agility quality model**

The results of this study found that NLC bindings were less agile than PBM in service-based systems. The results in Table 37 in 6.2.2.1 show that WSIF (NLC) was rated lower than SOAP (PBM) in terms of flexibility and interoperability. WSIF (NLC) was rated lower than SOAP (PBM) in terms of interoperability due to the lower rating of communication commonality. The reason WSIF (NLC) did not offer the same level of communication commonality as SOAP (PBM) is that the SOAP (PBM) protocol is recognised by a higher number of languages than WSIF (NLC). This was evident in the reengineered IBHIS broker, as unlike the SOAP (PBM) bindings, the WSIF (NLC) binding could not be used to access the system from a business process. The lack of communication commonality is also the cause of the

lack of flexibility. The architecture could not be expanded easily due to a tight coupling caused by the implementation of WSIF (NLC) in IBM Websphere 5.1.1, which meant that the system could not be called from a business process. In fact, some parts of the system could not be called at all.

As SOAP (PBM) was found to be better than WSIF (NLC) in these two aspects, the working hypothesis is supported and it can be said that a system reengineered for MCA using NLC will be less agile than one using PBM. This means that if agility is an important aspect when reengineering a system for MCA, the use of NLC will not be appropriate.

#### **7.1.2.2 Agility test cases**

Table 39 in section 5.2.2.2 shows that when creating a new service for use in a business process, using SOAP was found to be easier than WSIF. Although the number of code statements was the same for each service, the time taken to create the WSIF services was longer. The reason for this is that a SOAP web service can be automatically generated from Java classes; this includes the service description document. However, in order to create a WSIF service, the service description document had to be manually created, which is non-trivial and time consuming. Some tools existed in the development environment for creating WSIF WDSL documents, but these were not mature and were not even capable of creating the simple service used in this study. This lack of tool support contributed to the lack of agility as it had an impact on the time taken to carry out the changes. In this study the researcher had written the WSIF service description document several times during prototyping. This

meant that when conducting the test case, the service interfaces were created more easily than they would have been if the service was being created for the first time. Even with this prior experience of using the technology, manually creating the interfaces was difficult and time consuming. A further issue is that WSIF lacks detailed documentation and there are very few working examples, making it more difficult to learn and implement. SOAP however, has a great deal of examples and tutorials available.

Table 40 in section 6.2.2.2 shows that in terms of amending an existing service, it is faster to amend a service based on WSIF than it is a service based on SOAP. For the WSIF service, only a few lines of code needed to be changed. For the SOAP service a similar number of lines of code needed to be changed, but the process of creating a web service had to be carried out also. Section 6.2.2.2 shows that when replacing a service with an alternative identical service (SOAP or WSIF) the time and effort would be the same. The reason for the time being the same is that the operations (defined as a port type in a WSDL document) provided by the service would be the same for both types of binding. Also, section 6.2.2.2 shows that when trying to access a service from another server (BPEL), it is not possible to access a WSIF service outside of the virtual machine in which it is running. The SOAP service however, could be accessed from a BPEL server. Therefore, the time and effort required could not be measured, but this does mean that if a service needs to be called outside of the server it is running in then WSIF is not appropriate.

WSIF took more time and effort for agility test case 1, but for the agility test case 2, WSIF took less time than SOAP, however only marginally. For agility test case 3, the

two bindings would take an equal amount of time and effort. Finally, agility test case 4 found that WSIF cannot be called from an external business process unlike SOAP which meant that time and effort could not be measured. This means that in terms of the time and effort required, using WSIF results on a system is less agile than SOAP, as it would take more time to respond to change.

The implications for these findings are that using SOAP will be easier to use than WSIF, due to the fact that WSIF service interfaces have to be created manually and that there are very few resources for learning the technology. Therefore, in terms of the agility test cases PBM is thought to be more agile than NLC.

### **7.1.3 Summary**

In order to address the CS-QQ1, the results of CS-RQ1-P1 and CS-RQ1-P2 are combined and compared against the decision table from the planning phase of the case study (see Table 26). If the NLC (WSIF) binding outperformed PBM (SOAP), it is marked with 'NLC' otherwise it is marked with 'PBM'. As NLC did not improve performance and scored lower on the agility metrics and the agility test cases, outcome number five was obtained. Based on these findings the following conclusion was made: native language calls should not be used in a business process as neither performance nor agility is improved.

## **7.2 CS-RQ2**

For research question CS-RQ2 – *'Does the inclusion of extra layers required for MCA reduce performance?'* the aim was to investigate the affect of each additional

architectural layer required for MCA on system performance. The results in section 6.2.3 (Table 42) show that with each layer added there is an increase in the response time. However, there was not a significant difference between the message sizes. These results agree with the statements by Hasselbring, et al. (2004) and Krafzig et al. (2004), that adding extra layers causes a reduction in performance.

The mean response time for the medium message was found to be faster than the mean response time for the small message of the service. Looking at the data sets, the mean response times were similar across all three message sizes. The mean response time for the small message and the large message were equal and the slower mean response time for the medium size message appears to be caused by an outlier. There was one outlier in each data set which was outside of the standard deviation. For both the small and large messages this value was the same (31ms), however this was higher for the medium sized message (47ms) which would have affected the mean. One potential cause for the outlier could be the network used in the experiment. A home Wi-Fi connection was used, which may have had variable network speed caused by using internet. There were measures taken to reduce this variation (see Table 54). Another control that could have been used would be to use a private network instead of a public network, but this would have reduced the ecological validity as multi-channel systems are unlikely to be used under such controlled conditions. Also, the statistical analysis of variance used is non-parametric, which are robust at coping with outliers (Coolican, 1999).



For the business processes the mean response time for the medium sized messages were marginally faster than the mean response time for the small messages. As with the services, the mean response times are similar across message size, however the cause does not appear to be caused by outliers. Again, the cause for the mean response time being faster for the medium message could also be attributed to variation in the network speed.

A further consideration that may affect the performance is that the client technology would need to process the response message from the service once it is received. This would take longer for a mobile device than a desktop PC due to its limited processing power and would further increase the response time.

### **7.3 CS-RQ3**

In this section the original IBHIS broker and the reengineered IBHIS broker (SOAP binding) are compared in terms of the agility quality model to investigate if reengineering a system as an SOA improves the agility of a system for CS-RQ3 -

*Does reengineering a system as an SOA improve agility?*

#### **7.3.1 Product quality criteria**

Each of the product quality criteria will be discussed from the results in section 6.2.4 (Tables 43 and 44).

### 7.3.1.1 Granularity

In terms of the services provided, the original IBHIS system was thought to be too fine grained (Table 44). Web service interfaces had been created for a number of system classes. However, these were simply web service interfaces and do not appear to have been designed as part of an SOA. Having a large number of web service interfaces may be good for having many reconfigurations of the system, but this does not represent the system in its simplest form. For reengineering IBHIS for MCA, a small number of coarse-grained services were required to reduce the number of messages in the system. This was addressed in the reengineered IBHIS broker, which offered a set of more reusable coarse grained service interfaces that were aligned with the ‘business functions’.

### 7.3.1.2 Complexity

In section 6.2.4 (Table 43) the results show that the reengineered IBHIS broker improved on all aspects except *number of methods*. However, the original IBHIS broker did not have any values that would be considered problematic. The *number of classes* was nearly halved from the original IBHIS broker making the reengineered IBHIS broker easier to understand and manage. However, it should be taken into account that seven of the classes in the original IBHIS broker were old versions and test classes. The *average number of methods per class* increased for the reengineered IBHIS broker. Although this number was increased in the reengineered IBHIS broker from 3.041 to 3.506, both were well under 20, a recommended maximum by Lorenz (Kan, 2003). The reason for this increase is a result of the required ‘business object’ classes. These classes were implemented as Java beans and for each field an associated ‘getter’ and ‘setter’ method are created. These methods were not all used

but were kept in order to maintain the notion of a Java bean. The *average method size* was halved in the reengineered IBHIS broker from 17 LOC to eight LOC. The *average number of instance variables* was also halved. However, the number of instance variables in both systems was less than six, the number recommended by Lorenz (Kan, 2003). Halving this number would be more important if the system was larger. The number of children (subclasses) remained the same at 0.

### **7.3.1.3 Expandability**

In section 6.2.4 (Table 44), the results show that the reengineered IBHIS broker had a higher rating for architectural expandability than the original IBHIS broker. The reason for this was that in the original IBHIS broker there was a tight coupling between the GUI and system code. This meant that any extension to the architecture would involve redesigning the user interface code as well as the system code. With the reengineered IBHIS broker, this tight coupling was removed as the business logic that was in the GUI was moved into business logic classes.

### **7.3.1.4 Generality**

In section 6.2.4 (Table 44) the results show that the generality of the services in the reengineered IBHIS broker and the original IBHIS broker were similar. All of the services could potentially be used outside of the application to a limited extent. For example, the ontology could be used in another medical system but could not be used for another domain.

### 7.3.1.5 Modularity

The results in section 6.2.4 (Table 43) show that the modularity of the original and reengineered IBHIS broker was equal. The *coupling between classes* was slightly improved in the reengineered IBHIS broker. However, both systems were below the acceptable number of 14 suggested by Sahraoui et al. (2000) with the original IBHIS broker at measuring 1.170 and the reengineered IBHIS broker at 1.015. The *lack of cohesion in methods* metric slightly increased in the reengineered IBHIS broker. This increase is, again - likely due to the ‘business objects’ classes that it was necessary to create. For each parameter in a Java bean there are two methods (‘getter’ and ‘setter’) rather than many methods using the same variables. This results in more shared variables, which would decrease the cohesion.

### 7.3.1.6 Communication commonality

In section 6.2.4 (Table 44), the results show that the communication commonality was improved in the reengineered IBHIS broker. This was due to the use of SOAP between the business logic layer and presentation layer. This was an improvement on the original IBHIS broker, which used web pages (.jsp) to call Java beans. The reengineered IBHIS broker was not only accessible via the web channel but also other channels, thus improving the ability to integrate with other systems.

### 7.3.2 Summary

In terms of the external quality factors, the reengineered IBHIS broker was rated higher in flexibility, reusability and interoperability (section 6.2.4, Table 45). Therefore it was rated as being higher than the original IBHIS broker in terms of

overall agility. This would agree with the statement that reengineering a system as an SOA improves agility (Krafzig et al. 2004; Allen 2006; Erl 2007; Newcomer & Lomow 2005).

## 7.4 CS-RQ4

CS-RQ4 – ‘*Is McARM an effective method for reengineering a system for MCA?*’ investigated the use of McARM to reengineer an existing system toward a multi-channel architecture. McARM was successfully used to reengineer an existing system for MCA and was found to be easy to follow. No specialist tools were required for the reengineering or service identification stages. This may be a disadvantage in terms of the reduction of time but a manual approach would give developers more control when reengineering the system. Also, UML was used at all stages which is a well known notation, recognised in both industry and academia.

A limitation of McARM is the recoding stages due to the huge number of programming languages that exist. However, an appropriate methodology could be inserted into the McARM re-coding stages where required. For example, a method for re-coding a functional language could be used in this stage instead of the object-oriented XWIF, which was used in the study. The business process creation stage does not have a large amount of languages, so there is a possibility that language specific details could be given for this stage. In a real project, these stages may be less of a problem if developers are familiar with the languages and system technologies being used.

The main problems found with McARM were related to use of the technologies to reengineer the system rather than the method itself, such as recoding and creating the business processes (See section 6.2.5.10). For the business process stage, WSIF took a long time to understand as examples and tutorials were not readily available. For the recoding stage, there were various problems such as dependencies in the system, which meant it was necessary to employ work-around techniques or alternatives had to be found.

Also, for the system understanding stage of McARM, the size of the system represented a problem. The high number of class diagrams were difficult to manage. In order to make the diagrams more manageable, the system was classified by Java project and then by each package within each project. However, this did not show how the projects and packages interacted with each other. This problem was reduced by sequence diagrams, as they showed which classes were called and in which order. Although there was not a high level view of all of the classes in the system, when the system was recoded the required classes were moved to the new projects without difficulty.

It was felt that creating the clients to access the system should be included as a stage, or as a sub-stage of recoding in McARM. The creation of clients is as important as creating the system itself, yet it is not given importance in this method or any of the other methods found in the SLR, with the exception of Comerio et al. (2004). Even though the client types will not always be known in a multi-channel system, the ones that *are* known at the time of designing the system should be created formally. This would help to ensure that the clients were created correctly and also consistently. It is

also felt that the requirements elicitation and analysis stage needs more work. Currently, it consists only of recommendations rather than a structured approach.

When creating the McARM, eight reengineering stages were proposed. In terms of a reengineering method, these stages can be grouped into the reengineering phases described in section 1.3.1 (see Table 47):

Phase	Stage
Reverse engineering	<ul style="list-style-type: none"> <li>Requirements gathering/analysis</li> <li>System understanding</li> <li>Service identification in the existing system</li> </ul>
Transformation	<ul style="list-style-type: none"> <li>System redesign</li> <li>System recoding</li> </ul>
Forward engineering	<ul style="list-style-type: none"> <li>Business process modelling</li> <li>Business process implementation</li> <li>Service interface implementation</li> </ul>

**Table 47 McARM reengineering phases**

In section 1.3.2, there were six levels identified at which a system could be reengineered. Table 48 shows the reengineering levels, the reengineering phases and the stage at which these are performed in McARM.

Level of abstraction	Stage number		
	Reverse	Transformation	Forward
Code	2	5	6
Functional	2	5	6
Design	2,3	5	6,7
Architectural	2	4	8
Requirements	-	-	1
Conceptual	2	-	-

**Table 48 McARM coverage of reengineering lifecycle**

Table 48 shows that McARM covers most of the reengineering lifecycle. Only the requirements and conceptual level of the reengineering process are not adequately covered. In terms of the conceptual level, reference models are used as part of the system understanding stage, however, there are no steps for transforming or forward

engineering these stages. In terms of the requirements, the original requirements documents are not examined or transformed. The method assumes that a system already exists and that it is to be modified to suit the new requirements.

## 7.5 Agility quality model

After working with and using the agility model, it was found that certain aspects of the agility model could be improved. These were:

- The lack of comprehensibility
- The lack of maintainability
- The vagueness of granularity
- Missing product quality criteria
- Misinterpretation of ‘decoupling of functionality and technology’

The first aspect is the lack of assessment of documentation for comprehending the system. Good documentation makes the system easier to understand and therefore easier to change. This was one of the characteristics suggested by Krafzig et al. (2004) but was not included in the agility model as there was no direct mapping to McCall’s quality model. However, it was noticed that some aspects of *correctness* do measure the level of documentation in the system. In McCall’s model, correctness is composed of:

- *Completeness* - The degree to which required functional requirements have been successfully achieved.
- *Consistency* - The use of uniform design and documentation techniques throughout the software development protocol.



- *Traceability* - The functional independence of program components.

For documentation purposes, *completeness* is not relevant and so can be ignored. *Consistency* is an assessment of system design and documentation, which can be measured using a semantic differential scale. Example questions used to assess consistency are:

- Are the same diagrammatical forms used?
- Are the diagrammatical forms used consistently?

The *traceability* aspect looks at tracing specific component(s), from code to requirements going through all stages of analysis and design and can be measured using a semantic differential scale. This means that if *completeness* is removed from *correctness*, which is then renamed *comprehensibility*, it can then be added to the model.

The external quality factor *maintainability* was not included in the final list of characteristics. Although this was considered at an architectural level, the code level was not thought to be important at the time of creating the model. However, this is now thought to be important as making a change to the system could also involve changes to the system code as shown in the agility test cases that involves changing a service. It is thought that maintainability should be added as part of the agility quality model, as this is another indicator of the length of time required to reengineer as system.

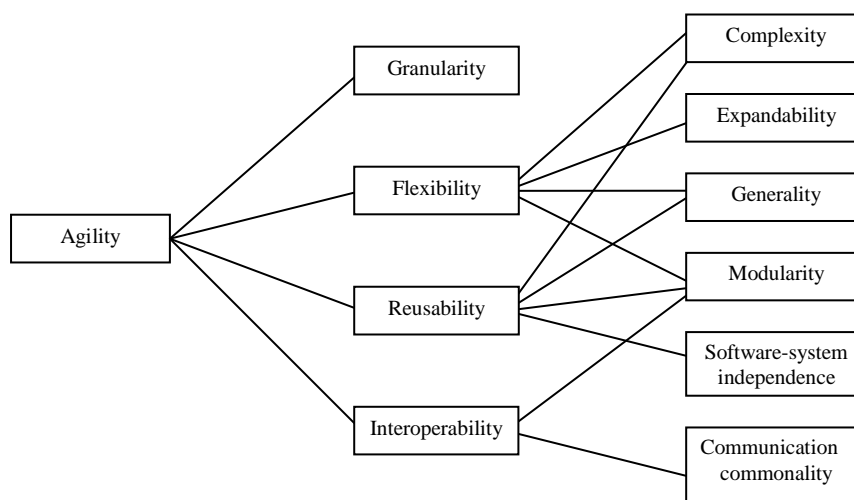
It was found that the agility model needed improvement as there were some external quality factors that needed to be added as well as expanded. The *granularity* external quality factor could be expanded as the original definition was too vague. This was investigated further and four types of granularity were found (Erl, 2007):

- *Service granularity* – the quantity of potential logic contained in a service which is broad in functional context.
- *Capability granularity* – the functional scope of a specific capability e.g. retrieving a document header is more fine-grained than retrieving the entire document.
- *Data granularity* – how big are the messages being sent? For example, retrieving a delivery address for an invoice is more fine-grained than retrieving the entire invoice.
- *Constraint granularity* – what is the level of detail of the constraints surrounding the data? For example, a document with many validation constraints is said to be more fine-grained than a document with little or no constraints.

*Constraint granularity* would not be included in the model as it was decided that this was too specific to service-oriented systems and would be hard to measure in systems that do not use XML schemas. However, it could be included if two service-oriented systems were being compared. *Service granularity* and *capability granularity* could be measured using a semantic differential scale. *Data granularity* could also be measured, using a semantic differential measurement, examining the potential message sizes that could be retrieved and the potential technologies that could be used.

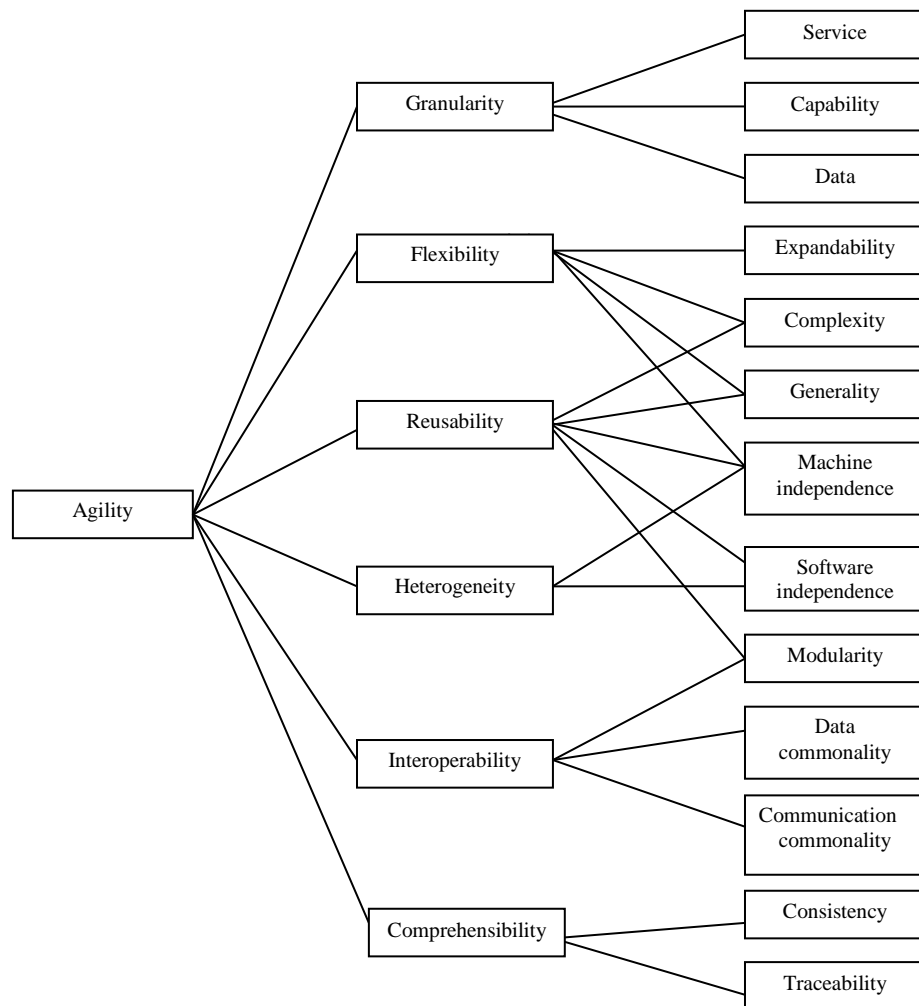
A potential further issue with the agility quality model was that *two product quality criteria* were not used. These were *data commonality (from interoperability)* and *machine independence (from reusability)*. These criteria were not included as they had been addressed by the IBHIS project. However, this does not mean that they should be excluded from the agility model.

Finally, upon further investigation, it was found that when conducting the study, that the interoperability external quality factor had been misinterpreted. The description from Krafzig et al. (2004) is ‘*Decoupling of functionality and technology – architecture must tolerate heterogeneity and change to its technical infrastructure. Business functionality must be decoupled from underlying technology*’. This was interpreted to mean *interoperability*. However, interoperability is still an important factor when measuring agility and therefore should be included in the agility quality model. Decoupling of functionality and technology has been re-labelled *heterogeneity* and added to the model. The product quality criteria for heterogeneity are software independence and hardware independence. Figure 37 shows the original model.



**Figure 37 Original agility quality model**

Figure 38 shows the revised agility quality model.



**Figure 38 Revised agility quality model**

Figure 38 shows the revised agility quality model which includes the additional aspects of quality that have been identified.

## **Chapter 8: Evaluation**

This chapter presents an evaluation of the research undertaken. The first section focuses on the SLR methodology. The second section focuses on the case study and the third section focuses on the data collection methods.

### **8.1 Systematic Literature Review**

A protocol and report were created as part of the SLR. This meant that it could be reviewed by supervisors and presented at workshops and conferences for peer review. The main strength of this SLR was that the authors of papers were contacted for further information on the same topic, if a paper was thought to be highly relevant. This improved the quality and quantity of evidence found by increasing the number of relevant papers.

Fifteen different service-oriented reengineering methods were found by the SLR. The analysis of these papers meant that a thorough understanding of service-oriented reengineering was acquired and that there was enough detail to help to create the McARM method. Only two methods specifically for MCA reengineering, were found by the SLR, which highlighted a gap in the research. Seven architectures for MCA were found. These were compared and contrasted in order to create McArc for the study. Finally, a clear set of challenges that could be encountered when reengineering a system as an SOA emerged from the evidence, which can be used to guide future practice and research.

A weakness with the SLR was related to the use of search engines. There were papers found in the pilot search on the Google search engine that did not appear in the main search. This was due to the dynamic nature of the web and search engines in particular, and highlights that the literature found by an SLR is a snapshot in time. The human element of the SLR could also be seen as weakness. Manual activities such as extracting data and deciding on papers could have error or bias. In this study, data extraction was validated by the supervising team in order to address this problem. Another weakness of the review was that the studies were not assessed for quality due to time constraints. Finally, only four data sources were used, it is possible that increasing this number may have increased the amount of included papers.

The weaknesses of the evidence from the review can be related to the research questions. For the first research question it was found that there was not a clear boundary between reengineering an existing system as an SOA and one that exposes an existing system using service interfaces. This made it difficult to decide whether or not a reengineering method should be included in the study. For the second research question, there were papers found which were not using a service-oriented approach and, therefore, excluded from the review. However, it is possible that only one of these approaches may have been a more suitable approach to the problem of reengineering the IBHIS broker for MCA. For the final research question, which looked at the types of issue found when reengineering a system as an SOA, the issue categories were decided upon by a novice researcher. These may have been misunderstood or could have been grouped more appropriately.

## **8.2 Case study**

In this section the case study is evaluated against a case study checklist and the validity of the case study is also evaluated.

### **8.2.1 Checklist**

The case study was evaluated against a checklist for software engineering case study assessment proposed by Höst & Runeson (2007). The aspects that were assessed were:

- Case study design
- Preparation for data collection
- Collecting evidence
- Analysis of collected data
- Reporting

#### **8.2.1.1 Case study design**

All of the items on the checklist were sufficiently covered except the definition of the case (see Table 49).

Question	Response
Is a clear purpose/objective/research question/hypothesis/proposition defined upfront?	Yes
Is the theoretical basis - relation to existing literature and other cases - defined?	Yes
Are the authors' intentions with the research made clear?	Yes
Is the case adequately defined (size, domain, process...)?	Yes – not thorough
Is a cause-effect relation under study? If yes, is the cause distinguished from other factors?	Yes
Will data be collected from multiple sources? Using multiple methods?	Yes
Is there a rationale behind the selection of roles, artefacts, viewpoints, etc.?	Yes
Are the case study settings relevant to validly address for the research question?	Yes
Is the integrity of individuals/organizations taken into account?	n/a

Table 49 Case study design evaluation

A clearer definition could have been given regarding the case definition. The integrity of the roles of the individuals was not taken into account as there were no external individuals or organisations used.

### 8.2.1.2 Preparation for data collection

All of the items on the checklist were sufficiently covered (see Table 50).

Question	Response
Is a protocol for data collection and analysis derived (what, why, how)?	Yes
Are multiple data sources and collection methods planned?	Yes
For quantitative data, are the measurements well defined?	Yes
Are the planned methods and measurements sufficient to fulfil the objective of the study?	Yes
Is the study design approved by a review board, and has informed consent obtained from individuals and organizations?	n/a

Table 50 Preparation for data collection evaluation



### 8.2.1.3 Collecting evidence

The only problem found was that the evidence had not been collected entirely as stated by the protocol (see Table 51).

Question	Response
Are data collected according to the protocol?	No a mock-up was made for RQ1(b) and RQ3
Is the observed phenomenon correctly implemented (e.g. to what extent is a design method under study actually used)?	-
Are data recorded to enable further analysis?	Yes
Are sensitive results identified (for individuals, organization or project)?	n/a
Are the data collection procedures well traceable?	Yes
Do the collected data provide ability to address the research question?	Yes

**Table 51 Collecting evidence evaluation**

Data was not collected exactly as outlined in the protocol due to problems with the original IBHIS broker code. This meant that it was necessary to create mock-up systems in order to undertake the performance testing and the agility test cases.

### 8.2.1.4 Analysis of the evidence

All of the items on the checklist were covered (see Table 52).

Question	Response
Is the analysis methodology defined, including roles and review procedures?	Yes
Is a chain of evidence shown with traceable inferences from data to research questions and existing theory?	Yes but not clear
Are alternative perspectives and explanations used in the analysis?	Yes
Is a cause-effect relation under study? If yes, is the cause distinguished from other factors?	Yes
Are there clear conclusions from the analysis, including recommendations for practice/further research?	Yes
Are threats to validity addressed in a systematic way?	Yes

Table 52 Analysis of the evidence evaluation

The only problem highlighted was the *chain of evidence* which could have been described clearer and in more detail.

### 8.2.1.5 Reporting

There were no problems found (see Table 53).

Question	Response
Are the case and its context adequately reported?	Yes
Are the research questions and corresponding answers reported?	Yes
Are related theory, hypotheses and propositions clearly reported?	Yes
Are the data collection procedures presented, with relevant motivation?	Yes
Are sufficient raw data presented?	Yes
Are the analysis procedures clearly reported?	Yes
Are ethical issues reported openly (personal intentions, integrity issues)?	n/a
Does the report contain conclusions, implications for practice and future research?	Yes
Does the report give a realistic and credible impression?	-
Is the report suitable for its audience, easy to read and well structured?	-

Table 53 Reporting evaluation

### **8.2.2 Validity**

The following section examines the validity of the case study (Yin, 2003). The types of validity are (Coolican, 1999):

- Construct validity
- Internal validity
- External validity
- Reliability

These will be explained further in the following sections.

#### **8.2.2.1 Construct validity**

Construct validity shows that the correct operational measures were planned for the concepts being studied. For this study there was a chain of evidence which had been outlined. This chain of evidence stated the aspects being measured and at which stage they will be measured. A protocol was created which was peer reviewed at two workshops (Jefferies et al., 2009).

The rest of this section outlines the potential threats to construct validity and describes ways in which they were avoided in the study (Coolican, 1999).

##### **8.2.2.1.1 Construct measures**

This is the extent to which measures actually relate to the concept. For this study, the main threat was that the metric models may not have been appropriate measures of

agility. There was a thorough review of literature and this work was also discussed at workshops and submitted as part of a conference proceeding (Jefferies et al., 2009).

#### **8.2.2.1.2 Mono-method bias**

Construct validity was maintained by taking different measures for the same concept. This is also known as triangulation, Höst & Runeson (2007). For the agility related research questions there were three different sources; code metrics, semantic differential measurements and the agility test cases.

#### **8.2.2.1.3 Levels of the independent variable**

The levels used for the independent variables may not have been sufficient for the size of the messages for the performance investigations. **If the size of the messages were not significantly different enough then the likelihood of there being a significant difference in response time is reduced.** For this study, the message sizes were measured in three sizes - small, medium and large. The small message size represents a query using a single field from the ontology, the medium size represents a multiple fields (half) query and the large query field represents a query of all fields. Although these may be representative of the range of message sizes for the study, they may not be representative of all possible SOAP messages.

#### **8.2.2.2 Internal validity**

The internal validity is threatened by the ways in which the results may have been caused by other factors. **Tables 54 examine the threats to internal validity of the performance measurements and describes the actions that were taken to control them.**

Rival explanation	Affect on result	Control Measures
Nesting of the XML documents could have an effect on the parsing time.	Deeper nesting requires more parsing which will reduce the response time.	Nesting of XML documents was kept to one level.
Using the internet to access web services could have an affect on performance.	Different days/times may have different speeds due to the number of users online.	Evaluation of the two bindings was performed on similar time/day to reduce the affect of internet speed differences.
The speed of a machine could affect the performance of a business process/web service call.	The speed of the machine will affect XML parsing, request initialisation etc.	The same machine was used for both bindings so that there was no difference between machine speeds.
Other programs on the machine could take away resources.	See above	Programs running on the machine were kept the same during testing.
The performance measurement tools are not accurate or reliable.	An inaccurate response time is recorded	This was not controlled but assumed to be accurate as they are part of the development environments used.

**Table 54 Threats and controls for performance measurements**

The performance measurements were controlled as much as possible but these could have been controlled further. For example, an experiment could have been conducted to determine if the level of nesting of the XML documents used in messaging made a difference in the response times. Another control that could have been improved was the speed of the network. This could have been improved by using a private network instead of a public network. However, this would reduce the ecological validity. **Table 55 examine the threats to internal validity of the agility metrics and evaluation of the reengineered system and describes the actions that were taken to control them.**

Threat to validity	Affect on result	Control Measures
The metric tool may not calculate the metrics correctly.	Incorrect metrics	The metrics measurement tool was not validated as this would have been too time consuming. However this is a popular open source project which has an active community surrounding it.
A novice researcher conducted the rating of some of the product quality criteria for the agility quality model.	The rating could contain inaccuracy or bias.	The semantic differential measurements were validated by a member of the supervising team. Although this person was responsible for coding the original IBHIS broker it was decided that it would be too much of a task to ask someone to understand the systems due to their size.
Practice effects - During the learning/prototyping phase of the study the researcher's understanding of the different technologies would have improved.	The researcher may have become more familiar with one of the technologies than the other. This would result in the tasks being performed quicker than if they had not worked with either of the technologies previously	It was not possible to control this, but, it was kept in mind when making conclusions about the technologies.

Table 55 Threats and controls for agility measurements

### 8.2.2.3 External validity

This section looks at the external validity of the study and how the effects can be generalised outside of the study (Coolican, 1999).

#### 8.2.2.3.1 CS1-RQ1

In terms of the *generalisation to the population*, the performance/agility trade-off is not atypical, and there are many other cases where the use of NLC and PBM could be compared, such as in chip design and network design. In terms of *ecological validity*, the reengineering scenario is not an atypical case as there are many systems that could benefit from being reengineered for MCA.

#### **8.2.2.3.2 CS1-RQ2**

In terms of the *generalisation to the population*, the addition of new layers in a system impacting on the performance is not atypical. In terms of *ecological validity*, the reengineering scenario is not a typical case as not all organisations will wish to expose their systems to multiple channels.

#### **8.2.2.3.3 CS1-RQ3**

In terms of *generalisation to the population*, although the term ‘agility’ is not often applied to systems that are not SOA based, they are discussed in similar terms such as flexibility which suggests that the results are generalisable. In terms of *ecological validity*, the reengineering scenario is not atypical as reengineering a system as an SOA is becoming increasingly more common in the IT industry.

#### **8.2.2.3.4 CS1-RQ4**

In terms of the *generalisation to the population*, not all systems are going to need to be reengineered for MCA. However, the results are applicable to any type of system that will be reengineered using service-based technologies. In terms of *ecological validity*, for systems that do need to be reengineered for MCA, a reengineering method will be needed.

#### **8.2.2.4 Reliability**

The case study had a protocol, which means that it is auditable increasing its reliability. The case study protocol was also validated in the form of peer review.

### 8.3 Data collection methods

This section evaluates the data collection methods in terms of their strengths and weaknesses. The data collection methods are:

- Response times
- Code metrics
- Semantic differential scale
- Agility test case data
- McARM evaluation

#### 8.3.1 Response times

A potential weakness with the response time data collection was the use of different tools used to measure the response times at different points in the system. The different tools may differ in how they take measurements which could have an affect on the results. **For example, when measuring a response time, some pre or post-processing may be included by one of the servers, but not included in the other server.**

For the original IBHIS broker measurements were taken from a browser within IBM Websphere application developer. For the reengineered IBHIS broker, measurements were taken from the business processes server (Oracle BPEL Process manager) and the measurements for the mobile client were taken from the Sun Wireless Toolkit 2.5.2. Control measures were put in place to minimise bias where possible (see Table 54).



### 8.3.2 Code metrics

The code metrics were gathered using the JHawk software. The algorithms in the tool could be incorrect or the tool could have bugs which would mean that the metrics are incorrect. However, as JHawk is a popular open source tool with a large community, the tool itself was not validated. The baselines used for the metrics are rules of thumb recommended by Lorenz (Kan, 2003) and were used as a guide only, as the two systems were being compared rather than being examined for values that may cause concern.

### 8.3.3 Semantic differential measurements

Using semantic differential measurements allowed the systems to be assessed on aspects that could not be measured by using code metrics, such as granularity. In order to reduce error, a member of the supervising team, who was familiar with the system, validated the results (See appendix H and I). There was mainly agreement, apart from some cases, where the reviewer had more knowledge of the system. For example, the reviewer knew that not all of the requirements were satisfied in the original system. The first reviewer had assumed that they were all satisfied as the project had been completed.

The main weakness of the semantic differential measurements is the fact that they are error prone and subject to bias (Coolican, 2003). For example the first reviewer had created the method and would therefore be biased in its favour. The validation reviewer may have been biased in that he would want to score the system favourably as he had worked on the original IBHIS broker. The validation reviewer was also the supervisor of the researcher which meant that he may have been biased toward

helping the researcher. However, the reviewer is an experienced researcher and there is no reason to believe that he did not perform a fair, unbiased review. An external validation reviewer could have been used but the time needed to perform the program comprehension would have been too long for a system of this size and complexity. There are both advantages and disadvantages for this decision. An external validation reviewer would have found it more difficult to interpret the code than the validation reviewer who had worked on the project previously. Also, an external validation reviewer would not know why certain decisions were made regarding the design/coding of the system and the effect that any changes may have.

#### **8.3.4 Agility test cases**

The agility test cases made it possible to measure the time and effort to make changes to the software. The agility test cases were derived from the focus of the investigation (agility of NLC and PBM). However, in a real system, the agility test cases would come from anticipated changes in business requirements. There are a large number of potential agility test cases, which means that there could be bias in the ones that were chosen. This could mean that the system is agile in the aspects tested but not in aspects that were untested. Another potential weakness of the agility test case data was the lack of assessment of the program comprehension. When a system is reengineered, the person carrying out the reengineering is not always the person that built the system; therefore there would be an impact on the time taken to make any changes to the system. For the purpose of this experiment it was assumed that this was the same person as prototyping of the WSIF technology was carried out in order to make sure that the technology could be used in the case study. This meant that there was already an understanding of what the system would do and how the technology

worked. This means that the results are less applicable when the person reengineering the system was not involved in creating the original system.

### **8.3.5 McARM evaluation**

The main problems are with the participant rating the result and their opinion of the output of the stage. As the study participant was also the study observer, there was the potential for bias. The participant was also the creator of the McARM method which could introduce bias toward the method.

## **Chapter 9: Conclusions and future work**

This chapter concludes the thesis and makes recommendation for future work.

### **9.1 Conclusions**

This section draws some conclusions about each research question, McArc, the agility quality model and the research methods used.

#### **9.1.1 CS-RQ1**

It is concluded in this case, that NLC should not be used instead of PBM for messaging between the services and business processes to improve the performance of a system reengineered for MCA. NLC should be used if the message size is large and interoperability is not important. For example, NLC would be beneficial if the services are going to be accessed by internal business processes, the data sets are large and the number of simultaneous invocations is high.

One of the main limitations of this part of the study was the fact that it was necessary to use mock up versions of the reengineered IBHIS broker instead of the broker itself. It was necessary to use mock ups of the reengineered IBHIS broker - due to the fact that the WSIF bindings could not be accessed by business processes outside of the server in which they are running. The mock up brokers matched the reengineered IBHIS broker query service functionality exactly, but there were some differences in

how the WSIF and SOAP services were created in Oracle BPEL Process Manager compared to IBM Websphere Application Developer.

A limitation for CS-RQ1-P1 is that the range of the message sizes used in this part of the study were only from the possible range from the reengineered IBHIS broker. However, these may not have been representative of the complete range of possible sizes when using services. Another limitation is that load was not taken into account. With a live server, multiple calls to a service will be made from numerous requesters. However, this work was concerned only with single response times only.

### **9.1.2 CS-RQ2**

It was found that with each additional layer required for MCA there was a decrease in system performance. This shows that the fewer layers there are for a multi-channel system, the quicker the response times. When a system is going to be accessed by devices with limited resources, a good understanding of the technologies used and user requirements is needed, to ensure that any additional layers employed in the architecture are justified. It was also found that message size did not decrease the system performance significantly. This shows that reducing the layers in architecture is more important than reducing the sizes of the messages sent.

A potential limitation with this part of the study is server load. This was not considered in this study, as the focus was on individual response times. However, load would be an important factor in a live system if there is a large volume of traffic. A further possible limitation is that it was not possible to perform the testing using a physical (mobile) device due to a problem with the messages returned from the Oracle

BPEL Process Manager. Therefore, it was necessary to use an emulator of a physical device and it is possible that this may have performed differently to a physical device, thus giving different results.

### **9.1.3 CS-RQ3**

It was found that reengineering a system as an SOA can help to improve the agility of a system. However, an SOA is not a silver bullet and there are cases where an SOA may not be an appropriate solution (Section 4.1).

A potential limitation with this research question was that only the system code was examined as no agility test cases were used. The reason for not having agility test cases for this part of the investigation was that only service-based systems were considered when the case study was planned. This meant none of the test cases could be applied to a non service oriented system.

### **9.1.4 CS-RQ4**

McARM is a comprehensive UML based reengineering method that can be used to reengineer a system for MCA. The case study provided an example of its use with a medium sized system. Compared to the methods found by the SLR for reengineering a system as a service-oriented architecture (including the two specifically for MCA), McARM is more comprehensive in terms of coverage. The four most comprehensive of the methods found only covered five of the stages required for reengineering a system for MCA (Zhang et al. 2005; Zhang & Yang 2004; Zimmermann et al. 2005;

Comerio et al., 2004). McARM covered eight of the stages required for reengineering for MCA.

### **9.1.5 Multi-channel Architecture**

For the study, a simple architectural layering was proposed (McArc) in order to expose the original IBHIS broker for multi-channel access. McArc was created by evaluating a number of potential multi-channel architectures. The implementation meant that the system was accessible from any device that supports web services, as well as business processes. This architecture was also loosely coupled at the service and business process level, which meant that it was agile as it could be changed easily.

### **9.1.6 Agility quality model**

This work proposed a method for assessing agility, which comprises an agility quality model and a set of agility test cases. The agility quality model can be used to assess the code of a system in order to state its level of agility. The test cases allowed the time and effort required to make potential changes to be assessed. In the case study, conducted as part of this work, the agility model was used to compare different systems and bindings.

One of the problems with the agility quality model was the difficulty in deciding how to define certain metrics. For example, when measuring lines of code, it was a difficult decision whether or not to use the statements only or to also include comments, lines with braces and empty lines. Only the *number of statements* was used

in agility quality model as it was decided that other aspects, such as empty lines, did not add to the complexity of the code - only how it looked.

#### **9.1.7 Research methodologies**

The SLR method was used for gathering information about service-oriented reengineering methods and issues as well as multi-channel access. The SLR method was found to be a useful and effective method from the point of view of a novice researcher. A useful structure was given to the researcher at the early stages of research, which was also found to be helpful later on. When creating a protocol for the SLR, there were many examples, which were found to be very helpful. It was found however that the SLR was merely a snapshot in time and did not lend itself well to rapidly changing areas of research.

The case study methodology allowed the research questions to be investigated in the way best suited to having multiple forms of measurement. The study used multiple sources to explore the performance and agility and the trade-off for systems reengineered for MCA. However, some aspects of conducting the case study were found to be difficult. For example, when planning the case study, there were not many guidelines and examples of deliverables such as a case study protocol. It was also found that when using multiple data sources and having multiple research questions, if a change was made to an aspect of the study or protocol, it is not always clear what impact the change would have on the rest of the case study.



## **9.2 Future work**

This section proposes future work concerning each research question, the multi-channel architecture and the agility quality model.

### **9.2.1 CS-RQ1**

For improving the speed between business processes and services, further work could look at other technologies such as the REST protocol. For the comparison of the agility of NLC and PBM, future work would involve testing on a different development environment, for example, a more recent version of IBM Websphere or Oracle BPEL Process Manager. The complexity metrics used were based on well known complexity measures. However, it is possible that there are other metrics that could have been included in the measure e.g. maintainability index and source quality that may have also been relevant. Future work is needed to investigate this further. For the agility test cases, future work could investigate a set of guidelines for creating agility test cases.

### **9.2.2 CS-RQ2**

Further work could be conducted to investigate the response times when accessing the business processes from a physical device. Due to the ‘empty header’ problem (See section 5.2.4.8), it was not possible to test the system on a physical device. Therefore, further investigation is needed into solving this problem so that business processes can be accessed by mobile devices.

### **9.2.3 CS-RQ3**

Future work could look at deriving a set of test cases for testing the agility of non service-based systems. Other future work could look at applying the agility testing methods to other reengineering projects for validation purposes or with a view to extending the method.

### **9.2.4 CS-RQ4**

Potential future work for McARM may involve investigating a potential generic method for the recoding stage of reengineering a system toward an SOA. An area that also requires further work is the requirements stage of the method. This could be detailed further with an appropriate structured method. Also required is provision for creating the clients that will access the system. This could be a stage on its own or it could be included in the recoding stage. The McARM method also needs to be independently tested in other cases and expert reviewed to ensure that it has been created in a credible manner.

The system that was used was chosen due to its availability; however it would have been useful to have a publicly available system that could be used as a benchmark for SO reengineering methods and experiments. One suggestion would be an open source system chosen by a number of experts in this area.

### **9.2.5 Multi-channel Architecture**

It was intended that this architecture could be applied to any system that requires reengineering for MCA. In order to investigate this, the architecture would need to be applied to other systems to see if it can be applied outside of this study.

### **9.2.6 Agility quality model**

The revised model will need to be tested further by using the model in another case study or by validating the model, for example by expert review. Also, the extra aspects added to the quality model need to be investigated further. Future work may also look at creating baselines for the metrics so that that level of system agility can be assessed in isolation.

## References

Adacal, M. & Benner, A. B. (2006). Mobile Web Services: A New Agent-Based Framework. *IEEE Internet Computing* 10(3) (May. 2006). pp 58-65.

Allen, P. (2006). Service-orientation – Winning Strategies and Best Practices. Cambridge, UK: Cambridge University Press. ISBN: 978-0-521-84336-2.

Alvaro A., Lucredio D., Garcia V., Prado A., Trevelin L., Almeida E. (2003). Orion-RE: A Component-Based Software Reengineering Environment. 10th Working Conference on Reverse Engineering (WCRE 2003). pp.248.

Anand, S., Chatterjee, A. M., Kumar, V., Raut, V. & Singh, V. (2005). Towards Legacy Enablement Using SOA and Web Services: Leverage legacy systems with SOA. [Online]

<<http://webservices.sys-con.com/read/164558.htm>> Accessed on 09.05.2007.

Arnold R. S. (1993). Software reengineering, IEEE Computer Society Press.

Arteta B. M. & Giachetti R. E. (2004). A measure of agility as the complexity of the enterprise system. In *Robotics and Computer-Integrated Manufacturing*, 20, 495-503.

Basilli V.R, Gianlugi. C and Rombach H. D. (1996). The Goal Question Metric Approach. [Online]

<<http://www.wagse.informatik.uni-kl.de/pubs/repository/basili94b/encyclo.gqm.pdf>> Accessed on 07.11.2010

Beck K. (2000). eXtreme Programming. AddisonWesley, Boston.

Berzins V, Shing M, Luqi, Saluto M & Williams J. (2000). Object-Oriented Modular Architecture for Ground Combat Simulation", *Proceedings of the 2000 Command and Control Research and Technology Symposium*, Naval Postgraduate School, Monterey, CA, June 26-28.

Bergey, J.; Smith, D.; Weideman, N.; & Woods, S.N. (1999) Options Analysis for Reengineering (OAR): Issues and Conceptual Approach. Technical Report (CMU/SEI-99-TN-014). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

Bertini, E. & Santucci, G. (2004). Modelling internet based applications for designing multi-device adaptive interfaces. In *Proceedings of the Working Conference on Advanced Visual interfaces* (May 25 - 28, 2004). Gallipoli, Italy. ACM, New York, NY. pp 252-256.

Bianco, P., Kotermanski, R. & Merson, P. (2007). Evaluating a Service-Oriented Architecture. Software Architecture Technology Initiative Technical report. CMU/SEI-2007-TR-015, ESC-TR-2007-015.

Bisdikian, C., Christensen, J., Davis, J., Ebling, M. R., Hunt, G., Jerome, W., Lei, H., Maes, S. & Sow, D. (2001). Enabling location-based applications. In *Proceedings of the 1st international Workshop on Mobile Commerce*, Rome, Italy. WMC '01. ACM, New York, NY. pp 38-42.

Blanvalet, S., Bolie, J., Cardella, M., Carey, S., Chandran, P., Coene, Y., Gaur, H., Geminiuc, K., Juric, M. & Zirn, M. (2006). BPEL Cookbook: Best Practices for SOA-based integration and composite applications development. Packt Publishing. ISBN: 978-1904811336.

Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G. & Merritt, M. (1978). Characteristics of Software Quality. North Holland Publishing, Amsterdam, The Netherlands.

Brereton, O.P., Gold, N.E., Budgen, D., Bennett, K.H. & Mehandjiev, N.D. (2006). Service-based systems: a systematic literature review of issues. Computer Science Technical Report, Keele University (TR/06-01).

Brereton, P., Kitchenham, B., Budgen, D. & Li, Z. (2008). Using a Protocol Template for Case Study Planning. *In Proceedings of Evaluation and Assessment in Software Engineering (EASE)*, (2008), Bari, Italy.

Brown, K. & Reinitz, R. (2003). Web Services Architectures and Best Practices. [Online] IBM. Available at [http://www.ibm.com/developerworks/websphere/techjournal/0310\\_brown/brown.html](http://www.ibm.com/developerworks/websphere/techjournal/0310_brown/brown.html) > Accessed on 27.07.2009.

Budgen, D. (2003) Software Design, 2<sup>nd</sup> Ed. Pearson Addison Wesley. ISBN 978-0201722192

Budgen, D., Rigby, M., Brereton, P. & Turner, M. (2007). A Data Integration Broker for Healthcare Systems. *Computer* 40, 4 (Apr. 2007). pp 34-41.

Chen, F., Li, S. & Yang, H. (2005) Feature Analysis for Service-Oriented Reengineering. In *the 12th IEEE Asia-Pacific Software Engineering Conference (APSEC)*, (Dec. 2005), Taipei.

Chen, S., Bao, L. & Chen, P. (2008). OptBPEL: A Tool for Performance Optimization of BPEL Process. *Software Composition 2008*. pp 141-148.

Chu W., Lu C., Shiu C., & He X. (2000). Pattern-based software reengineering: a case study. *Journal of Software Maintenance* 12, 2 (March 2000), pp. 121-141.

Chung, S., An, J. & Davalos, S. (2007). Service-Oriented Software Reengineering: SoSR. In *Proceedings of 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. pp.172c.

Cohen, F. (2001). Myths and misunderstandings surrounding SOAP. [Online] <<http://www.ibm.com/developerworks/library/ws-spmyths.html>> Accessed on 29.01.2011

Comerio, M., De Paoli, F., Grega, S., Batini, C., Di Francesco, C. & Di Pasquale, A. (2004). A service re-design methodology for multi-channel adaptation. In *Proceedings of the 2nd international Conference on Service Oriented*

*Computing*, (November 15 - 19, 2004), New York, USA. ICSOC '04. ACM, New York, NY. pp 11-20.

Coolican H. (1999). *Research Methods and Statistics in Psychology*, 3<sup>rd</sup> Ed. Hodder Arnold H&S. ISBN: 978-0340747605.

Dromey, R. G. (1995). A model for software product quality. In *IEEE Transactions on Software Engineering*, 21, pp 146-162.

Duftler, M. J., Mukhi, N. K., Slominski, A., Weerewarana S. & Watson T. (2001). Web services invocation framework (WSIF). [Online] IBM. Available at <http://www.research.ibm.com/people/b/bth/OOWS2001/duftler.pdf> > Accessed on 31.04.2008.

Eisenstein J., Vanderdonckt J. & Puerta A. (2001). Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, (January 14 - 17, 2001), Santa Fe, New Mexico, United States. IUI '01. ACM, New York, NY. pp 69-76.

Erl, T. (2004). *Service-Oriented Architecture – A field guide to integrating XML and Web Services*. Upper Saddle River: Prentice Hall PTR. ISBN 0-13-142898-5.

Erl, T. (2007) *SOA Principles of Service Design*. Prentice Hall. ISBN-10: 0132344823.



Ganesh, J., Padmabhuni S. & Moitra D. (2004). Web Services and Multi-Channel Integration: A Proposed Framework. In *Proceedings of the IEEE International Conference on Web Services*, (June 06 - 09, 2004). ICWS. IEEE Computer Society, Washington, DC. pp 70.

Hackmann, G., Haitjema, M., Gill, C. & Roman, G.-C. (2006). Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In *Proceedings of 4th International Conference on Service Oriented Computing (ICSOC 2006)*. Springer Verlag. pp 503-508.

Hasselbring, W., Reussner, R., Jaekel, H., Schlegelmilch, J., Teschke, T. & Krieghoff, S. (2004). The Dublo Architecture Pattern for Smooth Migration of Business Information Systems: An Experience Report. In *Proceedings of the 26th international Conference on Software Engineering*, (May 23 - 28, 2004). International Conference on Software Engineering. IEEE Computer Society, Washington, DC. pp 117-126.

Hohpe G. & Woolf, B. (2003). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. ISBN: 0321200683

Höst, M. & Runeson, P. (2007) Checklists for Software Engineering Case Study Research, in *Proceedings of 1<sup>st</sup> International Symposium on Empirical Software Engineering & Measurement (ESEM)*. IEEE Computer Society Press. pp 479-482.

Hunold S., Korch M., Krellner B., Rauber T., Reichel T., and Runger G. (2008). Transformation of Legacy Software into Client/Server Applications through Pattern-

Based Rearchitcturing. In *Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC '08)*. IEEE Computer Society, Washington, DC, USA, pp 303-310.

International Organization for Standardization (1991). ISO/IEC 9126: Software Product Evaluation - Quality Characteristics and Guidelines for their Use. Geneva, Switzerland. Addison-Wesley Professional.

Jacobson I. & Lindstrom F. (1991). Reengineering of old systems to an object-oriented architecture. In *Conference proceedings on Object-oriented programming systems, languages, and applications (OOPSLA '91)*. ACM, New York, NY, USA, pp. 340-350.

Jalote P. (2008). A Concise Introduction to Software Engineering (1 ed.). Springer Publishing Company, Incorporated.

Jarzabe S. (1993) . Software reengineering for usability. In *Computer Software and Applications Conference, 1993. COMPSAC 93. Proceedings., Seventeenth Annual International* . Phoenix, AZ , USA pp 100-106.

Jefferies, C., Brereton, P., & Turner, M. (2008). A Systematic Literature Review of Approaches to Reengineering for Multi-Channel Access. In *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, (April 01 - 04, 2008). CSMR. IEEE Computer Society, Washington, DC. pp 258-262.

Jefferies, C., Brereton, O.P. & Turner, M. (2009). A Comparison of Binding Technologies for Multi-Channel Access. In *Proceeding of the 2009 Conference on Techniques and Applications For Mobile Commerce: Proceedings of Tamoco*. Frontiers in Artificial Intelligence and Applications, vol. 201. IOS Press, Amsterdam, The Netherlands. pp 149-154.

Jong, I. de (2002). Web Services/SOAP and CORBA. [Online] <[http://www.omg.org/news/whitepapers/CORBA\\_vs\\_SOAP1.pdf](http://www.omg.org/news/whitepapers/CORBA_vs_SOAP1.pdf)> Accessed on 29.01.2011

Kan, S. (2003). Metrics and Models in Software Quality Management, 2<sup>nd</sup> Ed. Pearson. ISBN 0-201-72915-6.

Kapoor, R.V. & Stroulia, E. (2001). Mathaino: Simultaneous legacy interface migration to multiple platforms. In *Proceedings of the 9th International Conference on Human-Computer Interaction*, New Orleans, LA, USA, vol. 1. pp 51-55.

Kitchenham, B. (2004) Procedures for Performing Systematic Reviews. Joint Technical Report Software Engineering Group, Keele University (TR/SE-0401), United Kingdom and Empirical Software Engineering, National ICT Australia Ltd, Australia (0400011T.1).

Kitchenham, B. & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Keele University and Durham University Joint Report. Tech. Rep. EBSE 2007-001.

Kohlhoff, C. & Steele, R. (2003). Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems. In *Proceedings of WWW'03*, (2003), Budapest, Hungary,

Kotsiopoulos, I., Keane, J., Turner, M., Layzell, P. & Zhu, F. (2003). IBHIS: Integration Broker for Heterogeneous Information Sources. In *Proceedings of the 27th Annual international Conference on Computer Software and Applications*, (November 03 - 06, 2003). COMPSAC. IEEE Computer Society, Washington, DC. pp 378.

Koutsoukos, G., Andrade, L., Gouveia, J. & El-Ramy, M. (2006). Service Extraction. [Online] Sensoria. Available at  
 <[http://www.pst.ifi.lmu.de/projekte/Sensoria/del\\_12/D6.2.a.pdf](http://www.pst.ifi.lmu.de/projekte/Sensoria/del_12/D6.2.a.pdf)> Accessed on  
 08.05.2007.

Krafzig, D., Banke, K. & Slama, D. (2004). Enterprise SOA Service Oriented Architecture Best Practices. Upper Saddle River: Prentice Hall PTR. ISBN 0-13-146575-9.

Kreger, H & Estefan, J. (2009). Navigating the SOA Open Standards Landscape Around Architecture. [Online] OASIS. Available at: <[http://www.oasis-open.org/committees/download.php/32911/wp\\_soa\\_harmonize\\_d1.pdf](http://www.oasis-open.org/committees/download.php/32911/wp_soa_harmonize_d1.pdf)>. Accessed on 25.01.2010.

Lai, K. Y., Phan, T. K., & Tari, Z. (2005). Efficient SOAP Binding for Mobile Web Services. In *Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary*, (November 15 - 17, 2005). LCN. IEEE Computer Society, Washington, DC. pp 218-225.

Lewis, G., Morris, E., O'Brien, L., Smith, D. & Wrage, L. (2005) Smart: The service-oriented migration and reuse technique. Technical Report CMU/SEI-2005-TN-029, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Sep 2005.

Liem, I., Wahyudin, D. & Shatten, A. (2006). Data Integration: An Experience of Information System Migration. Available at: [http://cocoon.ifs.tuwien.ac.at/pub/iiwas/iiwas2006\\_2.pdf](http://cocoon.ifs.tuwien.ac.at/pub/iiwas/iiwas2006_2.pdf) Accessed on 19.02.2011

Lewis, G., Morris, E., Smith, D. & Simanta, S. (2008). SMART: Analyzing the Reuse Potential of Legacy Components in a Service-Oriented Architecture. Technical Report CMU/SEI-2008-TN-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, June 2008.

Lung C. (1998). Software architecture recovery and restructuring through clustering techniques. In *Proceedings of the third international workshop on Software architecture* (ISAW '98). ACM, New York, NY, USA, pp101-104.

Machado, A. C. & Ferraz, C. A. (2005). Guidelines for performance evaluation of web services. In *Proceedings of the 11th Brazilian Symposium on Multimedia and the*

Web, (December 05 - 07, 2005), Pocos de Caldas - Minas Gerais, Brazil. R. P. Fortes, Ed. WebMedia '05, vol. 125. ACM, New York, NY. pp 1-10.

Marchetti, C., Pernici, B. & Plebani, P. (2003). A Quality Model for e-Service Based Multi-Channel Adaptive Information Systems. *In Proceedings of Web Information Systems Engineering Workshops. WISEW'03.* pp. 165-172.

Mayhauser, A. V. & Vans A. M. (1995). Program Comprehension During Software Maintenance and Evolution. *IEEE Computer*, pp. 44-55, Aug.

McCall, J. A., Richards, P. K. & Walters, G. F. (1977). Factors in Software Quality, Volumes I, II, and III. US Rome Air Development Center Reports, US Department of Commerce, USA, 1977.

Menkhaus, G. (2002). An Architecture for Supporting Multi-Device, Client-Adaptive Services. *Ann. Softw. Eng.* 13(1-4), (Jun. 2002). pp 309-327.

Migliardi, M. & Podesta, R. (2004). Performance Improvement in Web Services Invocation Framework. *In Proceedings of 18th International Parallel and Distributed Processing Symposium (2004).* IPDPS'04. pp.110b.

Moller, A. & Schwartzbach, M. (2006). An Introduction to Xml and Web Technologies. Addison Wesley. ISBN: 978-0321269669.

Mukhi, N. (2001). We Service Invocation sans SOAP. [Online] IBM. Available at

<<http://www.ibm.com/developerworks/library/ws-wsif.html>> Accessed on 05.05.2008.

Mukhi, N. & Slominski, A. (2001). Web service invocation sans SOAP, Part 2: The architecture of Web Service Invocation Framework. [Online] IBM Available at <<http://www.ibm.com/developerworks/webservices/library/ws-wsif2/>> Accessed on 21.07.2009.

Neilson, J. (1993). Usability Engineering. Morgan Kaufmann. ISBN: 978-0125184069.

Newcomer, E. & Lomow, G. (2005). Understanding SOA with Web Services, Addison Wesley. ISBN 0-321-18086-0.

OASIS RM-CS (2006). Reference Model for Service Oriented Architecture 1.0. [Online] OASIS. Available at: <<http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf>>. Accessed on 15.02.2010.

Oh, S. & Fox, G. C. (2007). Optimizing Web Service messaging performance in mobile computing. *Future Gener. Comput. Syst.* 23(4), (May. 2007). pp 623-632.

Open Group (2009). SOA Reference Architecture. [Online] Open Group. Available at: <https://www.opengroup.org/projects/soa-ref-arch/uploads/40/19713/soa-ra-public-050609.pdf>. Accessed on 24.01.2010.

Park, G. C., Kim, S. S., Bae, G. T., Kim, Y. S., & Kang, B. H. (2006). An Automated WSDL Generation and Enhanced SOAP Message Processing System for Mobile Web Services. In *Proceedings of the Third international Conference on information Technology: New Generations* (April 10 - 12, 2006). ITNG. IEEE Computer Society, Washington, DC. pp 388-387.

Pautasso, C., Zimmermann, O. & Leymann, F. (2008). Restful web services vs. "big" web services: making the right architectural decision. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*. New York, NY, USA: ACM. pp 805-814.

Pellerin, R. (2007). The MoodS protocol: a J2ME object-oriented communication protocol. In *Proceedings of the 4th international Conference on Mobile Technology, Applications, and Systems and the 1st international Symposium on Computer Human interaction in Mobile Technology* (September 10 - 12, 2007). Singapore. Mobility '07. ACM, New York, NY. pp 8-15.

Pfleeger, S. L. (2001). *Software Engineering Theory and Practice*, 2<sup>nd</sup> Ed. Prentice Hall. ISBN: 978-0130290496.

Portier, B. (2007) SOA terminology overview, Part 3: Analysis and design. [Online] IBM. Available at  
<http://www.ibm.com/developerworks/webservices/library/ws-soa-term3/index.html>  
 Accessed on 29.08.2007.



Riaz Ahamed, S. S., (2010). An Integrated and Comprehensive Approach to Software Quality. *International Journal of Engineering Science and Technology*. 2(2). pp 59-66.

Rosenberg L. H. (1996). Software Re-engineering. Goddard Space Flight Center. NASA.

Roto, V. & Oulasvirta, A. (2005). Need for non-visual feedback with long response times in mobile HCI. In *Special interest Tracks and Posters of the 14th international Conference on World Wide Web* (May 10 - 14, 2005). Chiba, Japan. WWW '05. ACM, New York, NY. pp 775-781.

Runeson, P. & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14(2). pp 131-161.

Sahraoui, H. A., Godin, R., & Miceli, T. (2000). Can Metrics Help to Bridge the Gap Between the Improvement of OO Design Quality and Its Automation? In *Proceedings of the international Conference on Software Maintenance*, (October 11 - 14, 2000). ICSM. IEEE Computer Society, Washington, DC. pp 154.

Sandoz, P., Percias-Geersten, S., Kawaguchi, K., Hadley, M. & Pelegri-Llopart, E. (2003). Fast Web Services. [Online] Sun Microsystems. <<http://java.sun.com/developer/technicalArticles/WebServices/fastWS/>> Accessed on 26.06.2008.

Sneed H. (2005). An Incremental Approach to System Replacement and Integration. In *Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*(CSMR '05). IEEE Computer Society, Washington, DC, USA, pp196-206.

Sneed, H. (2006). Wrapping Legacy Software for Reuse in a SOA. In *Proceedings of Third GI-Workshop XML4BPM XML Integration and Transformation for Business Process Management*, (February 22, 2006), Passau, Germany.

Sneed, H. (2007). Migrating to Web Services A Research Framework. In *proceedings of CSMR 2007 11th European Conference on Software Maintenance and Reengineering "Software Evolution in Complex Software Intensive Systems"*, (March 21-23, 2007).

Sneed, H. & Sneed, S. (2003). Creating Web Services from Legacy Host Programs. In *Proceedings of 5th International Workshop on Web Site Evolution*, (2003). pp 59.

Sommerville I. & Sawyer P. (1997). Requirements Engineering: A Good Practice Guide. Wiley. ISBN 0471974447.

Strohmaier, M. & Lindstaedt, S. (2005). Beyond Flexible Information Systems: Why Business Agility Matters. In *Sixth Workshop on Business Process Modelling, Development and Support*, Porto, Portugal. BPMDS'05 during CAiSE'05.

Strohmaier, M. & Rollett, H. (2005). Future Research Challenges in Business Agility - Time, Control and Information Systems. In *Proceedings of the Seventh IEEE*

*international Conference on E-Commerce Technology Workshops* (July 19, 2005). CECW. IEEE Computer Society, Washington, DC. pp 109-1.

Tian M., Voigt T., Naumowicz T., Ritter H. & Schiller J. (2003). Performance Considerations for Mobile Web Services (2003). *Elsevier Computer Comm. J.* 27(11), (2004). pp. 1097–1105.

Tilley, S., Gerdes, J., Hamilton, T., Huang, S., Müller, H. & Wong, K., (2002). Adoption Challenges in Migrating to Web Services. In *Proceedings of the Fourth International Workshop on Web Site Evolution*. WSE'02. pp 21.

Tsourveloudis N., Valavanis K., Gracanin D. & Matijasevic M. (1999). On the measurement of agility in manufacturing systems. In *Proceedings of the European Symposium on Intelligent Techniques* (June 1999) AB-02, Crete, Greece. (ESIT'99).

Turner, M. & Charters, S. (2006). Protocol for a Systematic Literature Review of the Technology Acceptance Model and its Predictive Capabilities. Technical Report Keele University TR06/02.

Turner, M., Zhu, F., Kotsiopoulos, I., Russell, M., Budgen, D., Bennett, K., Brereton, P., Keane, J., Layzell, P. & Rigby, M. (2004). Using Web Services Technologies to create an Information Broker: An Experience Report, presented at *26th International Conference on Software Engineering (ICSE 2004)*, Edinburgh, Scotland.

Yin, R. K. (2003). *Case Study Research: Design and Methods*, 3rd Edition. Sage Publications. ISBN: 978-0761925538.

Zhang, J., Chung, J. & Chang, C. (2003) Architecture-based development of web service based applications. In *Proceedings of The First International Conference on Web Services (ICWS'03)*, (June 23-26, 2003), Las Vegas NV. pp 265-271.

Zhang, Z., Liu, R. & Yang, H. (2005). Service Identification and Packaging in Service Oriented Reengineering. In *Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, Knowledge Systems Institute Graduate School. pp 241-249.

Zhang, Z. & Yang, H. (2004). Incubating Services in Legacy Systems for Architectural Migration. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference (Apsec'04)*, (November 30 - December 03, 2004). APSEC. IEEE Computer Society, Washington, DC, pp 196-203.

Zhang, Z., Yang, H. & Chu, W. C. (2006). Extracting Reusable Object-Oriented Legacy Code Segments with Combined Formal Concept Analysis and Slicing Techniques for Service Integration. In *Proceedings of the Sixth international Conference on Quality Software* (October 27 - 28, 2006). QSIC. IEEE Computer Society, Washington, DC. pp 385-392.

Zimmermann, O., Doubrovski, V., Grundler, J. & Hogg, K. (2005). Service-oriented architecture and business process choreography in an order management scenario:

rationale, concepts, lessons learned. In *Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (October 16 - 20, 2005), San Diego, CA, USA. OOPSLA '05. ACM Press, New York, NY. pp 301-312.

Zimmerman, O., Krogdhal, P. & Gee, C. (2004a) Elements of Service-Oriented Analysis and Design: An interdisciplinary modelling approach for SOA projects. [Online] IBM. Available at <<http://www.ibm.com/developerworks/library/ws-soad1/>> Accessed on 12.09.2007.

Zimmermann, O., Milinski, S., Craes, M., & Oellermann, F. (2004b). Second generation web services-oriented architecture in production in the finance industry. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications* (October 24 - 28, 2004), Vancouver, BC, CANADA. OOPSLA '04. ACM Press, New York, NY. pp 283-289.

## **Appendix**

## Appendix A – Reengineering ordering

Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Requirements																		
Business process modelling	3	1	none	none	none	none	1	none	1	none	none	none	1	none	none	none	1	1
Web service Interfaces	4	2	none	1	3	none	4	6	none	none	none	4	5	7	3,4	none	3	4
Recoding/ restructuring	2	none	5	2	2	4	none	5	4	5	none	3	none	6	none	4	4	none
Redesign	1	3	2, 4	3	1	none	none	4	3	4	3	2	none	5	2,5,6,7	3	5	6
Service identification	none	none	none	none	none	3	none	none	none	none	2	none	3,4	none	none	none	none	3, 5
Code understanding	none	4	3	none	none	2	3	3	2	none	none	1	none	1,4	1	1,2	none	2
	none	none	1	none	none	1	2	1,2	none	1,2,3	1	none	2	2, 3	none	none	2	none
Classification	m	t	m	t	b	b	m	t	m	b	b	b	m	b	m	b	m	m
Coverage	4	4	4	3	3	4	4	5	4	3	3	4	4	5	3	3	5	5
m = meet-in-the-middle																		
b = bottom up																		
t = top down																		

## Appendix B – Functional testing

### Functional test plan 1:

Service	Input	Expected outcome	Actual outcome
Activation Service	<b>Username:</b> mark <b>Password:</b> we76yth	<b>Roles:</b> Doctor,Override,on_duty, <b>Teams:</b> MedicalTeam1, TeamOverride,	<b>Roles:</b> Doctor,Override,on_duty, <b>Teams:</b> MedicalTeam1, TeamOverride,
Query fields Service	<b>Roles:</b> Doctor, On Duty <b>Teams:</b> Medical Team 1	All roles and teams authorised. Fields returned: 1. FinancialNotes 2. Date_Of_Birth 3. MaritalStatus 4. referrersRelationship 5. Photograph 6. First_Name 7. Surname 8. Home_Telephone_No 9. ReasonForRequest 10. Occupation 11. CurrentHospital 12. ReferrerName 13. Address 14. RiskAssessment 15. ReferralSource	Only On-Duty is authorised. Correct fields returned: 1. FinancialNotes 2. Date_Of_Birth 3. MaritalStatus 4. referrersRelationship 5. Photograph 6. First_Name 7. Surname 8. Home_Telephone_No 9. ReasonForRequest 10. Occupation 11. CurrentHospital 12. ReferrerName 13. Address 14. RiskAssessment 15. ReferralSource
Query Service	Financial Notes, Date of Birth, Marital Status, referrersRelationship, Photograph, First Name, Home Tel No, Occupation, Current Hospital, Address	clientNumber 4572244 dateOfBirth 1998-12-12 maritalStatus child referrersRelationship clientImage familiarForename Philip telephoneNumber 01782456871 propertyNameNumber 158 street Stockholm Way district Hanley town Stoke-on-Trent postcode ST1-5RG	clientNumber,4572244, dateOfBirth,1998-1212, maritalStatus,child, referrersRelationship,blankvalue, clientImage,blankvalue, familiarForename,Philip, telephoneNumber,01782456871, propertyNameNumber,158, street,Stockholm Way, district,Hanley, town,Stoke-on-Trent, postcode,ST1-5RG,



**Functional test plan 2:**

Service	Input	Expected outcome	Actual outcome
Activation Service	<b>User:</b> lin horley <b>Password:</b> ag32the	<b>Roles:</b> CaseWorker, Override,on_duty,Doctor,CareManager, <b>Teams:</b> SolihullCareTeam2, TeamOverride,SolihullCareTeam1,	<b>Roles:</b> CaseWorker,Override, on_duty,Doctor,CareManager, <b>Teams:</b> SolihullCareTeam2, TeamOverride, SolihullCareTeam1,
Query fields service	<b>Roles:</b> Case Worker, On Duty <b>Teams:</b> SS Care Team 2	All roles and teams authorised. Fields: 1. FinancialNotes 2. Date_Of_Birth 3. MaritalStatus 4. referrersRelationship 5. Photograph 6. First_Name 7. Surname 8. Home_Telephone_No 9. ReasonForRequest 10. Occupation 11. CurrentHospital 12. ReferrerName 13. Address 14. RiskAssessment 15. ReferralSource	All roles and teams authorised. Fields: 1. FinancialNotes, 1. Date_Of_Birth, 2. MaritalStatus, 3. referrersRelationship, 4. Photograph, 5. First_Name, 6. Surname, 7. Home_Telephone_No 8. ReasonForRequest, 9. Occupation, 10. CurrentHospital, 11. ReferrerName, 12. Address, 13. RiskAssessment, 14. ReferralSource,
Query Service	<b>Surname:</b> Williams Client_Number, carer, Ethnic Origin, FinancialNotes, Date_of_Birth, MaritalStatus, referrersRelationship. First_Name, Home_Telephone_No, RiskAssessment	Nothing returned due to no Manchester data source.	Nothing returned.

**Functional test plan 3:**

<b>Part of system</b>	<b>Input</b>	<b>Expected outcome</b>	<b>Actual outcome</b>
Activation service	<b>User:</b> lin horley <b>Password:</b> ag32the	<b>Roles:</b> CaseWorker, Override,on_duty,Doctor,CareManager, <b>Teams:</b> SolihullCareTeam2, TeamOverride,SolihullCareTeam1,	<b>Roles:</b> CaseWorker,Override, on_duty,Doctor,CareManager, <b>Teams:</b> SolihullCareTeam2, TeamOverride, SolihullCareTeam1,
Query fields service	<b>Override:</b> Global	<b>Roles:</b> role override <b>Teams:</b> team override	<b>Roles:</b> IBHISOverride <b>Teams:</b> IBHISTeamOverride,
Query service	<b>Surname:</b> King FinancialNotes,	Financial notes will be visible	Financial notes visible

**Appendix C - PBM and NLC response times**

<b>NLC</b>			<b>PBM</b>		
small (ms)	medium (ms)	large (ms)	small (ms)	medium (ms)	large (ms)
172	375	203	172	172	125
203	172	140	235	157	219
156	156	94	390	156	203
187	125	141	172	235	156
219	156	156	156	172	157
172	140	140	125	266	141
188	156	203	250	140	157
187	156	172	328	156	172
141	156	187	140	156	828
203	156	266	172	156	438
156	141	672	188	157	172
937	156	141	219	140	188
188	906	110	219	140	141
188	140	125	140	156	172
172	172	141	172	187	203
140	125	157	125	172	187
188	141	219	156	172	156
172	109	125	156	172	157
141	141	94	187	141	140
156	141	125	218	313	156
<b>213.3</b>	<b>196</b>	<b>180.55</b>	<b>196</b>	<b>175.8</b>	<b>213.4</b>

## Appendix D – reengineered IBHIS broker (PBM) semantic differential scale – Primary reviewer

Area being evaluated	Response									
	Poor			Acceptable			Excellent			
Completeness	1	2	3	4	5	6	7	8	9	10
Comments: All functional requirements met.										
Consistency of design/documentation	1	2	3	4	5	6	7	8	9	10
Comments: UML is used at all stages										
Traceability of components	1	2	3	4	5	6	7	8	9	10
Comments: Was able to follow the activation service through all aspects right down to the code, but naming could have been more consistent										
Granularity:	1	2	3	4	5	6	7	8	9	10
Comments: The granularity was just right for the project. It was coarse grained enough to realise the requirements. It could have been more fine grained though (2 potential services were left out) but for good reason (see service identification).										
Expandability	1	2	3	4	5	6	7	8	9	10
Comments: In terms of the services, more services could be created easily. The architecture had now separated visual from business logic which means changes are easier to make.										
Generality	1	2	3	4	5	6	7	8	9	10
Comments: Two of the services were not very general and were very specific to the task (getQueryFields and query) but the activation could have been used in other scenarios.										
Software system independence	1	2	3	4	5	6	7	8	9	10
Comments: Unchanged										
Communication commonality	1	2	3	4	5	6	7	8	9	10
Comments: This was improved slightly as web services were used instead of .jsp for communication between business logic and presentation.										

## Appendix E – reengineered IBHIS broker (NLC) semantic differential scale – Primary reviewer

Area being evaluated	Response									
	Poor			Acceptable			Excellent			
Completeness	1	2	3	4	5	6	7	8	9	10
Comments: System was not accessible from business processes.										
Consistency of design/documentation	1	2	3	4	5	6	7	8	9	10
Comments: UML is used at all stages										
Traceability of components	1	2	3	4	5	6	7	8	9	10
Comments: Was able to follow the activation service through all aspects right down to the code, but naming could have been more consistent										
Granularity:	1	2	3	4	5	6	7	8	9	10
Comments: The granularity was just right for the project. It was coarse grained enough to realise the requirements. It could have been more fine grained though (2 potential services were left out) but for good reason (see service identification).										
Expandability	1	2	3	4	5	6	7	8	9	10
Comments: The architecture could not be expanded as well as the SOAP version due to the dependency created by using WSIF. The system was not accessible from business processes which mean you do not have the building blocks for a flexible architecture. The design of the architecture does easily allow for more blocks to be added and removed however.										
Generality	1	2	3	4	5	6	7	8	9	10
Comments: Two of the services were not very general and were very specific to the task (getQueryFields and query) but the activation could have been used in other scenarios.										
Software system independence	1	2	3	4	5	6	7	8	9	10
Comments: Unchanged										
Communication commonality	1	2	3	4	5	6	7	8	9	10
Comments: WSIF does not have the flexibility that is given by SOAP which means it cannot be used outside of Websphere. To make matters worse the implementation of WSIF in the version of Websphere used meant that the system could not even be called as the 'client' technology for calling WSIF would not allow it to access the drivers used in a mysql database. It has access to database drivers for its own class but I was unable to get it to use a database driver that is used by the system code. Even if this can be accessed the 'enterprise application client' can only be accessed by systems that support J2EE.										

**Appendix F – response times for CS-RQ2**

<b>Web service</b>			<b>Business process</b>			<b>Mobile client</b>		
<b>small (ms)</b>	<b>medium (ms)</b>	<b>large (ms)</b>	<b>small (ms)</b>	<b>medium (ms)</b>	<b>large (ms)</b>	<b>small (ms)</b>	<b>medium (ms)</b>	<b>large (ms)</b>
16	15	16	265	203	281	593	578	579
16	16	15	250	219	203	937	763	609
15	15	16	219	250	219	1000	640	578
16	16	16	235	203	250	609	703	563
15	15	16	219	218	265	937	625	703
15	47	16	219	250	328	609	625	578
16	16	31	219	266	203	578	593	672
31	16	16	234	218	250	578	610	578
16	15	16	218	250	297	578	578	578
16	16	16	250	250	328	594	609	625
16	16	15	187	234	219	609	953	578
16	16	15	250	203	234	593	1406	976
16	15	15	203	219	187	1046	593	593
15	16	16	218	265	234	641	531	578
16	16	16	218	203	203	593	609	547
16	16	16	281	235	234	578	625	609
15	16	16	235	219	219	688	1297	563
16	15	15	266	234	203	578	750	610
16	16	16	219	219	203	764	594	578
16	16	16	219	219	203	1000	578	2750
<b>16.5</b>	<b>17.25</b>	<b>16.5</b>	<b>231.2</b>	<b>228.85</b>	<b>238.15</b>	<b>705.15</b>	<b>713</b>	<b>722.25</b>

## Appendix G – original IBHIS broker semantic differential scale – Primary reviewer

Area being evaluated	Response									
	Poor			Acceptable			Excellent			
Completeness	1	2	3	4	5	6	7	8	9	10
Comments: Not having any original design documentation meant it was not known if they were therefore as the project was completed it was assumed to be correct to a certain extent										
Consistency of design/documentation	1	2	3	4	5	6	7	8	9	10
Comments: No documentation										
Traceability of components	1	2	3	4	5	6	7	8	9	10
Comments: No documentation										
Granularity:	1	2	3	4	5	6	7	8	9	10
Comments: 22 interfaces is too many (but better than none). There should only be a handful.										
Expandability	1	2	3	4	5	6	7	8	9	10
Comments: The expandability is ok. The system is based on service principles which are used for expandability. The only problem is there are some workarounds that were used which reduce this as well as the fairly tight coupling with the user interface and the business functionality.										
Generality	1	2	3	4	5	6	7	8	9	10
Comments: Most of the 'services' can be used outside the given context but not to a great extent.										
Software system independence	1	2	3	4	5	6	7	8	9	10
Comments: IBHIS has been ported to other operating systems and is written in the portable Java language. IBHIS is bound to the Websphere development server and uses the OWL language.										
Communication commonality	1	2	3	4	5	6	7	8	9	10
Comments: SOAP communication used for messaging. HTTP and XML used for servlets.										

## Appendix H – original IBHIS broker semantic differential scale - Validation

Area being evaluated	Response									
	Poor			Acceptable			Excellent			
<b>Completeness</b>	1	2	3	4	5	6	7	8	9	10
Comments: The original IBHIS prototype meets all of the functional requirements for the system, but some areas are not developed to the level that they were intended. For example, the Data Access Services are integrated into the broker itself, rather than being stand-alone independent services. The interface is also tied to the broker implementation somewhat.										
<b>Consistency of design/documentation</b>	1	2	3	4	5	6	7	8	9	10
Comments: On the whole the design is consistent, but there are some differences due to the use of three distributed programmers, each working on different PhDs (and therefore each with a different focus). The documentation is also incomplete.										
<b>Traceability of components</b>	1	2	3	4	5	6	7	8	9	10
Comments: Everything included in the system can be traced back to requirements.										
<b>Granularity:</b>	1	2	3	4	5	6	7	8	9	10
Comments: The system uses Web services to communicate internally, but an improvement would be to expose the whole broker as a service itself. Access to the system is also only via a single Web interface. Internally, a number of aspects of the system are grouped into a single service (i.e. the Data Access Services) and so in an ideal implementation the system would be made up of finer grained services.										
<b>Expandability</b>	1	2	3	4	5	6	7	8	9	10
Comments: The system has been extended subsequently, but due to inconsistencies in the coding this is not straightforward. It is also tied to a particular version of the server platform/database due to the use of particular class libraries.										
<b>Generality</b>	1	2	3	4	5	6	7	8	9	10
Comments: The services could be used within other systems, with amendments. Their functionality could be generalised.										
<b>Software system independence</b>	1	2	3	4	5	6	7	8	9	10
Comments: The system uses Web services and so should be independent, but in reality due to the particular versions of class libraries/servers used (see above), this is not the case.										
<b>Communication commonality</b>	1	2	3	4	5	6	7	8	9	10
Comments: The system uses standard protocols and technologies for the majority of communication.										



## Appendix I – reengineered IBHIS broker (PBM) semantic differential scale – validation

SOAP:

Area being evaluated	Response									
	Poor			Acceptable			Excellent			
<b>Completeness</b>	1	2	3	4	5	6	7	8	9	10
Comments: Assessed against live system and current requirements (not original requirements). Does not take into account agility										
<b>Consistency of design/documentation</b>	1	2	3	4	5	6	7	8	9	10
Comments: Everything in implementation seems consistent apart from small update of section										
<b>Traceability of components</b>	1	2	3	4	5	6	7	8	9	10
Comments:										
<b>Granularity:</b>	1	2	3	4	5	6	7	8	9	10
Comments: dependent on if coarse desirable										
<b>Expandability</b>	1	2	3	4	5	6	7	8	9	10
Comments: improved from original particularly at the interface but still used back end										
<b>Generality</b>	1	2	3	4	5	6	7	8	9	10
Comments: The same										
<b>Software system independence</b>	1	2	3	4	5	6	7	8	9	10
Comments: The same										
<b>Communication commonality</b>	1	2	3	4	5	6	7	8	9	10
Comments: SOAP interface improves communication commonality.										