University of Fort Hare
Together in Excellence

A P2P Middleware Design for Digital Access Nodes in Marginalised Rural Areas

A thesis submitted in fulfillment of the requirements of the degree of

Master of Science

In

Computer Science

University of Fort Hare

by Ronald Wertlen

January 2010

Contents

List of Figures

List of Tables

Preamble

Acknowledgments

I would like to thank –

- my wife and family (very lovingly) for their support: Work like this cannot occur without some sacrifices by the ones with whom I most like to share my time,
- the Departments of Computer Science and the Centres of Excellence at Fort Hare and also Rhodes University for supporting my work, and in particular Alfredo Terzoli,
- eKhaya ICT for material and research support (fees, etc.), and
- my wonderful team of proof-readers: Erika, Ford, Henry, Tanya. Thanks from the bottom of my heart.

General Notes

- Even when we break the rules, our English usage in this document follows an "open style" – as defined by the NASA writers' style guide [McC98].

Declaration

- This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Abstract

This thesis addresses software design within the field of Information and Communications Technology for Development (ICTD). Specifically, it makes a case for the design and development of software which is custom-made for the context of marginalised rural areas (MRAs). One of the main aims of any ICTD project is sustainability and such sustainability is particularly difficult in MRAs because of the high costs of projects located there. Most literature on ICTD projects focuses on other factors, such as management, regulations, social and community issues when discussing this issue. Technical matters are often down-played or ignored entirely. This thesis argues that MRAs exhibit unique technical characteristics and that by understanding these characteristics, one can possibly design more cost-effective software. One specific characteristic is described and addressed in this thesis – a characteristic we describe here for the first time and call a *network island*. Further analysis of the literature generates a picture of a distributed network of access nodes (DANs) within such *network islands*, which are connected by high speed networks and are able to share resources and stimulate usage of technology by offering a wide range of services. This thesis attempts to design a fitting middleware platform for such a context, which would achieve the following aims: i) allow software developers to create solutions for the context more efficiently (correctly, rapidly); ii) stimulate product managers and business owners to create innovative software products more easily (cost-effectively).

A given in the context of this thesis is that the software should use free/libre open source software (FLOSS) – good arguments do also exist for the use of FLOSS. A review of useful FLOSS frameworks is undertaken and several of these are examined in an applied part of the thesis, to see how useful they may be. They form the basis for a walking skeleton implementation of the proposed middleware. The Spring framework is the basis for experiments, along with Spring-Webservices, JMX and PHP 5's web service capabilities.

This thesis builds on three years of work at the Siyakhula Living Lab (SLL), an experimental testbed in a MRA in the Mbashe district of the Eastern Cape of South Africa. Several existing products are deployed at the SLL in the fields of eCommerce, eGovernment and eLearning. Requirements specifications are engineered from a variety of sources, including interviews, mailing lists, the author's experience as a supervisor at the SLL, and a review of the existing SLL products. Future products are also investigated, as the thesis considers current trends in ICTD. Use cases are also derived and listed. Most of the use cases are concerned with management functions of DANs that can be automated, so that operators of DANs can focus on their core business and not on technology.

Using the UML Components methodology, the thesis then proceeds to design a middleware component architecture that is derived from the requirements specification. The process proceeds step-by-step, so that the reader can follow how business rules, operations and interfaces are derived from the use cases. Ultimately, the business rules, interfaces and operations are related to business logic, system interfaces and operations that are situated in specific components. The components in turn are derived from the business information model, that is derived from the business concepts that

were initially used to describe the context for the requirements engineering. In this way, a logical method for software design is applied to the problem domain to methodically derive a software design for a middleware solution. The thesis tests the design by considering possible weaknesses in the design. The network aspect is tested by interpolating from formal assumptions about the nature of the context. The data access layer is also identified as a possible bottleneck. We suggest the use of fast indexing methods instead of relational databases to maintain flexibility and efficiency of the data layer.

Lessons learned from the exercise are discussed, within the context of the author's experience in software development teams, as well as in ICTD projects. This synthesis of information leads to warnings about the psychology of middleware development. We note that the ICTD domain is a particularly difficult one with regards to software development as business requirements are not usually clearly formulated and developers do not have the requisite domain knowledge.

In conclusion, the core arguments of the thesis are recounted in a bullet form, to lay bare the reasoning behind this work. Novel aspects of the work are also highlighted. They include the description of a network island, and aspects of the DAN middleware requirements engineering and design. Future steps for work based on this thesis are mapped out and open problems relating to this research are touched upon.

Chapter 1

Introduction

Where there's a will, there's a way.

Anonymous

The message must go through.

Pony Express Motto, 1860

1.1 Background

This thesis is written against the backdrop of Information and Communication Technologies for Development (ICTD) in impoverished and marginalised rural areas (MRAs). The main problem of ICTD projects has been sustainability, since such projects have generally only flourished as long as external resources were being delivered to the communities where the projects were based.

The aim of sustainability, one might say, is the holy grail of ICTD pilot projects, which in the past have struggled to show that technology introduction to MRAs makes economic sense[BEH$^+$03]. Falling hardware costs and greater global participation in a virtual knowledge based society are, however, tilting the scales in favour of ubiquitous ICTs – even in MRAs [KDQ09]. In the knowledge based society it is possible to generate revenues with what you know – i.e. just the information one has at one's disposal has a worth, and that worth is calculated on how many parties in the market are interested in that information. Thus, by bringing communities from MRAs into the knowledge society through ICTs, one can potentially open new doors to build revenues for the communities[Ken08]. This requires a set of novel applications, which offer their end users access to the tools they need to convert their knowledge into an economic profit.

This thesis provides a software development perspective of the problem and looks at what kinds of software designs could be of benefit in ICTD projects in MRAs. In particular, we argue that a particular kind of rural technological context is becoming common in MRAs and that such a context can be served particularly well with a particular software design. The specific rural context of this thesis is informed by the work of the University of Fort Hare and Rhodes University departments of Computer Science in an Eastern Cape MRA – the Siyakhula Living Lab.

This thesis builds on several years of experience in the field.

1.2 Aims

This thesis proposes a novel middleware for rural Digital Access Nodes (DANs), such as those running at the Siyakhula Living Lab (SLL).

The goal of the software is twofold:

1. To allow developers writing applications and services for the context of the SLL to be able to develop their applications more effectively and efficiently

2. To provide a framework for developers to better target the needs of the end-users in the given context.

Why is this important? To date a number of networked software applications have been developed within the context of the SLL (for a list, see table 3.2). These applications are almost exclusively stand-alone applications (in one instance they share a database table, in theory), which belies the fact that all the applications have a common goal. This thesis argues that the novelty and effectiveness of the applications could be increased by allowing the applications to collaborate, especially on a contextual level. For the purposes of this thesis, this is the level at which domain knowledge influences a middleware layer to: a) provide specific service interfaces relevant to the context and b) react transparently to requests and events (such as timeouts) in a manner which is beneficial in the context. The argument is explored through the implementation of a middleware layer, which is a typical technical solution to the problem of collaboration between applications. The middleware layer exposes standardised interfaces allowing developers to develop standard, reusable applications for the network. Through the middleware layer, developers thus have access to a set of services existing on the network, which in past projects were being replicated within each application. The main instance of such a service is the user authentication, which had been separately implemented by three services.

This thesis suggests that such a middleware layer can be distributed in order to allow applications transparent access to network services regardless of the location at which they are housed in the network. This is an important step as it allows the network to expand more easily since the creation of new physical nodes in the network is not associated with complete duplication of all software and hardware resources required. Naturally, access to the shared resources is controlled.

Further, this thesis explores avenues to build a middleware layer suited to the scenario of a network of DANs – a unique context. The network of DANs are meant to offer a variety of highly innovative applications to communities in MRAs. This is in contrast to the goals or contexts of other middleware solutions known to the author.

1.2.1 Objectives

The objectives of this project are thus, to:

1. Design a middleware framework which can cater for future applications in the context of MRAs.
    (a) The rural SLL context should be carefully considered in order to make some educated guesses as to what sorts of future applications will run on the system, as these need to be catered for as well. For instance, the SLL is at the time of writing negotiating the addition of GSM based resources (e.g. via the Wireless Village project by Nokia-Siemens Networks and Vodacom).
    (b) Further, the aspect of several DANs collaborating in an area is to be explored. It may be beneficial to allow DANs to share resources, to provide the user with a better service, allowing access from anywhere within the network.
2. Investigate open and compatible implementation technologies for the middleware framework .
    (a) The framework should take into account the heterogeneous nature of the development practices at the Universities and should cater for existing applications.
    (b) The SLL is particularly interested in the use of Open Source technologies to enable use and reuse of solutions thus promoting sustainability.

1.3 Definitions

In this section we define the meaning of some key terms, that are otherwise fuzzy in meaning or used in different senses by different authors depending on the perspective.

Component

Szyperski's definition of a component is succinct and captures all the facets of what a component is :

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." [SGM02][1] This definition highlights the idea of reusable independent units of functionality, whose behaviour is defined beforehand and is contained in interfaces, and which capture large pieces of stand-alone functionality that can be run on their own or in other contexts.

Services versus Applications

We use the word services to refer to basic utilities that manage one or more shared resources in a network environment. Examples of services are login and user management, logging, backup functionality and middleware. Since the word services is often used from a end user perspective to refer to the actual utilities provided (such as egovernment services) there is potentially a source of confusion, which we wish to avoid in this thesis. From an end-user perspective, however, it is not possible to tell where the boundaries of services lie. An egovernment service would typically make use of several services in the middleware, as well as core services of the platform and this is transparent to the user. To avoid ambiguity in this thesis, "end user services" are always labelled from a platform perspective as "application services", or simply "applications", and never from an end user perspective unless the point of view is explicitly expressed as an end user one. Such applications are usually components which are running on the network, in a web application server such as Apache web server + PHP scripts or Apache Tomcat servlet container engine. We use the word applications to refer to software running on the network and using the platform services to achieve some user-related aims.

Any service can also be a component.

Rural Digital Access Node

In this thesis a Rural Digital Access Node (abbreviated DAN [2]) is a facility which is accessible to the public for at least a portion of every day and is situated in an marginalised rural area (MRA), i.e. a rural area isolated from modern services and amenities, with an impoverished population, high rates of illiteracy and an unmotivated, uninformed community. The term "rural telecentre" is synonymous with DAN, with one exception: the DAN does not restrict users to physically visiting the centre to access digital services. Instead the DAN focuses also on affordable means of bringing digital services into rural homes, for instance via cheap GSM access technology or WiMAX (depending on regulatory restrictions). The more abstract "Rural Digital Access Node" thus emphasises the unspecific and experimental nature of the facility. This thesis makes no assumptions about the access type, whether "school-based", telecentre, Internet kiosk or at a so-called multi-purpose centre, or any of the many other kinds of facility. Nor does it restrict the kind of services offered to a specific industry sectors – any digital service should be accessible. To us, the most interesting sectors in a rural context

[1] In *[SGM02]*, the above definition appears after an analysis of several *dozens* of varying definitions of the concept of a component in computer science. Szyperski's exhaustive approach convinced us.

[2] We abbreviated the concept to DAN, because the rural context is given in this work.

are telecommunications, education, health, economics, and social development. All digital services thus converge within the scope of the DAN, and through the middleware layer are encouraged to collaborate to add further information value.

Middleware

We define middleware in section 2.2.1.

1.4 Scope

The scope of this thesis is a description of the research and development work behind the middleware being proposed. Since the requirements engineering step of the development work occurs against the background of the SLL communities, it is necessary to understand the broader context of MRAs. The middleware layer has much greater potential for the SLL than providing a single sign on capability. Applications that offer automatic assistance by matching user profiles to grant criteria, or that find pertinent information more easily and actively warn users about events that require their attention, could have a great impact in the SLL scenario. Such applications would push information to the users rather than waiting for the users to find things out for themselves. Further, work flows integrating applications and services could be more easily implemented. Such work flows could, for example, help ensure customer satisfaction with the eCommerce portal.

The thesis focuses on a software design, which captures and encourages such ideas of flexibility and extensibility.

Implementation is taken as far as proof-of-concept of core ideas.

Semantic web services, a promising future technology related to the middleware, are out of the scope of this thesis.

1.5 Structure

The thesis is structured as follows:

- Literature Review – this chapter has two main sections. It introduces major ICTD concepts in the first, highlighting factors and concepts that are of importance to software developers. It then looks at (software) technologies relevant to ICTD in the second.

- Methodology and Requirements Engineering – this chapter has four main sections. Firstly we introduce the methodologies that were employed in the thesis. Next, an analysis of existing and planned software provides vital information about how the system being designed will be used. The requirements engineering section uses this information to analyse the requirements of all the roles in known use case scenarios, as well as describing the requirements of the context. Finally, some technology decisions are motivated from the requirements.

- System Design and Architecture – this chapter derives a system design from the requirements engineering according to the methodology. It also provides a new design of the current eCommerce system (eMall), to illustrate the major concepts of the system design. An information model and non-functional considerations affecting the design each also have a section in the chapter.
- Implementation and Results – this chapter primarily addresses the implemented proof-of-concept and explains the Spring framework on which the demonstrator was based. Technologies looked at in smaller, isolated experiments, are mentioned. The chapter closes with lessons learned, general discussion about the development of middleware (including some important caveats!) and the ESTIMA project is also mentioned as a result.
- Conclusion – this chapter brings this thesis to a close and focuses the reader on the central arguments of this work.
- Publications – this chapter lists related publications.
- The List of References is self-explanatory.
- The Appendices bring some more technical odds and ends to light – for enthusiasts.

Chapter 2

Literature Review

The benefits [of awareNet social networking] *should not be overlooked. In this modern world we live in, being computer literate is no longer a luxury but a necessity. You must remember not long ago you couldn't get a job if you spoke only your mother tongue. These days it's becoming a prerequisite to have a driver's licence and to know something about the digital world.*

Thozamile Ngeju, Upstart Youth Newspaper Article (to be published 2010)

This chapter is structured in the following manner:

- Firstly we analyse the field of ICTD[1].
- Secondly, we research software technology relevant to our problem: open source middleware, ad-hoc distributed systems and Service Oriented Architecture.
- Finally, a concluding section sums up the main aspects of relevance discovered in the review of literature and prior art.

[1]ICTD may not classically be seen as a field within Computer Science. However, since software is required for all Information and Communication Technologies to operate, scientific investigations which look at the effects of innovative software and digital services that affect socio-economic conditions of persons, may very well qualify as Computer Science, as long as they are concerned primarily with the software aspects of the research and, to a lesser extent, with research into the impacts. The rigorous evaluation of impacts may require assistance from economists, anthropologists and ethnographers.

The selection of works introduced in this chapter are based on an examination of the context as well as the author's experience in working with Internet technologies and the Siyakhula Living Lab.

The aim of this chapter is to prevent this thesis from reinventing the wheel at any level.

2.1 Information and Communication Technologies for Development (ICTD[2])

According to Richard Heeks[3], the phenomenon of ICTs being used for development (in the sense of socio-economic upliftment, or as Heeks puts it, "International Development") has only been made possible recently by two main factors: Firstly, through the Internet, whose success is in turn based for a large part on low price commodity (unspecialised) hardware for communication and computing. And secondly, through the political will brought to bear on the subject by the United Nation's Millenium Development Goals (MDGs) agreed upon in 2000, but which were a result of a political process initiated in the early to mid-1990's[Hee08]. Heeks' observations are borne out by several others ([BEH$^+$03], [DO04], [oTU07], [DB09]). ICTD is a very broad, multidisciplinary field of study (and practice), since it considers all ICT aspects that affect international development efforts. We limited ourselves in our overview to the following research areas:

- *ICTD projects* – several projects have tried to improve disadvantaged communities by bringing ICTs to them.
- *Siyakhula Living Lab* – this is the field intervention which informs the context of this work.
- *Network topology* – here we present a case for our conceptual construct of "Network Islands" based on related literature.
- *Telecentre technology* – for a long time, telecentres were seen as the ideal vehicle to provide access to software and the Internet.
- *Novel ICTs for MRAs* – there is a wide palette of cutting edge technologies that are proven and that have open source (generic) versions, which might be applied with excellent results in the rural development context.

2.1.1 Large ICTD Projects

ICTD projects are necessarily multi-disciplinary undertakings. A review of literature reporting on ICTD projects shows a number of professionals working side by side to complete the projects. The professionals come from several traditional academic fields. We found the following fields were represented: engineering, computer science, informatics, economics, sociology, psychology, ethnology, anthropology, education, gender studies and political studies. This breadth of subject

[2] In a popular context it is also abbreviated ICT4D.

[3] Heeks is Professor of Development Informatics at Manchester University's Institute for Development Policy and Management.

matter and the volume of government reports and literature on ICTD projects made the literature review more difficult, as criteria for a selection of the literature had to be developed first.

Community Informatics and ICTD

We see Community Informatics (CI) as a subset of ICTD[4]. Its extensive theoretical underpinning was able to guide us in our selection of ICTD projects to review. CI is seen by many as the founder of the telecentre movement. This is a major point of overlap, as many of the early ICTD pilot projects involve access to computing and Internet, mainly via telecentres ([ACG08, Sey08]). Here we briefly introduce the field of CI, before reviewing projects.

The multi-disciplinary field of CI has been growing since the mid 1990s and has a firm theoretical background, unlike ICTD. The field has its roots in early digital community efforts in Scandinavian countries in the early 80's[MD05]. The Journal of Community Informatics[5] puts it this way, "Community Informatics (CI) is the study and the practice of enabling communities with Information and Communications Technologies (ICTs)". Discussing this definition, we can quickly get to the core of some CI issues. *Enabling communities*, the core of the definition, is meant in a general sense. Communities are not "enabled to do something". Unpacking the definition further, we understand that networks of people (communities) are transformed from dysfunctional units, which are unable to plan for contingencies and deal with matters that they themselves perceive as problems, into units that can make use of resources in the network to achieve things they want to. The enabling element is the Information and Communication Technology, which allows the community to communicate within itself and without itself, with the rest of the world. Information is communicated and allows the community to come to new realisations. CI is thus plainly engaged with complex networks of intelligent (human) agents, each of whom occupies a specific perspective and is embedded in a network of his/her own, which informs that perspective. Thus even as computer scientists, with perhaps limited social understanding, we can appreciate that any CI projects are complex ventures – not necessarily because of the technology, but because of the social aspects of such a project.

CI theories and lessons learned are essential background reading required to perceive the users requirements realistically, when creating a working system for rural communities. Community members in MRAs (in Africa) are not even familiar with other common forms of technology such as Automated Banking Tellers, home appliances, etc. A wide variety of skills and ethical sensitivity is required on the part of developers who work in this field [And05].

[4]This view may be contested, as CI involves itself with any community, including ones which are in developed countries and are not impoverished at all. The impacts of globalisation, the current economic crisis and demographic shifts such as aging, also put communities in developed areas under strain. These communities too are in need of development, otherwise they will become the impoverished communities of tomorrow.

[5]http://ci-journal.net/, downloaded 20th November 2009.

ICTD Project Review

To grasp the backdrop to this project properly, we looked at several relevant works, and influential authors, relating to the CI field[6]. We mention here the central points of some of the works, to highlight the major themes in ICTD and in particular in the telecentre or public access field.

[BEH$^+$03] (ICT for Development Contributing to the Millennium Development Goals: Lessons Learned from Seventeen infoDev Projects), is a concluding project report and overview of large well funded World Bank projects in the ICTD field. These projects are all concerned with access to information with respect to fulfilling the MDGs, and thus fall into Heeks' categorisation of ICT4D 1.0 projects, i.e. providing access. The lessons learned are universal, succinct and apply to all projects. Central among the lessons is the theme of community participation, as well as the theme of sustainability through application of existing cheap technologies in local language that are easy to use. Lesson Four is particularly interesting. It states that ICT4D projects may contribute more to MDGs when applied in rural areas. The only reason given by the authors for this conclusion is that 70% of the world's poor live in rural areas. While this reasoning is purely statistical and ignores the cost of projects undertaken in rural areas, the economic law of diminishing returns supports the conclusion that a dollar spent in a rural project may be more effective than one spent in an urban area, as it is more likely to contribute to MDGs. The only problem we see with this reasoning, is that there may be a critical mass of dollars required in order to make the rural project effective.

[UN 03] is an UN ICT Task Force review of "information kiosks" as far as sustainability is concerned. It is a collection of articles, many of which could be mentioned in this listing in their own right. Their in depth review brings out several major factors of sustainability. In addition to networking and sharing resources, they mention the selection of the right person ("social innovator") to drive the telecentre. This is something we cannot as developers influence, although we do have the ability to help this person by making the technology as easy to use as possible. Further, Stoll mentions (pp. 65) that financial sustainability is only achievable if as broad a mixture of services as possible is offered by the telecentre. This also supports our perspective that it must be possible to integrate as many existing services and as yet unknown services as possible into a rural telecentre platform. Worthy of mention are two further papers, one by Fuchs (pp. 80) that claims that it is critical to bring digital services into MRAs as quickly as possible and another by Wisner (pp. 110) that introduces a new measure of telecentre impact. Fuchs claims that digital technology must find its way into MRAs regardless of direct economic effects. It is the only way to spur innovation and adoption sustainably and in a manner that really empowers the community, since the community will be gathering experiences

[6]Importance of the works was determined in a survey conducted by us of works that are often cited in CI literature. A paper was presented on this work [Wer08]. Particularly works after 2001 were considered. The global recession at that time put pressure on funding agencies to give their ad-hoc pilot project interventions a scientific footing, which improved the quality of reports produced after 2001 [IDR06].

themselves. This is similar to arguments made for the teaching of computer science in developing countries, made by [DB09]. Wisner, introduces a framework which looks at the aggregate impact that a telecentre has on a community, over and above financial measurements. One of the important variables she introduces is capacity. Through the telecentre tool, the community gains capacity to undertake ventures.

[DO04] (A Rural ICT Toolkit for Africa) is another work that explicates the observations and experiences made in World Bank projects. It is also has a strong leaning on lessons learned from ICT4D pilot projects, and is addressed to legislators and governments primarily. It attempts to show how private-public partnerships can provide access to rural areas affordably. An interesting observation in this work is that the rural areas may have greater buying power than businesses generally think, if the rural diaspora is considered. Remittances from the diaspora need to be included in the business plan.

[JH06] is a study of a South African telecentre installation. Jacobs and Herselman found in the case of one rural village intervention, that repeatability of service was crucial to sustainability. Customers did not want to come back to the telecentre as their results varied tremendously, arousing skepticism. An integrated platform such as the one being developed at the SLL addresses this problem and introduces integrated solutions and some work-flow managed processes. Further the following broad categorisation of service types is defined (in order of increasing revenue possibility): Telecentre, DTP, Training and SMME Support.

[Med06] (Rethinking Telecentre Sustainability: How to Implement a Social Enterprise Approach – Lessons from India and Africa) stresses that a network of like-minded people following the same aim have a greater chance of success than if they are operating individually. This echoes findings of the UNICTF, as mentioned above. Telecentre.org, a network of telecentre activists, operating predominantly in the Americas and South Asia support this view, and provide an online platform for telecentre operators to share experiences[Tel09a]. Telecentre.org is funded by Microsoft and the IDRC. We note that allowing local telecentres to network automatically and share resources should offer advantages to their sustainability.

[Ken08] (ICT: Promises, Opportunities and Dangers for the Rural Future) is an economist's perspective on the future of ICTD in rural areas and it is in stark contrast to the other papers reviewed here. It echoes the lesson mentioned in [BEH^{+}03], namely that ICTD projects have to focus on sustainability, however Kenny takes a different angle and attacks projects that focus on the Internet and Internet-related industries, which have been promoted by government policy in some poor areas, particularly in India. His analysis shows that infrastructure investments in these industries alone have almost no impact on MDGs in MRAs (as well as peri-urban areas), i.e. they do not turn such areas into productive, self-sustaining areas. Further the infrastructure investments required are very large and they should not be risked. To put Kenny's criticisms of novel, Internet enabled ICTD projects in MRAs into perspective, the criticism is primarily raised against large government projects which end up benefitting corporations that cleverly make use of the instruments that are meant to benefit the poor (i.e. impact MDGs) nor does it take the large open source movement into account, which together with cheap mobile technology brings novel applications within reach of all, in a hitherto unrivaled manner.

[ACG08] (Information needs and watering holes: public access to information and ICT in 25 countries) reports the results of a survey of over 25000 telecentres (by all their various names) in 25 different countries, coordinated by Washington University's Center for Information & Society (CIS). The findings of the survey are principally related to social, economic, political and environmental factors, not technical ones. This is similar to the other papers reviewed in this background section. The findings in this report suggest that access can be improved, by converging media (Internet, mobile phone service, radio, etc.), as well as by getting telecentres of all types to cooperate and collaborate. Further, the paper points out that access points need to be seen as "cool" venues, as well as redefining what trivial and non-trivial use is (e.g. are social media trivial?). Related to this is another unpublished report from the CIS [Sey08], which besides strongly supporting Kenny's argument (see above) presents a succinct organisational, economic classification of the multitude of different types of public access modes that are similar to telecentres, and which have had studies published about them.

[KDQ09] (2009 Information and Communications for Development: Extending Reach and Increasing Impact) is the World Bank's 2009 report on ICTD. The latest insights in the report are that mobile applications and technology are an important vehicle of delivery and need to be considered to give the poor connectivity, and that broadband is an important enabling technology, whether mobile or fixed. The report does not herald any new revelations as all of these observations may be gleaned from ones discussed above. A much more positive appraisal of the IT and ITES (IT enabled services) industries are given by the report, it is almost euphoric in comparison with Kenny's appraisal, since it does not focus on MRAs. The report also repeats the caveat that the impact of ICTD is not yet measurable. In the following section, 2.1.2, we colour in some of these observations with additional observations from the field and look at why the ICTD scene is only evolving slowly.

It is simple to find broad pictures and overviews of the political, environmental and social aspects of telecentres and access. However, technical reports tend to be driven by hardware considerations, since a large part of the access problem in MRAs is on a hardware level. However, the software level does not seem to be widely explored, from a telecentre point of view. Even the survey run by the CIS [ACG08] of 25000 telecentres ignores the technical dimension. So we do not know about the software aspect. We presume that the reason for this is that technology arguments are very hard to substantiate scientifically. One need simply look at the arguments on the pros and cons of Linux vs. Microsoft operating systems, in general. Our own feeling is that the open source movement embodies the kind of empowering spirit that we see in the aims of Community Informatics. This is one of the reasons why we focus on FLOSS.

2.1.2 Siyakhula Living Lab (SLL)

This thesis forms part of a larger research project, the Siyakhula Living Lab (SLL), which is a member of the European Network of Living Labs. The SLL began as the "Dwesa project" at the Universities of Fort Hare and Rhodes. The original objective of the precursor rural testbeds project was to develop and field-test the prototype of a simple, cost-effective and robust, integrated e-business/ telecommunication platform for deployment in marginalised communities in South Africa. It has become a testbed for all innovative ICTs, with a focus on distributed ICTs and communication modes (GSM, WiMAX, and Web Services)[TSTC06]. It is situated in a marginalised rural area (MRA) at the communities of Dwesa, Cwebe and Nkwalini, in the Eastern Cape of South Africa in the Mbashe Municipality. Amalgamation of the original project with the SELF Solar Computer Lab network ([Wer09, eKh09]) resulted in a network of public access nodes at 8 rural schools, which will ultimately form a single island spanning 30km. The technologies deployed at the participating communities at Rural Digital Access Nodes (DAN) are geared to connect the communities with one another via a local WiFi/WiMAX network and with others via the Internet. The fact that DANs are connected in a high-speed WAN is crucial to this thesis. We contend that given a suitable substrate, useful eServices can flourish in MRAs as they are driven by the needs of the community and its increasing capability to use the virtual sphere. We believe that could be an ingredient in reaching one of the main goals of the SLL: to show that the technologies deployed can generate sufficient revenue to sustain the running costs incurred by the ICTs themselves and to maintain interest of the community in the project[DTMT07].

Technologies deployed (or soon to be deployed) at the SLL include the following:

- An eCommerce application which allows local traders to exhibit and sell their crafts on-line[NTM06], with a rewards based extension to encourage usage of the shop [JTT09];
- An equitable community-based cost management scheme, implemented initially for Internet access, but which is extensible to include other service costs[TTT07];
- An eGovernment platform, which mirrors commonly required home affairs documents, provides a communication transport with home affairs and provides fora for local government initiatives[JMW08];
- An eJudiciary – a system that is meant to help the tribal leaders track issues in the community to promote transparency and fairness, as well as providing important statistics about such issues in the community over time [SM08];

- BingBee, an unattended kiosk approach, which merges simple and novel hardware HCI elements, with a selection of customised content aimed at promoting logic, literacy and numeracy skills in young children[SWL06].
- GSM and radio channels to deliver information to the community from external and internal sources.
- Agent based service provisioning, which makes available an ontology built from local knowledge to route requests for eServices within the platform [TTC09].
- Extension to all systems which localise software for the particular area and language are being developed by native speakers who originate from the region [DMTT08].
- Java Advanced Intelligent Networks (JAIN) based architecture for the convergence of real-time communication technologies [TTW09].

In an effort to provide replicable service to villagers across nodes, a suitable middleware platform could be tremendously beneficial.

What becomes clear from the applied work in MRAs is how pervasive barriers to ICTs are: wired technologies are expensive in such areas, hence the high penetration of mobile communications as opposed to fixed telephone lines [DO04], and there is no electricity in most such areas, although there are ongoing efforts to provide basic solar power for instance at clinics (see the Solar Electric Light Fund (SELF) or Solar Light for Africa). Further the high level of illiteracy means that there is no support or maintenance readily available for ICTs. Harsh environmental conditions, moisture or heat and little secured housing are the norm in these areas. All these factors combine to create a particularly difficult environment for ICTs.

Apart from infrastructural challenges, the poorly educated community members and cultural divergence make it difficult for externally initiated projects to succeed. General psycho-socio-political barriers are reviewed in general in the previous section. The SLL attempts to address all the psychological and social barriers. For instance a local field base has been set up within the community so that student trips to the area are housed within the community. Further the long-standing relationship with the community – over three years – has built trust and acceptance within sections of the community.

2.1.3 The *Network Island* Phenomenon

Mesh Networks

Mesh networks are networks whose nodes are able to route messages between any other nodes in the network. Even if the network is damaged, messages can continue to pass to all destinations as long as any path between the communicating nodes exists. The original idea of the mesh comes from the topology of such a network, which resembles a lattice arrangement. This was a handy way of laying out hardware sensors or circuits (such as CMOS circuits). The sensors could be configured to pass on information from surrounding nodes adding their own inputs to the signal [BBLV04]. This meaning of mesh networks seems to have been lost in more recent works such as those mentioned below, which focus on wireless mesh networks.

There are several rural single radio mesh[7] or long-distance network pilots being researched in recent literature, see for example [Joh07, PNS+07]. For an example of such a network please see 3.2. We discuss the central findings of these projects in the section 2.1.5. These projects are typified by simple flat IP addressing at nodes, using inexpensive WiFi hardware, in situations where maintenance of equipment is complicated and other hindrances like failing power supply introduce network down time. Mesh networks provide a challenge to regular network technologies, owing to their dynamism. P2P applications however are engineered to function in such scenarios (see section 2.2.2).

Characteristics of Network Islands

Within the SLL, we have also created and observed a rural single radio mesh network. We used inexpensive long-distance wireless technologies to create rural islands of high speed connectivity. The wireless networks in the SLL span areas in the order of 10 - 30km, but these could be increased to several 100 kilometres. We noticed that the topology of our networks (as well as those of the other authors mentioned in this section) do not resemble a lattice at all. We thus call them *network islands*. We characterise network islands by the following: within the island, high-speed links provide excellent, redundant connectivity between nodes, while connectivity to the Internet can take place at any point in the network over *low-bandwidth links* such as VSAT and GPRS or EDGE technology. We thus postulate a subclass of wireless mesh networks, called network islands.

Advanced applications such video-conferencing may operate well within the network in a cost-effective manner. This can for example benefit users, by allowing replication of eLearning or other digital services from one node to others within the network. Within the SLL, researchers have been implementing and testing services and applications adapted to the local context, which will potentially be able to make use of these characteristics [DTMT07].

WiMAX Network Islands

[AR06] also support this idea in their work. They suggest that WiMAX technology will evolve from being a Wireless Internet Service Provision technology [8], to natively supporting a P2P WiMAX network architecture for communications in rural areas. They argue that the IEEE 802.16e extension to the WiMAX standard (Mobile WiMAX) will free the technology, so that communities will be able to utilise it in a similar manner to the way in which IEEE 802.11 has been

[7] If one considers these very inexpensive networks in terms of their hardware, then the name "single radio rural mesh network" is an apt description. This is because the nodes often only have a single radio for connectivity to other nodes in the network, owing to ease of setup and maintenance.

[8] WiMAX is currently commonly being touted for the role of backhaul technology to smaller WiFi networks or other Points of Presence (e.g. Telecentres)

used to create mesh networks. The authors specifically mention P2P as a technology, because it is the social aspect of networking that they wish to highlight. In fact, the WiFi technology they describe is identical to the Mesh networking technology. Angelov and Rao do mention the risks to their P2P WiMAX networking solution from regulators, and it does seem at this point that regulations and the high cost of equipment are putting the brakes on WiMAX adoption, however, they also caution that WiMAX adoption will take several years (from 2007), so their projection cannot be faulted yet.

2.1.4 Review of Telecentre Software

Since telecentres are such a large part of the ICTD effort to provide access, we review commercially available (also FLOSS) products which are used in telecentres[9]. Our review focused on two questions: is reusable telecentre software available that could ease our mission? And, can we learn from the architecture of such telecentre systems? An overview of publicly available telecentre software shows that there is a variety of telecentre software available, commercially and in the open source domain. The following products were selected to form the basis of the review in this subsection, as they are representative of the field:

- Cybera – an open source software available via sourceforge, using a client/server architecture
- TrueCafe, Cafezee and SmartLaunch – commercial telecentre software, using a client/server architecture

These products use a client/server architecture to provide the proprietor of the telecentre with control over the workstations being used by the clients. All of the commercial software reviewed uses the Windows operating system family. In the open source pool, some versions supported only Windows, others only Linux and yet others were cross-platform. At least in the case of the commercial software, very close integration with the operating system is intended to lead to a high degree of control. Thus, the software is tamper-proof as far as the end-user working on a workstation is concerned. The main features that seem to be required by proprietors are security, pricing and billing, and marketing. Thus these systems all offer common methods of charging customers according to the time of day, pre- or post-paid, time spent at the workstation, amount of traffic generated to the Internet and number of pages printed. The way in which the pricing and billing work are designed to minimise losses of income due to customers not being able to pay (pre-paid models) and due to theft by employees (ticket systems), as well as making use of demand by customers (premium time of day charges). Security is important as customers may not use workstations or other resources when they are not allowed to, or when the system is not logging their use. Typically this means that advanced programs like the command line or terminal are disabled. For purposes of marketing, workstations look and feel can be changed to match the telecentre's corporate image. Often these systems offer integrated management and accounting modules which allow the proprietor to view statistics on usage and revenues, as these are ultimately the factors that need to be optimised. Most Windows

[9]For our purposes, a telecentre is identical to a cyber cafe or Internet cafe as discussed in the previous section 2.1.1.

software requires licensing. A particular form of licensing is the network license, in which a particular software can be executing or running a certain number of times on a particular local subnet. In order to optimise licensing costs – which can be very high indeed – the software can also monitor and control how often a certain programme is run simultaneously on the workstations and block execution when license conditions are violated. Licensing control is not of interest to this thesis.

We note that the reviewed software packages are not suitable for reuse in our project. The main reason for this is that the assumptions underlying the software architecture are related to classical telecentres (or classical telecoms functions) and not to a DAN which offers its users all varieties of digital information and knowledge based or enabled applications. Classical telecentres are independent entities which provide very basic services to their users. These include typically, but are not limited to Internet access, electronic office services, other office services such as photocopying, and gaming. Telecentres do not typically offer their users a service infrastructure which provides additional custom-built applications.

Further, the applications we are attempting to introduce are aimed at bringing knowledge society concepts[10] within the grasp of ordinary customers by introducing a technological and social layer customised to local cultural expectations and conditions. Knowledge based services typically work with contexts, i.e. the context of the user and that of the service can influence the functioning of the service and affect which choices it offers. This is also true of the services at Dwesa. E.g. the eCommerce service distinguishes between specific shop keepers, customers and administrators.and may take the number of sales a shop-keeper makes into account when charging for the service. All of these context related facts should be encapsulated in services which provide a simple interface. For example, a call to the function $costPerUnitUsage(userToken)$ may return an amount which depends on the users context. The fact that the service itself used a further module (say a social charging module) to incorporate sales information in the cost calculation should not be required by the caller [MSZ01, TTT07].

2.1.5 Novel[11] ICTs for MRAs

In this section we attempt to come closer to relevant Computer Science literature and projects. Whereas, the previous sections inform our picture of the background to ICTD in a socio-political overview (section 2.1.1) and grassroots or applied (section 2.1.2) manner, this section attempts to find out what technological advances are a) relevant to the field and b) actually a result of work done in this field[12]. We could not find such an overview in the literature, e.g. [UN 03]in the section “Innovations to close the Digital Divide” on pp. 113 mentions just 1 software innovation in 5

[10]The discussion about the value of the knowledge society and fit to rural populations in Africa is not within the scope of this thesis. There is a large amount of literature relating to the nature of work in a knowledge society, and possibilities opened up to rural workers through tele-work, digital content creation, etc.

[11]We use the term “novel”, instead of “new” or “innovative”, since: the technologies applied in the context may be old ones, not new ones; and by innovation, we understand a novel technology application which has been widely adopted and is being used sustainably, which might also not be the case.

[12][DB09] point out that more work needs to be done by locals, in order for technology not to introduce unnecessary cultural barriers, which is why it may be interesting to see what local technologies are seen as novel in the literature.

projects mentioned (the others are all mentioned because of their innovative business models), and so we performed one ourselves. What we found was that since one of the main factors which assists ICT adoption in MRAs is cost, technology resulting from general market pressure and innovation is in fact the best friend of ICTD. Miniaturisation and solid state devices seem to be much more resistant to harsh conditions in MRAs (as noted by several projects including Aleutia [Ale07], OLPC [OLP09] and Inveneo [Inv08]) and, perhaps even more crucially, price reductions in wireless and general technology are making ICTs more common in MRAs, without any intervention from third parties. This finding was in part prompted by the observation that most ICTD projects use off-the-shelf technology and do not report on technical aspects (ref. section2.1.1).

Our analysis grouped novel ICTD technology projects according to their level of completeness. *Complete* projects involve application of novel technology in MRAs and they are and have been widely reproduced. An example of this is the OLPC XO technology, which is a complete product and has been released into MRAs in thousands. *Partial* projects are ones such as iPath.ch, which is an eHealth application developed for MRAs, or the ADEN project telecentre software. These software packages need further ingredients to be released into the field (e.g. hardware, power, etc.). *Situated projects* are projects that have been released into MRAs but have not been reproduced yet. *Events* are situated projects that only run for a limited time period. This classification proved to be problematic in retrospect, as there was no clear delineation of technical and other factors required for complete projects. Originally, our idea was to consider only technical aspects, however, complete ICTD projects necessarily require a socio-political and environmental dimension in order to be complete, which confused issues. This is evident when one reads our paper [Wer08]. We refer readers to our paper for an analysis of individual technology projects. At this point we look at the most relevant projects, and also mention the specific fields of Computer Science that are represented in these projects.

[BEH$^+$03] presents an overview of 17 projects financed by the World Bank in the area of ICT4D. Batchelor raises two relevant points over and above those mentioned in section 2.1.1: firstly, projects carried out in a rural area often have a greater impact than those carried out in peri-urban areas, and existing technologies (radio, TV and mobile phone) can be used to enhance the impact of ICT4D interventions. The former point illustrates the importance of the current initiative, the latter once again underlines that a wide variety of services must be integrated with the platform to achieve sustainability and maximum benefit for the communities.

Technology and Infrastructure for Emerging Regions (TIER) is a project at the University of California Berkeley which has been investigating innovative technology application in developing areas, including rural developing areas. The TIER WiLDNet (WiFi-based Long Distance (WiLD) networks) sub-project has tested and implemented long-distance wireless networks based on 802.11 technology. They note that current 802.11 standards do not provide good connectivity in WiLDs owing to interference from existing networks, TCP/IP collisions caused by the greater delay and inefficient link-layer recovery. Atmospheric phenomena can also influence the quality of connectivity [PNS$^+$07]. Lessons from the WiLDNet research are that the idea of high-connectivity islands in MRAs is feasible and that even with high connectivity wireless networks, bandwidth should never be wasted, as deterioration can occur owing to a number of reasons.

Aleutia and ByteWalla address the Internet connectivity problem, proposing similar solutions. Mobile devices that travel into the city every day could be carrying messages that are sent once the network is accessed in town. Aleutia and others mentioned this as an idea in 2008. Bytewalla on the other hand has a large number of supporters, as well as a clear and open technology and business plan that are available online. Bytewalla may be a good example of open source spurring adoption [JSN$^+$09][Ale07].

The Meraka Wireless Africa Mesh project is an good project to consider, as it contributes to the creation of high-speed wireless networks, potentially in rural areas. Situated mainly in peri-urban areas, the project uses the cheapest wireless equipment and an innovative wireless aerial made from a recycled tin can and open source software (BATMAN) to connect many distant nodes in a mesh network. The project is situated in northern areas of Tshwane and Mamelodi.[Joh07]

Femtocells and picocells – an example is the village connection system from Nokia – are a technology developed to improve local reception in places that usually do not offer any, for instance cruise liners (boats) and airplanes. These technologies are extremely interesting in the rural context. However as we have seen, regulations constrain the adoption of these technologies in a manner that could be efficient enough for the rural context as backhaul continues to be a problem. Combined with network islands, however, these technologies could be used to spur local development, if not taxed in a similar way to the way in which regional networks are taxed.

We only found one technically innovative result directly addressing telecentres holistically, using open source software – the ADEN project (as opposed to a host of commercial ventures selling a variety of mostly Microsoft Windows based software for telecentres, although we did also find Linux based ones). Free/libre open source software (FLOSS) does seem to generally be useful in the telecentre milieu [Baj07], it is however by no means prevalent. A closer look at technology in this milieu is given in section 2.1.4. The Fostering Digital Inclusion (ADEN) Project of the French Foreign Ministry is an international EU and French funded cooperation project, whose website moved to Angola in 2008 and was unavailable at time of writing. We are not aware of any studies or published reports on the project. The ADEN Pack is a Mandriva Linux based complete solution for telecentre managers. The software is released in a DVD form (ISO image), which is easily installable at telecentres. Networking of telecentres is not presumed. Support forums for the telecentre manager are mainly in French. We did not research why FLOSS is not reported upon often in the ICTD technical domain.

Another important access software, which is FLOSS, is the Digital Doorway, a Meraka (Council for Scientific and Industrial Research – CSIR) project in South Africa. The Digital Doorway is a self-service kiosk, which resembles modern tourist information points found in major cities nowadays. It is a stainless steel column housing a PC running Ubuntu Linux, which has up to 4 terminals connected to it. Internet connectivity is supplied via VSAT and an EDGE/GPRS connection connects the access point with a central server for the purposes of monitoring and maintenance. Several of the units are deployed in MRAs and several papers have been published about the system. The system is constructed with a variety of scripts automating common house-holding tasks and allowing access to services in a controlled manner. Interestingly, the platform was originally based on Microsoft Windows, but was migrated to Linux ([Gus08], and per interview with Mr. Gush).

Since the telecentre problem is principally one of communication – in a sense patching users to resources and performing exact accounting on the transactions in order to bill them afterwards, we also considered the use of communication management (switching) software. This was done because Rhodes University has a soft-switching software based on the Asterisk software package, which supports a wide variety functionality, all closely related to the communications industry. The use of the iLanga soft-switch would additionally provide access to legacy telephone communications for user, which is certainly a very important market for the telecentre [STB09, TTW09].

2.2 Software Technology

The main aims of this section are to review software technology available to us, as well as looking at software development methodologies that are related to these technologies. Specifically, we look at middleware technologies available to us and we introduce the benefits of ad hoc networking technology.

Since we do not find any reusable software frameworks in the ICTD domain above, we review generic software frameworks that might form the basis of our own attempt at solving the problem.

2.2.1 Open Source Middleware

"Middleware is a distributed-system software that resides between applications and underlying platforms (operating systems; databases; hardware), and/or ties together distributed applications, databases or devices."[Mid09] Middleware is an umbrella term for technologies that are primarily used to achieve one of the main programming optimisation techniques in the Computer Science toolbox – information hiding, or encapsulation [13] – in a networked environment. Specifically, calls made to middleware are agnostic of the location from which the service or method being called are being answered, and of the implementation of the service. The importance of middleware is thus that it helps software development projects and running systems successfully manage changes over time, which makes safe code reuse possible. Middleware is often linked to component-based programming paradigms which also focuses on safe code reuse and composition of components (see also Service Oriented Architecture (SOA), section 2.2.3) [Wik09a]. The programming paradigm underlying the middleware implementation is however not specified, and so in this section, we consider approaches from grid (parallel) programming and agent based approaches as well as the usual suspects.

A historical development of middleware as a concept may be understood in the following way. Original software

[13] The latter term usually is applied in Object Oriented programming, which uses objects to gather information and hide private information, cleanly exposing public information of objects. However the basic idea is much older and is notably captured in Parnas' work "On the Criteria to Be Used in Decomposing Systems Into Modules" [Par72].

programming was imperative (assembler, C, etc.) and procedural. Complex software systems evolved and expanded over time, which introduced problems in software written, compiled and linked in the classical manner. Advancements in programming practice (functional programming, logic programming etc.) were introduced early on, however the bulk of coding (common operating systems, etc.) was carried out in an imperative manner. The widespread introduction of 3-tier and in fact N-tier systems, which separated their views, business logic and database in separate code bases, and even separate programming technologies required glue components (middleware). Large complex systems with several programmers working simultaneously on different aspects, could be created more efficiently if specialised tools were used for each separate part, and if the interfaces between these parts could be managed and stabilised [author's experience, see also [SGM02]]. Several technologies were developed in the 90's to deal with this aspect of computing and they displayed various characteristics. The most commonly used and most widely hyped of the technologies at that time was most probably the Common Object Request Broker Architecture (CORBA). CORBA was hailed as the first open, standardised, enterprise capable technology by IT experts, who immediately saw the value of the system in a technological sense [author's experience]. CORBA did not live up to its expectations, although it is still in use today, because the business results did not match up to expectations, and this partly gave impetus to the SOA school of thought. We see middleware as a computer science concept and SOA as a related information systems or business science concept.

There are several classifications of middleware[Wik09a]. These classifications consider in what manner the various components are linked by middleware, e.g. are (asynchronous or synchronous) procedural calls to remote methods allowed, are objects exchanged, is the system transactional (can calls be rolled back), etc. Such classifications have been found to be quite theoretical and dated, as middleware tends to mix the various types of component linkage blurring such distinctions [SGM02]. For our purposes, thus, we do not need to look at examples of all types of middleware. Instead, we consider what the goals of our middleware platform are and review relevant technologies. Although the exact system requirements are mentioned in the following section (3.3), we know that the system we need must link applications developed by many developers, for a telecentre-like context, which offers services that may be expected (or not) in such a context. We also need to consider the theory behind Service Oriented Architectures, to gain high-level requirements that may be important. Since a lot of development will be done within the context of the SLL, FLOSS technology is preferred, however it might be better if the platform is open and does not restrict other IDEs. The middleware should ideally be able to link knowledge or database frameworks and make this information accessible via standardised Resource Discovery mechanisms. The latter aspect is discussed in the following section, 2.2.2. Using these criteria, we selected the following middleware technologies for consideration as the basis of our work:

- COMPONENT-BASED programming[14]

[14]The reader may notice that we have ignored a large class of popular middleware clustered around the .NET implementation. While there are open source implementations of .NET, they are beyond the scope of this work.

OSGi

– Refer to figure 2.1a. The Open Standards Gateway Initiative (OSGi) specifications "define a standardized, component-oriented, computing environment for networked services that is the foundation of an enhanced service-oriented architecture" in the OSGi Technical White Paper[OSG07]. These standards specify how applications, which are termed bundles, can be added to a system and managed throughout the life-cycle of their use. These bundles have access to a range of services offered by the framework and are subject to security restrictions, which can be programmed. The OSGi specification was initially targeted at mobile devices, to redress imbalances introduced by MIDP, but OSGi has proved to be very popular even in large server applications, through integration with the Spring Framework.[OSG07]

Java EE

– Refer to figure 2.1b. Java 2 Enterprise Edition is a complete platform for the implementation of distributed applications such as service oriented architectures or web applications. In version 5, Java EE includes an integrated application server, called Glassfish, allowing Sun to leverage its wide range of Java technologies in an orchestrated fashion [Sun09]. According to [SGM02], the Java EE Platform is also a complex component framework with Enterprise Java Beans at the heart of the framework. EJBs drive the application container (server) and provide most of the orchestrated functionality of the platform. By virtue of the nature of Java, access to EJBs is granted to most of the components. Through the various component libraries available in Java EE, a wide array of technologies are available to allow organised implementation of practically all tasks necessary for SOA development from resource discovery, through to distribution and load balancing. Glassfish is a potential technology basis for our middleware.

SOFA

– Refer to figure 2.1c. SOFA-2 is a framework and formal language, which allows developers to design architectures using hierarchical composition of components, and then to translate these into Java code in an explicit manner. The system addresses the fact that most service frameworks offer flat component libraries, which have to be pieced together by the developer, whereby the onus for the management of composition lies with the developer. SOFA-2 on the other hand (like AspectJ) uses annotations to semantically mark-up POJOs (plain old Java objects) in order for the composition and component description to take place. At run-time all the component descriptions are put into a system wide repository. The system is distributed via containers called SOFANodes, which may reside anywhere in the network. SOFANodes support business logic components in Controllers, and allow the logic to connect through connectors. Connectors are created at run-time in order to maximise use of context knowledge and optimise communications within the system.

Connectors can allow different styles of communication such as direct method invocation or communication buses (many-to-many communications) [BHP06, BHP+07].

JMX

– Refer to figure 2.1d. JMX is a Java standard which is intended to be used as a monitoring and managing API for Java applications, with which one can e.g. monitor errors and start and stop services. The framework is included in Java SE and Java EE since 2005. The JMX framework has 3 levels: the Instrumentation level, the Agent Services level, and the Distributed Services level. Because of its wide adoption, clean interfaces and well tested technology, JMX is an example of a middleware with a specific purpose, which is why we include it here. Since JSR160, it also has specific remote capabilities. Managed Beans (MBeans) are the basic building blocks of the service portion of the architecture, they expose functionality in the Agent Services and Distributed Services levels. Calls to MBeans are handled through a proxy, which allows the system to be completely distributed. Further, MBeans are more powerful than regular EJBs because they are able to access resources outside of their context, such as the disk [Jav06, IBM09].

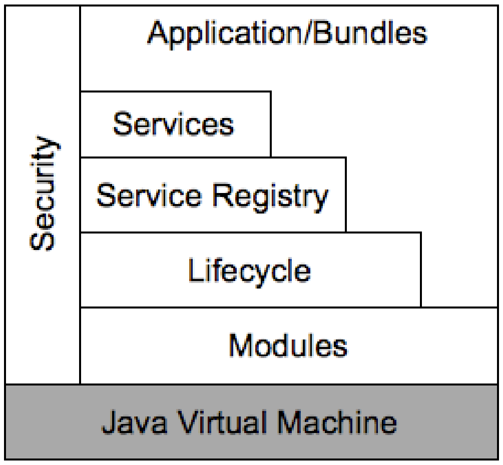
CORBA

– Refer to figure 2.1e (adapted from[Las96]). The Common Object Request Broker Architecture (CORBA) was first released by the by the Object Management Group in 1991 and has since then evolved little to a second version. Its early appearance is reflected in the way CORBA standardises remote procedure calls (RPC) in a tightly coupled, strongly typed manner. To invoke methods on a CORBA 2 servant, for example, the client may have needed to know the full signature of a method being invoked. CORBA uses the Interface Definition Language (IDL) to generate invokable method stubs for different programming languages. Thus although there is a generic type called "any" which does introduce looser coupling between client and servant, since the variable types being returned may need to be translated between languages, and since they may contain complex types such as arrays, parsing errors can occur (especially, if different ORBs were used). CORBA does not provide the same level of abstraction as do some of the more technologies described above [author's experience]. Essentially, although CORBA based infrastructures are still in use today, its lack of interoperability and other failings make it a middleware which is .[Hen06]

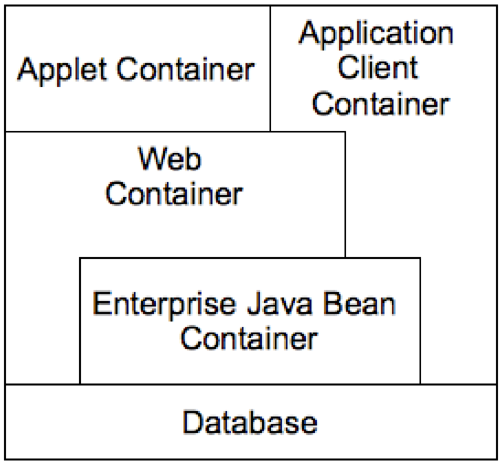
• Event Driven

Web Services: WSDL and SOAP, REST (AJAX, and Javascript and JSON)[15]

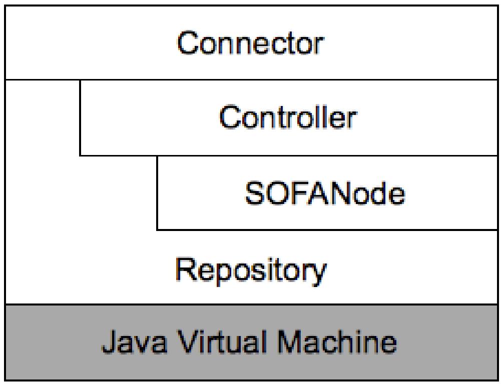
[15]SOAP – Simple Object Access Protocol; REST – Representational State Transfer; AJAX – Asynchronous Javascript And XML; JSON – Javascript Object Notation. These are common web services technologies in use today. We do not differentiate between these here.



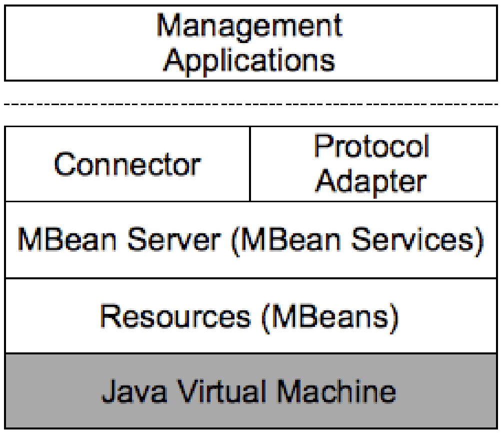
(a) OSGi



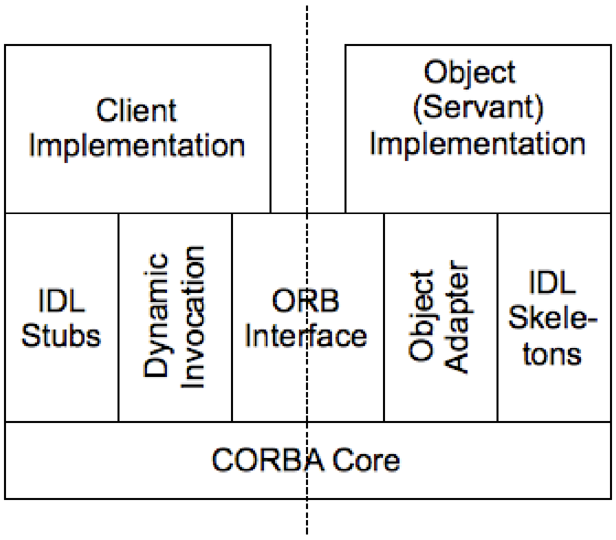
(b) Java EE



(c) SOFA 2.0



(d) JMX



(e) CORBA

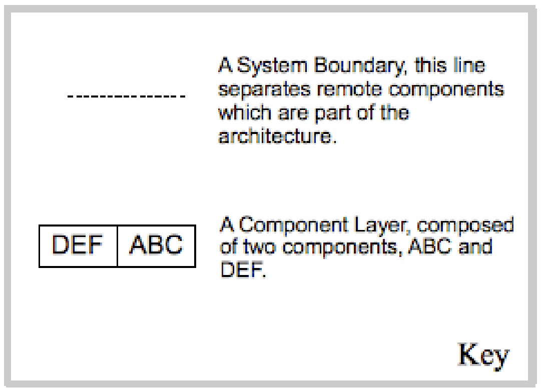

A System Boundary, this line separates remote components which are part of the architecture.
DEF
ABC
A Component Layer, composed of two components, ABC and DEF.
Key

Figure 2.1: Component layer diagrams of several middleware architectures

- Event driven architectures are becoming the de facto industry standard as far as distributed systems are concerned, especially ones which cross organisational boundaries. They provide interoperability with a vast number of services available from several companies accessible via the Web (certainly all the big Internet companies, Google, Amazon, eBay, and so on offer Web Service APIs). They are also used in operational research architectures in Grid system driven by the very popular Globus Toolkit. Last but not least, the initial SOAP specification standardised by the W3C, coming from Microsoft is the basis of the .NET architecture and leverages the enormous development community using Microsoft Visual Studio and the Microsoft Component Framework (Microsoft Foundation Classes).
- Figure (2.2) shows the atomic unit of interaction for a complete Web Services call. After steps 1 & 2 have been completed, resolving a particular service address, they do not need to be repeated for subsequent Service requests and responses (steps 3 & 4). What should be evident in the figure, is the simplicity of the collaboration required from the (few) roles involved. A component diagram for this scenario has not been included, as it is trivial. There is no definition in the specification of particular components so each application may have its own arbitrary layers, instead the standard only defines the protocol that has to be communicated across the wire. Further using URLs as resource location descriptions and XML as the on-the-wire format, the whole protocol is human readable, which has greatly improved use. This is in stark contrast to CORBA, whose binary IOR addresses and focus on client and server side IDL implementations unnecessarily restricted developers. A more detailed analysis of the technologies is given in the next section (SOA, 2.2.3).
- FLOSS Frameworks that support Web Services include PHP natively since version 5, prior to that via the PEAR component framework, Java natively since 1.6, prior to that through several projects (most notably Apache Axis and Spring-WS), Ruby on Rails, etc. Since Web Services are native in newer versions of Java, they are native to the Java EE application container server. They are also standardised in the OSGi standard (as implemented by the Spring Framework). Web Services are thus compatible with a number of other technologies mentioned in this section.

• Agent-based

Agent-based programming is steeped in the folklore of intelligently “acting” software agents, which act as proxies for their human counterparts in a particular (virtual) environment (such as the World Wide Web), by carrying out tasks for them. Naively seen[16], this potentially represents vast time-saving for the human user as enormous amounts of information can be processed by agents. Early software agent literature is filled with detailed wide typologies of

[16]This observation is naive, because although lazy humans do not want to browse through masses of information, the browsing process may be vital in assisting them to ultimately possess enough knowledge to be comfortable with their choice.

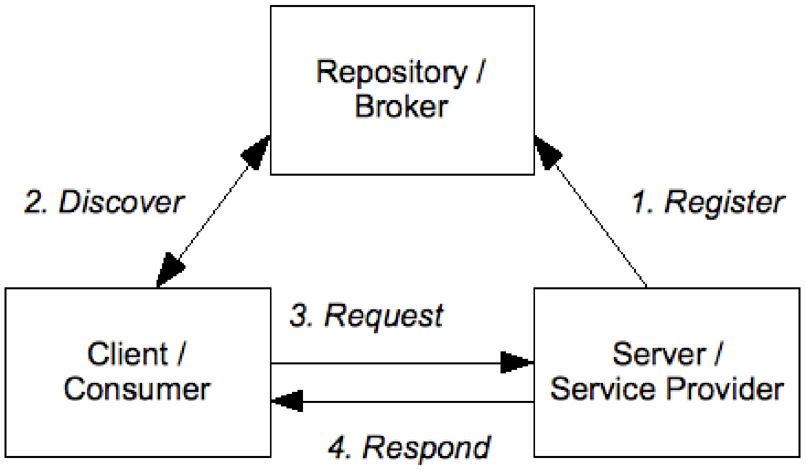


Figure 2.2: A distributed Event-Driven Architecture as a collaboration diagram: Web services collaboration diagram involving client, server and broker

the various types of agents [Nwa96], whereas modern literature has much simpler streamlined typologies [Wik09b]. In fact, Haag cited in [Wik09b] claims there are only 4 types of intelligent software agent: Buyer agents or shopping bots, user or personal agents, monitoring-and-surveillance agents and data mining agents[17].

Multi-Agent Systems (MAS) seem to be well suited to the implementation of a distributed middleware, such as the one we are trying to build, and JADE is one of the most popular scientific implementations of an agent platform that is freely available. Analogously to the abstract developments and understanding of agent systems, early JADE MAS platforms were implemented using CORBA middleware (IIOP) for inter-platform communications and JAVA Remote Method Invocation (RMI) for communication within a particular platform. Later JADE MAS extensions support Agent "platforms" built on the J2EE application container (using JBoss for instance or Spring Framework) and use web services for messaging[Tel09b]. Further [TTC09] demonstrated an architecture called PIASK using JADE MAS to provision eServices in the SLL. An analysis of their results is not available at the time of writing, so we cannot judge the efficacy of their method.

Figure 2.3 shows a component layer architecture of the FIPA Abstract architecture, which is embodied in the FIPA compliant JADE MAS platform. We see in the figure, that the basis of the FIPA architecture is its Message Transport Service (MTS), which connects all components of MAS and supports interoperability in terms of the Agent Communication Language(s) (ACL), resource directory services (via the Directory Facilitator) and content

[17]On the face of it, Agent Programming seems a noble aim to achieve, however, there is no evidence that the approach works. A quick look at the practical track record of the methodology shows no large scale, industrial implementations of such systems, where real intelligence could be located (e.g. Multi-Agent Systems show great promise of delivering truly new results). For instance the typology of intelligent agents suggested by Haag does not show any intelligence at all – these are merely lexical analysis automata which can process large amounts of data quickly and efficiently following simple or complex algorithms. Truly intelligent systems capable of logical reasoning are being built using semantic web technology (RDF, OWL, DAML+OIL, etc.), however semantic web systems are also still in their infancy. The "semantic web" has been heralded as the Web 3.0. Perhaps Agent systems have failed, because the WWW is not inherently a friendly collaborative place. Technophiles and hackers deliberately build traps for automata and spend great amounts of time reverse engineering complex processes to take advantage of programmed behaviour. [Art09]

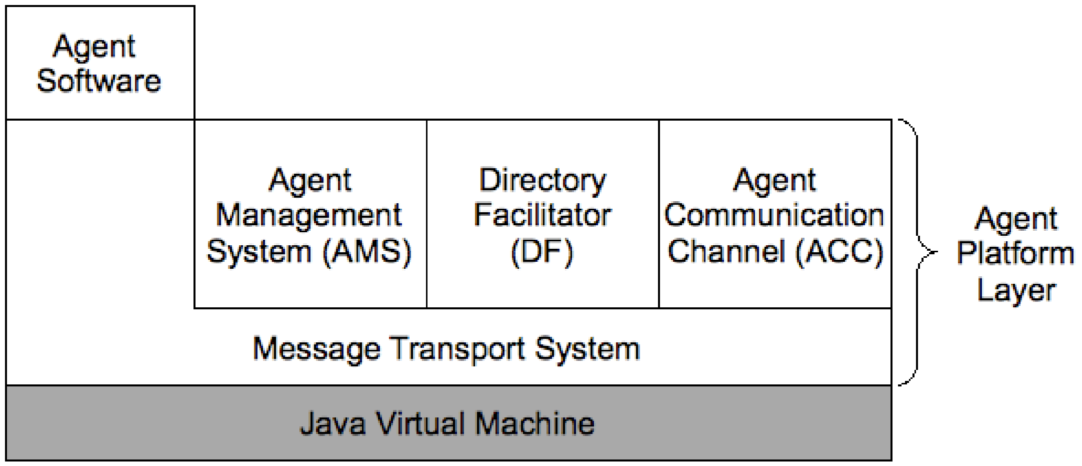

Agent Software
Agent Management System (AMS)
Directory Facilitator (DF)
Agent Communication Channel (ACC)
Agent Platform Layer
Message Transport System
Java Virtual Machine

Figure 2.3: Component layer diagram of the JADE / FIPA abstract architecture

languages used between differing FIPA agent implementations [Fou02]. The Directory Facilitator (DF) offers two types of resource directories: services and agents. Together with an Agent Management System (AMS) and Agent Communication Channel (ACC), the connector to other platforms, these facilities build a virtual environment in which programmed agents can be instantiated and can perform actions [BPR01].

- Parallel

  Grids and ad-hoc networks are contrasted in section 2.2.2 as they share many common aspects [FI03]. Here we look at the middleware characteristics of Grid systems. Specifically, we consider the most widely used implementation of the Open Grid Services Architecture (OGSA) / Infrastructure (OGSI), the Globus Toolkit. To see why Grid technology as embodied in the Globus implementation, is middleware, consider the following statement: "While every application [using the Globus Toolkit 4 (GT4) Grid software] has unique requirements, a small set of functions frequently recur. For example, we must often discover available resources, configure a computing resource to run an application or deploy a service, manage an application or service, move data from one site to another, monitor system components, control who can do what, manage user credentials and attributes" [Fos06]. GT4 thus implements most common middleware functionality, for specialised applications requiring large scale, efficient resource sharing, like the distribution of terabytes of astronomical data. Further, it is implemented using J2EE and Web Services, two basic middleware technologies discussed in this section, among others. This makes GT4 a very good candidate for middleware framework. We also found ICTD project in rural India constructing a service-oriented Grid as middleware for their eServices application [PD07]. Although the paper cites Foster's work on the Globus Project, it is not clear whether the authors used the Globus Toolkit as a basis for their implementation, which is called SORIG.

  Globus is clearly a larger framework of components than either of the technologies it uses, as is illustrated in

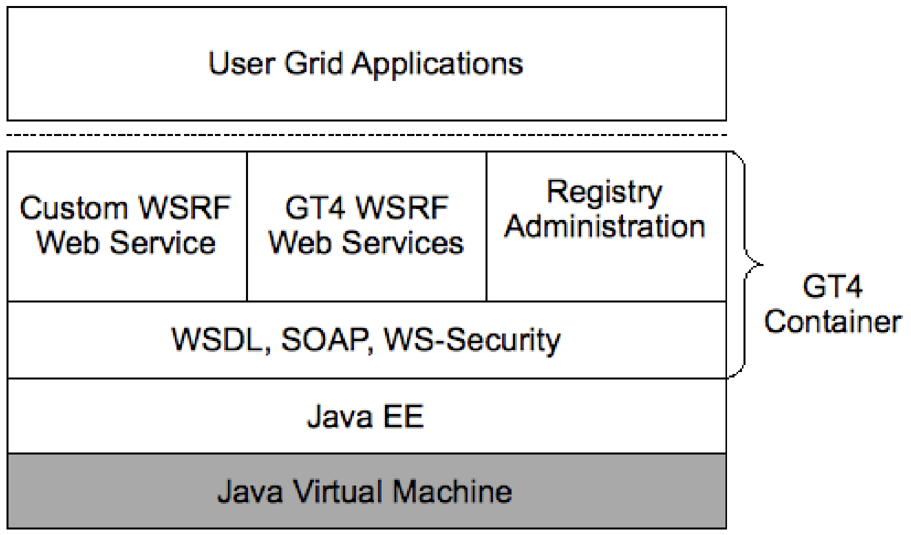


Figure 2.4: Component layer diagram of a *Java-based GT4 container*, as an implementation of the Open Grid Services Infrastructure.

the following figure 2.4. The Globus Toolkit (version 4) has three application containers: Java, C and Python. The Java container is the most advanced offering the most functionality. It builds on top of several Web Services specifications and provides implementations of these to take care of messaging, security, discovery and resources frameworks.

2.2.2 Ad-hoc Distributed Networks (P2P)

We understand P2P as an OSI application layer technology, which constructs mutable, ad-hoc overlay networks that have social significance through the principle of inclusion, i.e. all nodes in the network, even those on the edges participate in the function of the whole [WT09, FI03]. We discuss P2P technology because it is a good fit to the environmental or contextual conditions that we found in our field testbed. We argue that the main critical success factors of sustainable telecentre projects outlined in section 2.1.1, are a good fit to P2P technology.

We introduce them for 4 reasons:

1. *High Network "Churn".* Typically the WAN architecture will be as simple as possible to prevent expensive maintenance and blocking scalability etc. robustness in the face of high churn, through dynamic routing, and automatic routing setup (the complex network theoretic foundations can be found in Newman's work [New03]). Gummadi et al [GGG$^+$03]outline how Distributed Hash Table (DHT) topologies affect resilience of a network in the face of instability (churn). Huebsch et al [HCH$^+$05] describe the design decisions in creating an Internet scale database system using DHTs and P2P technology.

2. *Fully Distributed Resource Discovery.* Since the network can fragment through missing nodes, it must be possible

to use the resources that are still available in the network, even though significant parts are missing. Static resource tables or statically wired resource discovery services are not sufficient for the task. Peer-to-peer technology offers implementations of efficient fully distributed resource discovery algorithms, which can be used in such a scenario[ABZC08, Kro02, TAA$^+$02, VDB04].

3. *The Social Dimension.* The champion, who is responsible for the operation of a particular telecentre, owns the telecentre and thus feels responsible for operations. Being part of a network may dilute the feeling of responsibility, and P2P technology can help to introduce business cases that keep the ownership of local resources, data and customers with each telecentre [Wer03, SSDN02, AH00]. Further P2P overlay networks can help provide distributed models for service discovery and add a degree of robustness to the network, especially if redundant WiMAX / WiFi links are present in the network (i.e. if a mesh architecture is used for the underlying physical network) [TAA$^+$02, Ora01].

4. *Overlay Networks.* The lack of skilled personnel in the rural areas means that the setup of new nodes and maintenance of existing nodes must be kept as simple as possible. Overlay networks implemented by middleware software, which create secure virtual networks on top of (hence overlay) any physical network, obviate complicated hardware setup requirements for the network (a frequent source of problems in the SLL). The software layer is also very easy to setup and it actually works better when the network is not over-configured. There is a wide variety of literature on the properties of differently structured overlay networks (a good overview can be found in Harwood's PhD Thesis [Har02]).

Further, ad hoc networking is applicable to the field of service oriented architectures. [ABZC08] have developed a SOAP *binding* for the JXTA P2P technology [18]. The binding allows one to distribute SOAP endpoint descriptions in an ad hoc network. This prior work touches very closely on the sort of middleware we wish to design in this project. SOAP is related to the Service Oriented Architecture family of technologies, mentioned below.

2.2.3 Service Oriented Architecture

The Service Oriented Architecture (SOA) paradigm is a business perspective on systems that may be built with open middleware standards. The business management orientation of the SOA concept is immediately evident in the following definition: "SOA is an application architecture within which all functions are defined as independent services with well-defined invokable interfaces which can be called in defined sequences to form business processes" [CHT03], although there is some contention about an exact definition [Sta06]. Several sources of SOA literature we looked at stressed the importance of business risk management making ultimate decisions concerning the development of SOA based

[18]In their paper, Amoretti et al mention similar initiatives targetted at a Globus (Grid) environment, which should be considered within the context of the implementation of the middleware design.

systems[Mal09, Sta06, CHT03, AGA$^+$08, AP09]. SOA is used to structure enterprise-wide systems that can offer the entire enterprise access to information and shared services to maximise their utility within the enterprise, and also without (in the case of externally available services). [19]

[AP09] provide a good reason to look at the SOA paradigm when designing a new distributed middleware, which has to serve a fairly complex scenario in a flexible manner; they interviewed several Finnish SOA development companies who found that the main advantage of SOA was that it forces developers and systems architects to understand the business processes within an organisation and indeed across organisations. The benefits of this are that if the SOA design is performed with sufficient input from business, the system becomes flexible enough to very quickly create new business processes and models potentially with the inclusion of third party services, which translate into new products for the organisation. [20]

SOA is generally understood to use Web Services technology (SOAP, REST), although an underlying technology is not specified. SOA develops value out of information hiding (for instance the ability to attach legacy systems to a modern infrastructure), automated resource discovery and guaranteed service levels, as well as the ability to compose services into new, aggregated service hierarchies, which express business processes (as mentioned above) [JNR09]. Thus SOA adds SLAs and composition of services into aggregated services as two additional business requirements made of middleware, which as a technical construct does not necessarily address such requirements.[BBW09] interviewed German companies who regularly use SOA and who create SOA solutions. Their findings were very similar to [AP09] mentioned above, and they additionally stressed that the agility of development was greatly improved in projects using SOA. Since this is one of the main goals of our effort, it seems that the SOA methodology may hold some lessons for us. So what is the SOA methodology and what kinds of techniques are available?

SOMA is one of the main commercial tools available, and is part of IBM's Rational solution package, with which a SOA can be modelled in order to ultimately develop a working system based on the architecture. The technique seems to aid the development process at each step from design to deployment of the finished software project[AGA$^+$08]. In fact, in [AGA$^+$08], an IBM research team makes the claim that "the construct of a service and service modeling, although introduced by SOA, is a software engineering best practice".

SOMA and Rational Unified Process (RUP) are supported by proprietary tools. We looked for a similar open process, which would allow us to follow a methodology just as rigorous as that proposed by the RUP, of which SOMA is an extension. *UML Components* by [CD01] introduces a method of the same name which defines step-by-step, how UML can be used to model service-oriented systems using component programming, for which open source tools exist. In table 2.1 we juxtapose the RUP process as applied within SOMA with UML Components. Thus we are assured that

[19]It seems to us that the SOA methodology may have been introduced to explain the value of middleware to business persons that hold the purse-strings in an enterprise.

[20]This very idea is noted by [CD01] as being the primary business motive underlying the adoption of component programming by large companies already in 2001. Perhaps theory and practice don't differ all that much?

RUP (Rational Unified Process)	**UML Components**
Phases	*Phases*
Business Modelling & Transformations	Requirements
Identification	Specification
Specification	Provisioning
Realization	Assembly
Implementation	Test
Deployment	Deployment
Management	
Objects Specified and Modelled	*Objects Specified and Modelled*
Services	Interfaces
Components	Components
Flows	Contracts
Information Flow	
Service Policy	
Component Contracts	
Artifacts	*Artifacts*
Development Plan	Business Requirements
Iteration Assessment	Business Concept Model
Project Measurements	Use Case Models
Periodic Status Assessment	Technical Constraints
Work Order	Component Specifications
Issues List	Component Architecture
Risk List	Components
Scenario List	Applications
Change List	
Use-case Model	
Software Architecture	
Execution Architecture	

Table 2.1: Juxtaposition of RUP and UML Components approaches to developing service-oriented architectures. The table shows similarities in development phases, but a disjunction in the focus and artifacts of each approach.

by following the modelling process outlined in UML Components we will still be following software engineering best practice as advocated by IBM.

A more technical exposition of SOA principles, which was published by a technical division in a large corporation, can be found in [Sta06]. The architectural design patterns Stal introduces translate very well into software modelling and programming constructs which can help streamline the developed classes and code. He has looked at several SOA which are in use in the corporate environment to which he has access. He has reverse-engineered the implementations used to create the SOA and discovered a number of useful software development patterns that can be applied to help ensure SOA principles are adhered to in a system. The benefits of this approach are that the Development Process gets important technical input to enable implementation of business processes with some assurance of following industry best practices, even if the requirements are incomplete. Some of Stal's key insights are that loose coupling is very important for SOA implementors. Several indirection patterns should be used when constructing communication channels [21], message

[21]Stal also mentions the Enterprise Service Bus a technology agnostic construct, which creates a fully connected network of services and applications, as a possible blueprint to integrate different middleware into SOA. Presumably Stal is thinking of classical middleware

transport systems and Class models (interfaces) in order to preserve loose coupling. Some of the patterns are: Bridge pattern, Explicit Interface pattern, Proxy pattern, Forwarder-Receiver pattern, Reflection pattern (Factory pattern), Store and Forward, or Message Queue Blueprint (very similar to the Staged Event Driven Architecture), Interceptor and Reactor patterns.

Another relevant technical exposition on SOA implementation concerns the work of Amoretti et al [ABZC08], and can be found in section 2.2.2.

2.3 Conclusion

The aims of this literature review are twofold:

1. it bootstraps our understanding of the problems faced in ICTD projects, so that we can suggest an appropriate solution, and realistically gauge the feasibility of our project idea, and
2. it reviews useful technologies, which we might use to bootstrap our development effort in order to achieve goals more completely, and which might prevent us from reinventing the wheel.

We find that sustainability is the holy grail of ICTD projects. Since there are no ready solutions to the problem of sustainability, any solutions we present in this thesis should respect the best practices we mention in this chapter. We also find that our solution should be geared toward deployment in network islands, which are a kind of network topology we expect to emerge in the mid-term and then to vanish again.

We find that the family of telecentre software currently in use, does not offer us any possibilities for reuse. Similarities to the telecentre (DAN) we envisage are superficial. The difference lies largely in the fact that the software running on behalf of the end user (i.e. within the end user's process space), with the exception of the timer which is timing use on the workstation, is all agnostic of its runtime context. The software may as well be running on the home computer of the child of a wealthy family in an industrialised context, as on a telecentre infrastructure tailored for MRAs.

MRAs represent an environment, which requires dynamic, robust technology, in terms of hardware as well as software, and which stand the most to gain from ICT enabled solutions which connect people in the MRA with the outside world, so that at least some services can begin to flow freely *in both directions*.

Since we cannot find an open source telecentre system which covers our requirements concerning extensibility, scalability and open interfaces, we consider creating our middleware from open source components which implemented all the required functionality of the system. In that case our contribution could be to combine these and to specify the standard interfaces available for the system. The next chapter discusses the methodology employed to design and develop the

based legacy systems which need to be integrated into an SOA; he glosses over the fact that an SOA itself is implemented using particular middleware. This demonstrates the difficulties of dealing with abstractions such as middleware and SOA which combine technological and business perspectives to differing degrees.

software in greater depth. We note that the methodology discussed in the following chapter is already introduced in this section and some comparisons with other literature are made.

This section situates the project in a overarching (business) and technological context, which allows us to continue, well informed, with the following chapter.

Chapter 3

Methodology and Requirements Engineering

"T'ain't What You Do (It's the Way That You Do It)"

Melvin Oliver and James Young (1939)

3.1 Methodology

This thesis, and the development work performed in its context, followed several cooperating methodologies. They apply to various aspects of the project.

Project Management Aspect – this project is an academic exercise and as such it fits very well into the first stage of a traditional project life cycle, which is called the project feasibility stage and consists of analytical and development work required to produce initial code supporting some hypothesis [Schwalbe, 2006]. We create a concept and implement a demonstrator. Deployment to the SLL and adoption by other projects as well as related evaluations, fall outside the scope of this project. Collaboration with the other SLL programming projects takes place in the sense that one or two existing projects could be integrated into the demonstrator. This eliminates the risk introduced by working with projects still under development, which are following their own critical path. Within this scope, the project was developed in an agile manner, i.e. following an iterative project life cycle. Agile project management, as described by Cockburn in [Coc05], does not attempt to plan an entire project from begin to finish rather it attempts to optimise investigative interactions in a team of programmers by keeping communication flowing. The goal is thus iteratively approached as one learns more about the problem itself and the possible solutions. One can thus say that agile project management is not designed for one person projects. However, since the definition of the problem was very imprecise, part of the project would involve experimentation to gain added insights into the problem, which fits well to an iterative method of development. Thus bottom-up and top-down approaches could be combined to allow technological insights, which were initially unknown, to shape the overall project design and requirements.

Architecture Design Aspect – the methodology we use in the technical architecture design for the distributed web services system followed industry standard methods for service-oriented architectures. We use the UML Components method, approximately, including some aspects of the Rational Unified Process. These methodologies can not be applied completely, not only because of time constraints, but because the project caters for future services that are not clearly defined. It also lacks customers who can provide the stories necessary to model complex and clear interface requirements, as is usual in industry projects. The essence of the methodologies is applied to bundle the available information and to clarify the clusters of services and data models that were applicable. Use of the model also lends the project some security as far as completeness of the work is concerned. The phases that are carried out are mentioned in table 2.1, with the exception of deployment.

Development Aspect – in keeping with the agile project management methodology, the development methodology adopted was one of test driven development (TDD), along with the agile "the code is the documentation" methodology. The reason for this approach is that in iterative projects, one wishes to rapidly develop working code and refactor at a later time, once changes to the requirements have been noted. Having a good test suite is essential to this manner of development. Further, good code documentation can ease adoption of the middleware being developed, and is essential.

A simple prototype as proof-of-concept is the aim of the development, so the emphasis in this thesis is on Architecture design and concept modelling.

The methodologies have been chosen according to the nature of the project and the constraints that have governed the project from the outset, apart from the technical constraints that are presented in detail in this section. These high-level constraints concern the nature of this particular Masters programme, namely, that students work largely on their own proposals and implementation of proposal and any group work is done very loosely within the confines of the SLL. The project is being developed by a single person who has to cover all aspects of the project implementation within a fixed amount of time.

3.1.1 Requirements Gathering Methodology

Our methodology for gathering the middleware requirements consisted of identifying the domains from which requirements could emerge, and then analysing and synthesising the requirements from each domain. Normally, requirements engineering is kept simple by focusing on the requirements of a client, who requests that the software embed some

Domain	**User Role**	**Investigative Technique**	**Notes**
Rural context	Village inhabitant	Observation, interview	This is the context of ICTD and SLL as described in section 2.1.
University context	Programmer	Publication review, code review, mailing lists use, interview	This is the context of the programmers, who use the middleware platform to develop programmes.
Business context	DAN operator	Case study review, publication review, brainstorming, interview	This is the context of the DAN operator, who needs to perform business transactions in order to ensure sustainability of the DAN.

Table 3.1: Context domains, the corresponding user roles and requirements elicitation (investigative) techniques used

"business logic"[1] in order to attain specific goals. Table 3.1 explains the domains we considered and the user roles (perspectives) considered in each domain. In our case, we used a combination of investigative techniques to elicit the requirements. Different methods predominated in different environments; the predominant methods are also mentioned in the table.

3.1.2 UML Components Methodology

The UML Components methodology described in [CD01] is a an excellent fit to the middleware application we are designing and modelling. There are three main reasons for this:

1. There is no main customer for the software, who might have a clear concept of the end-user goals, future business extensions and software design understanding, which could drive the development process. A structured process helps to keep any project on track.

2. UML Components is focused on "the server side of things" as the authors put it. Middleware is inherently related more to server software than end-user software like GUIs. In fact, in this project, we did not develop any GUIs – user interaction is restricted to log messages.

3. The methodology helps to produce a cleanly designed component based system, which has a service oriented architecture, and which is a good fit to the *a priori* requirements with which we set off into this project.

[1]The term business logic is a bit of a misnomer, as it does not refer to business in the commercial sense. E.g. the term can also apply to mathematical formulae in scientific software, which attain the goals of the scientist, but which are in no way commercially oriented.

3.2 Applications and Services Analysis

3.2.1 Current Applications

Current applications were surveyed by the author in detail, within the context of a supervisory role, by the author at the University of Fort Hare during 2007, 2008 and 2009. In this time, the author provided practical advice and gained insights into several student projects, as well as the overall methodology of the institute, the Telkom Centre of Excellence.

Additionally, the investigation of current applications revealed the following sorts of assets that may be relevant to our system. The most commonly used assets that could also be shared among the applications are their data models. These data models inherently express business models relevant to the context. These data models could also be shared among the applications with good effect. For example, the concept of a rural user, including demographical information and background, etc., as captured by the DAN operator at registration time, could be used by an eGovernment module to suggest grant or bursary applications that the user might be able to make. Another kind of asset are physical resources, such as printers and physical storage media that might be shared. Bandwidth and traffic are network related assets that may apply to the WAN island (i.e. accessing other DANs) and on the Internet link(s). Finally, software components (i.e. services) are also assets to the system. An example of services as assets may be the Media Platform Service, which offers VoIP or Video Conferencing functionality to applications.

3.2.2 Future Applications

In general, future applications will make much more use of mobile access to all services delivered via the middleware. GSM push services for audio and text should soon be included and are present in the eGovernment work. Applications running on mobile phones, that are connected to the network resources, either via WiFi or a cellular technology, promise to expand the size of the network and increase churn greatly, since each mobile device could represent a node in the network. Such services are already postulated in the eCommerce work, although not implemented. Semantic web technologies, knowledge systems, and other techniques making use of Ontologies and Reasoning systems are also a strong future thrust, as they have the potential of ably assisting the community members (who lack knowledge) in using the available services.

For each potential future technology, we suggest a solution that is taken into account in the requirements:

Alternative communications channels – the middleware does not ignore alternative communication channels such as legacy and mobile telephony. We suggest that Mobicents, the framework and container of the real-time communication solutions employed in the SLL, be linked into the middleware as a service that controls a resource. Likewise, the middleware system would act as a resource in the Mobicents world by granting access to DAN resources. Inter-process

Project	**Roles**	**Technology**	**Description of the Work**
eCommerce & extensions	Shop owner, system administrator	PHP 4, MySQL 4 (planning is proceeding on a rewrite using OSCommerce shop framework)	This is an eCommerce website with multilingual shop owner back end, customer front-end [Nje07] and extensions [JTT09, DMTT08].
eGovernment	Community member, system administrator, Home Affairs official	PHP, Zoop framework, Linux, MySQL 5, Apache 2	This is an eGovernment site with Governmental services for the eJudiciary site with services for tracking cases etc. [Jak09]
eJudiciary	Tribal leader, system administrator (champion), community member	PHP, Linux, MySQL 5, Apache 2	The eJudiciary web application allows the community to track tribal justice proceedings (which are a legal form of arbitration in South Africa), providing transparency and better information use and security [SM08].
Billing and Rating Engine	Subscriber, system administrator	PHP 4, Cake Framework, HotCakes Framework, FreeRadius, Linux, MySQL 4, Apache	This is an accounting system with novel billing and rating parameters, which works together with captive portal technology. It is flexible and extensible [Tar07].
Wireless Network Security Investigation	Intruder, hacker, user	FreeRadius, PHP	This is an overview of the assets (resources) on the network and a risk analysis of possible effects by intruders into the network [Muc08].
Media Services Platform	Caller, callee, administrator	Java, Mobicents Framework and Java Advanced Intelligent Networks (JAIN) / SLEE standard	The Mobicents based system is a converged communications system, which links Asterisk based Public Switched Telephone Networks (legacy telephony), with all IP (Session Initiation Protocol – SIP) based forms of communication, including Voice and Video. [TTW08]
Ontology Service	Indigenous tribal member, domain specialist, administrator	Java, JADE framework	This is an Indigenous Knowledge Systems (IKS). It uses a Multi-Agent Platform and Ontologies technology to capture and represent knowledge of community members. It also provides access to eServices in a semantic manner. [TTC09]

Table 3.2: Comparative List of Applications and Services Currently Implemented within the Siyakhula Living Lab

communications could be programmed in both worlds, on the one hand in a middleware service and on the other hand in a mobicents Resource Adapter.

Huge number of nodes – this problem needs to be addressed by specifying an additional topology or structure for the ad hoc network mechanism, i.e. P2P layer, of the middleware. It is common practice in P2P networks (e.g. Gnutella, Kazaa, JXTA, BitTorrent [SRS05, TAA$^+$02, Coh03]) to track usefulness of nodes in terms of duration of presence, network capacity and processing speed. The most useful nodes are often promoted to super-peer nodes. Using two-tier or multi-tier organisation the P2P layer can be optimised to support ever greater numbers of services running on ever greater numbers of peers [2].

Semantic technologies – semantic technologies mainly affect the resource discovery layer and the data model layer of the middleware. Semantic web services are an object of much research [MSZ01]. We suspect that the technologies being investigated in that field can quite simply be applied to the communications layer being proposed in this thesis.

3.3 Requirements Specification

3.3.1 High Level System Envisioning

The middleware platform is a "glue" component which allows easy development, installation and administration (or operation) of functionality required at a network of rural Digital Access Nodes (DANs). More generically expressed, the middleware controls how components find and gain access to other components in the network.

IT staff can easily create new nodes in the network, because the middleware automatically creates an overlay network over the configured physical network. The middleware automatically finds new nodes if permitted by the physical network configuration and spreads service descriptions in the network.

Developers don't create stand alone applications, instead they encapsulate their functionality as applications or services extending the sum functionality of the system. In doing so, their modules may become discoverable on the network to the benefit of other applications and may gain access to the assets available on the network. Developers can thus focus on their core concern which is solving a particular problem correctly. Users of the network may rapidly gain new functionality as core functions do not need to be developed repeatedly.

Operators run DANs to earn a living. They control the services available to community members at a particular node and they also control how assets available from the node can be accessed by other nodes, if at all. Operators can easily install new services and applications from online repositories, as well as controlling services (start, stop, allow access to particular applications, deny access to particular applications).

The middleware allows users to access (or use) services in a more uniform manner. For example, they can log into

[2]For example, a simple heuristic for optimising the network, once such a two-tier approach were in place, might be never to allow mobile devices to become super-peers, because of their limitations in terms of power, bandwidth and processing speed.

the system once and use all services with a single set of credentials. Users must register to use the system. Their usage of the system is logged and they may be required to pay for system usage. Their use of the system is mediated by the operator, who can give them advice about which applications they may want to use. The operator also may provide assistance with all other aspects of system use. Users can be recognised by their traits – in our current context that could be a South African ID number.

All components in the system can subscribe to notifications about events that concern them.

3.3.2 Business Concept Model

Figure 3.1 depicts in a mindmap the concepts mentioned at various places in this document up to now. The three components (Service, Application and Middleware) have already been defined in the Introduction (Section 1.3), since they are key concepts to this work. The ideas of Assets and Data Models as they pertain to Services and Applications have already been touched on in the requirements analysis carried out in section 3.2. Users and different user types (or roles) are also mentioned there. The mindmap reduces these concepts to a minimal set: all types of community member who are characterised by low literacy levels and especially low computer literacy levels are members; operators are community members who are responsible for the operation of a DAN; administrators are technically versed IT staff who are able to troubleshoot networks and install software correctly. The neighbourhood is a networking concept which is the set of all the DANs that are directly connected to this DAN in a logical or physical manner. The requirements analyses also highlighted the issue of security. The middleware uses the two concepts of user credentials and access control lists (ACLs) to help maintain confidentiality, integrity and availability (CIA), which are widely accepted cornerstone concepts of information security.

3.3.3 *A Priori* Requirements

Contextual Requirements

For the contextual requirements analysis, we grouped the university and business contexts together, since they were identical in the SLL [3].

Table 3.3 lists the various requirements that we derived directly from these contexts for any arbitrary software solution, which might be required to run in the SLL. The requirements themselves are discussed in the literature review in section 2. These requirements are split into functional and non-functional groups. Although non-functional requirements are generally not a priority for functional prototypes (as the name suggests), non-functional aspects of applications running in the SLL determine some of the functional aspects of a middleware, so we need to look closely at these here. The third column ("Applies") in each subtable denotes whether we took the particular requirement into

[3]Business activities in the SLL were still being started at the time of writing, and they were led by the University partners

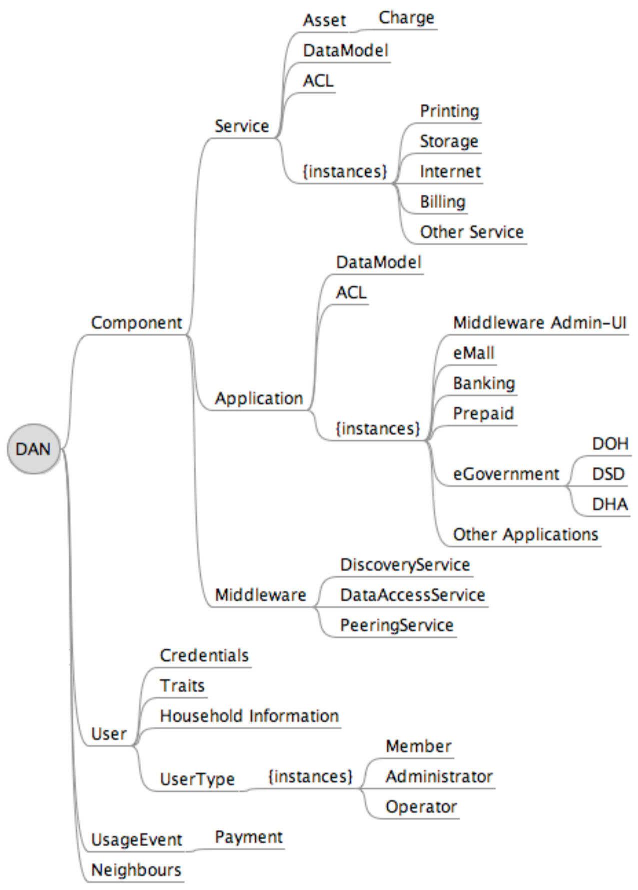


Figure 3.1: Business Concept Model Mindmap, showing the relevant business concepts that are dealt with by the middleware layer.

Non-Functional Requirements	**Importance**	**Applies**
Is Open Source	Essential	Yes
Is Robust	High	Yes
Is Inherently Secure	High	Yes
Installs on Diverse Architectures (Cross-Platform)	Medium	Yes
Uses Low Bandwidth (Internet)	Medium	Yes
Does not restrict Bandwidth (WAN)	Medium	Yes
Has Low Power Demands	Medium	No

(a) Rural Context: Non-functional requirements

Functional Requirements	**Importance**	**Applies**
Has Existing eServices	High	Yes
Allows New Useful eServices	High	Yes
Allows Mobile Access	High	Yes
Is Easily Managed (Authorisation, Access and Accountability)	High	Yes
Is Easily Administrable (Self-Configuring and Self-Healing)	High	Yes
Allows Rich Content (Audio / Video)	Medium	Yes

(b) Rural Context: Functional requirements

Non-Functional Requirements	**Importance**	**Applies**
Follows Best Practices	Essential	Yes
Provides Transparent Access to Resources	High	Yes
Is Programming Language Independent	High	Yes
Documents of Interfaces and Code	High	Yes
Contains Examples for Developers	Medium	Yes

(c) University/Business Context: Non-functional requirements

Functional Requirements	**Importance**	**Applies**
Loose Coupling of Components	Essential	Yes
Open Interfaces	High	Yes
Extensible Architecture	High	Yes
Support for Callbacks	High	Yes
Clear Component Architecture	Medium	Yes

(d) University/Business Context: Functional requirements

Table 3.3: Contextual requirements

consideration in the functionality of the middleware.

3.3.4 Assumed Network Characteristics

We next examine the physical networking context within which the middleware will be deployed. In 2005, Mishra et al (part of the TIER group at Berkeley) performed an analysis of the wireless technology options for the creation of the Akshaya network in a hilly rural area in India, which apart from population density seems to be similar to many African scenarios including the SLL. Although prices have changed, their core findings still hold. For rural underdeveloped areas, a low cost per unit of demand needs to be established. The sustainability of the system may in fact thus rest on the way it scales, which makes these assumptions important [MHF+05]. A further example network for a rural school network near the SLL in the Mbashe district is provided in figure 3.2.

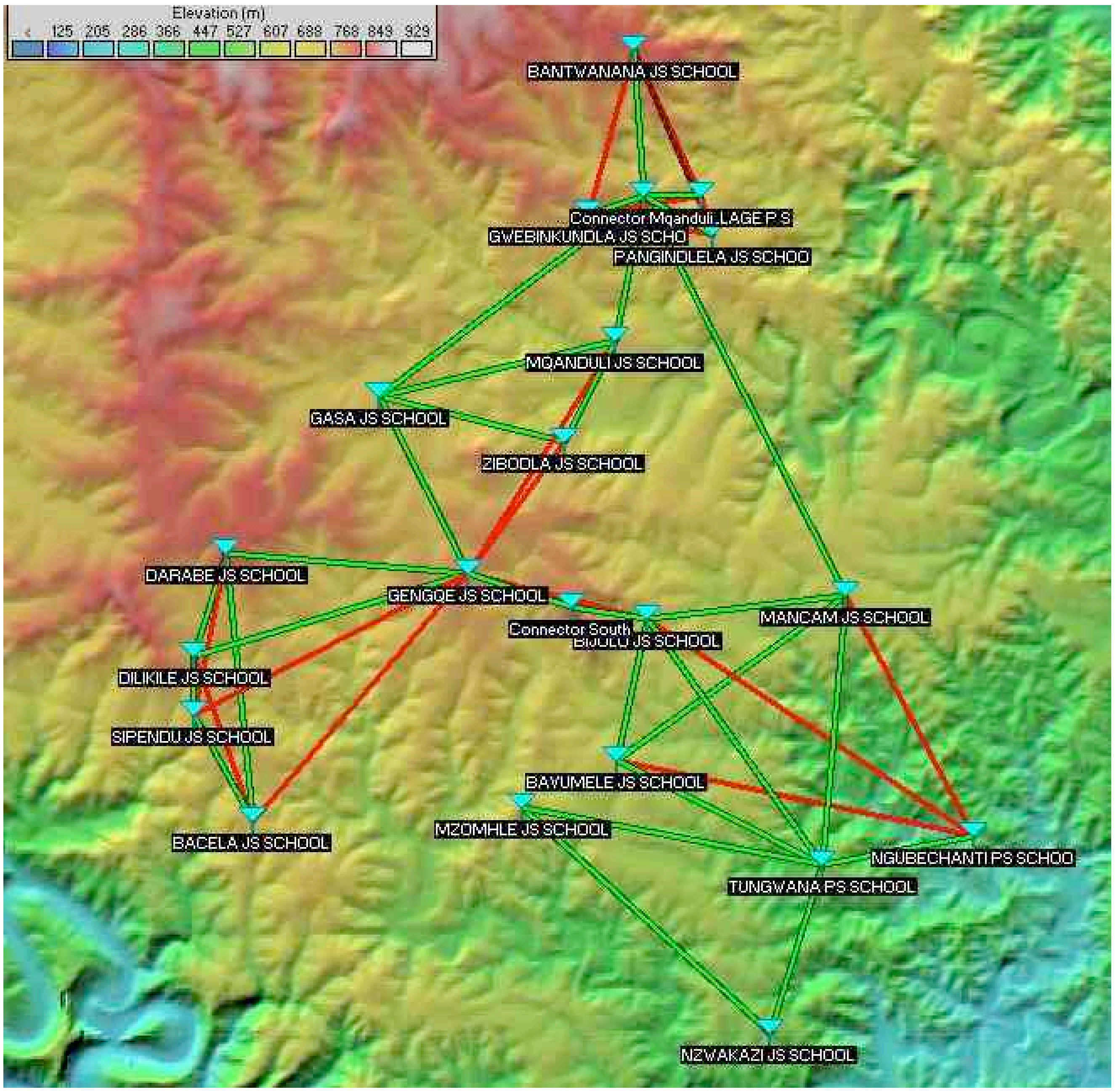


Figure 3.2: Radio survey graph of a 20 school network spanning 20km near Mqanduli, South Africa. Paths between nodes that are darkly coloured do not allow line-of-sight (LOS) between the nodes. Links between nodes can only be established along the lighter coloured paths.

The list of network characteristics in table 3.4 is derived from the characteristics of graph theoretic complex networks mentioned in [New03], as well as case studies. These are assumptions that will affect both the kinds of physical infrastructure required for the correct proportioning of the middleware system (i.e. regarding particular numbers of users, numbers of digital access nodes and characteristics of the network topology), as well as allowing context based predictions of usage patterns relevant to the middleware system (as analysed in section 4.5.2). To sum up the table, we expect a loosely connected network, with high local uptimes, and potentially a high number of nodes and users, although in the immediate case, node and user numbers will be low.

- We note that the uptime characteristic, is interpreted as

  $$uptime = \frac{(MTBF - MTTR)}{MTBF}$$

  where MTBF = mean time before failure, and MTTR = mean time to recovery, as per [Bre01]. We chose the values of 0.95 for WAN uptime, which is slightly higher than 0.9 for Internet uptime. These are rough experiential values and they take into account that it is more difficult to get the ISP to fix problems than SLL technical staff.

- The clustering coefficient of the network allows us to determine how many links there will be in the network on average given a number of nodes. The clustering coefficient can be worked out for existing graphs, by counting the number of unique connected triangles in the network. Another question this value allows us to answer is: ”Given that A and B, and A and C are friends, what is the likelihood that B and C are friends?” The current testbed network has a clustering coefficient of zero. There are no redundant links, and the star topology employed means that only one node has “friends”. This is not a desirable state for the network. The Akshaya network mentioned above also had a clustering coefficient of zero. The network shown in figure 3.2 was designed with redundant links where possible, in hilly terrain similar to the Akshaya and SLL contexts. The (maximal) clustering coefficient in this network is:

  $$C = \frac{6\,\times\,number\,of\,triangles\,in\,the\,network}{number\,of\,paths\,of\,length\,2} = \frac{6\,\times\,8}{140} = 0.343$$

  The equation is taken from Newman [New03]. We will use this value as our C value. While many real networks have a value tending to zero or actually zero, rural mesh networks and an easier lie of the land may result in much higher C values, so we feel this choice is justified.

- The PFNet project final report includes several detailed usage statistics describing how the network was used by the community in a deep rural Pacific Ocean context [CLS+05]. In this report, 3.1%[4] of the population living near each DAN made use of it. 4 of the SLL DANs are situated in communities totalling 15000 persons [PPKG]. Using the PFNet data to extrapolate the information from the SLL, we might expect about 117 users to make use of the

[4]Statistical outliers were removed to derive this figure.

Network Characteristic	**Estimated (Assumed) Value**
Test Network Nodes (n)	$n \approx 10$
Max Number of Nodes (N) in Use Case	$N \approx 10000$
WAN Uptime (t_w)	$t_w \approx 0.95$
Internet Uptime (t_i)	$t_i \approx 0.9$
Clustering Coefficient (C)	$C \approx 0.343$
Average number of seats per DAN (s)	$s \approx 10$
Number of individual users per month, initially (u_t)	$u_t \approx 117$
Average number of user visits per month, initially (v_u)	$v_u = u_t \times 3 \approx 351$
Average number of uses per user visit, initially (u_u)	$u_u \approx 1.5$
Number of individual users per month, later (u_θ)	$u_\theta \approx 350$
Average number of user visits per month, later (v_μ)	$v_\mu = u_\theta \times 8 \approx 2800$
Average number of uses per user visit, later (u_μ)	$u_\mu \approx 3$
Average Number of Users per Node at peak time(p_u)	$\begin{cases} if\, p_u \lessapprox s, & p_u = \frac{Number\, of\, Visits\, p.mo.}{30\times6\div7\times2\times3} \approx \frac{v}{154.3} \\ if\, p_u \gtrapprox s, & p_u = s \end{cases}$

Table 3.4: Networking assumptions relating to working within Network Islands

DAN per month. Further the PFNet data suggests that these will visit the DAN roughly 3 times a month and use it for 1.5 different tasks (on average). These estimates are reflected in the table.

- We presume that usage statistics will increase once the eServices are released, and these assumed figures are also reflected. We assume a roughly 200% increase in the number of users (350), two DAN visits per week (roughly eight per month), and usage of the DAN for three different tasks.

- We also introduce a formula (p_u) for the number of users at a DAN at peak time, based on a three step distribution: level zero (zero users in twelve hrs), level one (50% of users in nine hours), level two (50% of users in three hours). Average seats per lab is taken to be ten (we have roughly 73 seats in seven DANs). DANs are closed one day per week and the number of days per month is taken to be thirty. Peak time users cannot exceed the maximum number of work places at the DAN and respect the day off (i.e. peak times only occur 6 days of the week). Further users are present for an average of an hour per visit, to keep things simple.

P2P networking assumptions

As presented in the section 2.2.2, ad-hoc networks these days are not naively structured, assuming a fully connected graph. Instead nodes are clustered into neighbourhoods, which is a concept we believe can work well in our context. N as assumed by us does not pose a major stumbling block to even naive P2P networks. Gnutella introduced a two-tier topology only when the order of a million nodes was reached [SRS05]. Project JXTA presents a P2P protocol and implementation, which in a naive operational mode can support N nodes [TAA$^+$02]. We can thus safely assume, that we do not need to concern ourselves with specifying a P2P topology and can leave this aspect to future work.

3.3.5 Use Cases

In this section we introduce a few detailed Use Case descriptions, which focus on core issues in the system.

The use cases highlight why loose coupling is essential in the system. They demonstrate that security is a concern in the middleware which must be addressed in the implementation. They also illustrate what sorts of associations the following constructs have in the middleware: notifications, remote components (components located at other DANs), the roles of human beings and software components as actors within the middleware context.

A point to note is that the user interfaces, which are used to allow the human roles access to the middleware system layer, are not within the scope of this project. The user interfaces themselves can be seen as an application, as indicated in figure 3.1.

The use cases are essential to the methodology we employ, as the component interfaces we will design in the next section are modelled on this information .

Add DAN

Initiator: Administrator Goal: Add a node to extend the system

Main Success Scenario:

1. Administrator starts a fresh node.
2. Administrator waits until that the node has booted onto the network.
3. Administrator logs onto the network.
4. Administrator verifies the current node on the network.

Extensions:

- 2.a. Node network boot fails.
  - Fail. In this case, the Administrator must troubleshoot the node and network connections.
- 3.a. Login fails
  - Allow several retries, then fail if still not successful.
- 4.a. Verification fails
  - Fail.

Register User

Initiator: Operator Goal: Add a new user's information to the system

Main Success Scenario:

1. Operator logs onto the DAN.
2. Operator captures which applications the user wants to use.
3. Operator captures user information into the middleware.
4. The middleware notifies the neighbourhood of a new user in order to verify uniqueness. [5] It does not share private information over and above e.g. ID number, in this step.
5. The middleware verifies user uniqueness.
6. The middleware notifies components that have registered for the "New User" event.
7. Components verify the captured user information.
8. The middleware returns a new user ID to the operator. The registration was successful.

Extensions:

- 1.a. Login fails
  - Allow several retries, then fail if still not successful.
- 5.a. The user already exists on another DAN.
  - The "Clone User" use case is invoked instead, with the user ID found, and this use case terminates.
- 7.a. Component verification fails at one or more components.
  - If access is denied to a specific application for whatever reason, then the scenario fails for this component, but succeeds otherwise.
  - If data verification fails, then the operator must correct the problem (e.g. telephone number mistyped) and resubmit the information.
  - Component specific errors may occur, they cannot block the registration of the user in general. Their success or failure affects only the component itself not the entire scenario.

[5] Since messaging is asynchronous, notification steps always succeeds.

Clone User

This use case is interesting because it concerns other DANs.

Initiator: Operator Goal: To create a local account for a user with a remote account.

Main Success Scenario:

1. Operator logs onto the DAN.
2. Operator enters user ID to clone.
3. System creates a local account for the user (which is available even if the remote DAN is offline).
4. Operator updates user information pertinent to the clone account.

Extensions:

- 1.a. Login fails
  - Allow several retries, then fail if still not successful.
- 2.a. The user ID cannot be verified in the neighbourhood.
  - Fail. A new user account should be created.
- 2.b. The original user account information is inaccessible (DAN may be offline).
  - Queue a clone request, and
  - create a clone account without the original information (this can be marked as pending).

Start Component

The operator needs to know which services and applications are available on her DAN, and thus she is required to explicitly authorise the running components. Some of these may share assets of the DAN.

Initiator: Component Goal: To authorise a component to perform actions in the DAN and in the neighbourhood

Main Success Scenario:

1. Operator logs onto the DAN.

2. Operator authorises the component to start locally (within the DAN).
3. Operator authorises the component to start remotely (it has access to the neighbourhood and vice-versa). The neighbourhood is notified.
4. Operator sets which users or user groups may use the component.

Extensions:

- 1.a. Login fails
  - Allow several retries, then fail if still not successful.
- 2.a. The component cannot start.
  - Fail. In this case, the operator has to perform some trouble-shooting. However, compatibility issues should be excluded, because the system should only allow the operator to start components whose versions have been checked for compatibility.
- 4.a. Component dependencies are broken. The operator selects a too restrictive use setting and dependent components cannot continue functioning.
  - This is not an error, it is a feature of the system. The operator can restrict access to any non-core services and influence all dependant components.
  - The operator is notified of the dependancy impact of her action.

Use Service

All UsageEvents in the system are generated by an interaction between components and this use case comes into play.

Initiator: Component Goal: To successfully complete a Usage Event

Main Success Scenario:

1. Component looks up the required service, using the DiscoveryService of the middleware.
2. Component sends request to the service, accompanied by credentials for the specific request.
3. Middleware checks credentials.
4. Service processes request, resulting in a UsageEvent and returns a response to the request.

Extensions:

- 1.a. The service being requested cannot be found.
  - Fail. In this case, the Operator must be notified of the malfunction of an application. Troubleshooting steps for this error should be followed.
- 2.a. Service is at a remote location which is offline.
  - Is this a time-critical UsageEvent? If yes, fail.
  - Otherwise, queue the message for later transmission when the service is available.
- 3.a. Credentials do not have the rights required by the UsageEvent in the services ACL.
  - Fail.
- 4.a. An error occurs while processing the request.
  - Time-critical error – the event occurs too late, e.g. submitting a grant proposal after the deadline.
  - Communication error – the process involves communication with a 3rd party, e.g. Home Affairs, and the communication could not take place:
    - If the request can be queued, queue it and continue.
  - System error – a technical error occurred.
  - Return appropriate error response.

Check Credentials

This is an included use case, which is used by all processes that require a user to be logged in to the system.

Initiator: Component Goal: To check whether the component's session token is in order.

Main Success Scenario:

1. A request specifying caller and callee components as well as a session token, is presented to the middleware.
2. The middleware verifies the session token is in order.

3. The middleware approves the token.

Extensions:

- 1.a. There is no session token.
  - User credentials are supplied by the user.
  - If the credentials are correct, a new session token is issued, and the process continues with step 4.
  - If the credentials are incorrect, fail.
- 2.a. The session token is stale or incorrect.
  - Continue with step 1.a.

Check User Rights

This is an included use case, which is used when attempting to access any component for the first time in a session.

Initiator: Component Goal: To check whether the component may gain access to a particular component.
Main Success Scenario:

1. A request specifying caller and callee components is presented to the middleware.
2. The middleware verifies that the ACL of the callee allows access.
3. The middleware allows access.

Extensions:

- 1.a. The callee is remote.
  - The security check is passed to the remote middleware instance.
  - If the remote service allows access, proceed to step 3.
  - If not, fail.
- 2.a. The ACL does not allow access.
  - Log unauthorised attempt to use a service.
  - Fail.

3.3.6 Use Case Synopsis

The UML Components method stresses the fact that as much precision is required in the initial business modelling as is in the later software engineering. As programmers, we often approach a problem with a set of programming patterns and pre-existing knowledge which help shape how we view the problem at hand. For instance, Java programmers may tend to naturally stay away from the concept of multiple inheritance in business objects, without a deeper realisation of the motivation for the design decisions. C++ programmers might not be so shy. As Software Architects, we are required to distance ourselves from the implementation and to model the business processes and operations as they are required by the end users and business owners.

For this reason, figure 3.3 includes end-user roles in the use case diagram, even though we will not be creating any user interfaces in this project. Considerations of how user roles will interact with the system are a problem of implementation and can be ignored at this point. But it is essential to understand what potential actors are going to be doing with the system, in order to model the operations and interfaces required for our middleware.

Figure 3.3 is a very high level diagram which shows how the concepts introduced in the mindmap in figure 3.1 interact. This figure is also meant to demonstrate how simple the operations we are trying to model in the middleware are from a business perspective and to which user roles the actions are available (we list the most basic user type as "(Community) Member", because basic members will usually be community members, however, they may also be teachers or others, who are not community members). It is quite clear that the middleware performs very few actions, which is correct in our opinion[6]. For this reason, two of the use cases presented in section 3.3.5 involve software components as initiators. Software components represent the actor roles within the system and can be classified into two major types. Their usage interactions are demonstrated in figure 3.4. This is a technical business process design which implies that we will be using the proxy design pattern to protect assets. Services can thus be understood as proxies.

3.3.7 Shared Assets and Access Rights

Assets in the system must be protected. Often these are resources that are consumed through use (in our parlance, they are consumed through *UsageEvents*), such as Internet Traffic limits and hard disk space. We try to make provision here for all the core assets of a Digital Access Node. In addition to access rights, accountability also needs to be maintained. Accountability is essential to maintain sustainability of the telecentre, as comprehensible charging for usage needs to be implemented.

Contrast the user models in specific applications: administrators or operators of the eGovernment service, or DHA official have different rights in that system. These are protected by the application and for the purposes of the middleware

[6]Very basic components that are potential single points of failure should be as simply and clearly designed as possible to prevent errors and bottlenecks in operation. This is especially true when a number of different programmers from different backgrounds are intended to use the system. These ideas echo *Occam's Razor* – a logical principle attributed to William of Ockham – that can be used to guide abstract design processes, in its formulation as a law of economy. Typically it is used when formulating scientific theories.

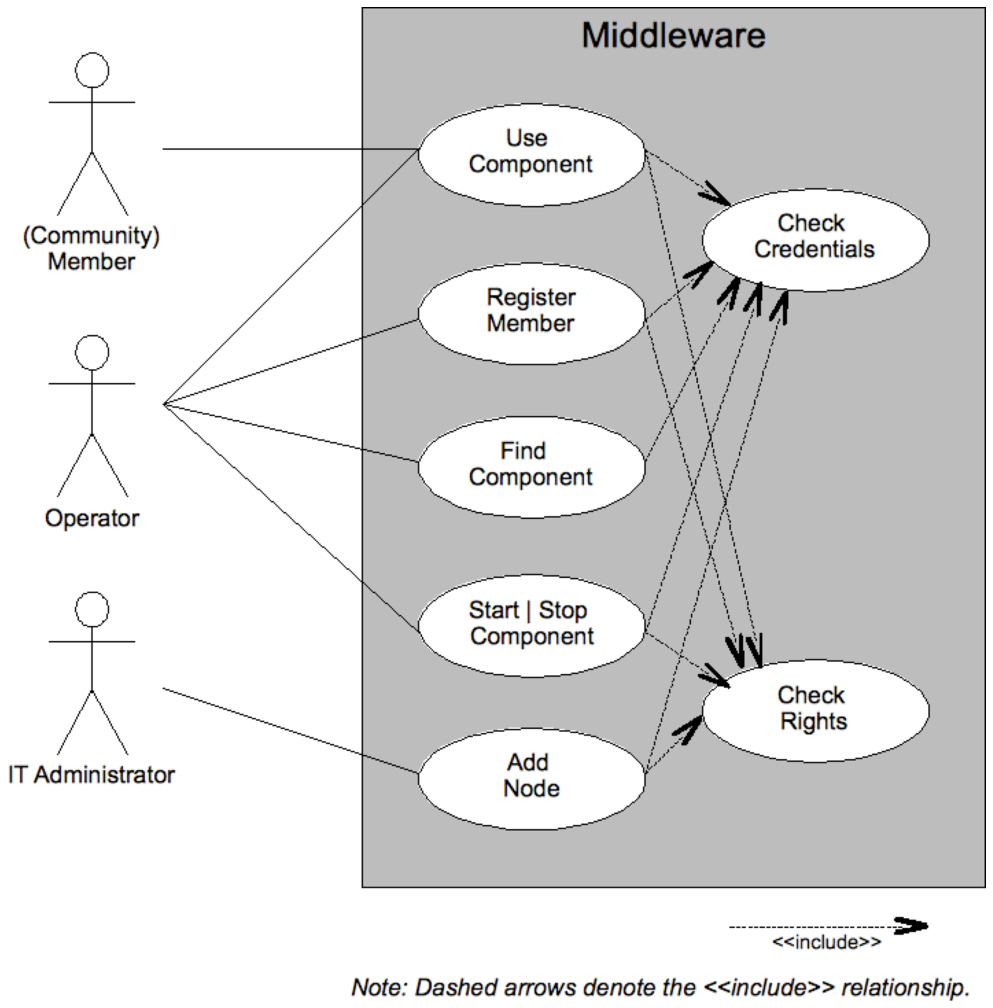


Figure 3.3: High Level Use Case depicting Roles of Actors useful to the Business Process Modelling

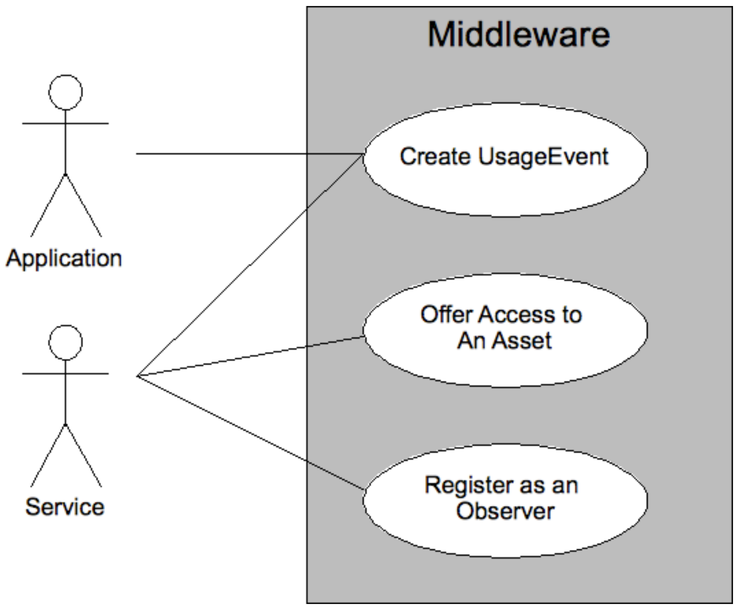


Figure 3.4: Internal Use-Case showing Usage Scenarios for the different Component Types: Applications, Services and Middleware.

these are just ordinary members. However, the application must be able to attach its own data models to the entities provided by the user manager. Indeed, the eGovernment service must be able to access information stored by the eHealth service, for instance, in order to assess which grant applications can be made. The access to the eHealth services information is made with the credentials of the user as issued by the middleware. Abuse by the application is limited, because the credentials' validity will lapse with time.

Thus the identity of the user of an application is paramount to which operations can be performed by the application and the proxy services acting on its behalf. The user credentials are passed down the chain of operations and and are used to either allow or disallow each operation in the chain.

3.4 Technology Decision-Making

3.4.1 Motivation for Open Source

We see Open Source philosophy as a good fit to the requirements context of this project, aside from the fact that it is an essential requirement to the project, owing to this project being part of the Siyakhula Living Lab.

The functional requirement that the middleware "allows new, useful eServices" seems to point to open source systems. They can potentially spur adoption among "hobbyists" and entrepreneurs who will provide necessary input about required eServices as well as usage information. The open source community may feel inclined to assist with the development of the project for altruistic reasons. Entrepreneurs will know that they can adapt the software to their own needs and students will be able to study the code and suggest improvements. Users will be empowered and may be activated to develop their own alternatives in the medium-term.

The functional requirement that the middleware present "Open Interfaces" to developers also points at open source systems. While it is perfectly normal for closed source systems to present open interfaces, with time the documentation effort grows as documenters struggle to keep up with code changes that may affect interfaces. This affects sustainability of the system. Open source systems are not immune to this problem, but they do offer developers some assistance in keeping interfaces transparent [author's experience].

Finally, our results show that FLOSS is not only free but that the technical quality of several relevant FLOSS projects is of a very high standard (see section 5.4).

3.4.2 Motivation for Java

Most of the applications developed within the Siyakhula Living Lab are written in PHP (see table 3.2) and use the LAMP stack. PHP is very well suited to rapid application development (RAD) to test proofs-of-concept and to create efficient web applications. Especially for smaller applications and when using optimising caching systems such as those provided for free by the Zend system, PHP through its close binding to the Apache web server performs very well. This is why many developers of smaller projects prefer to use PHP [author's experience]. A middleware platform on the other hand has other requirements. While efficiency and rapid application development are par for the course in a Masters project, a middleware platform is potentially a single point of failure for the entire system [Wik09a] and as such should be developed using the same requirements as we would have, say, for an operating system as regards stability and security. We examined the requirements lists for further requirements that could influence these considerations. The following non-functional requirements were found to be relevant in addition to the two already mentioned (robustness and security): "follows best practices" and "cross platform support". While there are several general purpose programming languages around, including Java, C#, PHP and others such as C++, python, ruby, we refrained from performing a contentious and time-consuming comparison of all these, and focused on finding out whether Java fulfills the main technology requirements, i.e. whether it is *good enough*[7].

Robustness – it is easy to write poor applications that can fail in most programming languages, and thus the question of system stability is often more dependant on the developer and system designers than on the programming language. Java's concept of compiling into byte-code and then executing controlled byte code is a model that has since also been adopted by the Microsoft ASP framework's C# language. The method allows implementers of the Java Virtual Machines

[7] While the phrase "good enough" has become associated with agile software development, the concept has also gained ground in the study of complex ecosystem and evolutionary principles in Biology. The combined studies of genetics and symbiosis describe complex networks of heterogeneous organisms that co-evolve to remain alive and flourish in a competitive environment. Ultimately bio-diversity is a positive aspect of any ecosystem and affords it a buffer against environmental interference. As time goes by, the system co-evolves to solve new problems for mutual benefit [PA00]. By a similar consideration, the middleware system should allow various competing perspectives, that will allow an evolving network of functionality. The choice of programming language from this perspective is irrelevant.

(JVMs) to employ elaborate garbage collection techniques to recycle system resources. It is mainly this aspect of Java, which lends performance boosts to complex and dynamic Java systems such as examined in [TAW03]. Java also has a longer track record as far as test suites are concerned. JUnit is currently in its 4th version and can be integrated into development projects with minimal code impact using Java aspects. JUnit has been in use since 2002 at the latest and has since been ported to several different languages and programming environments.

Security – A security model is built into the Java language, so that all aspects of security can be addressed from loading of code from archives (so called JAR files) to selective execution models. Further, Java security is enhanced by its security frameworks, which allow e.g. single sign-on in enterprise systems.

Best practices – Java has been an enterprise server open source technology of choice for some time [GvLPF01]. Many mission critical backend systems at banking institutions in South Africa and Internet portal backends in Germany are based on the technology [author's experience]. As mentioned above, several modern programming languages reuse ideas originally developed in the Java world. The Java Community Process (JCP) is a managed community process which has produced several of the useful extensions to Java. It is an integral part of Java's success allowing Java to grow through a series of vetted and peer-reviewed extension to the language and framework.

Cross-platform support – Java has a proven cross-platform record, which means that middleware installations will be able to run on most host operating systems, including mobile systems (perhaps with some adaptation to the middleware) through the Java 2 Mobile Edition (J2ME). This is important in the heterogeneous DAN environment, in which a wide variety of donated and machines purchased by third parties, co-exist side by side.

3.4.3 Motivation for Web Services and especially SOAP-based Web Services

We present an argument for (SOAP based) Web Services in bullet form here. The argument for a web service based technology is quite simple to make, while the extension of the argument to SOAP follows from the arguments for openness and best practices.

- SOAP is a stable standard since 1999.
- It is widely used, e.g. by Amazon, Google, banking systems in South Africa, etc.
- It is implemented and supported in a wide variety of technologies.
- It is thoroughly typed and well-defined, thus eliminating ambiguities in communication.

Counter-arguments are that it is very slightly bulkier than REST (a small constant order larger - not an order of magnitude), and also requires more heavy-weight marshalling (e.g., XML vs. Javascript objects (JSON)) of objects. The counter-arguments are generally a result of the last advantage listed above.

3.4.4 Motivation for P2P (/ Grid) vs. Central Solutions

In section 2.2.2, we presented the key benefits of ad hoc networking as represented by P2P and Grid solutions and some of the literature that supports these positions. Here we provide further motivation related to the specific requirements of the system, as analysed in this chapter.

Ad hoc networking contrasts against client-server technology, which is also a very commonly used form of distribution of information. Systems which handle mission critical data are often designed as client-server systems as one can reduce security complexity and data duplication by having all critical access (e.g. to a bank account) occur via a single gatekeeper, who is responsible for and can guarantee data integrity. We have to weigh the data integrity guarantees made easier by the client-server architecture against the *single point of failure* introduced by a single gatekeeper to any services. According to our assumptions, each node will on average be offline from the rest of the network for 18.25 days per year (see 3.4, we calculate, $(1 - t_w) \times 365 = 18.25$). This means that any node, which does not host the central service, may lose access to the central service for 36.5 days in the year. Several day long down times can cause users to lose faith in the service and not wish to use it any longer [author's experience].

Using ad hoc networking, services can be replicated in the network much more easily. Any DAN, which later installs an Internet connection, can offer this resource to others in the network to share costs. The network becomes more easily extensible – a key requirement of our system.

Another important argument for the introduction of P2P or Grid techniques in this work is the end user view of the DAN. Once an end user (e.g. community member) has grasped how a service is to be utilised, she does not need to be concerned with how or where she accesses the service. The DAN may be any gateway to electronic services, such as a cell-phone, telecentre operator (the human interface), or a PC in a computer lab. While client/server applications require that the server be accessible all the time, a P2P application need not be. This is particularly relevant for push applications where users create content (or data) of some sort. For example, a community crafter can perhaps offer new artifacts for sale using her cell phone, while in the field and with no network connection. She can perform the steps she has learned from anywhere in the network[8] (using for instance a USB drive at a connected school laboratory).

3.4.5 Conflicting Requirements

The technology underpinning the middleware platform must take cognisance of conflicting requirements. As mentioned in table 3.1, requirements come from several contexts, which often makes prioritisation and detection (at design time) of the conflicts difficult. We identified the following main conflicting requirement domains, which need to be balanced by the

[8]This introduces the problem of phishing. Clearly, users who do not understand what they are doing and instead blindly follow a recipe do pose a security threat to the system, as they can be led to type credentials into the wrong interfaces, thus handing over their credentials to unauthorised third parties. A number of technical means can be used to reduce this threat. For instance, a cell phone can be configured to represent the user, thus allowing the authorisation step to be skipped when accessing the system from the cell phone. The user might then restrict themselves to only accessing the system via their cell-phone, which would increase safety.

solution and technology proposed: System Flexibility vs. Efficiency (during runtime) and Ease-of-use vs. Complexity (at development time). Essentially, both of these domains are attributable to the requirement of "Loose Coupling". Loose coupling can speed software development in almost all cases, however it slows runtime execution because of the levels of indirection required to implement it. Some non-functional requirements affecting the design are analysed in the following chapter. Taking a quick peek ahead (see section 4.5.2), we find that run-time efficiency is not a concern. To keep customers happy and save power though, latency must be kept to a minimum.

Openness vs Security is another perceived conflict of priorities. However, we do not believe that it really is a conflict. Linux is one of the most open operating systems, yet security problems concerning the Linux operating system hardly ever arise. Open interfaces mean that a water-tight security scheme needs to be in place to protect the system.

Rich Content and Low Bandwidth Requirements are potentially conflicting requirements. This conflict is handled by allowing rich content on the high-speed WAN, but restricting it across the Internet connection. As Internet access improves, this restriction will gradually be able to fall away.

Chapter 4

System Design and Architecture

Lego blocks are ideal components. The blocks can easily be used together and an elementary block can be used in many kinds of toys. Some pieces may be more specific, which make it possible to implement some parts of a toy with a more detailed look.

Lego Pamphlet 2009

4.1 A Fitting Design

In this section we try to create a fitting design. An intuitively fitting design, is one which:

- takes care of the requirements specified above,
- does not ignore conflicting requirements,
- and is implementable with the resources available.

Less obviously, the requirements demand that the design

- reduces dependencies between components (to allow loose coupling and component clarity) [CD01],
- and keeps component interfaces as simple and clear as possible (according to SOA principles – see section 2.2.3),

- in order to be backward compatible (this final point is discussed in the next section).

The middleware software design we propose here is suited to rapid application development and agile processes. We set up an infrastructure for easy pluggability of new services in the middleware, should new resources become available. These stipulate that the design incorporate flexibility into the system. An important aspect of our design is to balance the flexibility against the overall performance of the system. As many others have pointed out before (see e.g. [OW07]) system complexity comes at a cost, and is further inversely related to the performance of a system. Performance is also important, as systems based on efficient software require less powerful hardware, which in turn (in general, although this is not always true at low N) use less resources such as power. Power and other expenses can directly influence the sustainability of the project and must be saved wherever possible.

In this chapter, we initially look at a high-level view of the design as introduced in initial stages of the project to set the scene. The following sections derive interfaces and components from the requirements chapter (3) according to the described methodology. After that we compose the components into a system architecture. An example section presents a component decomposition for the eCommerce Application, demonstrating how other application builders can can use the middleware framework to shape their applications according to the needs of a rural DAN. Finally a flexible and extensible information model is proposed.

4.1.1 Top-Down View

An initial high-level design of the system (figure 4.1), which had informed initial implementation experiments is presented in this section as the basis for a high-level discussion about some of the core design considerations behind the middleware platform. The initial design underwent several stages of refinement[1], although at a high level, the concept has remained unchanged. A component-wise decomposition of the concept fits with the analysis in Chapter 3, thus changes have mainly taken the form of extensions and elaborations of the ideas.

Figure 4.1 (slightly amended from [WT09][2]) shows how the middleware forms an intermediary layer between Applications and Services, and allows interactions with other middleware installations. The figure also shows the physical communications network, illustrating that DAN's are expected to relay messages if no direct link exists between two communicating nodes. Since the P2P communications layer allows communications between all nodes, applications and services can assume that they are operating in a fully-connected neighbourhood of nodes. The middleware thus allows access to services at other DAN's transparently.

Initial implementation experiments show that the high-level design can be implemented in a Web-Services based framework. The experiments inform subsequent design steps such as the determination of the initial boundary of the

[1]The simultaneous use of top-down and bottom-up (see Methodology, section 3.1) approaches ensured that an overall view was maintained, while performing low-level implementational experiments.

[2]The figure published in [WT09] pre-dates naming standardisation. Originally the concept of a DAN was named rural telecentre (RT), as can be seen in the publication.

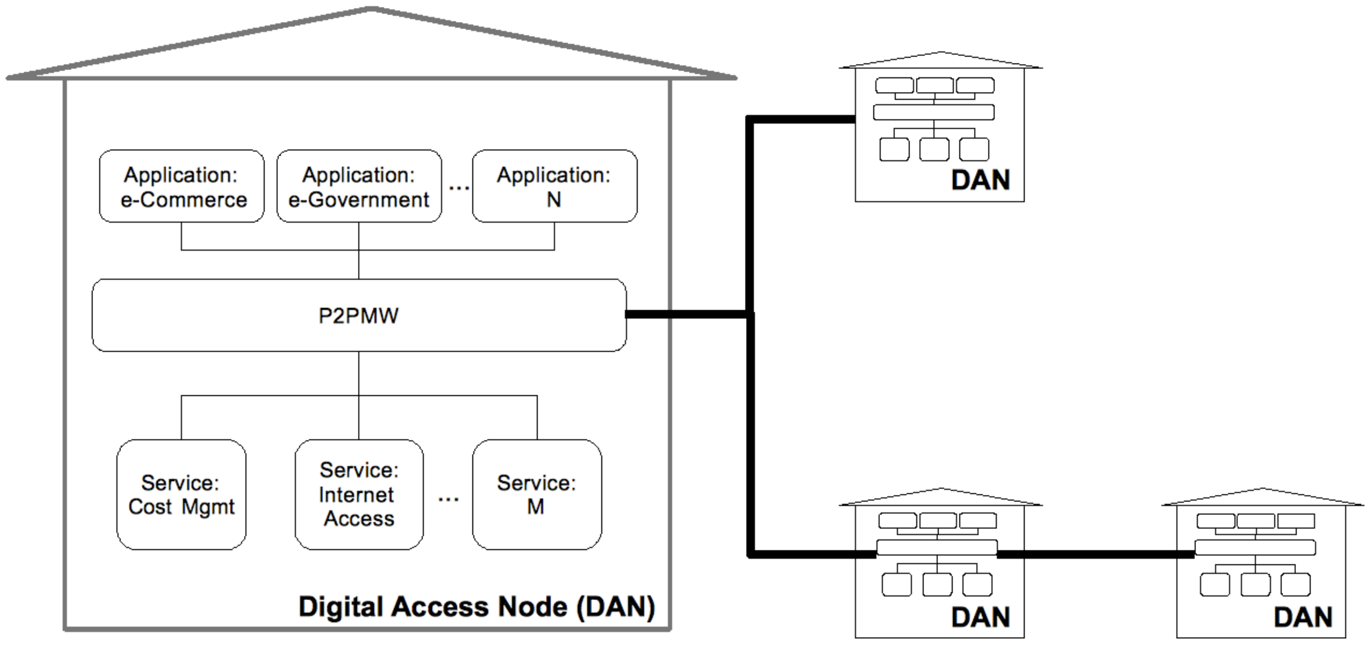


Figure 4.1: Initial design of the system architecture, showing components and component interactions.

middleware. Since the applications being glued together by the middleware have often been designed as stand-alone applications, their construction has duplicated functionality that could be supplied by the middleware. We did not design the middleware by factoring any common functionality out of existing applications. That method of design would have bloated the middleware introducing unwanted functions. It may also have omitted required functions. Instead, the most basic use cases required to make the platform useful were selected and these formed the basis of our design, as it is presented in the rest of this chapter. We illustrate this argument in the following section 4.1.2.

4.1.2 Supporting Billing

Here we present the reasoning that led to the exclusion of the billing function from the middleware core service. A simplified UML Use Case diagram (figure 4.2) explains how the billing application can be used. The billing system has several unique features which make it a reasonable candidate for inclusion in the middleware, despite our decision to exclude it.

- The billing system can potentially be linked to every application and service in the middleware, as one can potentially charge for any usage of any service or application in the system, including such low-level items as the amount of time that the user is logged in, etc.
- The billing system must potentially be tightly coupled with some services, for instance to allow prepaid usage of such a service. Prepaid usage of a service by a user often requires that the service be interrupted according to a specific event occurring, such as a time limit or resource usage limit being reached. A common example is

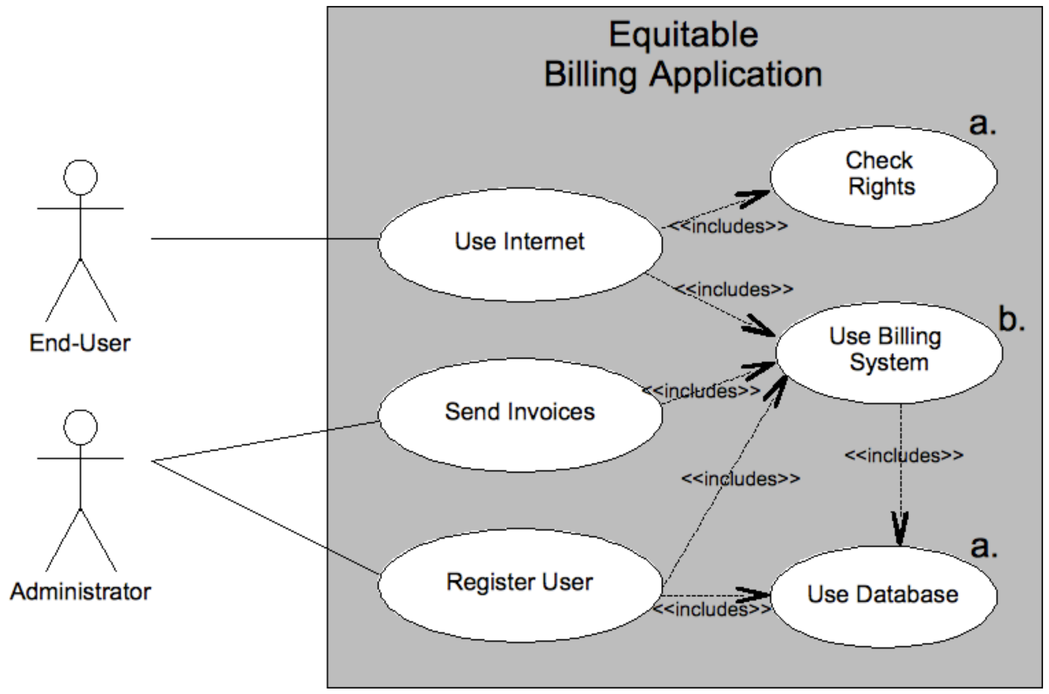


Figure 4.2: UML Use Case Diagram showing main use case of the billing system (using Internet) as developed in [Tar07], and underlying use cases that are provided by the middleware (a.) and core use cases (b.) of the application.

prepaid Internet usage by traffic. As soon as a certain number of Megabytes have been up- or downloaded, the user connection must be severed.

The use case can further be complicated if the user were to increase the limit on the fly by increasing the prepaid amount. The component providing the service would have to be able to liaise with the billing system in near real-time to provide this useful service. There are several designs that solve this extension of the use case, including: restricting functionality by only allowing prepaid services linked to variables such as time, which do not require a dependency between components; or passing control over certain prepaid billing scenarios to the real-time component, which updates the billing system after-the-fact (in a batched modus). Both of these possible solutions to the prepaid problem are compatible with the way we have designed the system and either can be employed.

4.1.3 More Top-Down Considerations

Considerations such as those discussed above, led to the exact division of services that the middleware platform had to provide and the description of the components that would provide the service. Additional functionality can be added as plug-ins. This step was important, as it determined how future applications and services will interact with existing services. It also added order to the already created applications and services and provided a framework for understanding them. Perhaps most importantly, the design decision regarding which operations are supported by middleware and which not places restrictions on new application development as these are expected to fulfill business assumptions made of

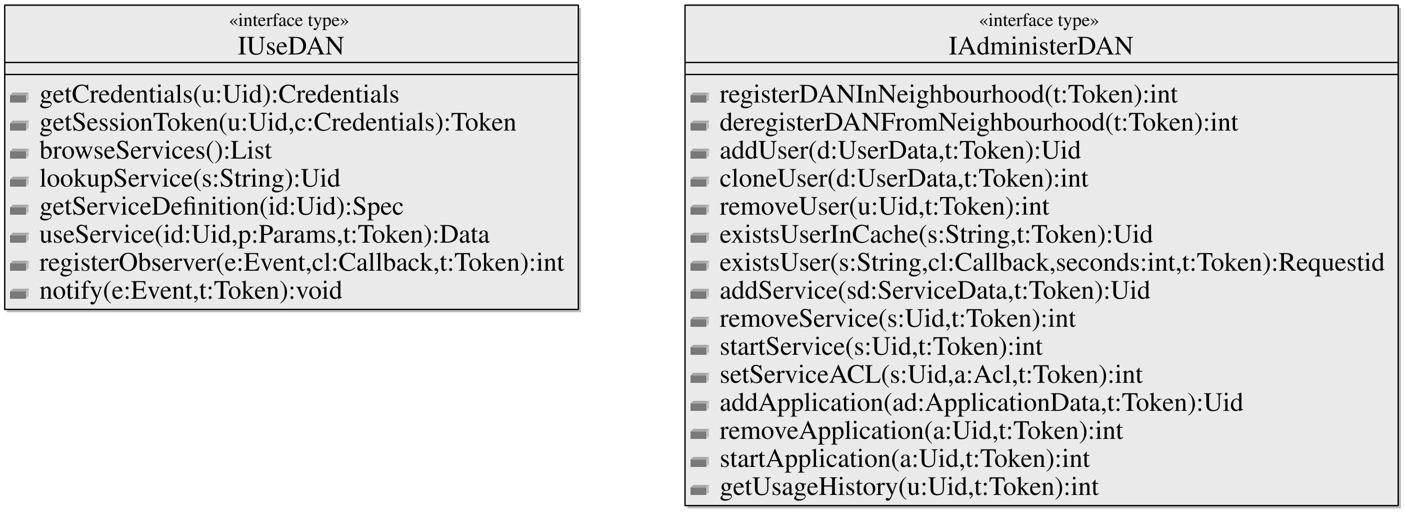

«interface type»
IUseDAN
getCredentials(u:Uid):Credentials
getSessionToken(u:Uid,c:Credentials):Token
browseServices():List
lookupService(s:String):Uid
getServiceDefinition(id:Uid):Spec
useService(id:Uid,p:Params,t:Token):Data
registerObserver(e:Event,cl:Callback,t:Token):int
notify(e:Event,t:Token):void
«interface type»
IAdministerDAN
registerDANInNeighbourhood(t:Token):int
deregisterDANFromNeighbourhood(t:Token):int
addUser(d:UserData,t:Token):Uid
cloneUser(d:UserData,t:Token):int
removeUser(u:Uid,t:Token):int
existsUserInCache(s:String,t:Token):Uid
existsUser(s:String,cl:Callback,seconds:int,t:Token):Requestid
addService(sd:ServiceData,t:Token):Uid
removeService(s:Uid,t:Token):int
startService(s:Uid,t:Token):int
setServiceACL(s:Uid,a:Acl,t:Token):int
addApplication(ad:ApplicationData,t:Token):Uid
removeApplication(a:Uid,t:Token):int
startApplication(a:Uid,t:Token):int
getUsageHistory(u:Uid,t:Token):int

Figure 4.3: UML Class Diagram, depicting business interfaces that may be relevant to user of a DAN, and the operations that they allow.

them. For instance, the idea that no component should duplicate middleware services. In order for this to be effective, the middleware must be simple to understand (i.e. know when you are not duplicating a service) and use – developers should be encouraged to make use of the facilities, because they offer a real gain.

Also from a high-level perspective, a minimal set of functionality in the middleware would allow future extensions to the middleware, without compromising backward compatibility. Interfaces could be honoured in succeeding generations of the middleware, without breaking existing installations in the field.

4.2 System Design

4.2.1 Business Operations and Rules

These are the business interfaces group operations that were presented in the Use Cases and they express what any person or company can "do" with a DAN. The target audience of the business interfaces is a product manager or business owner, whose business plan is based on the functionality that will be delivered by the software. The business interfaces are what allow the business plan to integrate with other (real or planned) aspects of the business[CD01]. In our instance, there are several possible business partners that may need to interface to the system. As discussed in Chapter 2, these can include eGovernment, eCommerce, audio and video communication systems that may need to interface to our system on a business level. As expressed below, business operations involve principally usage events and administrative events. Business users can use the system or offer use of the system to third parties through the IUseDAN interface, and they can administer their DAN via the IAdministerDAN interface. The operations that can be carried out on the system can be read from the UML diagram in figure 4.3.

Next we describe the clusters of operations that can be performed and list the pre and post-conditions for the most important operation(s) in each group. The pre- and post-conditions make explicit the rules that concern the parameters of the component interfaces as well as specifying the intended results of the business operations. Business rules tell us more about the assumptions behind the components we are creating. They provide detail for the component design that needs to be implemented.

Application – { Start | Stop | Add | Remove } This set of operations deals with the publication (release) of functionality in application[3] components. The add and remove operations make a new component available or unavailable to the platform[4]. The administrator or operator can start and stop the application via a user interface.

Start:

Pre:	The application to be started is a valid installed application (i.e. it can be located by name, its version numbering indicates compatibility with the middleware version).
Post:	The number of running applications has increased by one. The most recently started application is the one that was started. The application is able to trigger usage events on public services. The application and its description is locatable in the middleware's directory. The location can be bookmarked and may be accessed directly obviating the middleware's directory.

Stop:

Pre:	The application to be stopped is a valid running application (i.e. the post conditions mentioned above under *start* apply).
Post:	The number of running applications has decreased by one. The name of the stopped application is the one missing from the list of running applications. The application is no longer accessible via the middleware's directory. The application is also unavailable to bookmarks that directly lead to the launching of the application, without consulting the middleware's directory. Services should ignore usage events originating from the application. All credentials held by the application must be invalidated.

Service – { Start | Stop | Add | Remove } This set of operations deals with the publication (release) of functionality in service components. These services can be intimately bound to the functioning of the middleware, extending its set of

[3]Definitions of the terms "service" and "application" are given in section 1.3, it is worth repeating at this point.

[4]As mentioned in the final sub-section "System Interfaces and Types", the business interfaces mention operations that are relevant from a business perspective. It may for instance later become important to be able to add an explicit add operation, as the DAN operator may be charged to expand the functionality at the DAN. On an implementation level, we notice that the operation occurs "automatically". Components can simply be included in the middleware container and they will appear at startup.

services, or they can be specific to an application, allowing the application to provide services to other applications in a controlled manner. As with the application components above, the add and remove operations make a new component available or unavailable to the platform. Unlike the above, a service component must be added together with a description of its interface (Service Data). The administrator or operator can start and stop the service via a user interface. The main difference between the service start and stop operations relate to the difference in use case scenarios between the two types of component (see figure 3.4).

Start:

Pre:	The service to be started is a valid installed service (i.e. it can be located by name, its version numbering indicates compatibility with the middleware version). A boolean value indicates whether the service is *neighbourhood public*.
Post:	The number of running services has increased by one. The most recently started service is the one that was started. The service is able to trigger usage events on other services. The service and its description is locatable in the middleware's directory. The location of the service can be bookmarked and may be accessed directly by any application in the neighbourhood, thus obviating the middleware's directory. [5] Since a service component can offer several – at least one – service operations (e.g. a billing service can offer a whole set of billing-related service operations like finding invoices, determine account limits, etc.), a number of new service operations may be advertised in the middleware's directory. The latest added service operations must refer to the new service location. If marked as "neighbourhood public" the location of the service is shared with other DANs in the neighbourhood, if it is not marked as "neighbourhood public", then the service may only be used by applications within the service's DAN. By sharing we mean that the service locations and all information about service operations are being propagated[6] within the DAN's neighbourhood and may be accessed by any application with valid credentials within the middleware constructed network.

Stop:

Pre:	The service to be stopped is a valid running service (i.e. the post conditions mentioned above under *start* apply).

[5]The (in-)transience of locations is an important consideration for this operation. If a new location is assigned to a service every time it starts, then a restart of the middleware system at a DAN will invalidate all old bookmarks pointing at that DAN's services. This is not desirable if the system must preserve state between middleware restarts. On the other hand, since services that are public may advertise direct links independently of the middleware, the argumentation is only relevant to proxied services. The resource location semantics are defined in section 4.4.2.

[6]It is important to note that propagation of information in the network takes time, while post conditions apply immediately after a process has completed. Thus, some post conditions may have effects that do not show immediately.

Post: The number of running services has decreased by one. The name of the stopped service is the one missing from the list of running services. The service is no longer accessible via the middleware's directory. The service's operations have been removed from the middleware's directory, i.e. service operation advertisements in the directory do not refer to the resource location of the service that has been stopped. The service is also unavailable to bookmarks that directly lead to the launching of the service, without consulting the middleware's directory. Other services should ignore usage events originating from the service. All credentials held by the service must be invalidated. The neighbourhood may be notified.

DAN – { Register In | Deregister From } Neighbourhood This set of operations deals with the initialisation of a new DAN within the network of all DANs. This operation may be initiated via a GUI by an administrator.

Register:

Pre: A DAN process has been created. It has a local name, and a globally unique ID.

Post: The DAN's core services and other services that have been started (and not subsequently stopped) are available via the middleware's directory service. The DAN has a neighbourhood of which it is part – the neighbourhood may consist of more than one DAN. The middleware's directory service also shows public services available to the whole neighbourhood.

Deregister:

Pre: The DAN is running (the post conditions described in the "Register" operation apply).

Post: The DAN no longer has a neighbourhood and cannot be contacted by other DAN's. Middleware services, including its directory, are not reachable.

User – { Add | Remove | Clone } This set of operations deals with users as far as they concern a single DAN. Since we assume that the network is not 100% reliable, and that each DAN must manage their own information carefully, user management operations are very important. The difference between adding and cloning a user is explained in the use case description (section 3.3.5).

Add:

Pre: A valid user trait is supplied, with which the user can be identified on a network level (i.e. the trait is independent of the DAN).

Post: The number of listed users has increased by 1. The most recently added user record in the list has a trait corresponding to the one supplied. A middleware unique ID (UID) and credential set was generated for the user and stored with her record. The user record created has an attribute showing that this user is a *local user*[7]. The middleware notified all relevant services that a new user has been created via an event notification. This event is being propagated to the neighbourhood of this DAN.

Remove:

Pre: A valid middleware unique ID (UID) is supplied. The post conditions for the above operation apply.

Post: The user record was marked as deleted. The user credentials have been invalidated. The middleware dispatched an event notification to all relevant services that the user has been removed. This event is being propagated to the neighbourhood of this DAN.

Clone:

Pre: A valid middleware unique ID (UID) is supplied.

Post: The number of listed users has increased by 1. The most recently added user record in the list has a trait corresponding to the one supplied. A credential set was generated for the user and stored with her record. The user record created has an attribute showing that this user is a *remote user* (local users are created with the "add user" operation).

User – Exists (in Neighbourhood | in Cache) This operation deals with users in a network context. A version of the operation (*in Cache*) is supplied that simply searches cached information and returns immediately. An asynchronous version exists as well (*in Neighbourhood*), to provide more a thorough search given network latencies. It should be noted that finding resources in a P2P network is not a trivial exercise. Processes that use the search function must be designed to cope with incomplete result sets and latencies. It is possible that a user cannot be found even though she exists within the network, due to DAN down time.

Exists:

Pre: A valid user trait is supplied, with which the user can be identified on a network level (i.e. the trait is independent of the DAN). A valid callback location is supplied and the callback implements the ICallback Interface. A number of seconds which the calling process is willing to

[7] All other records for this user in the network are considered to be copies, and this record is an original

wait is supplied (this should be greater than zero otherwise the cache searching version should be used).

Post: A request ID was returned [8]. After the number of seconds specified, the callback function will be called with the request ID and a UID. If the UID is zero then the specified user could not be found in the time given[9].

Service Usage – { Browse | Lookup | Get Definition | Use } These operations are provided by the middleware to access services in the network. The Browse and Lookup operations are fairly straightforward – an application can browse the service directory and use this information to present a list of services to a user or it can look up the location of a service given a name. It is transparent to the application whether the service is local or not. An application can ask for more details about a service with the "Get Definition (Define)" operation. Some services can choose to allow usage only via the middleware (indirect or proxied usage), whereas others can be used via the middleware or directly via their location information. These two modes of operation are depicted in figure 4.4. The direct call will not reflect within the middleware and service implementers are obliged to notify the middleware of the usage event[10]. The indirect use operation is further explained below.

Define:

Pre: A valid unique ID that refers to a service is supplied, along with authorisation.

Post: An interface definition for this service was returned, along with middleware related data, such as whether the service is local or not and access control information for the service.

Use:

Pre: A valid unique ID that refers to a service is supplied, along with the data required by the service for execution (parameters), and authorisation. [11]

Post: The list of usage events has been increased by at least one. At least one of the usage events has a source matching the authorisation information supplied in the usage request and a sink that is this service. All remaining usage events generated can be chained such that their source and sink parameters match each other[12]. An integer was returned showing the result of the usage event.

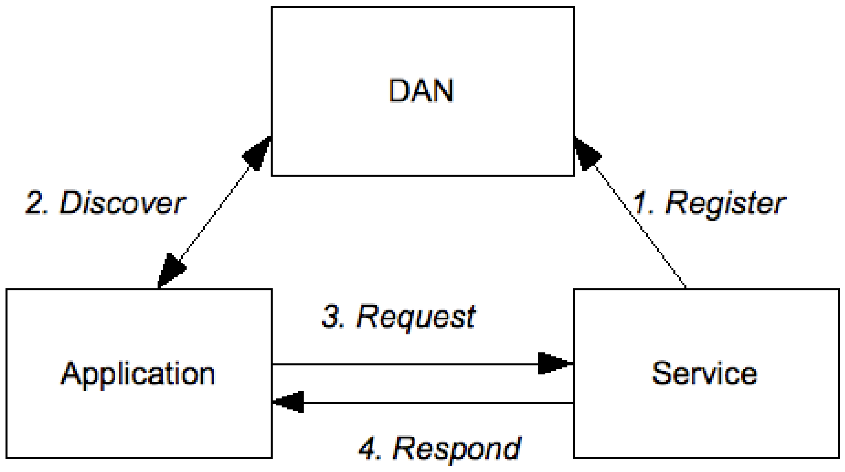
[8]In the case the case of the synchronous version of this operation, the return value is either a valid UID or zero is returned. If the value is zero then the user does not exist in the network. Otherwise, the user exists in the network and has the returned UID.

[9]It is important to note that duplicate users can thus exist in the system. This is not something that we can avoid in a network with our assumptions. The damage can be minimised by occasionally running administrative tasks that look for duplication and resolve it. Further the design of the system is such that it can continue working with duplication and is aware that it exists.

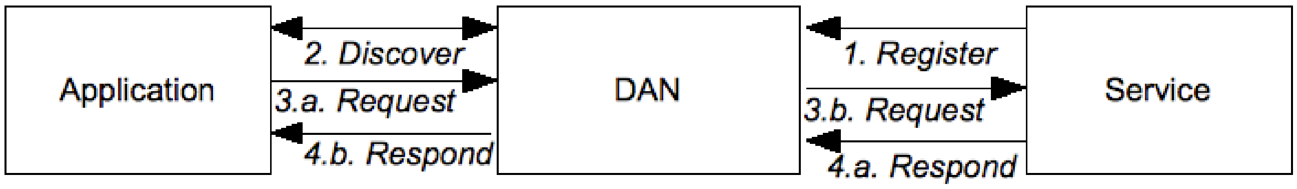
[10]An implementation level solution may provide built in detection of such events through interception or by some other means.

[11]The middleware evaluates the information supplied and passes the call on to the service if the information was valid, immediately logging a usage event.

[12]If we describe usage events by their source and sink parameters, as pairs (A, B), and if (A, B) is the first usage event in our chain, then a *chain of usage events* produced could be represented as the following set, e.g. $\{(A, B); (B, C); (C, D)\}$.



(a) Direct Usage (Web Services Architectural Pattern)



(b) Indirect Usage (Proxy Architectural Pattern)

Figure 4.4: Direct vs. Indirect modes of service usage.

4.2.2 Business Types

Business types describe what kinds of objects we will be dealing with in the system. The UML Components methodology distinguishes between the business interfaces and objects (called types) and implementation objects or system objects. They represent an intermediate step, which show at a business level how the concepts are arranged in the system. Business types are separated into core types and undifferentiated or normal types. The core types are the major concepts on which the other concepts depend. [13]

As we see in figure 4.5, the business type model is derived from the concept model (figure 3.1). Generally more abstract concepts, i.e. ones appearing higher in the concept tree, tend to be the core types and their helpers are the sub-concepts. The type model is a predecessor of the information model discussed later.

As may be expected, the core types in our system are Users, Services, Applications and the DAN (which is represented by the middleware). Figure 4.5 also shows basic data associated with each of the business types. This data is part of the Information Model discussed in section 4.4.2.

[13]The conceptual split between core and other types is common in programming. It is analogous to the distinction between main and auxiliary tables in database theory.

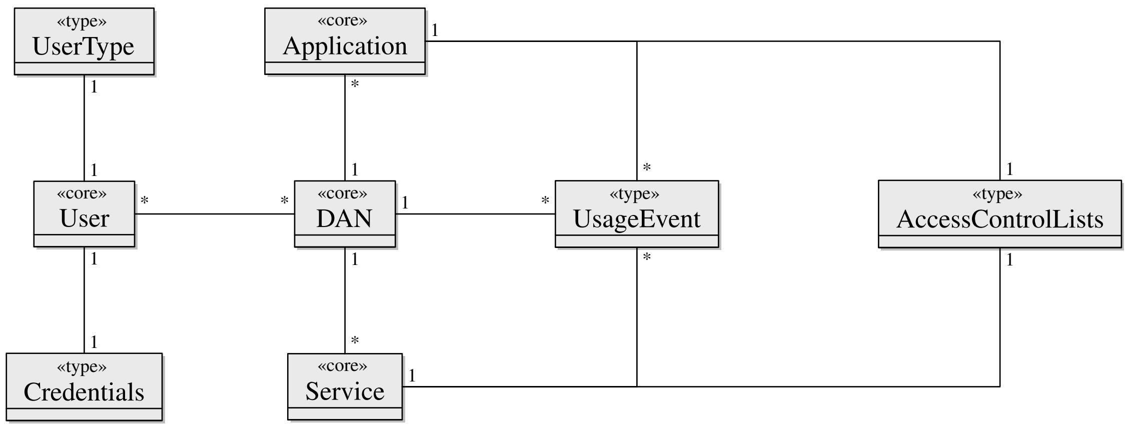


Figure 4.5: UML Class Diagram of business types used by the interfaces. The business types translate directly to an interface information model, which is what the middleware data layer must represent and store.

4.2.3 System Interfaces and Types

According to the UML Components methodology, system interfaces specify detailed interface rules, which developers must adhere to when programming the interface's implementation. The system interfaces are therefore a subset of the business rules, since business rules apply to a broad range of business cases, while the system interfaces represent the functionality in a single version of the software. Further, the system interfaces are expected to reorganise the business interfaces. However, maintaining two sets of rules can be cumbersome and there are cases where the business rules and system rules and interfaces can correspond [CD01]. This project assumes that the business rules and system interfaces *do correspond* for the following reasons: the team working on the business and programming aspects is very small; the scope of the project is clearly defined; and the close cooperation of business and development processes fits well to the agile method [author's experience]. In this design section, we refer to the business interfaces *qua* system interfaces, but some refactoring is performed to cluster operations in a manner which suits creation of components better.

At the beginning of section 4.2.1, we presented business interfaces that the middleware will present to users of the DAN. Developers (including third party integrators) are presented with a somewhat different set of interfaces. These provide access to core middleware functionality and business types. Functionality required for end-users is composed from a subset of functionality required by developers. Additionally, there are some interfaces that reflect requirements for asynchronous functionality. These are based on the generic ICallback interface, e.g. ISearchCallback.

The ICallback interface must be implemented by components that wish to perform asynchronous searches or other asynchronous usage events. The data which is passed into the *notify* operation on the interface depends on which asynchronous operation was called. A request ID must be provided by the caller, so that the component can determine to which asynchronous operation the response should be marshalled. Notifications with no request ID should be dropped.

The manager and services interfaces in figure 4.6 directly give rise to components available in the system. An explanation of the most important system operations is given in the section on business operations (section 4.2.1), since

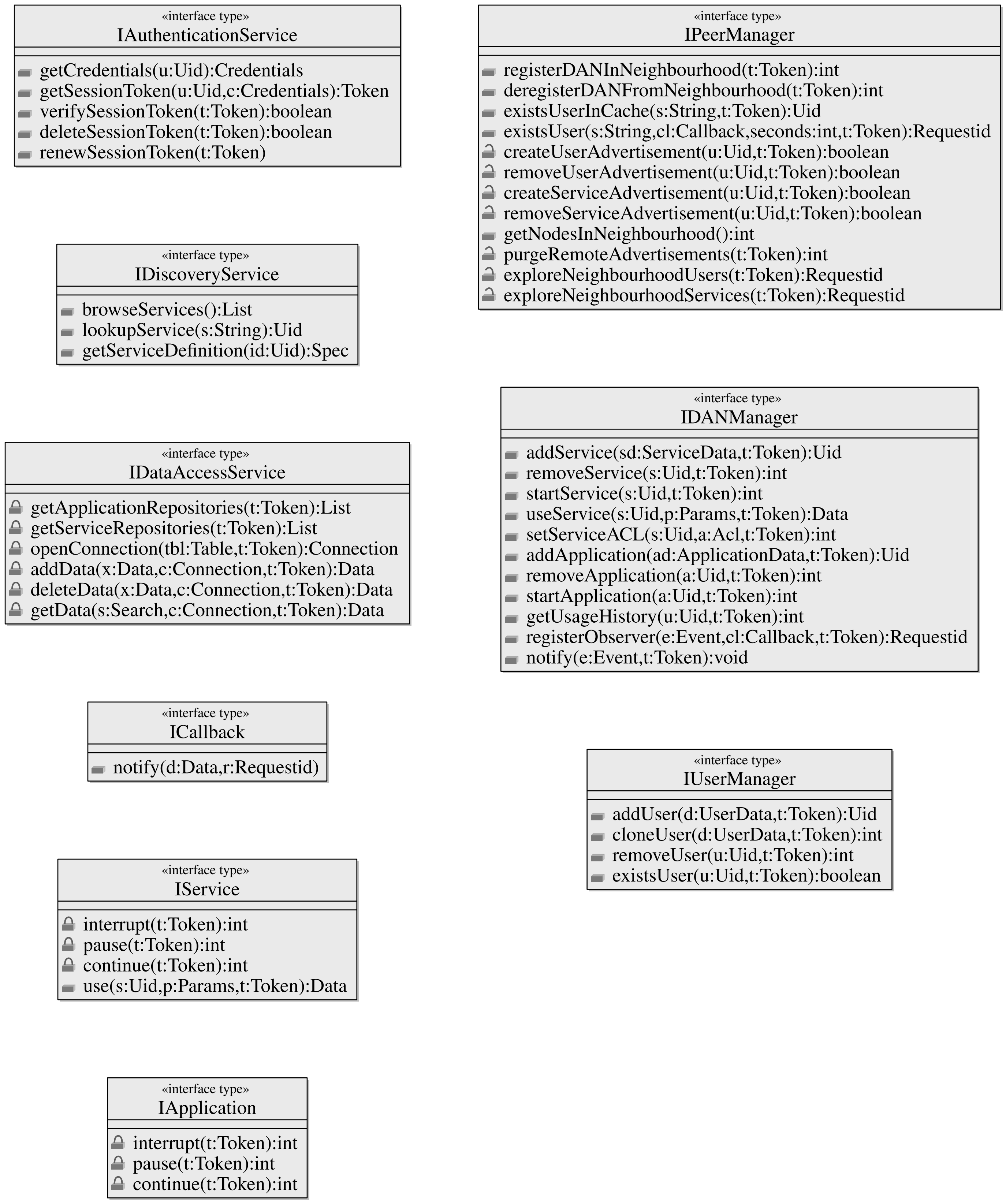

«interface type»
IAuthenticationService
getCredentials(u:Uid):Credentials
getSessionToken(u:Uid,c:Credentials):Token
verifySessionToken(t:Token):boolean
deleteSessionToken(t:Token):boolean
renewSessionToken(t:Token)
«interface type»
IPeerManager
registerDANInNeighbourhood(t:Token):int
deregisterDANFromNeighbourhood(t:Token):int
existsUserInCache(s:String,t:Token):Uid
existsUser(s:String,cl:Callback,seconds:int,t:Token):Requestid
createUserAdvertisement(u:Uid,t:Token):boolean
removeUserAdvertisement(u:Uid,t:Token):boolean
createServiceAdvertisement(u:Uid,t:Token):boolean
removeServiceAdvertisement(u:Uid,t:Token):boolean
getNodesInNeighbourhood():int
purgeRemoteAdvertisements(t:Token):int
exploreNeighbourhoodUsers(t:Token):Requestid
exploreNeighbourhoodServices(t:Token):Requestid
«interface type»
IDiscoveryService
browseServices():List
lookupService(s:String):Uid
getServiceDefinition(id:Uid):Spec
«interface type»
IDANManager
addService(sd:ServiceData,t:Token):Uid
removeService(s:Uid,t:Token):int
startService(s:Uid,t:Token):int
useService(s:Uid,p:Params,t:Token):Data
setServiceACL(s:Uid,a:Acl,t:Token):int
addApplication(ad:ApplicationData,t:Token):Uid
removeApplication(a:Uid,t:Token):int
startApplication(a:Uid,t:Token):int
getUsageHistory(u:Uid,t:Token):int
registerObserver(e:Event,cl:Callback,t:Token):Requestid
notify(e:Event,t:Token):void
«interface type»
IDataAccessService
getApplicationRepositories(t:Token):List
getServiceRepositories(t:Token):List
openConnection(tbl:Table,t:Token):Connection
addData(x:Data,c:Connection,t:Token):Data
deleteData(x:Data,c:Connection,t:Token):Data
getData(s:Search,c:Connection,t:Token):Data
«interface type»
ICallback
notify(d:Data,r:Requestid)
«interface type»
IUserManager
addUser(d:UserData,t:Token):Uid
cloneUser(d:UserData,t:Token):int
removeUser(u:Uid,t:Token):int
existsUser(u:Uid,t:Token):boolean
«interface type»
IService
interrupt(t:Token):int
pause(t:Token):int
continue(t:Token):int
use(s:Uid,p:Params,t:Token):Data
«interface type»
IApplication
interrupt(t:Token):int
pause(t:Token):int
continue(t:Token):int

Figure 4.6: UML Class Diagram of interfaces that the middleware supports and that may be relevant to developers or integrators (contrast with figure 4.3).

we assume that the business interfaces and system interfaces correspond. In the component specification below we explain the purpose of the components and the interfaces they support.

4.2.4 Component Specification

Components offer interfaces whose names they share in this initial version of the middleware design.

DiscoveryService The discovery service is a distributed lookup service which allows one to look up resources on a network wide basis. Resources include users and services, and there are corresponding operations for this. The discovery service performs a similar function to UDDI or LDAP, and may be implemented using either of these technologies.

DataAccessService The data access service provides access to the Information Model of the middleware and its components. Operations allowed on data stored within the data layer include field based search (which can return record sets), insertion, update and deletion of data. Further, services can authorise third parties to access their data. The Information Model represents a point at which contextual information is concretely included in the middleware and made accessible to components and is thus a unique feature of this middleware.

PeerManager The peer manager handles interactions external to the DAN within the peer to peer network. It acts using notifications called advertisements, to spread information within the network. Further, the peer manager should handle communication transport matters such as relaying of messages to peers that are not directly connected – such operations are not accessible via the middleware API and do not appear here.

UserManager The user manager handles basic user information, as required for security and identification purposes. Components can augment user information via the data access service, since the user manager stores user data in the data layer. Credentials are specially handled and encrypted (with a salt) in the data layer. The user manager supports the concept of a session, whereby a user can for a certain amount of time not be required to log in repeatedly to access different services. Instead a session token is passed between components, and this token is used to authenticate the user.

DANManager The DAN manager component composes the other manager and service components into a unified middleware component. It also provides access to meta-operations which control the middleware such as, starting and stopping the middleware, adding and removing components. Figure 4.7 shows how the components might fit together in a UML Component Diagram[14].

[14]UML Component diagrams can show both the interfaces that a component offers or implements, as well as interfaces that are used or expected by the component. Interfaces that are implemented are denoted by a closed curve (circle) at the end of a line extending from the component. Expected interfaces are denoted by an open curve ("U") at the end of a line extending from the component. More information about UML diagrams can be gleaned from any of the many UML references, e.g. [PP05].

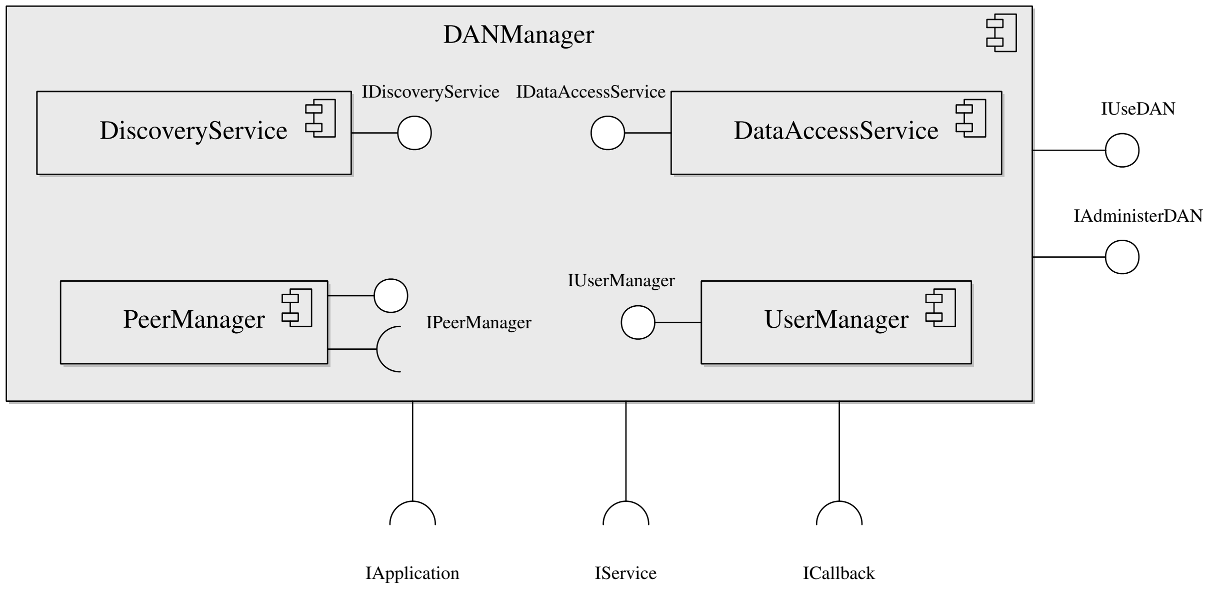


Figure 4.7: UML Component Diagram showing the middleware components and their interfaces.

4.2.5 Proposed Component Architecture

The proposed component architecture is no more complicated than the middleware component specification shown above. The architecture is diagrammatically presented in figure 4.8. The figure shows two isomorphic DANs – in terms of component topology. They have the same number of services and applications and the same dependencies. Further, the two DANs are dependent on each other. The DANs connect to each other transparently using the IPeerManager interface – implicitly the PeerManager sub-component connects the two DANManagers. Note that actual network boundaries are irrelevant to the architecture and components can be added to the chain of DANManagers as necessary. Additional services and applications can also be added to each DANManager. These components could all be situated at different physical DANs, since they are connected using web services or an equivalent distributed technology. This is fine for network testing purposes where virtual networks can be constructed in a lab. In practice, only locally hosted components should be accepted by a DANManager, i.e. remote components may only connect via the PeerManager component.

P2P Network Transparency

The PeerManager enables the DiscoveryService to keep track of resources at neighbouring DANs. This transparency at the neighbourhood level could potentially complicate matters for components. As an example, consider a user management operation, such as credential authentication. The PeerManager may transparently connect an application with different remote UserManager components to verify a remote user's credentials (see figure 4.9). If the PeerManager is able to choose an arbitrary UserManager at peer nodes, the system could behave in a random manner, occasionally

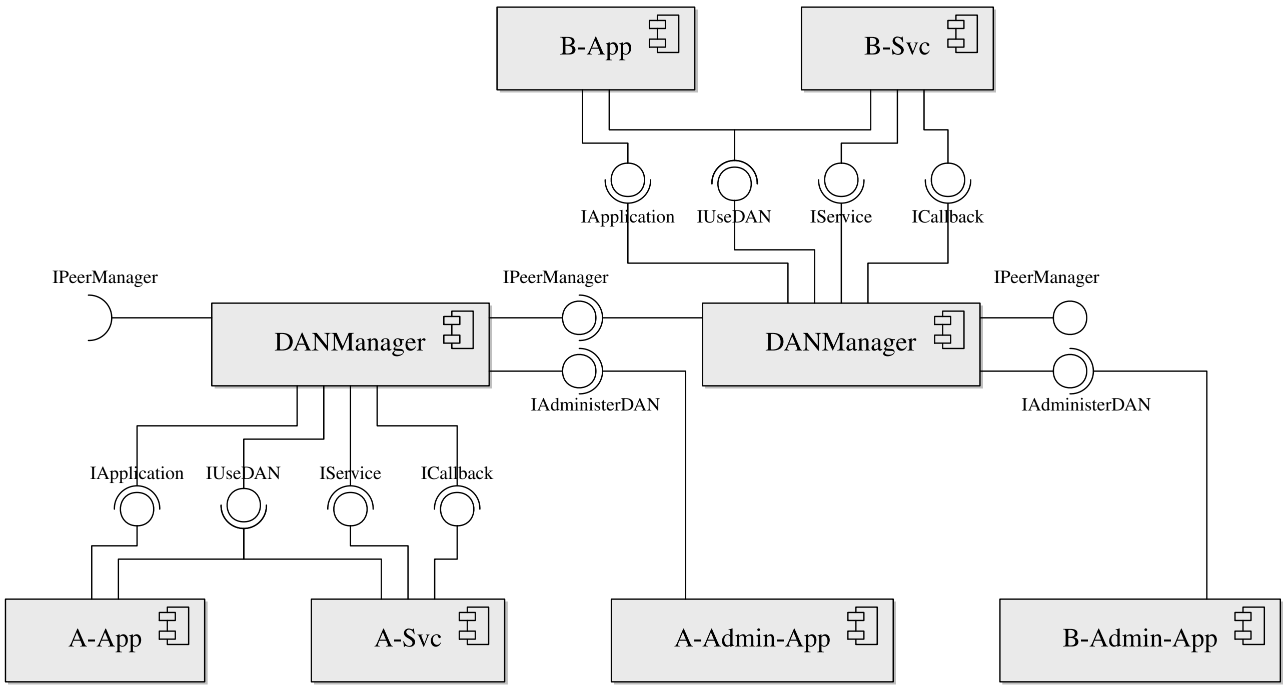


Figure 4.8: UML Component Diagram of the Component Architecture

verifying credentials and occasionally not verifying them. This example is perhaps somewhat artificial, but it illustrates a real issue. If a component has to be directed to a unique resource, such as a user, or a particular service, then it should always be directed to the same resource. Since multiple services of the same kind are allowed within the neighbourhood, this is potentially a critical issue.

The solution we propose here preserves loose coupling from both the perspectives of the middleware as well as the application or service. It depends on variable referencing of resources – resources may be referred to by their unique ID or by their (non-unique) name (see also section 4.4.2):

- The middleware makes sure that some core resources (such as user management) are always accessed locally and that these services are unique locally. Replicable resources, e.g. an Internet connection, may be accessed locally or remotely in the same manner. A component using the middleware need not know the difference between these. Thus a component accessing a core resource (such as user management) will always address the resource by name and not by UID, as it is up to the middleware to provide the correct resource.

- Service and application components *must* always use UID (unique IDs) of resources when they need a resource specific to their product specific objectives. They should use names when any resource of a certain named class will do. The middleware need not know about product specific application and service dependencies.

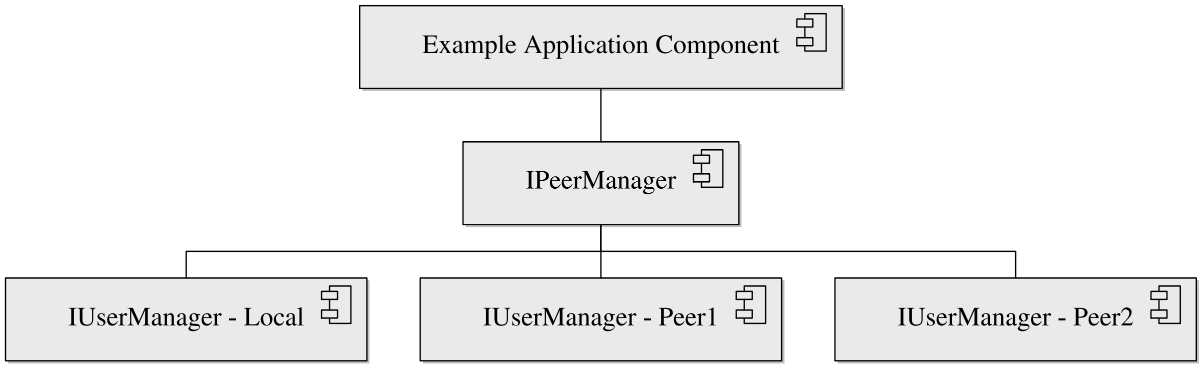


Figure 4.9: UML Component Diagram showing how transparency at the neighbourhood level could potentially complicate matters for components, using the example of credential authentication.

4.3 Example Components: eCommerce

In a business sense, there is a difference between applications and services. Applications are purely consumers of resources, while services can consume resources as well as providing them (see figure 3.4 for the business use case illustrating this point). Thus applications, which want to share data with other applications, need to specify a service component as well. A case in point is the eCommerce product. The eCommerce product makes some of its data available for community friendly[15] billing purposes, but the data is sensitive and should be protected as it includes money transactions. In addition to the single sign-on facility, the middleware can act as a proxy for the eCommerce "Mall Service". This allows an administrator to specify which applications, services and users may have access to the Mall Service, adding a further layer of protection.

In this section we model the eCommerce product to illustrate how a software product can be implemented using the middleware concepts proposed by us. This conceptual work should be performed for any new service or application that needs to be included in the DAN, since it ensures that the new service or application fits to the concepts of the middleware and that it can integrate with the existing infrastructure.

The Mall Service exposes the functionality of the mall as described in [Nje07]. Table 4.1 summarises the functions allowed by the eMall and shows how the middleware and Mall Service take over operations – without exposing too much detail about how the operations themselves work. The first column in the table shows the operations that were allowed by the monolithic eMall application. The second column shows how existing middleware interfaces take over much of the administration functionality; it also shows one new interface that needs to be added for the service.

The new interface – *IMallService* – is discoverable using the discovery service and it is offered as a virtual interface by the middleware. A component diagram illustrates these new dependencies in figure 4.10. Calls can be directed to the

[15]"Equitable Billing" systems (proposed e.g. in [Tar07]), are rating systems which rest on socially acceptable or democratically decided rules. For instance, a rule may say that those with most equity (the greatest stake) in the system must pay more for use and upkeep of the system.

eMall operations by user type [Nje07]	**MallService operations by component**
Mall-Administrator	*IUserManager*
Log in	Register
Manage users	Login
View purchase orders	*IDataAccessService*
Manage own profile	Handle sharable, mall-specific info for:
Manage shop owners	* user (e.g. type)
Shop Owner / Service Provider	* sales info
Manage own shop (products)	* product info
Manage purchase orders	*IMallService*
Manage own profile	Manage user profiles
Customer	Manage purchase orders
Login	Manage products
Manage own profile	*ShoppingApplication*
View own purchases	*ShopManagementApplication*
Make purchase orders	*MallAdministrationApplication*
Anyone	
Register	
Browse the Mall	
Fill shopping cart	

Table 4.1: Juxtaposition of operations as defined in the monolithic eMall, and the same operations as delegated to components.

service via the middleware by addressing the interface virtually through the *useService()* operation. In the figure, we show how a third party service can be integrated with the platform while maintaining loose coupling – the middleware is agnostic concerning the semantics of the new service, but is still able to support it and offer a new service through a virtual interface. On the application side of the diagram, the following features are demonstrated: decomposition of user interface functionality into separate components[16]; the minimal set of interfaces required for an application to access the functionality; an administration application also has to make use of middleware functionality via *IAdministerDAN;* and finally, a BillingService component also may make use of the MallService component via its interface.

In figure 4.11 we provide some more detail about the newly added interface, with example operations that flesh out the generic operations mentioned in table 4.1. This further describes the level of contractual commitment required from service interfaces in order to be useful to other developers.

This section demonstrates the added value of this component-wise decomposition. Other applications can take advantage of the operations provided by the MallService component (e.g. a BillingService component), something that was not really possible with the eMall application. Further UI functionality can be changed for products based on the Mall Service, without affecting these 3rd party services, or the Mall Service itself. Upgrades to the MallService component immediately apply to all the users of the service. By adhering to interfaces, applications will continue to work even if new functionality is added or existing functionality is adapted in a compatible manner.

[16]This is done for illustration purposes only – the UI component could also be a single component.

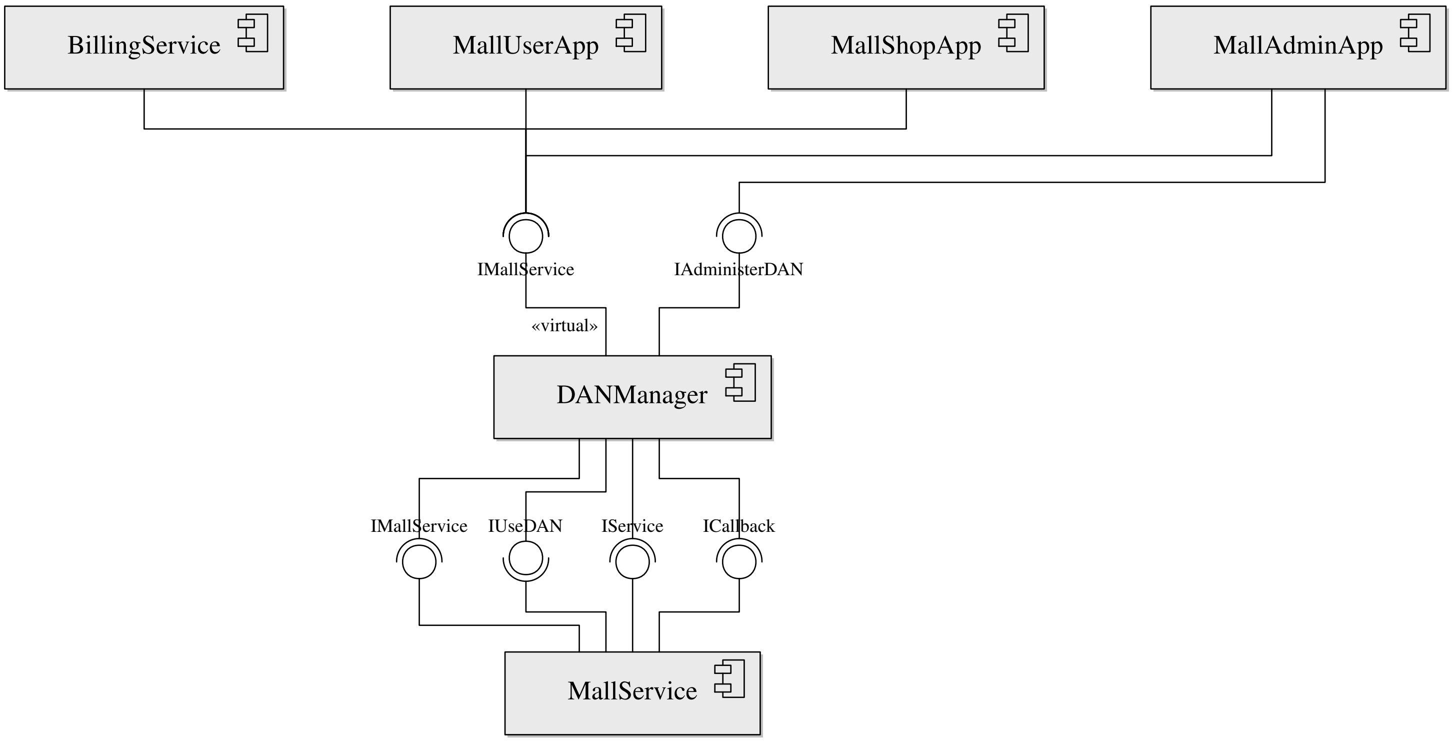
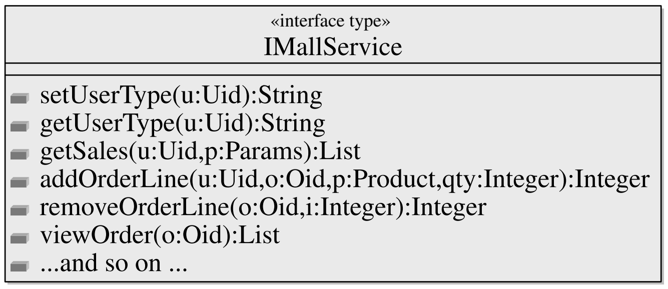


Figure 4.10: UML Component Diagram showing the middleware architecture with two example services, *MallService* and *BillingService*.



```
setUserType(u:Uid):String
getUserType(u:Uid):String
getSales(u:Uid,p:Params):List
addOrderLine(u:Uid,o:Oid,p:Product,qty:Integer):Integer
removeOrderLine(o:Oid,i:Integer):Integer
viewOrder(o:Oid):List
...and so on ...
```

Figure 4.11: UML Class Diagram of the *IMallService* interface.

4.4 Information Model and Core Data Types

We propose a flexible extensible information model. This is derived from the fact that each application potentially brings its own data model to the system and may use the middleware to share this data with other instances of the application (for example at other DANs, if necessary and if authorised to do so). The data is private to the application component instances and is not shareable with other components. The application also has access to the core information model and can access globally shared information, depending on access rights. At the very least, each application can check credentials and use the single sign-on functionality offered by the system (UserManager). Applications also have access to the DiscoveryService, DataAccessService etc. Using these services, they can preserve their state in a distributed, transparent fashion and provide new ways of accessing resources.

The extensible information model is needed because we cannot a priori create a data model, which captures all possible aspects of the data needed by all components in the system, as would be normal practice in a centralised system. We also know that modern XML databases support this kind of functionality efficiently and that XML and web services integrate well and thus we are assured that this strategy is technically feasible. Since our database of records does not need to have a fixed data schema, selection of records according to certain criteria (search), is less like selection in a relational database and more like a search in indexed free-text documents. Therefore, in addition to XML Databases, we also have the option of using hierarchical indexers such as the Apache Xindice project to manage our data in the data access layer efficiently[17].

In this section we describe the core data model in a preliminary manner. The core data model is also extensible and the component architecture is guaranteed not to break because of the data model extensions. Security is assisted by not allowing applications to share data. Only services may share data, and since services can allow only indirect use, a security / trust layer can automatically be added to preserve CIA for sensitive data.

4.4.1 Core Information Model

The core information model is presented graphically in figure 4.12. It is clear that the skeleton of the Information Model is derived directly from the business type model. In order not to over-engineer our solution, we have kept the core information model minimal and used only ideas from the Requirements section 3.3 for this.

A further reason to keep the core information model minimal is the principle of loose coupling. 3rd party information services that are not tightly bound to the middleware, may be better at managing the information model required for the DAN. For instance, relevant semantic knowledge representation work has been done by [TT09]. Such a service can be included in the network as a semantic component, able to extend the overall information much further and make the

[17]While our assumptions show that ultra-high performance is not initially essential to the system, good design must balance functionality with efficiency, as a sluggish system will affect the user experience and present an additional hurdle for use to developers and community members alike.

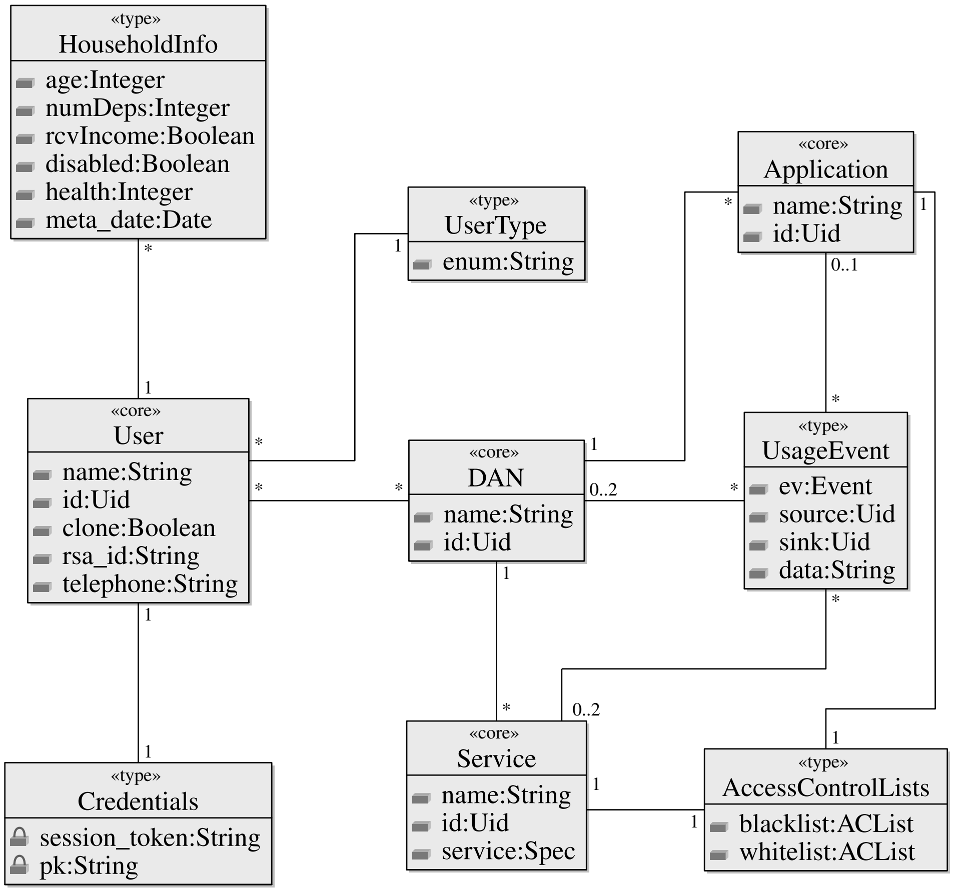


Figure 4.12: UML diagram of Core Information Model

knowledge captured available to several applications.

In light of the above, it is necessary to explain the addition of the type – "Household Information". Household information helps place the model in the rural setting of the DAN. It makes contextual information available to application developers and enables inclusion of this data in and across components. The attributes of the type are merely sketched from information commonly used by governmental agencies such as the Departments of Public Works, Social Development, Health, Labour, Water Affairs and Forestry, Education, Home Affairs and other organisations such as the South African Social Security Agency (SASSA), etc.[MC08]

4.4.2 Core Data Types

In this section we consider some of the important data types from figure 4.12.

UID The most important semantic to consider here is that of the UID (Unique ID), which can act simultaneously as a resource identifier and locator. Resource location by UID is a common practice in P2P networks that use *content addressing*. The specific method of using content hashing to address items in a manner that scales well was first described in this paper [RFH+01][18]. The authors described how *Distributed Hash Tables* (DHTs) could be used to partition the naming space of resource in the P2P network into a d-dimensional space. Succeeding works modified this method to create networks with other characteristics [GGG+03]. These methods will become applicable once the number of resources reaches into the millions. Until then, we can use simpler methods.

The concept of URIs (Uniform Resource Indicators), a W3C standard, is used primarily to identify a resource and can simultaneously be used to locate it. A URI based naming schema for the middleware can help developers and administrators quickly understand what is happening in the system – more so than if the UID were just an arbitrary number[19]. We thus propose a UID scheme such as:

$$uri : DAN.Name/\{S|U|A\}/\{Service|User|Application\}.Name - Hashnumber$$

The following is an example UID: *uri:Ngwane/S/emall-38029*

Name Our solution to the problem brought up in section 4.2.5 required that a naming scheme also be implemented for resources. While the naming of resources can be left entirely up to the resource developers, a similar scheme could be followed as with the UID. By simply leaving off the *Hashnumber* in the UID, a class of resources at a particular DAN could be identified. One might also leave the *DAN.Name* parameter blank, to address all resources of a certain type in the neighbourhood. Since URI expressions are strictly defined, we suggest that the names of the

[18]Altnet, an American company, claimed to have patented content addressing in 1997 in the following patent [FL99]. We analysed this patent and found that it did not predate the ground-breaking work of Ratnasamy et al[Wer06].

[19]This was a lesson that was learnt in the CORBA technology, whose opaque IOR resource locator IDs did not help acceptance of the technology [Hen06].

resources not be expressed as URIs, but that they simply follow the format, roughly. We suggest the following naming scheme:

$$\{DAN.Name\}/Resource.Name$$

The following is an example name: $/emall$

Spec The service specification should be rigourously defined. Since we plan to be using SOAP based web services in our system, it is logical to describe them using the official W3C web service definition language (WSDL).

Validations Several data types can undergo type checking. It is important that the middleware data access layer natively validate core data types in order to keep core data valid. Invalid data should not be accepted. Here are a few obvious checks that can be made: rsa_id should be a valid South African ID Number; $0 \leq age \leq 150$; the UserType can only be one of 'administrator', 'operator', or 'user' ; etc.

4.5 Designing Robustness

4.5.1 Pre- and post-conditions

By including detailed pre- and post-conditions in the design, implementers are assisted in their test-driven development. The specified conditions may be adapted in several cases into fairly intelligent checks of functionality. Such checks have been shown to assist in rapid application development and the production of high quality software despite shifting goal posts, refactoring etc. [CD01]

4.5.2 System bottlenecks

We can identify some bottlenecks in advance. In this section we look at potential bottlenecks and how they can be minimised. Often bottlenecks only appear during actual operation, because runtime assumptions as set out in section 3.3 may not be accurate[20].

The Messaging System is the bottleneck.

How many messages can we expect on the bus at one time? Apart from direct communications with services, all events such as user additions and removals, service additions and removals, usage events and so on result in notifications which are middleware messages flowing in the system. Our architecture very much resembles a bus architecture, in which all communications flow along a bus.

[20]This is the classical difference between lab environments and real environments, contextual factors may have been misjudged and thus portions of the software over-designed and others under-designed. This principle is also visible in the extreme programming paradigm as expressed by Cunningham ([Cun09]) and others.

We propose equations (4.1) - (4.4) to determine the peak number of messages in the bus at any one time[21].

$$Messages\, due\, to\, local\, usage\, events: \; n_{LU} \approx p_u \times u_\mu \times constant_{messages-per-use} \tag{4.1}$$

$$where\, constant_{messages-per-use}\, is\, estimated\, to\, be = 5$$

$$Messages\, due\, to\, local\, authentication: \; n_{LA} \approx p_u \times constant_{messages-per-auth} \tag{4.2}$$

$$where\, constant_{messages-per-auth}\, is\, estimated\, to\, be = 3$$

$$Messages\, due\, to\, peers\, leaving\, or\, joining\, the\, network: \; n_{PC} \approx \frac{n}{2} * (1 - (t_w)^n) \tag{4.3}$$

$$Peak\, number\, of\, messages\, in\, the\, bus: \; n_{peak} = (n_{LU} + n_{LA} + n_{PC}) * constant_{buffer} \tag{4.4}$$

$$where\, constant_{buffer}\, is\, a\, safety\, factor = 4$$

Notes to the derivation of the equations:

(4.1) The number of messages due to local events is derived by multiplying the average number of users at peak time by the number of "uses" of the system by the users. This intermediate result is multiplied by the number of system messages produced on average by a system "use". The constant ($constant_{messages-per-use}$) needs to be empirically verified.

(4.2) The number of messages due to local authentication is simply the number of users using the system at peak time, multiplied by the number of messages per authentication (estimated at 3).

(4.3) Here the size of the neighbourhood plays a role, as information about peers joining or leaving the network must be propagated through the network. The likelihood of such events (join / leave) is related to the WAN uptime. Also since, the network is relatively loosely clustered, rebroadcasts of messages will probably only occur n/2 times. This equation shows that the impact of peer churn only really makes a difference once very large values of n are achieved.

[21] *NB:* These equations are merely a starting point for discussion and empirical measurements. They raise awareness of important non-functional parameters, rather than serving as a gold standard by which to measure the system.

(4.4) The number of messages in the bus during peak times is simply a sum of the previous factors. In equation (4), we choose to add a safety buffer ($constant_{buffer}$) of 4 times the amount of traffic we calculate for the peak time, since performance of the network also degrades at saturation. Messages generated by services registration, starting and stopping, as well as messages due to new user registration, can be safely ignored, as their occurence is physically constrained by the requirement of intervention by the DAN operator. We have also ignored remote usage events, since we do not know how many usage events will be remote and how many will be local (remote usage events may add a constant number of messages to each use). If such events occur during peak times, their presence is accounted for by a safety buffer.

Applying these equations to table 3.4, we arrive at the following value for peak number of messages in the system per hour:

$$n_{peak} = (10 \times 3 \times 5 + 10 \times 3 + \frac{10}{2} \times 0.4) \times 4 = 668$$

With an average expected message size, set at 8kb, the system may expect 5.10 Mb/hr in terms of message throughput at peak values assuming all DAN workstations are occupied and users are busily creating usage events. This amount of traffic can comfortably be handled by the systems currently in place in addition to the existing traffic. The system can certainly absorb several Gigabytes of data per hour.

Caveats:

1. Clearly, the middleware is not intended as as a communications bus for real-time streaming communications. It is a control channel which handles management data allowing applications to co-exist and resources to be shared. Real-time audio and video content may not be streamed via the notification queues.

2. Shared data may feasibly be in the Megabyte order of size. E.g. in the eGovernment application, large scanned application form appendices may need to be uploaded. While one may want to pass these on to the data access layer for storage, in order to share these, clearly a reference to a service local store would improve matters and prevent huge messages from burdening the system.

The Data Access layer as a bottleneck

Supporting the entire data access layer with one database system or a single database may be simple and effective, initially. Evaluations need to take place once the system has been running for a while and new applications have been created to test how many components make use of a local database and how many use the shared data store as their database. The additional processing added by a web services or communication layer which connects components may discourage many components from using it to too large a degree, so that only shared data is actually stored there. On the other hand, ease of programming and sufficient resources may encourage developers to use the shared data store. There are several well-known optimisations for real-time data stores, including a separation of the write and read indexes[22], and parallelisation of the database according to content domains – to name just two [author's experience].

4.5.3 System Abuse and Student Projects

Security restrictions within a DAN and within its neighbourhood via the mechanisms of the ACL and credentials are designed to prevent malicious abuse of confidentiality, data integrity and authentication of identity. We assume that since the amounts of money in the communities and the system are small, and since Internet connections are limited, there will be little interest to maliciously abuse the system. In the long-term, security of the system must be reviewed. Since interaction occurs via predefined web service interfaces, these should be secure. Automated updates and the possibility of unskilled operators loading and starting *Trojan Horse* modules needs to be guarded against.

A primary weapon against intentional system abuse is the usage logs kept by the middleware. Using these, a system administrator should be able to trace any component that is maliciously misbehaving and stop its components[23]. Also malicious attackers can potentially be traced in this manner, so it is important that logs be kept securely and in a tamper-proof manner by the system.

We also assume that the system can generally rely on the good intentions of student and third party developers, as their business goals are closely aligned to that of the DAN. It is nonetheless feasible that components developed by students will contain errors which may cause those components to malfunction. Since it is reasonable to expect that student projects will in future be rolled out onto the middleware platform to the benefit of the community, buggy components must be assumed. Buggy components can produce the following kinds of error situations:

- System Crash. The component provokes a bug in the middleware or in the underlying technological container. These errors need to be handled by finding the cause of the problem, adjusting test cases to cover the incident if necessary, and programming a patch. The patch could be released back to the DAN in need of it via a secure automatic update mechanism.

- Incorrect message. The component cannot influence the middleware through generation of incorrect messages, as professional components are expected to have been tested thoroughly. In this case, the buggy component should simply stop functioning, as it gets no feedback from the system.

[22]The altavista web search engine first popularised this concept (building on ideas that had been around for a while [?]) – an index writing process would update indexes creating an incremental update index, which would periodically be merged with the full index. An incoming search would have to place two searches – one to the main (full) index and one to the update index. This separation of tasks represented a vast increase in efficiency, as writing to a huge index is a very costly operation, while writing to a small index is not. Further the indexes can easily be served from different machines or clusters of machines.

[23]We assume that the middleware has not been infiltrated – of course it is possible to rewrite the middleware to support malicious activity, but we see this as more of a management or business problem than a technical one.

- Denial of Service Attack (DoS). A component which generates a very large number of messages (notifications, requests, or responses) can unintentionally perform a DoS attack on the system. The design of the architecture can prevent this. Message queues in the middleware can prioritise and even drop messages in special circumstances, thus dampening the effect of the DoS.

- Infinite loops. Since usage events can be chained it is possible that a service can call itself infinitely, thus wasting system resources. The middleware should try to prevent such mishaps.

- Incorrect content. Messages that are correct in syntax, but which are erroneous in other ways are very difficult to detect as they rely on complicated application dependant semantics. Thus, a buggy component, might erroneously subtract X Rands from Client Y's account, causing real damage to a client. While the solution to this problem is outside the scope of this project, we suggest that student components be marked in a way which will prevent them from using services indirectly – and we will assume that critical services may only be used indirectly.

Chapter 5

Implementation of a “Walking Skeleton” and Results

"There is nothing new under the sun. It has all been done before."

Sir Arthur Conan Doyle: A Study in Scarlet (1887)

In this section we discuss the implementation work done as part of this thesis (section 5.1 - 5.3) and the general results in terms of lessons learned and general observations (section 5.4).

We investigated several frameworks in order to identify the most suitable frameworks for our design. While the methodology employed does go to some lengths to prevent a gap between design and implementation, this thesis leaves many aspects (most notably the messaging subsystem) unspecified. The chosen framework should thus provide some guidelines or best practices as to the messaging system. The main intention was not to reinvent the wheel.

As outlined in the requirements we developed demonstrator components in a test-driven manner.

We chose Eclipse as the Integrated Development Environment to be used for the implementation. Minor experiments were also performed directly from the command-line.

A technology demonstrator was developed for the Spring Framework. The demonstrator was not suitable for release to actual users, but was rather intended as a proof-of-concept. Minor experiments were run for JMX and EJB partial solutions. The demonstrator and experiments are discussed in the rest of this section.

This section draws heavily on the author's experience as software programmer and systems designer for international companies in the USA, Germany and South Africa. Where references are not explicitly given, the reader can assume that the statements are the author's opinion, drawn from experience.

5.1 Technology Demonstrator based on the Spring Framework

5.1.1 Demonstrator Overview

An end-to-end technology demonstrator was implemented. It implements one operation (*registerService*()) and joins the most diverse aspects of the middleware, from an out-of-process PHP client to the data access layer. It does not set up the final architecture. Rather it is a "walking skeleton" as proposed by the Crystal Clear implementation methodology (which is an agile method) [Coc05]. It does link several main architectural components with an implementation framework.

The demonstrator layer diagram appears in figure 5.1. As we see in the diagram, several different components (appearing in grey) were implemented. Lower layers of the demonstrator used existing software[1]. The aim of the demonstrator was to show how PHP could be connected to the Java based middleware. An initial attempt at establishing the data access layer was also made. Both of the implemented functionalities used the Spring Framework to achieve these ends. The demonstrator successfully allowed a PHP client to make calls to its web service interfaces, and implemented an in-memory and JDBC (HSQLDB) based data access layer stub.

Additionally, a user interface (UI) was programmed for the demonstrator using servlet functionality. Specifically JSP (Java Server Pages) and STL (Standard Template Library) technology was used to create a rudimentary web front-end. The demonstrator shows how two servlet based applications (UI and web services) can co-exist in the same Spring-controlled address space. The web services could be addressed via a URL such as http://localhost:8080/p2pmwt/mwService/ and the UI was accessible within the same address space as http://localhost:8080/p2pmwt/siyakhula.htm. The configuration required that the address space be segmented for the applications via a Spring *ViewResolver*. This method could be used to implement middleware services.

5.1.2 Spring Framework Technology

This section describes how the Spring framework technology was used. Snippets of code and pseudo-code explain the basic system principles. Of all the technologies presented in Chapter 2, Spring was chosen because it represents an industry standard framework for communications and back-end applications written in Java, and is thus a potentially

[1]The existing packages used in figure5.1 are commonly used Java tools. Clearly all Java programs exist in a JVM. The Tomcat servlet container is what enables the Java servlet technology, which is used by the web service as well as the web front-end. The Spring framework is present in the form of the core framework as well as DAO, WS and View packages.

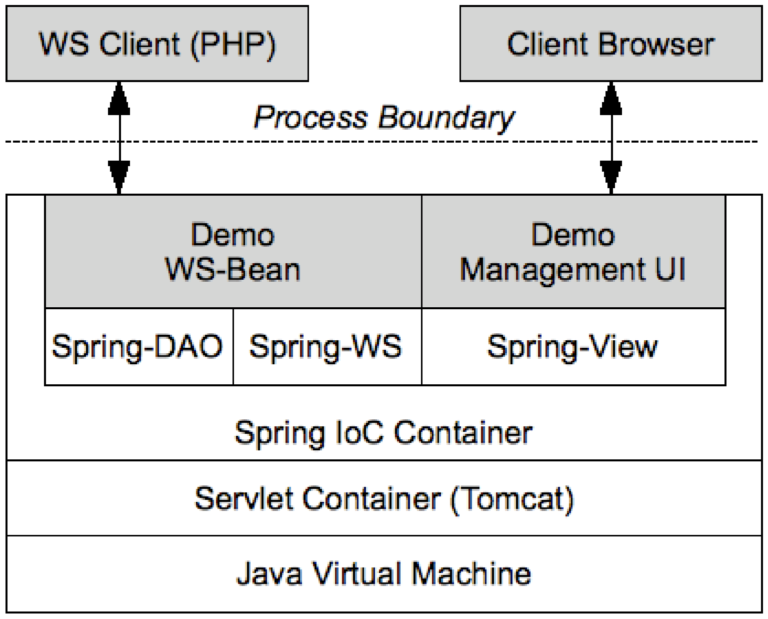


Figure 5.1: Component layer diagram of the demonstrator. Components that were implemented appear shaded.

good fit. Its Inversion of Control (IoC) container allows the developer to focus on business logic and to develop an application with a UI using the Model-View-Controller design pattern. We have not mentioned the MVC pattern in the design chapter, because the UI is not within the scope of this thesis, however, we do mention it here because it is a part of the demonstrator implementation.

Spring works by creating beans out of POJOs (plain old Java objects) which embed the functionality and operations required. The IoC pattern means that a developer's own code is not in control of the flow of events. Instead, POJOs are called by the Spring Framework in response to certain pre-defined events[2]. The main advantage of this is that components can be written in a way which promotes loose coupling and the benefits that come with that (as required, see section 3.3).

The first feature of any Spring application is its configuration. In figure 5.2 we present the configuration of our Spring *application context*. The application context is the method that Spring uses to create the beans that drive the processes being implemented. An alternative mode of operation that may be used is the Spring *BeanFactory*, which we do not explore further here. The application context configuration introduces the major components (which are realised as Java beans at startup time) and shows their dependencies (a priori). This process is often called *dependency injection*, a term originating from Aspect Oriented Programming. The Spring Framework documentation often refers to "injection" or "bean injection" and we use this language below [**?**].

Figure 5.2 shows three beans that will be instantiated. The first two are part of the middleware, i.e. *DANMan-*

[2]Spring is often called "convention based", because the naming of methods and bean properties is not arbitrary. Instead it is linked by convention (or by an exact specification) to the information given in the configuration files. See also discussion on bean getters and setters below.

```
<bean id="serviceManager" class="za.ac.ufh.middleware.SimpleDANManager">
  <property name="serviceDao" ref="dataAccessService"/>
</bean>
<bean id="dataAccessService" class="za.ac.ufh.repository.SQLDataAccessService">
  <property name="dataSource" ref="dataSource" />
</bean>
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
  <property name="driverClassName" value="${jdbc.driverClassName}" />
  <property name="url" value="${jdbc.url}" />
  <property name="username" value="${jdbc.username}" />
  <property name="password" value="${jdbc.password}" />
</bean>
```

Figure 5.2: XML configuration snippet showing how bean dependencies are expressed in Spring. Here, the DANManager bean requires a bean called dataAccessService.

ager and *SQLDataAccessService*. The third bean is a Spring class which attaches a database for simpler use. The dataAccessService bean is injected into the serviceManager bean, and the dataSource bean is injected into the dataAccessService bean. This injection occurs after construction of the object, through a setter on the bean whose name follows the convention "set" + "property name"; in the case of the first bean, this would be *setServiceDao*().

In the previous section (see 4.2.5) a thought experiment had shown how components could call other components unintentionally. Spring attempts to mitigate this problem by introducing the idea of a "bean scope". The default scope with which a bean is created is the "singleton scope". This scope uses the singleton pattern to construct the bean object. That means that the constructor is private and Spring's bean factory will always return the same bean of the type that was first created (in this case that will have been at startup). Spring offers other well controlled methods of scoping[3] beans including the prototype scope and method injection. The latter is a manner of linking beans with different life-cycles to each other and in particular re-injecting beans with shorter lifespans into singleton beans at runtime.

Figure 5.3 shows how we set up the middleware messaging service; specifically our SOAP web services endpoint (or sink) is defined. We reproduce the configuration here to introduce the nomenclature that appears in the following pseudo-code algorithms (algorithms 5.1 and 5.2). According to the method introduced with figure 5.2, we see in figure 5.3 a *WSEndpointHandler* bean that we implemented, which has the DANManager bean injected at construction time and a Spring web services bean which has our *WSEndpointHandler* bean injected. The *PayloadRootQNameEndpointMapping* bean implements the web service communication details and calls our endpoint handler once a message has been received. It passes on the full XML message that was received from the Root Qualified Name of the XML document (hence the name of the bean class). Another feature of the configuration below are the names of the web service calls that will trigger our endpoint. They are specified in the two *prop* elements and are *ApplicationRequest* and *RegisterServiceRequest*. All other requests are ignored by this configuration.

Algorithm 5.1 explains at a very high level, how the message service boots up. Included in the sketch is the loop with which the Tomcat webserver listens for calls of the web service. As mentioned above, Spring inverts the control

[3]Adding a scope to a bean or amending a bean's scope is referred to as "scoping" here.

```
<bean id="servicesEndpoint" class="za.ac.ufh.middleware.ws.WSEndpointHandler">
   <constructor-arg ref="serviceManager" />
</bean>
<bean class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">
   <property name="mappings">
      <props>
         <prop key="{http://ufh.ac.za/schemas/mw}ApplicationRequest">servicesEndpoint</prop>
         <prop key="{http://ufh.ac.za/schemas/mw}RegisterServiceRequest">servicesEndpoint</prop>
      </props>
   </property>
   <property name="interceptors">
      <bean class="za.ac.ufh.middleware.ws.SimpleXmlLoggingInterceptor" />
   </property>
</bean>
```

Figure 5.3: XML configuration of the web service communication channel of the middleware.

Algorithm 5.1 High level pseudo-code explanation of how the Spring Framework invokes the demonstrator web service.

```
// At startup time, the IoC container creates the WSEndpointHandler bean, and
// passes in its dependency -- the DANManager bean, that has already been
// created.
handler = WSEndpointHandler(danManager); // Here our constructor is called
// the following line represents the listen loop of the webserver
while ( true ) {                          // Loop forever

    // The webserver tries to get a new message (blocking)
    msg = getMessage();
    // A message has been received... after some initial processing,
    // Spring invokes the handling of the message by our WS-Endpoint
    response = handler.invokeInternal(msg);
    // And returns the response from the handler to the caller
    dispatch( response );

}
```

and so all calls are initiated by the Spring Framework. This is reflected in the pseudocode algorithm which explains how code we wrote is called by the IoC Container and the higher level web server container.

Algorithm 5.2 explains how our web service endpoint handler calls a business operation on the DANManager. The endpoint is responsible for further unpacking the message and making the correct call on the DANManager. At a later stage security validations that require message level information might be added here.

5.1.3 Web Services Technology

To introduce web services, we will take a quick tour of the development of the web. What is the web? It is originally a decentralised human readable network that is based on the twin technologies of HTTP (the Hyper-Text Transfer Protocol) and HTML (Hyper-Text Markup Language). HTTP is the communication protocol, which relies on Transmission Control Protocol over Internet Protocol (TCP/IP) as a communication basis. A client opens a TCP socket on port 80 of a server and makes a request (see 7.1 for such a conversation). The server answers and the session is closed. This means that requests were originally thought of as being stateless. The answer is meant to be read by a human in a web browser. Early versions of HTML thus mixed formatting and content. Currently, the W3C recommends that XHTML – which is XML (eXtensible Markup Language) – be used to carry the meaning and related CSS stylesheets

Algorithm 5.2 Core demonstrator web services algorithm in pseudo-code showing how the web services end point passes control to the DANManager, which registers a new service in this case.

```
// Definition of Function  invokeInternal( message )
// This is what our message handler does when it is called by
// the container.
// first we note that we received a message
log.info ( ‘‘called with message: ‘‘ msg.id );
// Unpack the XML to find out to which component it is addressed.
service_identity = evaluateIdXPath(message);
operation = evaluateOpXPath(message);
params = evaluateParamsXPath(message);
// Let the DANManager handle the actual operation required
// and get the result of the operation.
result = ourDANManagerBean.evaluate(operation,service_identity,params);
// Create a web services response object -- the result is embedded in XML.
response = createResponse( result );
// And the response is returned
return response;
```

carry the formatting information. The split allows a host of other XML technologies to leverage the meaning of the message, including allowing machines to process[4] the content. Thus XML is often referred to as a “machine-readable format”[Wor10].

XML documents are hierarchically structured. They consist of nodes, which are named (represented by XML Tags) and may contain some content (which can appear in the node, or be attached to the name of the tag as a name-value pair – an attribute of the tag). The first XML technology we need to mention at this point is the XML Schema Definition (XSD). An XSD defines valid content for a certain type of XML document. XHTML documents conform to a specific XSD, and are thus a particular type of XML document. Web services transmit machine readable information in messages and are commonly stateless – as was the original web. Their message format is commonly XML, which is also understandable to humans (although namespace information does tend to clutter XML). Web services advertise their services in a suitable WSDL file. The Web Services Definition Language (WSDL) also belongs to the XML family of specifications.

Since we are transmitting XML in our web service messages, we can define the valid format of our messages in an XSD. Our schema is fairly short and is reproduced in the appendix 7.1. The schema used for the demonstrator simply defines the operations available on the middleware, which are very limited. These would have to be extended to include all the operations in interfaces *IUseDAN* and *IAdministerDAN*.

Spring offers a simplified process to automatically generate WSDL files from our XSD.

5.1.4 JUnit

JUnit is a an open source library commonly used to implement test-driven development in Java. The convention based functioning of the library has evolved since the first versions and JUnit functionality is now also available through Java annotations. We created a parallel package structure to the source and placed all our tests in that parallel structure.

[4]They still cannot understand the content – but let us not mention the story of Semantic Web Services here.

Algorithm 5.3 Java test algorithm for adding a service

```
public void testRegisterService() {
    int svcs_a = danManager.getServices().size();        // check number of services beforehand
    // register a new service, i.e. add a service
    danManager.registerService(B_IDENT + svcs_a, B_DESCRIPTION + " " + svcs_a, "http://dummy.wsdl.loc");
    int svcs_b = danManager.getServices().size();     // check number of services after the addition
    assertEquals(svcs_a + 1, svcs_b);                    // there should only be one additional service
    // make a snapshot of services and check that the identifier of the latest one matches
    List<BasicService> services = danManager.getServices();
    BasicService service = services.get(svcs_b - 1);
    assertEquals(B_IDENT + svcs_a, service.getIdentifier());
}
```

This is common practice with testing and ensures that tests can easily be matched to the source code that they test [author's experience]. The package structures can be seen in the appendices in section 7.1.

Algorithm 5.3 shows the logic used to test the addition of a service to the DANManager. The test quite clearly echoes the business rules that were analysed in the design (see section 4.2.1, "Adding a service") . The comments in the code link clearly to the business rules discussed earlier.

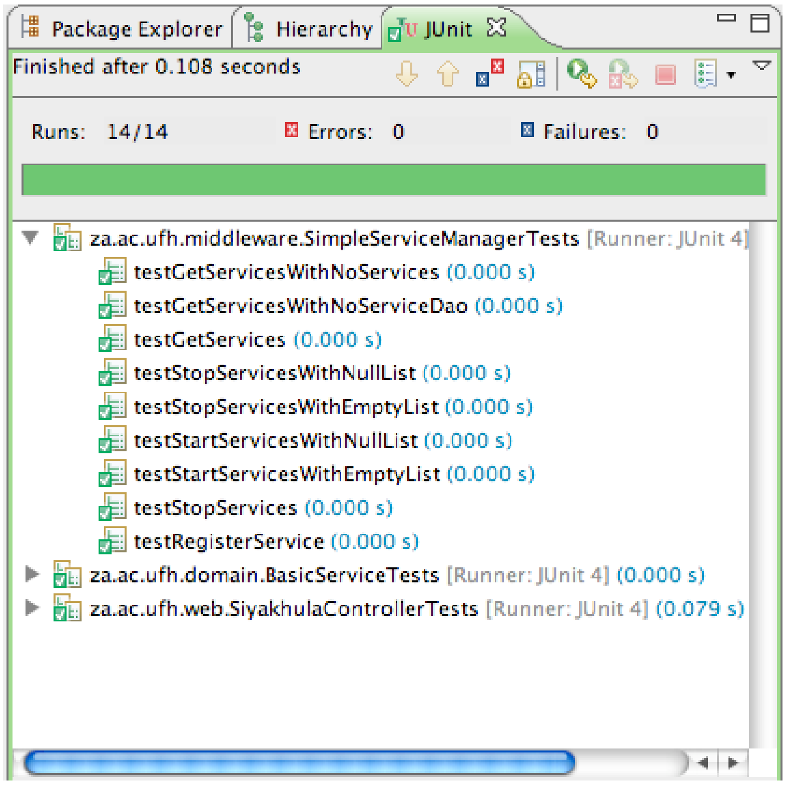
The convenience of JUnit is demonstrated in the following screenshot (figure 5.4). JUnit is integrated into the Eclipse IDE and can be run at any time. Unit tests promote an agile development methodology, because they provide a good indicator that basic functionality is still intact after wide-reaching changes have been made. Such changes become necessary, e.g., when interfaces change and code refactoring takes place. This contributes to the sustainability of the development process, when the final goals of the development are not precisely laid out (as e.g. in this thesis).

Testing the web services is a little more complicated, because the web services endpoint described in algorithm 5.2 takes a complex object type as an argument. The argument is an XML document (named *message* in the algorithm) and needs to be recreated for the test. There are two common methods to solve the problem of testing with complex objects: either create a *TestObjectFactory*, which generates test objects and passes them into the test without requiring an entire web services infrastructure to be started, or use the Mock Objects pattern[CP02]. Mock objects (similarly to agents) implement the same interface as the argument required by the method we are testing. However, unlike the XML Document a mock object will include logic designed to test certain aspects of the method being tested, by incorporating delays, throwing exceptions etc. The mock object is thus more like a test agent than a simple data structure, and it is able to provoke certain expected responses in the code.

5.1.5 Extending the System

In terms of the implementation, extending or amending the core system follows these steps:

1. install IDE
2. import project
3. make changes



```
Finished after 0.108 seconds

Runs:  14/14        Errors:  0        Failures:  0

▼ za.ac.ufh.middleware.SimpleServiceManagerTests [Runner: JUnit 4]
      testGetServicesWithNoServices (0.000 s)
      testGetServicesWithNoServiceDao (0.000 s)
      testGetServices (0.000 s)
      testStopServicesWithNullList (0.000 s)
      testStopServicesWithEmptyList (0.000 s)
      testStartServicesWithNullList (0.000 s)
      testStartServicesWithEmptyList (0.000 s)
      testStopServices (0.000 s)
      testRegisterService (0.000 s)
▶ za.ac.ufh.domain.BasicServiceTests [Runner: JUnit 4] (0.000 s)
▶ za.ac.ufh.web.SiyakhulaControllerTests [Runner: JUnit 4] (0.079 s)
```

Figure 5.4: Screenshot of JUnit in Eclipse IDE

Integrating your project on the other hand, for instance if you have a specific technology that you need to use, will follow these steps:

1. Design the system, taking care to decompose it into an application and a service components (if required)

2. Construct the information model and plan what information sharing and using needs to take place.

3. Implement: ensure that developed components are interface compliant (test using mock objects).

4. Integrate: Upload the system.

5. Run: register application and service, and perform end-to-end tests on the new functionality.

In both cases a design phase that covers the topics explained in section 4.3, must be performed.

5.2 Integrating PHP

An exemplary integration intended for the eCommerce system, was programmed and linked to the Spring framework demonstrator. We created a PHP function which registered a dummy service with the middleware demonstrator. The core elements of the PHP function can be seen in algorithm 5.4. The following is a line-by-line explanation of the code:

Algorithm 5.4 Algorithm in PHP 5 showing exemplary integration with Spring based demonstrator.

```
$wsdl_url    = 'http://localhost:8080/p2pmwt/mwService/middleware.wsdl';
$wsdl_ns     = 'http://ufh.ac.za/schemas/mw';
$client                 = new SoapClient($wsdl_url,              //  (1)
                        $options);                               //
$registerservicerequest = new stdClass();                        //  (2)
$registerservicerequest->Identifier                              //
                        = "eCommerce Service";                   //  (3)
$registerservicerequest->Description                             //
                        = "eCommerce Service \                   //  (4)
                       test registration";                       //
$registerservicerequest->WsdlUrl                                 //
                        = "http://test.com/wsdl";                //  (5)
$rsrType                = new SoapVar(                           //
                        $registerservicerequest ,                //  (6)
                        SOAP_ENC_OBJECT,                         //
                        "addServiceRequest",                     //
                        $wsdl_ns);                               //
try{                                                             //
       $response        = $client->RegisterService(              //  (7)
                        $rsrType);                               //
} catch(SoapFault $e) {                                          //
       trigger_error(                                            //  (8)
                        'Something soapy went wrong:'            //
                        .$e->faultstring,E_USER_WARNING);        //
}                                                                //
```

(1) Create new SOAP client. The SOAP Client is a built-in PHP 5 object which can call Web Services for one transparently.

(2) Create a PHP RegisterServiceRequest object, to which we will attach the parameters that are required by the call.

(3) Set the Identifier of our service

(4) Set the Description of our service

(5) Set the WSDL address of our service

(6) Create the SOAP "*addServiceRequest*" call, including the object created as a parameter.

(7) Here our SOAP client makes the call to the middleware, receiving a response (*$response*).

(8) The SOAP client may throw an exception or fault, which we want to catch and process.

Simple PHP functions such as the one described above can be defined for all the operations that are required from the middleware. It is important to note that the above work is of a prototypical nature, because the interfaces to the middleware have not been defined yet. Some more work needs to be done on the actual type signatures of the middleware interfaces before the PHP code can be cleanly implemented.

5.3 Technology Experiments

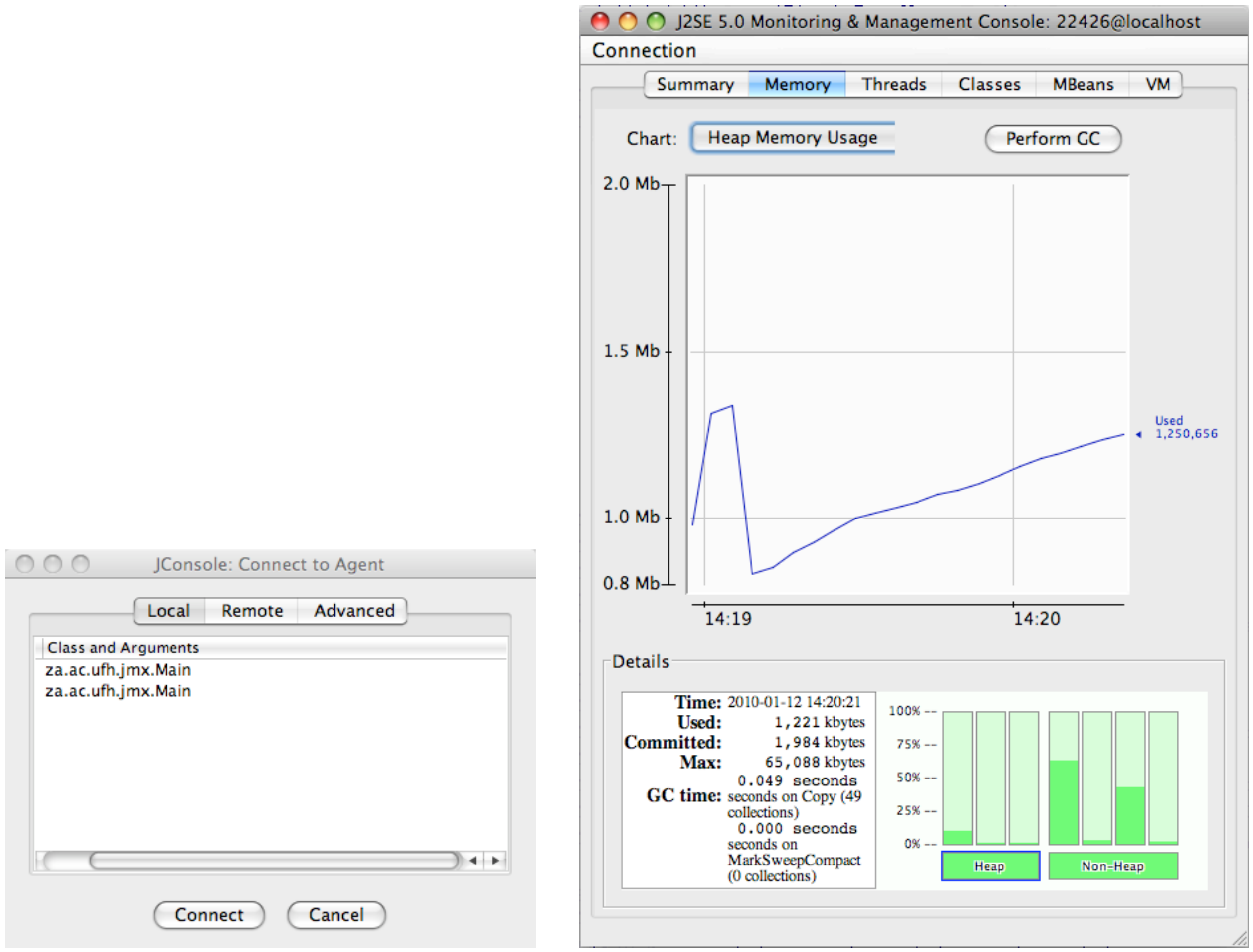
5.3.1 JMX

DANManager subcomponents are contained within the DAN Manager and are composed to present a facade with a wide variety of functions to the web services world. Internally, these components should not communicate using web services, as web services impose a performance penalty because of marshalling and unmarshalling of objects as XML information. The distinction between regular services and DANManager subcomponents is clearly visible in the design diagrams (see e.g. 4.8). It is also possible to define several communication channels for services, to allow services written in Java to be more easily accessible from the middleware than via web services.

Spring and JMX are compatible technologies, which can assist in this task. JMX already includes all the services required to control a service oriented architecture as explained in section 2.2.1. We repeat here, that JMX does not conform to our requirements of language independence, which is why we chose not to make JMX (and thus by implication Java Remote Method Invocation (RMI)) the method of communication between all components in the middleware. We know of no bindings for PHP etc. to Java RMI, nor of any initiatives attempting such a coupling. For a tighter coupling than web services, CORBA may be used.

The objective of the JMX experiment was to see how easily one can access methods and data on a bean via JMX. We implemented the experiment, by altering the Sun JMX tutorial, allowing public access to methods on a bean. We then used the standard Sun JMX management application (JConsole), which is delivered with the Sun Java SDK, to attach to, inspect and also control our bean. In figure 5.5, we see (a.) the available JMX agents to which one can attach, and (b.) basic information about the bean and its process. We can see summary information about the process uptime, memory usage (in detail), the numbers of threads, class loading information and JVM information. Such information can be very important when optimising a distributed SOA.

In figure 5.6 we see how we can manipulate the bean via the JConsole interface. Such manipulation and notification occurs via the JMX API (which JConsole uses to implement its functionality). All functionality displayed by the JConsole application are also available to us through the API. Thus, basing this experiment on JConsole is a quick method of getting more insight into the possibilities afforded by JMX.



Class and Arguments
za.ac.ufh.jmx.Main
za.ac.ufh.jmx.Main

Time: 2010-01-12 14:20:21
Used: 1,221 kbytes
Committed: 1,984 kbytes
Max: 65,088 kbytes
GC time: 0.049 seconds seconds on Copy (49 collections)
0.000 seconds seconds on MarkSweepCompact (0 collections)

(a) Attaching JConsole to available JMX agents on localhost.

(b) Richness of detail in the JMX standard application – JConsole.

Figure 5.5: JConsole standard screens, showing ease of use and richness of information.

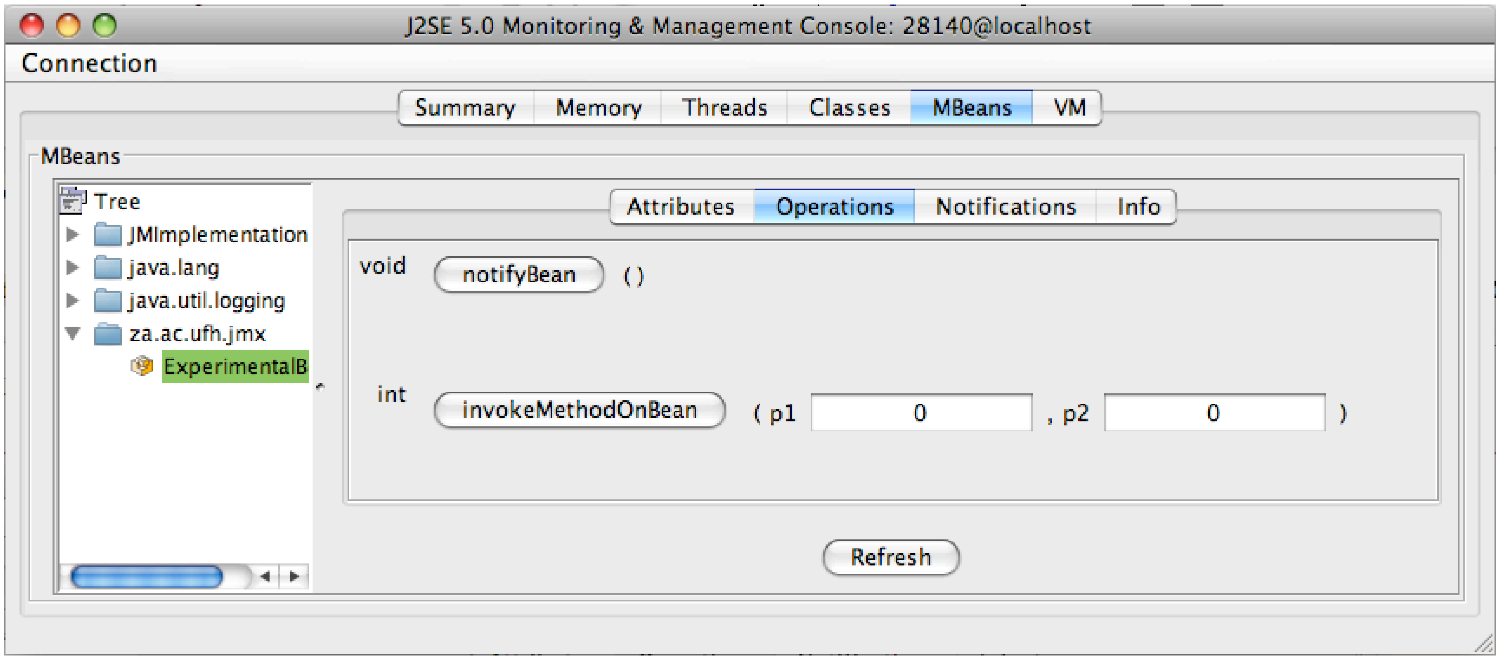


Figure 5.6: JConsole demonstrating dynamic MBean method and attribute discovery via Java reflection. JConsole natively allows us to notify the bean and invoke its public methods.

5.3.2 Notes

Servlet containers and EJBs

Since we have opted to use Java, it is logical that we will be working with a container[5]. We used the Apache Tomcat container in our demonstrator because it is very well integrated with the Eclipse IDE. A typical implementation deployment scenario from the author's experience uses an Apache webserver with a Tomcat servlet container. The Apache webserver is optimised for serving static pages and can implement optimised caching strategies, as well as allowing access to a host of other services. It passes requests for dynamic pages on to the Tomcat container. This scenario can be used with other servlet containers, e.g. JBoss and GlassFish.

At the time of writing a new GlassFish server version 3 implementing the new Java Enterprise Edition (Java EE) 6 APIs was released by Sun. As mentioned in section 2.2.1 GlassFish is a strong contender for middleware framework technology along with EJBs. GlassFish (version 2) natively includes JXTA P2P capabilities. GlassFish uses JXTA for communication between web servers [6], which is an implementational view of the task of the PeerManager.

EJBs also offer the advantages of IoC and may moreover offer several communication channels we should like to use, such as communicating via web services and SOAP with third party services and via RMI with our own

[5]Several tried-and-tested containers which encapsulate the communication stack exist for Java (Tomcat, JBoss, OAS, etc.). The alternative is an implementation of an own transport stack, which in the most extreme case uses java.nio packages for socket communication. In a less extreme case, one could build an Apache Webserver module, or embed a Java webserver, such as Jetty, into one's system. That way, the lower (HTTP-level) transport layers would already be implemented in a managed fashion.

[6]Specifically JXTA is used for load balancing and clustering of web servers. Although our system latencies are greater than in the aforementioned use contexts, there are strong similarities between the contexts.

services (regardless of whether they are sub-components or not). However with an increase in abstraction often comes a performance penalty. Further experiments regarding container technology, especially concerning GlassFish and EJB functionality and performance, should be undertaken.

Single Sign-on (SSO)

The login process is so trivial as to have been relegated to a minor task in most previous projects as reviewed in table 3.2. However, several of the projects in which we were involved started with the implementation of this functionality, because it was a "quick win"[7]. It is our opinion that these efforts were wasted, as the project implementors should have been concentrating on their core business, not on an auxiliary functionality that was only tangential to the project. Further, login functionality, if developed from the ground up must be implemented securely, and that can prove to be deceptively complex. For instance, there are several attacks that can be made on schemes which naively save hashed passwords in a user database, e.g. the rainbow table attack, and such considerations were usually neglected in the projects reviewed by the author.

We compiled a list of open source single sign-on solutions, which are widely used and respected: GIS-CAS (used in the Grid world, e.g. by GT4 – see section 2.2.1), OpenSSO, JOSSO, Open ID (which is often not considered a SSO scheme) [DEL97, AMM04]. The most interesting of these is Java Open SSO (JOSSO) as it is also supported by the GlassFish servlet container. This is a further argument to experiment with the GlassFish server.

An experiment with SSO functionality should be undertaken.

5.4 Results and Discussion

5.4.1 Lessons Learned

A key result and lesson learned of this project is the design process that we carried out and documented in Chapter 4. The design presented contains a minimal and necessary set of operations for a DAN middleware. It takes into account basic non-functional aspects, such as security and performance, as well as functional aspects, as expressed in the use case analysis and requirements engineering. State of the art service oriented architectural principles are applied to satisfy the business requirements. Performance and flexibility are carefully weighed up in order to accommodate future services and applications.

[7]A quick win is an easy task which helps one achieve a simple or minor tactical goal, but thereby securing some strategic objective, like customer trust, knowledge of tools, etc.

Frameworks lessons

We looked at several major frameworks in the scope of the implementation as seen above. The general ease of use of these free/libre open source tools was excellent. They all included documentation and tutorials to speed implementation tests and adoption. Reference documentation was thorough and even more complex questions such as method scoping were addressed in detail. This is in stark contrast to the state of some of the FLOSS libraries the author used in the early 2000's. We attribute this change to the facts that:

- we only considered mainstream FLOSS software,
- the packages we considered were usually supported by a large company,
- and they had a large user base.

FLOSS has attained a level of maturity, such that it is natural for thousands of programmers to use FLOSS tools to build a wide variety of systems. The small percentage of programmers that does put something back into the projects are sufficient to keep the tools up-to-date and correct.

Web Services compatibility: Java <-> PHP

Theoretically, this is not an issue[8], but we did find practical issues. Java very cleanly implements the XML specification and handles XML namespaces in a manner which is consistent with the standards and what one would expect. Web services capability is fairly new in PHP, appearing only in PHP 5. Prior to that one had to use the PEAR library or other solutions. We found that PHP did not add the specified namespace to the included XML payload, and gave elements in the payload the default namespace (i.e. no-namespace, for the documents we were sending). This caused problems with the Java web services marshalling performed by the Spring framework, which expected the payload also to have a namespace provided. The Spring behaviour could be overridden, but we could not override the PHP behaviour[9]. We could not find a way to include namespaces into the message payload by using the web services API. We would have had to perform XML processing ourselves to achieve this. Debugging this behaviour was time consuming.

This incompatibility was reminiscent of CORBA Inter-ORB compatibility test the author undertook in 2003 as part of other projects. While the Java ORBs were able to communicate well with each other, Java C++ ORB compatibility only worked for atomic objects, and not for vectors. Theoretically, the ORBs, using CORBA 3.2, should have been able to communicate with each other, since CORBA 3.2 specifies its wire format.

[8]Since web services define their interfaces as well as the "wire format" – the format in which data is passed between applications – integration should be trivial. This is a contrast to the case with early versions of CORBA, which were not explicitly compatible across development technologies (see later discussion on CORBA).

[9]A further work-around to this problem, which we used successfully in the past when namespaces were inconsistently used by software, is to use XPath expressions that target the *local-name()* of XML elements. This XPath expression ignores name-spaces and allows one to check namespaces separately in the cases where they are required as a security or consistency check.

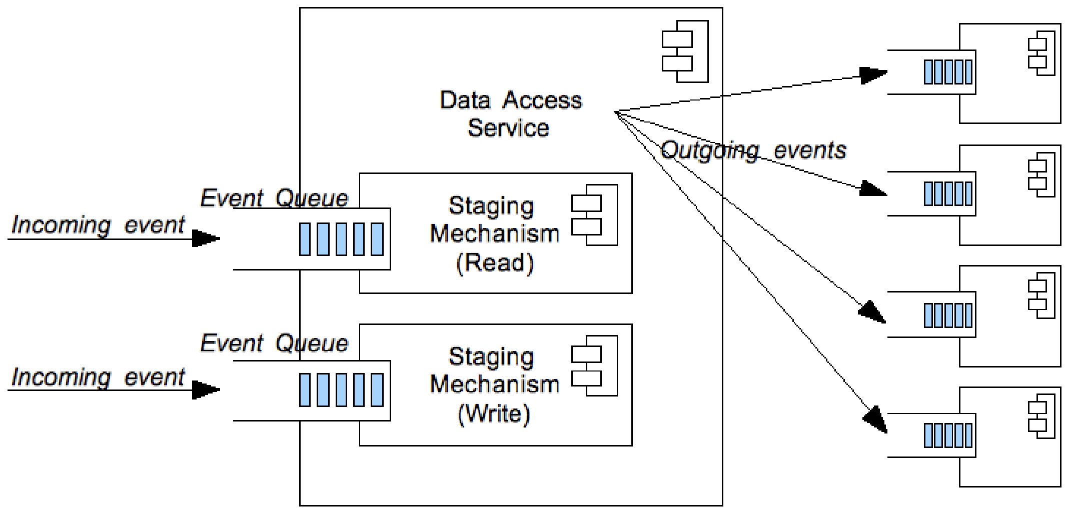


Figure 5.7: Component Architecture showing event queues connecting basic platform components, in a staged event driven manner (after Welsh [WCB01]).

Since we are using open source software, it is possible to delve deeply into the problem and to find the exact cause of the problem and even to fix it – if it is critical to the project.

Staging Components and Loose Coupling

One of the lessons that we expect will appear when stress tests are performed on the system is that components should be connected in a staged manner. The IoC pattern is an excellent decoupling mechanism and should be employed whether via Spring, EJB or combination of the technologies. The POJOs which implement the components should be implemented using principles from the staged event driven architecture (SEDA) or Message Queue Blueprint (see section 2.2.3).

From an implementation aspect, these components should extend a Staging Mechanism (an interface with abstract implementation), whereby all messages arriving are first queued, before being processed in a threaded manner. The number of threads performing the processing for each stage, and the maximum length of the queues would be dynamically controlled according to demand in the system via JMX MBean interfaces. This could occur automatically using algorithms such as those described in [WCB01], or manually via a GUI. It is highly likely that Spring and other containers already have dynamic means to support such processing requirements.

In figure 5.7, we see the staged event driven architecture applied hypothetically to a component in our component architecture illustrating statements made in this section.

5.4.2 Middleware is Invisible

A middleware project can be a complex undertaking. This section is based on the author's experience in three projects which involved large scale middleware implementations, DFN S2S [Wer03] and the implementation of a distributed real-time communications platform in a startup with more than 350 million Euros of startup capital (name may not be disclosed) as well as a medium scale implementation – the Kapenta framework [Str10] which drives awareNet[Vil10].[10]

The contexts mentioned above shared the following characteristics:

- Developers were faced with a new problem context – one with which they were quite unfamiliar.
- Neither a business owner with a clear business plan, nor a system specification, nor a technical work contract were present.
- In each case a first version of the software was developed by experienced programmers (and system designers on one case). I.e., the programmers applied general principles that they had learned elsewhere, to the problem domain.

In all three cases, after initial research, the software teams decided to develop their own framework, which was specific to the problem domain. The advantage of this method is that the entire concept and code base comes from a single source and is often internally very consistent. Shifting specifications from an unclear source can quickly be implemented, once the framework is completed. The disadvantage of this method is that the entire concept and code base comes from a single source (the team) and misses lessons learned that are embedded in other prior art – research literature, software packages that address portions of the problem, etc. Combining developer blindness and project owner desire for tangible visible results – middleware projects often end in the unsatisfactory state where ecstatic developers expound the virtues of the "emperor's new clothes" to perplexed and irritated business owners. Two of these projects were shelved ultimately, even after personnel changes had been made and systems rewritten. Kapenta continues to function, because it is domain specific (it doesn't try to solve too many problems) and is a small and efficient package. Kapenta, too, has been rewritten several times, however the simplicity of scope and simpler business plan have ensured that it remains in use.

The particular problem encountered with middleware is that it is difficult to construct a satisfactory walking skeleton on which to hang functionality. The messaging system and many of the components seem to initially have no purpose but to talk to each other in a closed circuit kind of fashion. Eventually a stage may be reached in which the middleware functions correctly, as proved by stress-tests. However, this is not a walking skeleton as it fails to go from end to end of the functionality chain, it only covers non-functional aspects of the project. Besides producing log messages, this type of demonstrator produces nothing tangible. Middleware is by nature invisible – it does not however do nothing. It facilitates interactions between compositional units, across contractually specified interfaces and making

[10]Not all of the following comments apply to all of the three projects mentioned, nor do all of the projects mentioned necessarily have technical flaws. Rather this section is a psychological analysis of the dangers of middleware development.

explicit dependencies between such units. The problem with the walking skeleton is thus not the middleware, but the functionality one that should be plugged into the middleware. Such functionality needs to be as clearly specified as the middleware itself and aspects of it need to be demonstrated in the walking skeleton.

We feel that we have avoided these traps in this project. By concentrating on the design and not trying to completely implement the middleware, we have managed to create an end-to-end skeleton using existing FLOSS frameworks. We have been able to make educated technology choices which could potentially bootstrap the actual implementation of such a middleware considerably, by allowing developers to focus on the core business elements and letting business owners know what they should expect. We feel that we have lit a way forward which could now more easily be followed by those that come after.

5.4.3 ESTIMA Project

The eServices and Telecommunications Infrastructure for Marginalised Areas (ESTIMA) project proposal was submitted to the South African and Finnish knowledge partnership on ICT (SAFIPA) in September 2009. The proposal outlines a business plan for rural Digital Access Nodes and rationalises the adoption of such a scheme in the Eastern Cape of South Africa and beyond. The proposal is related to this thesis, as it requires a technological platform similar to the one proposed in this thesis. The approval of the ESTIMA project, which was stringently reviewed, is in a sense a ratification of some aspects of the ideas presented in this thesis.

In addition to some technical data presented in the proposal, the following use case was mentioned to demonstrate the kinds of functionality that may be created in the ESTIMA project. What is visible in this use case is the width of application functionality that can be supported by the architecture presented here.

In this use case, a DAN receives and installs the platform with an eCommerce service, a mobile service and an cost sharing service (which allows equitable sharing of the service within the community). The software may be obtained for free in the case of an individual centre, which is trying out the system, or it may be deployed to the centre by regional government in the context of a region wide roll-out. The telecentre operator (a champion) is responsible for the running of the platform (paying costs associated with the system to keep it running). The operator is trained by a partner of the factory, such as Schoolnet SA or the Rhodes University Education Department (initially). The operator explains the system to the community and makes sure that it is used by artisans. Artisans place their goods into a portal for sale in the Internet. The unique value addition provided by the platform is:

- The ecommerce or shopping system itself can be any current shopping system (there are good Open Source varieties available for free).
- Adding support for platform interfaces allows the shopping system to be easily integrated :
  - with platform mobile services
  - with a communal billing and rating service, which provides equitable charging of the individuals to keep telecentre services running
  - into the distributed platform security
- A shop owner's workflow can track the completion state of the order. The shop owner is sent reminders via SMS.
- A novel mobile phone interface, which can also be accessed, via bluetooth services to upload large numbers of orders at any telecentre will be available.

Standardised services can all make use of similar shared services available on the platform which are easier to develop and easier to understand for operators who have to explain them to the users. Through the use of modern process technology such as workflow systems, transparency can be added to the system.

Table 5.1: Extract from ESTIMA proposal

Chapter 6

Conclusion

"The dream of reusing off-the-shelf components has taken longer to materialise than first envisaged."

Cheesman and Daniels 2001.

This thesis meets all the primary objectives as set out in the introduction. Objective 1 (designing a middleware package after a thorough requirements engineering) has been satisfactorily attained. Objective 2 (investigation of FLOSS frameworks for the implementation of the middleware) has been completed in a theoretical sense – a number of software frameworks were reviewed and analysed. In a practical sense, frameworks that are *good enough* to fulfill the requirements were cursorily examined – a further practical investigation of the technologies needs to be undertaken. We present a précis of the core arguments presented in this thesis, in summation of the results of this project.

In Chapter 2, we argued the following:

(1) We recognise a new sort of topology in rural village networks. Through cheap, high-speed, wireless point-to-point equipment (among other forms of technology) village networks will, in future, span large tracts of land containing several villages in what we call "network islands". These networks will have poor Internet connectivity and they are a sub-class of mesh networks (which do not define any connectivity quality to other networks).

(2) We point out prior work which supports the idea that electronic services can assist in developing marginalised rural areas. We also look at several services being developed for the rural context in a stand-alone fashion, within our work group as well as by others.

(3) We point out prior work that supports the idea that a wide variety of eServices can assist an unsustainable rural telecentre in making the quantum jump to sustainability and point out that resource sharing can greatly increase the number of eServices offered at any given point in a rural network. We also introduce the idea of a DAN to differentiate our work from the pre-existing concept of a telecentre. A DAN is always part of a network island, just as a node is always part of a network (in graph theoretic terms).

(4) We deduce from (1), (2) and (3), that a middleware solution can facilitate technical resources at DANs within a common network island.

A further chain of argumentation follows, spanning Chapters 3 and 4:

(5) We analyse the requirements of such a middleware looking at functional and non-functional requirements. These are further grouped into rural and development laboratory requirements (in Chapter 3). We also compose a list of assumptions underlying a functional network island (see tables 3.3 and 3.4).

(6) We loosely deduce from (5) the key components of a system which would fulfill the requirements (in Chapter 4). A key feature of the design is the integration of a shared data access layer, which could store context-dependant information models that could be shared among applications and could bootstrap new applications by allowing developers to access resources they would not normally have been able to (thus enabling them to develop new context-related applications).

(7) We propose formulas derived from (5) and (6), that may predict how the network usage at peak times would affect network traffic and bandwidth (see equation 4.4). These demonstrate that the middleware can operate effectively in network islands and add to sustainability.

In Chapter 5 we argue that Spring and a combination of other technologies, such as JMX and JUnit, could be used to develop the system from (6). We also show that other technologies such as PHP can be integrated seamlessly. We argue that using open source software can also improve sustainability of the DAN middleware project.

The core innovations (or novel insights) of this work can be found in the following:

- The concept of network islands (in Chapter 2);
- the requirements of a middleware for DANs (in Chapter 3);
- a component based design for an ad hoc distributed DAN middleware containing an integrated contextual information model (in Chapter 4); and

- key lessons learned from several projects which were undertaken to implement middleware solutions (in Chapter 5).

As mentioned in Chapter 5, a key result of this project is the design process which was followed to produce the DAN middleware design. It is worth noting that the combination of SOA and P2P principles, is fairly novel with little research in this direction. The alignment of the design with the business analysis of the DAN do make the result new. Further we hope that the discussion of the process itself may be revealing to readers hoping to find out about component based software design.

Programming and software design is a lot about dreams and visions. In that sense it can be a dangerous undertaking with undesired ramifications (see the quote at the beginning of this Chapter and section 5.4.2). To the author it is a manner of interacting with the world which is at once `{ within && without | virtual && real | theoretical && applied }` and thus one of the most deeply satisfying pursuits available to us all.

6.1 Future Work

Throughout this thesis, there are references to work which needs to be done to either refine the logical argumentation or to add certainty to *a posteriori* statements (i.e. determined from empirical data). In this section we gather all of these statements of future imperative, so as to make it clear what kinds of work may follow on from this project.

Mobile aspects relevant to the thesis

Mobile technology is very important to sustainability of rural ICT projects, as it is a very important access technology (see e.g. section 2.1). While the mobile device as a viewing mechanism is out of the scope here (as are all UI issues), mobile devices could be included in the middleware architecture if they have a Java Virtual Machine allowing execution of the middleware. With IEEE 802.15 (Bluetooth) communication methods, such mobile devices can add substantial weight and new applications to the middleware system without necessarily incurring costs[1] on the owners of the devices.

Thus this important technology extension to the architecture should be evaluated from a functional perspective, in terms of new use cases possible in the DAN. For instance, it could allow email, chat or browsing via the middleware from mobile devices connected to the DAN, when there are no seats free in the DAN and even from outside of the DAN during closing hours. Further, many of the assumptions in table 3.4 would need to be reassessed, i.e. a non-functional reappraisal would also have to follow.

[1]Excluding electricity costs – if the device is not charged from a renewable cost free source, such as e.g. a private photovoltaic panel.

P2P topologies

There are several reasons why a structured P2P topology could need to be introduced to the network. Fundamentally these are: a very much larger maximum number N of nodes in the use cases, and the need for more predictability in the network despite peer churn. In the former case, using the naive neighbourhood idea (as in early Gnutella protocols, such as v0.2), separate P2P network islands within the physical network islands may be formed. Also network query flooding might damage response times. In the latter case, structuring the peer network can reduce unpredictability and irrational behaviour on the network. This can be done automatically, with no set up or administrative overheads, i.e. without compromising important requirements concerning administrative overhead. Structuring of the network can occur either via a peer differentiation, i.e. creating a super-peer network (see e.g. [Wer03]), or via data replication, typically using DHTs (see e.g. section 2.2.2).

Refinement of mathematical predictors

Predictions about the non-functional performance can be made using equation 4.4, etc. The constants used in these equations need to be examined as deployment occurs in order to reduce error.

Completion of the walking skeleton

Here we list some of the topics raised around the walking skeleton:

- A messaging subsystem needs to be implemented in a manner that fulfills performance and integration requirements, within and between components.
- GlassFish and EJB functionality and performance experiments should be undertaken.
- An experiment with SSO functionality should be undertaken. An evaluation should follow of a middleware design with stringent SSO and ACL checking policies to determine whether such a design would be secure to the problems introduced in section 4.5.3.
- Concrete use cases for the staged event driven architecture should be investigated on a component by component basis, especially at components likely to be handling the majority of messages. Simulations can be run to determine effectiveness of the method in a particular component architecture.

Development of a prototype

The skeleton should be fleshed out with advanced use cases, such as those presented in sections 4.3 and 5.4.3. The prototype should then be tested with end-users at a very early stage in order to get early feedback and correct the problem of a lack of business owner in the project. In this manner, the end-users may be motivated to co-create the product.

A prototype with advanced functionality could also allow experimentation with the sorts of technology which might not occur to grassroots users. Such advanced experimentation should also occur in the lab[2].

[2] **Advanced topics for future consideration:**

- Integration of Semantic Web Services.
- Integration of Real-Time Streaming Service, including voice applications (telephony) and video conferencing.
- Security for Mobile Money and Micro-Billing Services.
- Etc. (creativity is the only limit)

Chapter 7

Publications

Publications by the author that directly stem from this work:

- Ronald R. R. Wertlen and Alfredo Terzoli. *Peer-to-Peer Web Services for Distributed Rural ICTs* (2009) [WT09]
- Ronald R. R. Wertlen. *An Overview of ICT Innovation for Developmental Projects in Marginalised Rural Areas* (2008) [Wer08]

7.1 Related publications

These are publications that are in progress or were peer-reviewed and published at scientific proceedings and that have direct relevance to this work, but which are not a direct result of the work:

- Ronald R. R. Wertlen. *Description of a Solar Computer Lab Network in a Marginalised Rural Area of the Eastern Cape* (2009)[Wer09]
- Bobby Jakachira and Hyppolite Muyingi and Ronald Wertlen. I*mplementation of a web-based E-government proxy for a marginalized rural population* (2008) [JMW08]
- Ronald R. R. Wertlen. *DFN Science-to-Science: Peer-to-Peer Scientific Research* (2003) [Wer03]

List of References

[ABZC08] M. Amoretti, M. Bisi, F. Zanichelli, and G. Conte, "Enabling peer-to-peer web service architectures with jxta-soap," in *IADIS 2008: Proceedings of the IADIS International Conference on Mobile Learning, Conference CD, Algarve, Portugal*, 2008.

[ACG08] R. Ambikar, C. Coward, and R. Gomez, "Information needs and watering holes: public access to information and ICT in 25 countries," in *5th Prato Community Informatics & Development Informatics Conference 2008: ICTs for Social Inclusion: What is the Reality? Conference CD, Prato, Italy, Centre for Community Networking Research, Monash University*, 2008.

[AGA+08] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, "SOMA: A method for developing service-oriented solutions," *IBM Systems Journal*, vol. 47.3, pp. 377 – 396, 2008.

[AH00] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, vol. 5:10, Available online: http://www.firstmonday.dk/issues/issue5_10/adar/index.html. Last visited: 2005-11-09, 2000.

[Ale07] Aleutia, "The aleutia project," 2007, available online. http://www.aleutia.com/ see also http://wiki.aleutia.com/. Last visited: 2007-04-13.

[AMM04] P. Aubry, V. Mathieu, and J. Marchal, "ESUP-Portail: open source single sign-on with CAS (central authentication service)," *EUNIS 2004: Proceedings of the European University Information Systems conference, Bled, Slovenia*, 2004.

[And05] N. Anderson, "Building digital capacities in remote communities within developing countries: Practical applications and ethical issues." *Information technology, education and society*, vol. 6, no. 3, 2005.

[AP09] J. Antikainen and S. Pekkola, "Factors influencing the alignment of SOA development with business objectives," in *Proceedings of the 17th European Conference on Information Systems (ECIS 2009), Verona (Italy)*, September 2009.

[AR06] B. Angelov and B. Rao, "The progression of WiMAX toward a peer-to-peer paradigm shift. IEC Newsletter http://www.iec.org/newsletter/oct06_2/ (visited 2009-05-03)," 2006.

[Art09] C. Arthur, "It's about money, not reputation," *Guardian Weekly*, vol. 181, no. 23, pp. 30–31, 2009.

[Baj07] F. R. Bajwa, "Telecentre technology: The application of free and open source software," 2007. [Online]. Available: http://www.apdip.net/news/apdipenote19[Lastvisited:2009-01-24]

[BBLV04] G. Barrenechea, B. Beferull-Lozano, and M. Vetterli, "Lattice sensor networks: capacity limits, optimal routing and robustness to failures," in *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks.* New York, NY, USA: ACM, 2004, pp. 186–195.

[BBW09] A. Becker, P. Buxmann, and T. Widjaja, "Value potential and challenges of Service-Oriented Architectures - a user and vendor perspective," in *Proceedings of the 17th European Conference on Information Systems (ECIS 2009), Verona (Italy)*, September 2009. [Online]. Available: http://tubiblio.ulb.tu-darmstadt.de/35831/

[BEH$^+$03] S. Batchelor, S. Evangelista, S. Hearn, M. Pierce, S. Sugden, and M. Webb, *ICT for Development Contributing to the Millennium Development Goals: Lessons Learned from Seventeen infoDev Projects.* Washington D.C.: World Bank, 2003.

[BHP06] T. Bures, P. Hnetynka, and F. Plasil, "SOFA 2.0: Balancing advanced features in a hierarchical component model," in *SERA '06: Proceedings of the ACIS International Conference on Software Engineering Research, Management and Applications, Seattle, USA*, 2006, pp. 40–48.

[BHP$^+$07] T. Bures, P. Hnetynka, F. Plasil, J. Klesnil, O. Kmoch, T. Kohan, and P. Kotrc, "Runtime support for advanced component concepts," in *SERA '07: Proceedings of the ACIS International Conference on Software Engineering Research, Management and Applications, Busan, Korea*, 2007, pp. 337–345.

[BPR01] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE: a FIPA2000 compliant agent development environment," in *Agents*, 2001, pp. 216–217.

[Bre01] E. A. Brewer, "Lessons from giant-scale services," *IEEE Internet Computing*, vol. 5, no. 4, pp. 46–55, 2001.

[CD01] J. Cheesman and J. Daniels, *UML Components.* Addison-Wesley Professional, Boston, 2001.

[CHT03] K. Channabasavaiah, K. Holley, and E. Tuggle, "Migrating to a Service-Oriented Architecture, part 1." 2003, available online. http://www-128.ibm.com/developerworks/library/ws-migratesoa/. Last visited on 2009-11-15.

[CLS$^+$05] A. Chand, D. Leeming, E. Stork, A. Agassi, and R. Biliki, “The impact of ICT on rural development in solomon islands: the PFnet case,” University of the South Pacific, Suva, Fiji, Tech. Rep., 2005.

[Coc05] A. Cockburn, *Crystal Clear: A human powered methodology for small teams*, O’Hagan, Ed. Pearson Education Inc., Addison-Wesley, Upper Saddle River, 2005.

[Coh03] B. Cohen, “Incentives build robustness in bittorrent.” in *In Proc. First Workshop on Economics of Peer-to-Peer Systems, Berkeley, USA.*, June 2003.

[CP02] A. Chaffee and W. Pietri, “Unit testing with mock objects,” 2002, available online. http://www.ibm.com/developerworks/library/j-mocktest.html. Last visited 2010-01-12.

[Cun09] W. Cunningham, “Programming principles (dry, yagni),” 2009, available online. http://c2.com/. Last visited: 2010-01-05.

[DB09] M. B. Dias and E. Brewer, “How computer science serves the developing world,” *Communications of the ACM*, vol. 52, 6, pp. 74 – 80, 2009.

[DEL97] T. S. Dare, E. B. Ek, and G. L. Luckenbaugh, “Method and system for authenticating users to multiple computer servers via a single sign- on,” US Patent 5,684,950, US Patent Office, Washington DC, 1997.

[DMTT08] S. Dyakalashe, H. N. Muyingi, A. Terzoli, and M. Thinyane, “Cultural and linguistic localization of the shop-owner interfaces to an e-commerce platform for rural development,” in *SATNAC 2008: Proceedings of the Southern African Telecommunication Networks and Applications Conference, Wild Coast, South Africa*, 2008.

[DO04] A. Dymond and S. Oestermann, “A rural ICT toolkit for Africa,” Washington D.C., 2004. [Online]. Available: http://www.infodev.org/en/Publication.23.html

[DTMT07] L. Dalvit, A. Terzoli, H. Muyingi, and M. Thinyane, “The deployment of an e-commerce platform and related projects in a rural area in South Africa,” *International Journal of Computing and ICT Research*, vol. 1, pp. 9–17, 2007.

[eKh09] eKhaya ICT, “SELF solar computer lab project. http://ekhayaict.com/ekhaya ict zwelenqaba.php (last visited 2009-04-24),” 2009. [Online]. Available: http://ekhayaict.com/eKhayaICTZwelenqaba.php

[FI03] I. T. Foster and A. Iamnitchi, “On death, taxes, and the convergence of peer-to-peer and grid computing,” *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers*, vol. 2735, pp. 118–128, 2003.

[FL99] D. A. Farber and R. D. Lachman, "Data processing system using substantially unique identifiers to identify data items, whereby identical data items have the same identifiers," United States of America Patent 5,978,791, 1999.

[Fos06] I. Foster, "Globus Toolkit version 4: Software for service-oriented systems," in *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS*, vol. 3779, 2006, pp. 2–13.

[Fou02] Foundation for Intelligent Physical Agents (FIPA), "FIPA abstract architecture specification," FIPA - SC00001L, Tech. Rep., 2002.

[GGG$^+$03] P. K. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The impact of DHT routing geometry on resilience and proximity," in *SIGCOMM*, 2003, pp. 381–394.

[Gus08] K. Gush, "Towards a more personalised user experience and better demographic data on the Digital Doorway public computer terminals." in *5th Prato Community Informatics & Development Informatics Conference 2008: ICTs for Social Inclusion: What is the Reality? Conference CD, Prato, Italy, Centre for Community Networking Research, Monash University*, 2008.

[GvLPF01] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster, "Multiparadigm communications in Java for grid computing," *Commun. ACM*, vol. 44, no. 10, pp. 118–125, 2001.

[Har02] A. Harwood, "High performance interconnection networks," Ph.D. dissertation, School of Computers and Information Technology, Griffith University, Queensland, Australia, 2002.

[HCH$^+$05] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi, "The architecture of PIER: an Internet-scale query processor," in *CIDR*, 2005, pp. 28–43.

[Hee08] R. Heeks, "ICT4D 2.0: The next phase of applying ICT for international development," *Computer*, vol. 41, no. 6, pp. 26–33, 2008.

[Hen06] M. Henning, "The rise and fall of CORBA," *Queue*, vol. 4, no. 5, pp. 28–34, 2006.

[IBM09] IBM Corporation, "Java Management Extensions (JMX)," 2009, available online. http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp. Last visited 2009-08-28.

[IDR06] IDRC, "Acacia prospectus 2006-2011," Program and Partnership Branch, International Development Research Centre (Canada), Tech. Rep., 2006.

[Inv08] Inveneo, "Inveneo." 2008, available online. http://www.inveneo.org/. Last visited 2008-08-12.

[Jak09] B. T. Jakachira, "Implementing an integrated e-government functionality for a marginalized community in the Eastern Cape, South Africa. (MSc Thesis)," Master's thesis, University of Fort Hare, 2009.

[Jav06] Java Community Process (JCP) - JSR3, JSR160, "Java Management Extensions (JMX)," 2006, available online. http://java.sun.com/javase/6/docs/technotes/guides/jmx/. [Last visited: 2009-12-01].

[JH06] S. Jacobs and M. Herselman, "Information access for development: A case study at a rural community centre in South Africa," *Issues in Informing Science and Information Technology*, vol. 3, pp. 295–306, 2006.

[JMW08] B. Jakachira, H. Muyingi, and R. Wertlen, "Implementation of a web-based e-government proxy for a marginalized rural population," in *Southern Africa Telecommunication Networks and Applications Conference, Wild Coast Sun, South Africa.* Telkom, Sept 2008.

[JNR09] C. Janiesch, M. Niemann, and N. Repp, "Towards a service governance framework for the Internet of services," in *Proceedings of the 17th European Conference on Information Systems (ECIS 2009), Verona (Italy)*, September 2009. [Online]. Available: http://tubiblio.ulb.tu-darmstadt.de/35831/

[Joh07] D. Johnson, "Evaluation of a single radio rural mesh network in South Africa," in *Proc. 2nd IEEE/ACM ICTD*, 2007.

[JSN$^+$09] J. Jiang, L. Shan, H. Ntareme, A. Doria, M. Zennaro, and B. Pehrson, "Bytewalla: Delay tolerant networks on android phones: Business idea," 2009, available Online. http://www.tslab.ssvl.kth.se/csd/projects/092106/sites/default/files/Business

[JTT09] N. R. Jere, M. Thinyane, and A. Terzoli, "Development of a reward based program for an e-commerce platform for a marginalized area," in *International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government (EEE'09), July 13-16, 2009, Las Vegas, USA.*, 2009.

[KDQ09] M. Khalil, P. Dongier, and C. Z.-W. Qiang, *2009 Information and Communications for Development: Extending Reach and Increasing Impact.* World Bank, 2009, ch. Overview, pp. 3 – 17.

[Ken08] C. Kenny, "ICT: Promises, opportunities and dangers for the rural future," in *Proceedings of the Rural Futures Conference*, M. Warren, Ed. University of Plymouth, April 2008.

[Kro02] A. Kronfol, "FASD: A fault-tolerant, adaptive, scalable, distributed search engine," Ph.D. dissertation, Princeton University, May 2002.

[Las96] C. Lassenius, "CORBA basics," 1996, available online. http://www.soberit.hut.fi/cls/corba

[Mal09] P. Malinverno, "SOA: Where do i start?" Gartner Research, Tech. Rep. G00165547, March 2009.

[MC08] L. Mavuso and P. Cheriyan, "System prospectus: Poverty eradication programme," 2008, hardcopy may be available from the Eastern Cape Government, under Document No. SDEC-00007. Classified.

[McC98] M. K. McCaskill, *Grammar, Punctuation, and Capitalization: A Handbook for Technical Writers and Editors*, nasa sp-7084 ed., NASA, Hampton, Virginia, 1998.

[MD05] M. J. Menou and P. Day, "Developing a sense-making framework for collective learning in latin american community telecenter assessment," in *Paper presented at a non-divisional workshop held at the meeting of the International Communication Association, New York*, 2005.

[Med06] M. Meddie, "Rethinking telecentre sustainability: How to implement a social enterprise approach - lessons from India and Africa," *The Journal of Community Informatics, Special Issue: Telecentres*, vol. 2:3, no. 3, pp. 265–279. [Online] Available: http://ci–journal.net/index.php/ciej/article/view/324/265, 2006.

[MHF$^+$05] S. M. Mishra, J. Hwang, D. Filippini, R. Moazzami, L. Subramanian, and T. Du, "Economic analysis of networking technologies for rural developing regions," in *WINE*, 2005, pp. 184–194.

[Mid09] Middleware, "ACM/IFIP/USENIX 10th International Middleware Conference, Urbana Champaign, Illinois, USA," 2009, available online. http://middleware2009.cs.uiuc.edu (visited on 2009-11-25).

[MSZ01] S. McIlraith, T. Son, and H. Zeng, "Semantic web services," *IEEE Intelligent Systems. Special Issue on the Semantic Web*, vol. 16, no. 2, pp. 46–53, March/April 2001.

[Muc08] T. Muchenje, "Investigating security issues in a converged IEEE 802.x wireless network. (MSc Thesis)," Master's thesis, University of Fort Hare, 2008.

[New03] M. E. J. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, pp. 167–256, 2003.

[Nje07] S. G. Njeje, "Implementation of a robust, cost effective, e-commerce platform for disadvantaged communities of the Eastern Cape, South Africa. (MSc Thesis)," Master's thesis, University of Fort Hare, 2007.

[NTM06] S. Njeje, A. Terzoli, and H. N. Muyingi, "Software engineering of a robust, cost-effective ecommerce platform for disadvantaged communities," in *Southern Africa Telecommunication Networks and Applications Conference, Cape Town, South Africa.*, 2006.

[Nwa96] H. S. Nwana, "Software agents: An overview," *Knowledge Engineering Review*, vol. 11, pp. 205–244, 1996.

[OLP09] OLPC, "The one laptop per child (OLPC) project," 2009, available online. http://www.laptop.org/. Last visited 2009-04-05.

[Ora01] A. Oram, Ed., *Peer-to-Peer - Harnessing the Power of Disruptive Technologies,.* O'Reilly, Beijing, 2001.

[OSG07] OSGi Alliance, "About the OSGi service platform, revision 4.1," 2007, available online. http://www.osgi.org/Links/Documents. Last visited 2009-12-03.

[oTU07] U. C. on Trade and D. UNCTAD, "ITU/UNCTAD 2007 World Information Society report: Beyond WSIS," Geneva, 2007.

[OW07] A. Oram and G. Wilson, Eds., *Beautiful Code: Leading Programmers Explain How They Think.* O'Reilly, Sebastopol, 2007.

[PA00] S. Paracer and V. Ahmadjian, *Symbiosis: An Introduction to Biological Associations. 2nd ed.* Oxford University Press, 2000.

[Par72] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[PD07] M. R. Patra and R. K. Das, "SORIG: a service-oriented framework for rural information grid – an implementation viewpoint," in *ICEGOV '07: Proceedings of the 1st international conference on Theory and practice of electronic governance.* New York, NY, USA: ACM, 2007, pp. 49–52.

[PNS+07] R. K. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. A. Brewer, "WiLDNet: Design and implementation of high performance WiFi based long distance networks," in *NSDI*, 2007.

[PP05] D. Pilone and N. Pitman, *UML 2.0 in a Nutshell*, J. Gennick, Ed. O'Reilly Media Inc., 2005.

[PPKG] C. Pade, R. Palmer, M. Kavhai, and S. Gumbo, "Siyakhula Living Lab: Baseline study report," available online. http://www.dst.gov.za/links/cofisa/document/. Last visited: 2010-04-10].

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A scalable content-addressable network," in *SIGCOMM 2001: Proceedings of the conference of the Special Interest Group on Data Communication, San Diego, USA*, 2001, pp. 161–172.

[Sey08] A. Sey, "Public access to ICTs: A review of the literature," 2008, available online: http://www.globalimpactstudy.org/wp-content/uploads/2009/02/ipai-lit-review-10-08.pdf [Last visited: 2009-11-23].

[SGM02] C. Szyperski, D. Gruntz, and S. Murer, *Component Software: Beyond Object-Oriented Programming.* Addison-Wesley Professional, Boston, 2002, iSBN 0-201-74572-0.

[SM08] M. S. Scott and H. N. Muyingi, “Investigation and development of an e-judiciary service for a citizen oriented judiciary system in rural community,” in *SATNAC 2008: Proceedings of the Southern African Telecommunication Networks and Applications Conference, Wild Coast, South Africa*, 2008.

[SRS05] D. Stutzbach, R. Rejaie, and S. Sen, “Characterizing unstructured overlay topologies in modern P2P file-sharing systems,” in *Internet Measurment Conference*, 2005, pp. 49–62.

[SSDN02] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, “HyperCuP- hypercubes, ontologies and efficient search on P2P networks,” Bologna, Italy, 2002.

[Sta06] M. Stal, “Using architectural patterns and blueprints for Service-Oriented Architecture,” *IEEE Software*, vol. 23, no. 2, pp. 54–61, 2006.

[STB09] Z. Shibeshi, A. Terzoli, and K. Bradshaw, “Developing toolkit for video-oriented services in the NGN environment,” in *SATNAC: Proceedings of the Southern African Telecommunication Networks and Applications Conference, Royal Swazi Spa, Swaziland*, 2009.

[Str10] Strix, “Kapenta framework (unpublished technical documentation),” 2010, available online. http://kapenta.org.uk/. Last visited 2010-01-12.

[Sun09] Sun Microsystems, “Java Enterprise Edition (Java EE),” 2009, available online. http://java.sun.com/javaee/. Last visited 2009-12-03.

[SWL06] H. Slay, P. Wentworth, and J. Locke, “BingBee, an information kiosk for social enablement in marginalized communities,” in *SAICSIT ’06: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, Gordon’s Bay, South Africa.* South African Institute for Computer Scientists and Information Technologists, 2006, pp. 107–116.

[TAA$^+$02] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, C.-J. Hugly, E. Pouyoul, and B. Yeager, “Project JXTA 2.0 super-peer virtual network.” Sun Microsystems. Available online. http://research.sun.com/spotlight/misc/jxta.pdf. Last visited 2009-04-29., Tech. Rep., 2002. [Online]. Available: http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf

[Tar07] P. Tarwireyi, “Implementation of a network cost management framework for marginalised communities. (MSc Thesis),” Master’s thesis, University of Fort Hare, 2007.

[TAW03] L. Titchkosky, M. Arlitt, and C. Williamson, “A performance comparison of dynamic web technologies,” *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 3, pp. 2–11, 2003.

[Tel09a] Telecentre.org, "http://www.telecentre.org/notes (last visited 2009-04-23)," 2009. [Online]. Available: http://www.telecentre.org/notes

[Tel09b] Telecom Italia Lab, "Java Agent Development Framework," 2009, available Online. http://jade.tilab.com/. Last visited: 2009-12-08.

[TSTC06] M. Thinyane, H. Slay, A. Terzoli, and P. Clayton, "A preliminary investigation into the implementation of ICT in marginalized communities," in *SATNAC 2009: Proceedings of the Southern African Telecommunication Networks and Applications Conference, Stellenbosch, South Africa*, 2006.

[TT09] M. Thinyane and A. Terzoli, "Universal digital inclusion: Beyond connectivity, affordability and capability," in *ICTD2009 conference, Doha*, 2009.

[TTC09] M. Thinyane, A. Terzoli, and P. Clayton, "EServices provisioning in a community development context through a JADE MAS platform," in *IEEE/ACM International Conference on Information and Communication Technologies and Development, Doha, Qatar*, 2009.

[TTT07] P. Tarwireyi, A. Terzoli, and M. Thinyane, "Implementation of an Internet access cost management system for disadvantaged communities," in *SATNAC 2007: Proceedings of the Southern African Telecommunication Networks and Applications Conference, Sugar Beach Resort, Mauritius*, 2007.

[TTW08] M. Tsietsi, A. Terzoli, and G. Wells, "An open service delivery platform for adding value to softswitch based telephony environments," in *SATNAC 2008: Proceedings of the Southern African Telecommunication Networks and Applications Conference, Wild Coast, South Africa*, 2008.

[TTW09] ——, "Mobicents as a service deployment environment for OpenIMSCore," in *SATNAC 2009: Proceedings of the Southern African Telecommunication Networks and Applications Conference, Royal Swazi Spa, Swaziland*, 2009.

[UN 03] UN ICT Task Force (UNICTF), "Connected for development: Information kiosks and sustainability," New York, 2003. [Online]. Available: http://www.unicttaskforce.org

[VDB04] K. Vanthournout, G. Deconinck, and R. Belmans, "A taxonomy for resource discovery," in *ARCS 2004: Proceedings of the International Conference on Architecture of Computing Systems: Organic and Pervasive Computing, Augsburg, Germany*, 2004, pp. 78–91.

[Vil10] Village Scribe Association, "awarenet," 2010, available online. http://code.google.com/p/awarenet/. Last visited 2010-01-12.

[WCB01] M. Welsh, D. Culler, and E. Brewer, "SEDA: an architecture for well-conditioned, scalable Internet services," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 230–243, 2001.

[Wer03] R. R. R. Wertlen, "DFN Science-to-Science: Peer-to-peer scientific research," in *TNC 2003: Proceedings of the Terena Networking Conference, Zagreb, Croatia*, 2003.

[Wer06] ——, "Re: Altnet goes after p2p networks with obvious patent (mailing list thread)." 2006, available online. http://www.mail-archive.com/p2p-hackers@zgp.org/msg00260.html. Last accessed 2009-12-31.

[Wer08] ——, "An overview of ICT innovation for developmental projects in marginalised rural areas," in *5th Prato Community Informatics & Development Informatics Conference 2008: ICTs for Social Inclusion: What is the Reality? Conference CD, Prato, Italy, Centre for Community Networking Research, Monash University.*, 2008.

[Wer09] ——, "Description of a solar computer lab network in a marginalised rural area of the Eastern Cape," 2009, author's work in progress for the Centre of Excellence.

[Wik09a] Wikipedia, "Middleware." 2009, available online. http://en.wikipedia.org/wiki/Middleware. Last visited 2009-07-29.

[Wik09b] ——, "Software agents," 2009, available online. http://en.wikipedia.org/wiki/Software-agent. Last visited 2009-12-07.

[Wor10] World Wide Web Consortium (W3C), "The xml 1.0 standard." 2010, available online. http://www.w3c.org/. Last visited 2010-01-04.

[WT09] R. R. R. Wertlen and A. Terzoli, "Peer-to-peer web services for distributed rural ICTs," in *SATNAC 2009: Proceedings of the Southern African Telecommunication Networks and Applications Conference, Royal Swazi Spa, Swaziland*, 2009.

Appendices

Implemented package hierarchy

```
src/za/ac/ufh/domain                   test/za/ac/ufh/domain
src/za/ac/ufh/middleware               test/za/ac/ufh/repository
src/za/ac/ufh/middleware/jxta          test/za/ac/ufh/middleware
src/za/ac/ufh/middleware/ws            test/za/ac/ufh/middleware/jxta
src/za/ac/ufh/repository               test/za/ac/ufh/middleware/ws

src/za/ac/ufh/web                      test/za/ac/ufh/web
```

Figure 7.1: Spring demonstrator package structures, left Java source, right tests.

HTTP conversation

This appendix is included to demystify the way the web works. The web is driven by human readable string processing – a powerful argument to follow in its footsteps.

// Open a TCP connection to the web server which is running on port 8080 here.

```
rrrw@Iridium-2:~>telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

// Our request to the web server:

```
GET /p2pmwt/siyakhula.htm HTTP/1.1
Host: localhost:8080
```

// The web server responds. It sends a header and then a message body. The message body is encoded in HTML.

// The header....

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Content-Language: en-US
Content-Length: 626
Date: Sat, 09 Jan 2010 13:29:48 GMT
```

// The body ...

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Siyakhula P2P Middleware </title>
```

XSD

This appendix is the XML Schema Definition defining the *http://ufh.ac.za/schemas/mw* namespace, which is the XML namespace of all the public middleware messages that can be used to communicate with the middleware. Here we define in an exemplary fashion, the Request and Response required for our demonstrator, namely, the *RegisterServiceRequest* and *RegisterServiceResponse*.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://ufh.ac.za/schemas/mw" elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://ufh.ac.za/schemas/mw">
<xsd:element name="Request">
<xsd:complexType>
<xsd:choice minOccurs="0" maxOccurs="1">
<xsd:element ref="RegisterServiceRequest"/>
<xsd:element ref="ServiceRequest"/>
<xsd:element ref="ApplicationRequest"/>
</xsd:choice>
</xsd:complexType>
</xsd:element>
<xsd:element name="RegisterServiceRequest">
<xsd:complexType mixed="true">
<xsd:all minOccurs="0" maxOccurs="1">
<xsd:element ref="Description"/>
<xsd:element ref="Identifier"/>
<xsd:element ref="WsdlUrl"/>
</xsd:all>
</xsd:complexType>
</xsd:element>
<xsd:element name="RegisterServiceResponse" type="xsd:string"/>
<xsd:element name="Description" type="xsd:string"/>
<xsd:element name="Identifier" type="xsd:string"/>
<xsd:element name="WsdlUrl" type="xsd:anyURI"/>
<xsd:element name="ServiceRequest">
<xsd:complexType mixed="true">
<xsd:sequence>
<xsd:element ref="Identifier"/>
</xsd:sequence>
</xsd:complexType>
```

```
</xsd:element>
<xsd:element name="ApplicationRequest">
<xsd:complexType mixed="true">
<xsd:sequence>
<xsd:element ref="Identifier"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:schema>
```