



Development of software models to integrate distributed RFID technologies into management information systems.

By

Fidel Nunez

September 2010

A Thesis Submitted for the Degree of M.Eng

Galway Mayo Institute of Technology

Supervisor: Dr. John Owen-Jones

Submitted to the Higher Education And Training Awards Council. September 2010

Development of software models to integrate distributed RFID technologies into management information systems.

Fidel Nunez

Abstract

The growth of Radio Frequency Identification (RFID) technologies within inventory and asset tracking processes during recent years, has caused a corresponding increase in the amounts of tracking data being generated at the edge of the network. This data requires filtering, processing and refining in order to produce information that can be used to provide more efficient business processes and aid decision making.

New legislation such as the Sarbanes-Oxley Act has enforced the asset tracking process in organisations, with RFID enhanced asset tracking technology being proposed as an answer. This however, requires the creation of an economical software system to consume and filter the data being generated by the hardware readers, and provide integration via a middleware solution with existing Enterprise Resource Planning (ERP) systems.

This thesis presents a method of collecting and processing data generated by RFID hardware. The manner in which the open source based web application¹ that was designed for this project interacts with ERP systems via Microsoft BizTalk messaging system is also described. Finally this thesis looks at how users of the Juno system are presented and can interact with this information, to help in the asset tracking process.

Security considerations are important due to the sensitive company financial information in use and this is considered later on.

¹ Referred to as “Juno” within this project.

Declarations

Copyright and DAI Statement

I hereby grant the Galway Mayo Institute of Technology or its agents the right to archive and to make available my thesis or dissertation in whole or part in the Institute libraries in all forms of media, now or here after known, subject to the provisions of the Copyright & Related Rights Act, 2000. I retain all proprietary rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

I also authorise Institute Microfilms to use the 350 word abstract of my thesis in Dissertation Abstracts International.

I have either used no substantial portions of copyright material in my thesis or I have obtained permission to use copyright material; where permission has not been granted I have applied/will apply for a partial restriction of the digital copy of my thesis or dissertation.

Signed: _____

Date: _____

Authenticity Statement

I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.

Signed: _____

Date: _____

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at GMIT or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at GMIT or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Signed: _____

Date: _____

Acknowledgements

With the completion of this project and thesis an important chapter of my life comes to an end. I wish to say thanks to the following people for their help and support throughout this project:

- My supervisor, Dr. John Owen-Jones for advice and guidance on the project and this thesis.
- Enterprise Ireland and their financial support, without which this research and project would not have been possible.
- GMIT for providing the facilities to accomplish this research.
- James O'Shaughnessy, a fellow researcher with whom it was a pleasure to work with on this project, and for sharing his broad ranging knowledge on RFID and networking technologies.
- Mr. Andrew Shields for his expertise and work on the BizTalk middleware integration.

I am delighted to have been presented this wonderful opportunity to learn a wide range of Software Engineering skills and work with emerging technologies.

Table of Contents

| | |
|---|------|
| Abstract | i |
| Declarations | ii |
| Acknowledgements..... | iii |
| Table of Contents | iv |
| List of Figures and Tables | viii |
| Chapter 1 – Introduction..... | 1 |
| Motivation for developing Juno fixed asset tracking system..... | 1 |
| Aims and objectives of the research work | 3 |
| Research Methodology | 6 |
| Thesis Structure | 6 |
| Chapter 2 – Literature Review | 8 |
| RFID..... | 8 |
| Overview | 8 |
| Readers..... | 9 |
| Tags | 10 |
| Middleware | 14 |
| Review of existing RFID middleware systems | 17 |
| Microsoft BizTalk..... | 17 |
| IBM WebSphere RFID..... | 18 |
| Review of existing RFID asset tracking systems | 20 |
| “Global SAW Tag” RFID system | 20 |
| “OATSystem” and “Reva” RFID Asset tracking systems..... | 22 |
| RFID Asset Tracking Alternatives..... | 23 |

| | |
|---|----|
| Bokode's..... | 23 |
| RuBee | 23 |
| Software architectures and Design Patterns..... | 25 |
| Multi-Tier Architecture | 26 |
| Model View Controller | 28 |
| Front Controller..... | 31 |
| Registry Pattern..... | 32 |
| Chapter 3 – System Requirements and Technologies | 34 |
| Requirements | 34 |
| System Usage Scenario | 37 |
| Technologies..... | 39 |
| Linux OpenSUSE | 39 |
| VMware..... | 41 |
| MySQL | 43 |
| Nginx..... | 44 |
| PHP | 45 |
| JavaScript..... | 48 |
| OpenBravo | 50 |
| Chapter 4 – System Design, Development and Implementation | 54 |
| Unified Modelling Language..... | 54 |
| General System Architecture | 55 |
| RFID Driver and Abstraction Layer Design | 57 |
| Juno Design..... | 60 |
| Initial Prototype Design..... | 60 |
| Database Design..... | 61 |

| | |
|--|-----|
| Model View Controller Framework..... | 61 |
| Authentication, Session control and Secure HTTP..... | 65 |
| Required Software and configuration..... | 66 |
| User Interface Design | 67 |
| ERP Integration via BizTalk Server | 69 |
| Development Techniques and Tools | 72 |
| Teamwork Breakdown | 73 |
| Chapter 5 –Testing and Security | 75 |
| Testing and Validation | 75 |
| Unit Testing | 75 |
| Integration and Usability Testing..... | 76 |
| Client-Side Testing and Debugging | 77 |
| Security..... | 78 |
| XSS..... | 79 |
| CSRF..... | 80 |
| SQL Injection | 81 |
| Chapter 6 – Conclusions and Future Work | 82 |
| Conclusions..... | 82 |
| Future Work | 86 |
| Bibliography | 90 |
| Glossary | 100 |
| Appendix | 102 |
| 1. Juno Entity Relation Diagram and schema | 102 |
| 2. Juno UML Class Diagram for the main Model Classes | 103 |
| 3. UML Deployment Diagram..... | 105 |

| | | |
|----|--|-----|
| 4. | Juno source code outline and structure | 106 |
| 5. | RFID tag data filtering and processing | 107 |
| 6. | BizTalk Integration, Development Iterations..... | 111 |
| | Development Iteration One – Flat File..... | 112 |
| | Iteration two – Message Queues MSMQ..... | 113 |
| | Iteration Three – Database | 117 |
| | Iteration Four – Web Services..... | 117 |
| 7. | BizTalk Orchestration Schedule | 119 |
| 8. | ERP interaction events and processing..... | 120 |
| 7. | Legacy A-Track application, user interface | 121 |
| 8. | Juno screenshots..... | 122 |

List of Figures and Tables

| | |
|--|----|
| Figure 1 - RFID system overview | 9 |
| Figure 2 - WaveTrend RX900 Active RFID Reader | 10 |
| Figure 3 - Mixture of active and passive tags used in the project..... | 12 |
| Figure 4 - Printable Battery (source: Fraunhofer ENAS)..... | 13 |
| Figure 5 - BizTalk RFID stack (source: Pro RFID in BizTalk Server 2006 r3)..... | 17 |
| Figure 6 - IBM WebSphere Solution map..... | 19 |
| Figure 7 - SAW RFID tags and tagged flight-bag (source: NASA)..... | 21 |
| Figure 8 - NASA version of the SAW 704 reader with PDA (source: NASA)..... | 22 |
| Figure 9 - RuBee tags (source: Visible Assets)..... | 24 |
| Figure 10 - Sample 3 Layer Application | 27 |
| Figure 11 – Classic Model View Controller relations..... | 29 |
| Figure 12 - Front Controller Sequence Diagram..... | 32 |
| Figure 13 - Simple Registry class | 33 |
| Figure 14 - Virtualized System overview, hosted architecture | 42 |
| Figure 15 - OpenBravo Architecture (source: openbravo.com)..... | 53 |
| Figure 16 - Main Hardware and Networking components..... | 55 |
| Figure 17 - Multitier RFID Architecture | 56 |
| Figure 18 - Juno web application, system overview and dataflow | 64 |
| Figure 19 - Juno login screen (non HTTPS) | 68 |
| Figure 20 - Typical BizTalk middleware system overview | 70 |
| Figure 21 - Mapper component configuration for OpenBravo assets within BizTalk Server 2006 | 71 |
| Figure 22 - Firebug Debugger, network connections overview of Juno main dashboard..... | 78 |

Chapter 1 – Introduction

Motivation for developing Juno fixed asset tracking system

The main motivation for migration to an RFID aided asset tracking system is to reduce costs for the user. RFID offers the promise of cost reductions and increased precision in fixed asset tracking, therefore producing real business benefits and gains by cutting out the labour-intensive work (1). Asset management enhanced by RFID can potentially present lower operational costs and increased efficiencies, an example of which could be the ability to find and redeploy equipment in order to meet the needs of a rapidly evolving business operation (2); this would not be possible without a detailed inventory of the condition and availability of assets.

In recent years there has been an explosion of interest in the use of RFID technologies, mainly due to falling tag and reader costs and the wide availability of fixed and wireless communications networks. A middleware solution is required in order to process the large amounts of data being generated at the edge of the network by RFID systems into usable and refined business information. RFID is not a line of sight technology, it allows data to be read from “tags” without contact (3) unlike barcodes, this makes it attractive to a wide range of industries such as: healthcare (4), pharmaceutical (5), retail (6) and warehousing. The wireless nature of the communication between reader and tags raises issues when it comes to data validity; for example metal objects interfere and reflect the signals while liquid containers absorb it.

Older fixed asset tracking systems made use of barcodes in order to keep track of assets for the purposes of financial accounting, maintenance and theft deterrence. Recent adoption of RFID systems as a replacement for barcode systems has been motivated by falling prices for RFID components and the advantage of not requiring line of sight identification during audits. IBM is one example of a company where this shift to RFID asset management occurred (7); here a legacy system which involved

the use of barcodes, paperwork and line of sight manual verification, has been replaced by an RFID aided process in order to save time and reduce costs.

New legislation such as the Sarbanes-Oxley Act (8) has enforced the asset tracking process in organisations, with RFID aided asset tracking being proposed as a possible solution (9). This tightening of legal requirements was enacted in response to corporate accounting scandals caused by companies such as Enron (10), and imposes new stricter requirements on auditing and accounting processes with regards to fixed assets.

RFID aided asset tracking requires the integration of physical hardware such as RFID readers and tags with enterprise software via the use of a networked infrastructure and a middleware solutions. The data from these sensors is generated on the edge of the networks and needs to be collected, filtered and processed before useful information is extracted and integrated via middleware solutions with existing management information systems. Forrester (11) defines RFID middleware as *“Platforms for managing RFID data and routing it between tag readers or other auto-identification devices and enterprise systems”*. There is a range of existing commercial middleware solutions in use, and these will be discussed in detail in next chapter, along with the design and configuration of the Juno middleware.

The rise of open source software as a viable and economical alternative to costly proprietary systems has opened up the possibility of creating cost effective software solutions. Hence the use of this kind of software can help bridge the physical interface of RFID readers with the existing financial and enterprise software interfaces, in order to provide a fixed asset tracking system that is; secure, low cost, fit for purpose and works well with the multitude of other third party enterprise systems. Free and open source software has played a large role in the design and creation of the Juno middleware system. Open source software gives the user the freedom to examine, change, use, copy, distribute, and improve the software. Unfortunately, there are various degrees of “freedom” afforded by a wide range of licenses, with the more commonly used ones being, the GNU Public License (GPL) (12) and the BSD License

(13), one needs to be aware of the differences, nuances and cons of these licenses. Open source software can usually be downloaded from a generic open source project hosting sites, such as Google Code (14) and SourceForge (15) or from a project specific homepage such as jQuery (16). Use of open source software is not without caveat, and issues exist when it comes to; overall quality, code revision control, availability of clear documentation and reliability. These concerns arise as open source projects, which have usually evolved to suit the interests of the participants who might have followed a different path from other interested parties.

Yet another motivation for the development of the Juno asset tracking system, came from the industrial partner who had in the past unsuccessfully tried to design and develop a similar asset tracking system, and who were hoping to use this research project as a stepping stone towards a future full product development and commercialisation project venture with Enterprise Ireland and GMIT. As outlined later on in this thesis in the requirements section, the past experience and lessons learned from this proprietary asset tracking application, which unfortunately failed to meet its objectives has helped in ensure that the same mistakes are avoided in this project.

Aims and objectives of the research work

The overall aim of the project is to determine if the fixed asset tracking process may be made more efficient using RFID. This would be developed by designing and implementing an integrated system, by following these main objectives:

- Provide RFID hardware abstraction, data gathering and filtering, and keeping the asset register up to date with reconciling the tag information.
- Offer data storage, asset related information input and retrieval for system users, and help the user in the decision making processes.
- Supply interoperability with various third party management and information systems via the use of a third party messaging middleware system.

In order to meet the above objectives, an effective fixed asset tracking solution needs to be researched, designed and developed. The main features of such a solution would include usability, security, extensibility and integration with enterprise financial software.

Usability and a clean user interface (UI) are essential features that directly support the users in their interaction with the system. A good user interface should be intuitive, consistent, and also clean of clutter and distractions. In the context of this project, all of these features should help the user to interact and participate in the asset tracking process with a little or no training. Although, the interface is a small layer, similar to the icing on a cake, the importance of getting this component of the system right cannot be underestimated, as this layer is directly visible to end users. The key goals of user interface design are: productive use of the system via an increase in learning speed and decrease in interaction error rate (17), and allowing the user to act and make decisions on the presented information (18).

Since the system would make use of a wide variety of different technologies and would be primarily client / server based, security considerations are paramount. Sensitive financial company data would flow through the system, therefore, the risk of a data leak or unauthorised access by parties such as a competitor, could put the whole business at risk. Hence, tight access control, security precautions and other security aspects are explored in this research and a section in Chapter 5 is dedicated to security aspects of the project.

Extensibility is a much coveted aspect of any software system and is often hard to achieve. The development of the Juno system using modularised techniques is explored in detail in the later chapters, the software framework designed for this system helps to easily add modules (also known as controllers within the Model View Controller architecture) to the system and modify and extend existing ones. A well documented internal Class structure and APIs allows us to have loosely coupled modules, meaning that these modules can interact with the data models and inputs and outputs of the system, through stable interfaces that also follow the “Information

Hiding Principle” (19) where one module doesn’t concern itself with the internal implementation of another one, and a change in one area doesn’t not unexpectedly cascade through the system, leaving us with a solid, reliable and maintainable software system. These metrics are of high importance and as illustrated by Frederick Brooks in his classic software engineering piece (20) get harder to achieve as the complexity of software system rises nonlinearly with size.

Yet another objective is to look into integration options for the multitude of hardware and software components to be used in this project, and to explore the pros and cons of these technologies. This includes a review of software systems ranging from relational databases and web servers to underlying the operating system and high level programming languages.

Since the project has wide scope and is team based, the principle objective for the author is the development of the Juno RFID asset tracking web application, and integration with the RFID Reader Driver Abstraction layer on one side and ERP system via BizTalk middleware on the other side.

Several other main objectives and tasks for the author include:

- Conducting a literary review of and researching the structure and architecture of existing RFID middleware systems.
- Analysis of existing asset tracking systems and the design and development of an improved system.
- Working closely with the other researcher on the project and outside consultant (Teamwork Breakdown section in Chapter 4 goes into further detail) in order to integrate hardware systems and communicate via third party middleware.
- Identifying and using suitable software systems and finding the best fit with the overall project objectives and system requirements.
- Working with the industrial partner in order to gather requirements and incorporate any changes resulting from feedback received.

Emphasis was made by the industrial partner with regards to; usability, security, extensibility and integration.

Research Methodology

The approach taken in this research project is as follows:

- Literary review.
- Research into web application and network security.
- Requirements analysis and system design.
- Infrastructure and technology review.
- Development and testing of the system.

Thesis Structure

Chapter 1 presents the motivation, aims and objectives for the project and this thesis along with methodology and thesis structure.

Chapter 2 contains a literary review of RFID technologies and its history, the work being done in the field and current state of technology, and finally a look at related emerging technologies.

Chapter 3 lists the system requirements gathered from the industrial partner. Also included is a detailed look at the pros and cons of the wide range of technologies used in this project, along with the reasoning behind the selection of underlying software.

Chapter 4 examines the analysis and design steps taken in order to find the correct development approach and to meet the project requirements and objectives. Also

described are the implementation of the solution and the detailed structure of each of its three sub-systems.

Due to the risks of possibly sensitive company information being accessed by unauthorised users, half of Chapter 5 is devoted to the important security issues in relation to the developed system. Current security threats and issues are discussed and presented to the reader. This chapter also outlines the testing methodology used throughout the development of the system.

The conclusions are presented in Chapter 6 and the success of applying the developed Juno solution to an RFID based asset tracking process is determined. Future direction for the system and recommendations for further work are also discussed here.

Chapter 2 – Literature Review

In order to design an effective system, it is important to examine the technologies used, and also gain an understanding of the history and the current and future state of these technologies.

RFID

Overview

The paper by H. Stockman in 1948 (21) is regarded as the first work to be published on the subject of RFID but, despite the technology itself being rather simple, it took many decades for the required engineering disciplines of antenna and circuit design, as well as the rise of software engineering and new circuit printing techniques, to be responsible for the recent widespread deployments and large scale applications of RFID systems.

RFID is usually used to describe a system of identification involving electronic readers sensing the presence of tags attached to items. These readers use electromagnetic field variations to communicate with and read information from the tags within range of the antennae. The data that is collected by the reader(s) in this fashion is then transmitted via a network to an RFID Middleware system, which is a collection of software that analyses, filters and processes the received data.

The main difference between RFID and other wireless transmission techniques is the asymmetric method of communication, where the reader acts as a transmitter and the tag takes the role of a responder, which reflects or modulates the electromagnetic waves from the reader. This is one of the reasons for the relative success of RFID technology, it allows complex readers with several antennae to interrogate and communicate with a range of small and low cost tags. Also note that the 3rd main

component of an RFID system (22) is called a middleware server (see FIGURE 1 – RFID SYSTEM OVERVIEW) or controller; this is a workstation or server running control and data processing software.

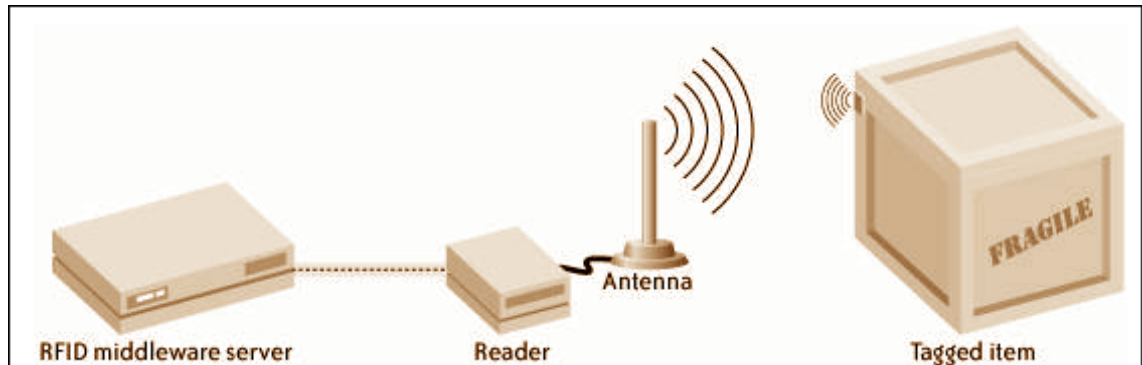


Figure 1 - RFID system overview

It is important to highlight that RFID technology suffers from several limitations, one being the physical constraints of metals and liquids blocking radio waves and the other being interference of communication between the readers and defective tags (23).

Readers

RFID readers are expensive but, powerful devices that are one of the main components of RFID systems. These RFID readers contain the following components (24) :

- One or more antennae attached to the reader, this allows for a greater interrogation area with just one reader.
- Radio interfaces responsible for modulation, demodulation and transmission and reception.
- A Microcontroller, digital signal processing and cryptographic modules.
- A Networking interface to allow for communication with software applications.

Currently one reader can simultaneously communicate with multiple tags within the read range, with a 98% accuracy rate when scanning 1000 tags per second reported (25).

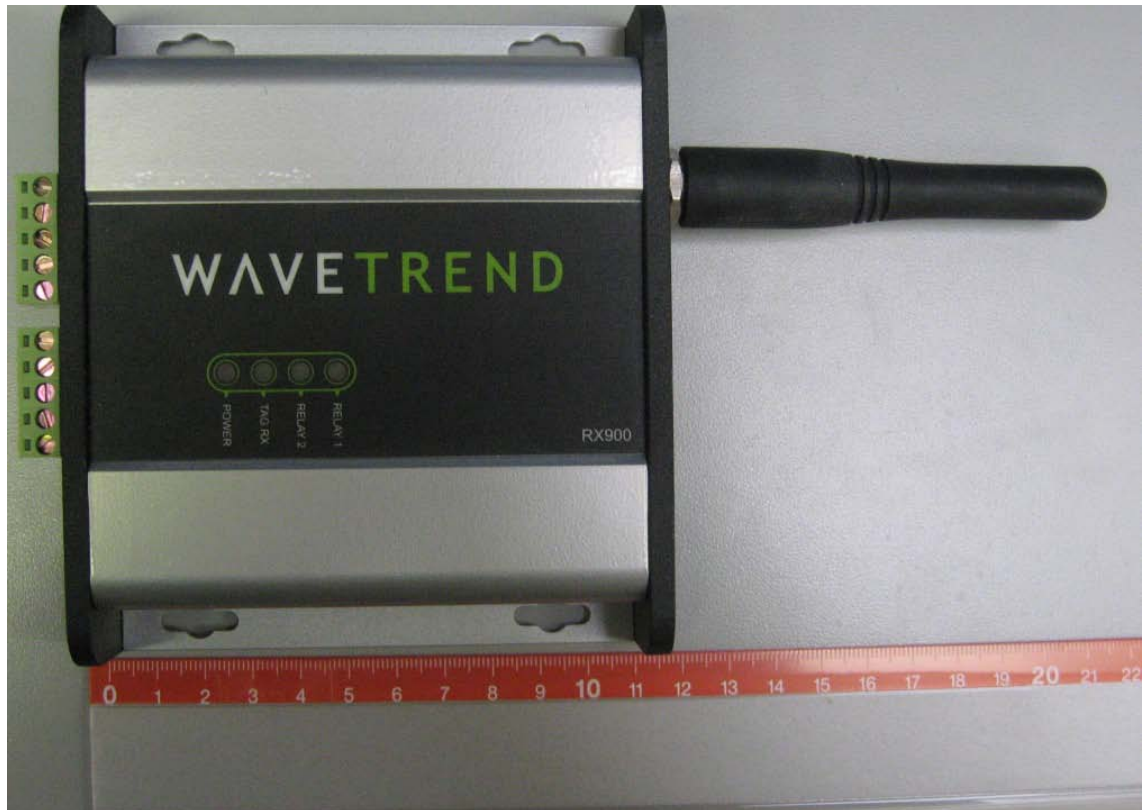


Figure 2 - WaveTrend RX900 Active RFID Reader

Tags

RFID tags are small and relatively cheap and are one of the main components in RFID systems. When queried by a reader through the transmission of a radio frequency signal, the purpose of these tags is to provide a unique identifier (26) in order to help identify the object the tag is attached to, and in some cases small amounts of data that is stored on the tag is also provided. Tags usually contain the following electronic components; a microchip, an antenna and a capacitor. These components are getting smaller and smaller over time, with new technologies such as the “coil on chip” design allowing for creation of tags 6mm in diameter and 1mm in depth (27). These

electronic tags (also sometimes referred to as transponders) can be divided into two main types; passive and active or a mixture of these two.

Passive tags are powered by the signal from the reader which interrogates the tag, they do not have a power source such as a battery and instead derive their power from the electromagnetic signal from the reader, this severely limits their range to under 3 meters, while on the positive side they have a long lifespan (28) and require no maintenance. Passive tags are the most commonly used variety and are cheap (29) to manufacture.

Active tags contain an onboard power supply usually in the form of a battery; this allows for communication over greater distances and through more interference than passive tags. Due to the availability of onboard power, environmental sensors such as a temperature sensor (30) or a humidity sensor (28) can be added to enhance the functionality of the active tag. Unfortunately active tags are currently larger in size and more complex than passive tag implementation, and the additional power requirement adds to the costs of the tags. The current cost differential between active and passive tags is large; with active tags costing up to \$US 20.00 and passive tags ranging in price between \$US 0.20 and 0.50 (31). There are two types of active tags: transponders and beacons (32). Transponders are woken up once they receive a signal from a reader, an example would be signal from a toll booth waking up the transponder on a car windshield; their main design advantage is increased battery life due to them only being used when woken by a reader signal. A beacon emits a signal at defined intervals and is usually used in location systems and needs to be picked up by at least three or more antennae in order to triangulate tag position.

Roy Want from Intel Research outlines two fundamentally different ways of transferring power from the reader to the tag and hence allowing communication; these being magnetic induction and electromagnetic wave capture (33). Magnetic coupling limits the operating range to under a meter; this is due to the magnetic field strength generated by the reader rapidly weakening with distance as described by Want: “The range for which it is possible to use magnetic induction approximates to $c/2\pi f$. Thus, as

the frequency (f) of operation increases, the distance that near-field coupling can operate over decreases (c being a constant, the speed of light)” (34).



Figure 3 - Mixture of active and passive tags used in the project

High frequency (HF) near-field RFID tags using magnetic induction operate at a frequency of 13.56 MHz and have spawned a range of standards such as ISO 14443 for ticketing, ISO 15693 for access control and ISO 18000 for item tracking (35). An example of this technology in use can be seen in key cards used by authorised personnel to gain access to an area by holding a card in front of a reader.

Far-field RFID tags capture the electromagnetic waves from the reader antenna, which powers on the electronics in the tag and the tag responds via a process known as back-scattering, the use of a backscattering link results in savings of cost and power (36). These tags must operate at specific EM ranges due to regulations on UHF communications, but they do have a much greater operational range, on the downside

these tags are affected by presence of water in the environment. These tags are recognisable by their large dipole antennas.

One interesting new technological advance with positive implications for active RFID tag technology is the development of printable batteries, research and work being carried out by the Electric Nano Systems (ENAS) at the Fraunhofer Research Institution (37). These offer the promise of millimetre thin batteries that can be cost effectively produced by printing processes. These batteries are less than a millimetre thick and weigh less than one gram, and they also provide a voltage of 1.5V and contain no mercury (38).

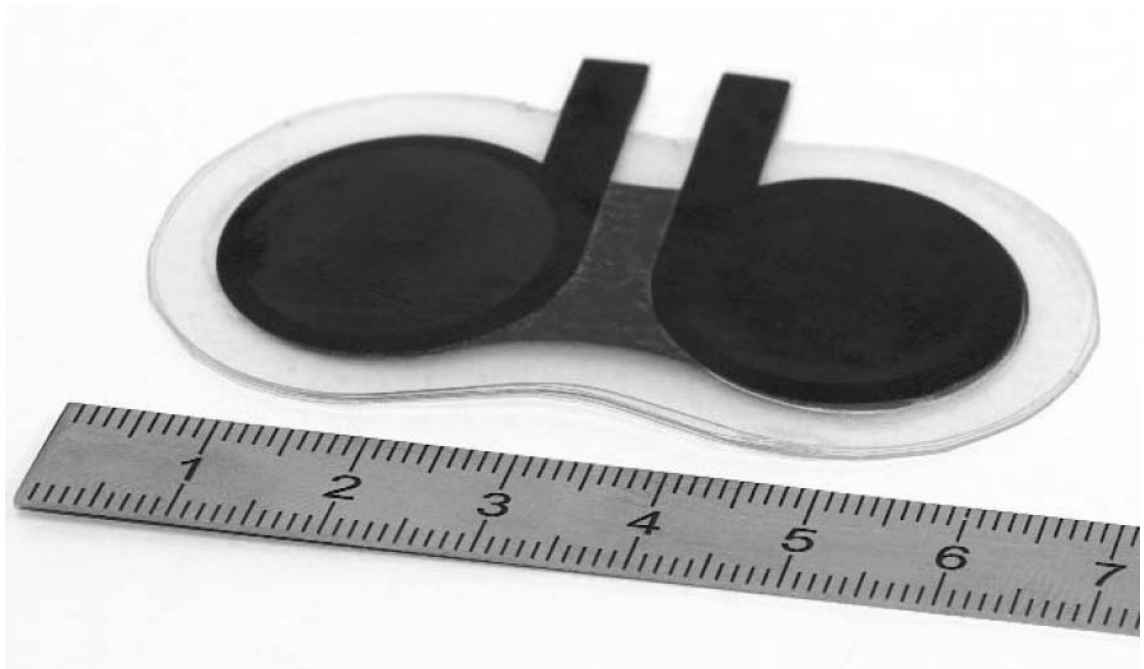


Figure 4 - Printable Battery (source: Fraunhofer ENAS)

This development would mean physically smaller and cheaper active RFID tags, which could open doors for more applications and make it more attractive to use active RFID in asset tracking.

Middleware

As outlined earlier the third component of RFID systems is the middleware server; which collects via a network the data generated by the readers, and then processes it into useful information. Listed are the main functions of an RFID middleware system:

- Collecting data from readers, this process is triggered by periodic events being called by a scheduler.
- An abstraction layer is required in order to get data into a common format from a multitude of readers, each of which use differing data transmission formats.
- Optionally authenticating and decrypting the data.
- Filtering the data to remove duplication and erroneous reads.
- Storing the processed information in a data store.
- Providing an interface for higher level applications.

Sunder Kekre, a professor at Carnegie-Mellon University expressed his concerns (39) about the flood of potential data from RFID readers, data that needs to be analysed and processed into practical information in order to be useful in business applications:

“RFID will generate vast amounts of data. But how will you utilize that data to predict problems, isolate issues, and improve processes to drive efficiencies and create value? We need better methods for developing business intelligence”

The main job of RFID middleware server is to process the data retrieved from readers, then filter out incomplete or duplicate readings and store in a data store.

Retrieving data from RFID readers is rather tricky since there are no commonly agreed communication standards in use at this time, and each reader currently uses its own proprietary and often very different interface and data formats. Hence one of the main functions of a middleware system is to provide a common and uniform API for higher level applications, and to solve the differences in communication formats between readers via the use of plug-ins designed to handle each reader’s unique interface. This

makes use of the computer science principle of encapsulation, where the internal mechanisms and workings of a system are hidden behind a well defined interface.

The authentication of tags and decryption of received data is performed in some middleware systems, in order to enhance security and privacy in RFID systems while minimising the extra overhead added by encryption features. One approach proposed by Keunwoo Rhee et al (40), works by combining an encryption algorithm with a password derived key and can be applied to low-cost RFID systems.

The core function of RFID middleware is to filter the data received from readers; this process involves reducing the amounts of data and converting it into more relevant information, which can be used by applications dependant on this data. This process can be explained using an example of an RFID tagged object moving along on a conveyor belt;

1. An Event is triggered by the middleware and the reader queries the tag asking for an identifier.
2. The Tag responds to the query, this is read by the reader and the data is read via a reader interface. There could be other tagged objects on the belt, so a large set of data is collected.
3. Duplicates and erroneous reads are removed, sometimes an object is not read due to interference but, the middleware system can compensate due to the frequency of reads.
4. The data is added to the data store.
5. An inventory application might query the store via an API in order to find all objects of type X that passed through the read area.

This duplicate removal process seen in step 3, along with continuous querying in order to catch any tagged items that might not have been read due to interference, is a characteristic of all RFID middleware systems such as the LIT Middleware outlined by Ashad Kabir et al. (41).

The final component of an RFID middleware server is the application level interface; this is a set of APIs, which allow business applications to interact with RFID systems and request useful data from them. Currently most of the RFID middleware systems use proprietary interfaces but, GS1 EPCglobal recently updated their ALE (Application Level Events) specification to version 1.1 which is described as:

“This EPCglobal Board-ratified standard specifies an interface through which clients may obtain filtered, consolidated Electronic Product Code™ (EPC) data from a variety of sources.” (42)

ALE is similar to SQL in the way it allows applications to interact with data without dealing with the quirks and complexities of the underlying system.

Another way of looking at RFID middleware is the manner in which the three layer system architecture is being used; design patterns are explored in further detail later on in this chapter. Feng Li et al (43) have identified three main layers:

- RFID hardware management layer (RHML), provides the connection between RFID devices and the middleware, hides the implementation detail of RFID reader communication, and provides an interface for the data management layer.
- RFID data management layer (RDML) processes the collected data.
- RFID application interface layer (RAIL) provides access and delivers information to other systems.

The above layers can be identified in all RFID middleware implementations encountered during the research.

Review of existing RFID middleware systems

Microsoft BizTalk

RFID middleware can be considered a subset of Message-Oriented middleware systems; this is a system that uses messages to transmit information between components. The best known of such messaging systems is Microsoft's BizTalk server, which also includes RFID adapters in the most recent versions (BizTalk server 2006 and 2009). Weiwei Sun et al. describes BizTalk RFID as a system that (44):

"... provides an open interface based on XML and Web Services, various hardware (such as RFID, bar code, IC card) compatible with DSPI (Device Service Provider Interface) can be Plug and play in Microsoft Windows, besides that OM/API is provided, the application program can be coded in managed code (C#)."

BizTalk RFID stack consists of several layers, see **FIGURE 5 - BIZTALK RFID STACK** (SOURCE: PRO RFID IN BIZTALK SERVER 2006 R3)

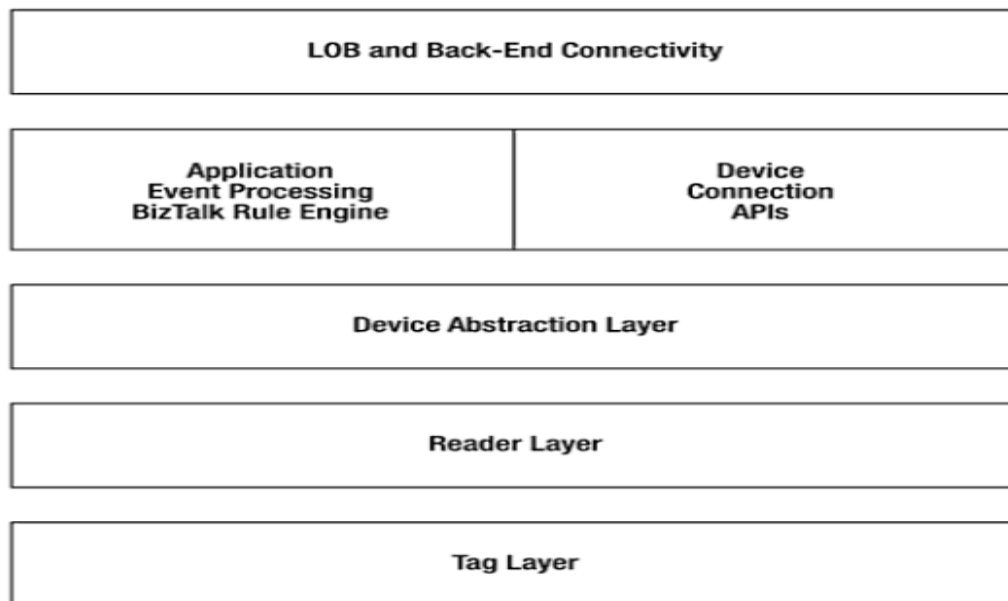


Figure 5 - BizTalk RFID stack (source: Pro RFID in BizTalk Server 2006 r3)

Looking from the bottom hardware layer we have 5 layers as described by Mark Beckner et al. (45):

- Physical Layer – Lowest layer representing real world physical entities such as RFID tags, barcodes and various other sensors.
- Reader Level – This layer is responsible for the communication between the physical layer below and the Device Provider Layer (DSPI) above
- DSPI – Provides uniform abstraction to the above layer for hardware devices using plug-in drivers which are called “device providers” within BizTalk and are usually provided by reader manufacturers.
- Application Level Services – This is a collection of edge application level services designed to be called by end user code, This includes;
 - Data abstraction API’s similar to ADO.NET model for databases.
 - Execution environment for BizTalk Rule Engine (BRE) rules.
 - Event processing model called RFID business process; similar to XML based Windows Workflow Foundation (WF) workflows are defined.
- Back end connectivity layer – The top layer of the stack, includes BizTalk orchestrations, real time monitoring, analytics and connectivity to other systems.

IBM WebSphere RFID

IBM’s WebSphere RFID middleware system provides the infrastructure and backbone support for RFID solutions, with the aim of; connecting tags, readers and enterprise information systems (46). The main components (also illustrated in **FIGURE 6 – IBM WEBSHERE SOLUTION MAP**) are:

- WebSphere RFID Device Infrastructure.
- WebSphere RFID Premises Server.
- WebSphere Business Integration Server.
- Custom Business Logic dependant on a particular deployment and usage scenario.

IBM WebSphere RFID is not a complete off the shelf solution, and does require writing custom application software (preferably by a costly IBM consulting team), in order to:

- Process RFID events received from the edge controller, and optionally modifying the event processing and device controller logic.
- Implement premise specific business logic to interpret the event data, a sample application is included.
- Collect and correlate the data into back-end business processes and systems.

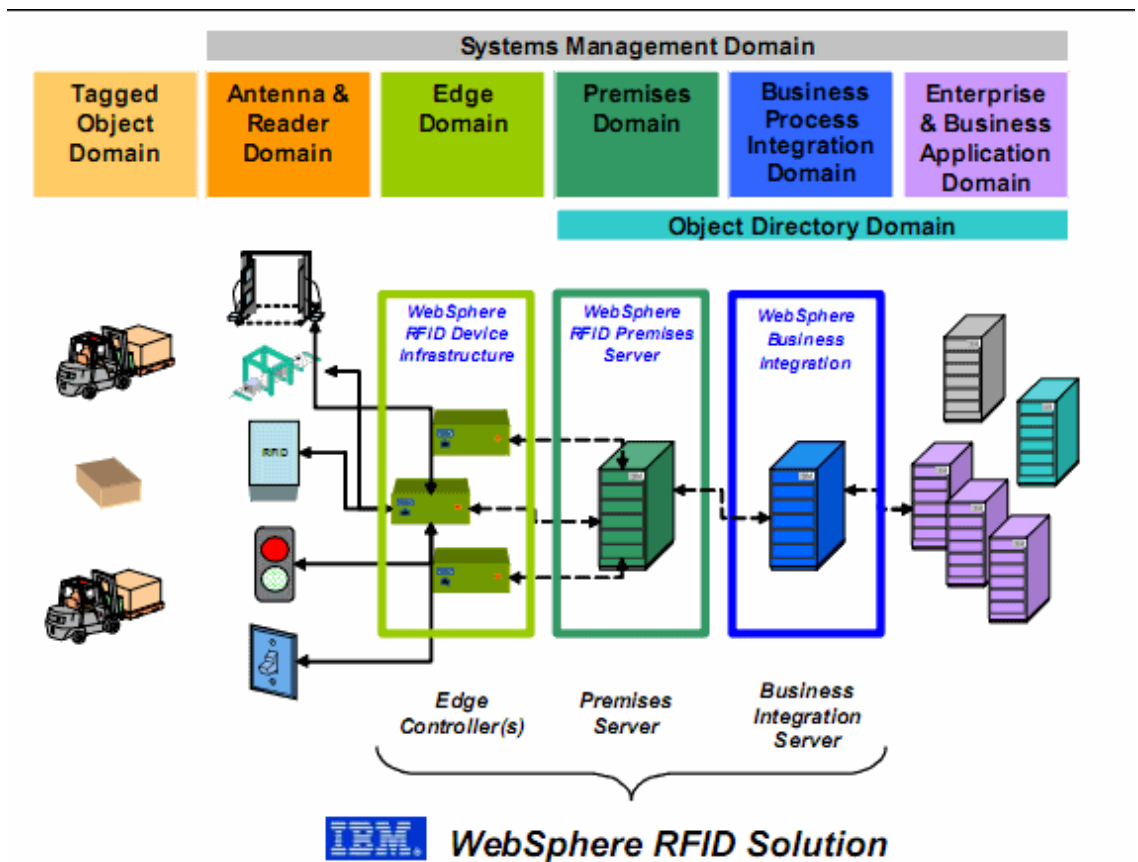


Figure 6 - IBM WebSphere Solution map

In some respects WebSphere RFID is similar to BizTalk server; they are both message driven systems. Some of the key concepts include:

- Events – These are XML messages containing information about an event, usually tag reads from readers.
- Queues – Events flow through the system using a series of message queues. The originating sender just places the message into a queue and is not concerned with what component or components will receive and act on the message.
- Tasks – A publish and subscribe model is used within WebSphere, when an event is placed in a queue; WebSphere MQ notifies one or more subscribed components, which process the events accordingly.

Review of existing RFID asset tracking systems

“Global SAW Tag” RFID system

GST RFID is one of the more interesting examples of RFID technology in use in an asset tracking process (47). The main objective of this system is the tracking of assets on the International Space Station (ISS) in order to reduce costs, which are very high (\$20,000 per kilogram (48)) when it comes to bringing objects and supplies into orbit.

The ISS presents a whole range of unique asset tracking issues, which can be solved with the use of RFID technology. Due to the zero gravity condition, assets need to be tied down, and are hidden from view inside flight bags. Crew also used to spend up to 20 minutes a day updating and synchronising with the legacy barcode asset tracking system, reduction of this time to 4 minutes a day with the use of RFID, yields an estimated saving of \$20,000 per day (30). Also being able to quickly find and resupply certain assets creates a safer environment for the crew.



Figure 7 - SAW RFID tags and tagged flight-bag (source: NASA)

The Saw technology can provide a complete inventory of a flight-bag, with 98% accuracy, in well under a minute, compared to 30 minutes for manual inventory in orbit. FIGURE 7 - SAW RFID TAGS AND TAGGED FLIGHT-BAG (SOURCE: NASA) shows a close up of the tags and a flight-bag and FIGURE 8 - NASA VERSION OF THE SAW 704 READER WITH PDA (SOURCE: NASA) has a photo of a reader and attached PDA.

The system was tested in 2008 in orbit and has displayed key advantages such as:

- Anti-collision capability.
- Large ID number space.
- Small physical tag size.
- Signal working with liquids and metals.
- Low, safe RF reader power.



Figure 8 - NASA version of the SAW 704 reader with PDA (source: NASA)

“OATSystem” and “Reva” RFID Asset tracking systems

The RFID Journal provides us with two interesting case studies of RFID asset tracking systems; these are OATSystem in use at the Super Bowl and the Reva system in use by Sony (49).

OATSystem is used by the Sports Video Group since 2007 in order to tag and track expensive and sophisticated video production equipment, which is used to televise the Super Bowl event. The technology helps personnel to find the equipment quickly.

The Reva system in use by Sony Europe helps reduce shrinkage and increases shipping efficiency. An interesting twist in this system is the use of cameras, which are triggered to record and store video when a tagged item passes thru the read area.

RFID Asset Tracking Alternatives

At the time of the writing of this thesis, there are several emerging technologies, which hold the promise of possible applications in asset tracking.

Bokode's

The MIT Camera Culture Group has put forward a paper on *Bokode's* at SIGGRAPH 2009 conference (50). Within this paper they detail a novel optical barcode design, which may be read with an out of focus camera at a relatively long range by exploiting the *bokeh* effect of ordinary camera lenses. Despite the increased read range and being able to store more information compared to barcodes, bokode's might not be useful in asset tracking applications when compared to RFID, since they use the visual part of the electromagnetic-spectrum, hence requiring a line of sight between the camera reader and tag. The MIT researchers reason that this line of sight requirement increases security compared to RFID tags and doesn't require an on-board power source.

RuBee

Another possible alternative is RuBee, a new development from Visible Assets. RuBee is based on a new standard IEEE P1902.1 "RuBee Standard for Long Wavelength Network Protocol" (51):

"RuBee will allow for networks encompassing thousands of radio tags operating below 450 KHz. RuBee networks provide for real-time inventory under harsh environments, e.g., near metal and water and in the presence of electromagnetic noise. RuBee radio tags, which can be either active or passive, have proven battery lives of ten years or more using inexpensive lithium batteries"

Despite being described as an alternative to RFID, RuBee can be considered a variant of RFID and contains the same core characteristics as RFID technology (52). RuBee uses active LF (low frequency) tags containing an on board battery and clock. One of the more interesting features of RuBee is how IP (Internet Protocol) is used to address

tags via their own unique IP address, also since each tag is a transceiver it can connect to other tags in range and establish a peer-to-peer network, this enables the base station reader to exchange information with tags that are not within the read field (1 to 30 meters) of the reader itself (53).



Figure 9 - RuBee tags (source: Visible Assets)

RuBee is a relatively new technology and has a lot in common with active RFID; for example the use of batteries enables the tags to be usable for a period of several years. There are also differences between RuBee and RFID technologies such as:

- RuBee tags are active long wavelength transceivers working in the 131 kHz to 450 kHz band (54), unlike RFID which works in the VHF or UHF bands.
- Almost all of the energy radiated by a RuBee base station is in the magnetic field as opposed to the electric field. Vipul Chawla et al. note that (55): “Long-wave magnetic signalling has a great advantage; it is highly resistant to performance degradation near metal objects and water, a serious problem for UHF and Microwave far-field RFID”.

The ability to work near liquid and metallic objects in harsh conditions, gives the technology an edge when it comes to asset tracking compared with RFID, and is leading to adoption of RuBee technology by companies such as Motorola, Panasonic, Sony and IBM (56).

RuBee poses several disadvantages:

- Data transmission rate is very slow (1200 baud) compared to other packet based network standards and the packet size is limited to few hundred bytes.
- If multiple tags are present in a small space, transmissions from various tags overlap, this leads to the tag identifier and other information transmitted to be missing.
- The use of IPv4 protocol is short-sighted considering that IPv4 addresses are becoming scarce; using IPv6 would have been a better idea.

Software architectures and Design Patterns

An overview of several software architectures and design patterns is required, in order to understand and appreciate the design choices made during the development of the Juno system. The following section provides a review of relevant, tested and proven design patterns. A combination of these methods along with several architectural differences, were used in the design and the development of the Juno system, the specific implementation details are outlined in later chapters.

Design patterns and pattern based development provides software developers with proven design techniques, in order to efficiently solve common programming design issues. Frank Buschman et al. write (57):

“Patterns help you build on the collective experience of skilled software engineers. They capture existing, well-proven experience in software development and help to promote good design practise. Every pattern deals with a specific, recurring problem in the design or implementation of a software system. Patterns can be used to construct software architectures with specific properties.”

Design patterns prove useful when designing and developing large software systems, as they help to visualise complicated processes into a common pattern or set of patterns. These models also help in facilitating communication of concepts between

designers and developers of a large system, which is usually developed in a team environment.

Multi-Tier Architecture

Multi-Tier Architecture, also referred to as N-Tier or Three Layer, is a manner of separating presentation from data management and business logic. The most commonly used variation is the Three Layer architecture, often used in client/server systems.

Layering is a common technique used in many software and hardware systems, best described as cake (58); here each layer rests on a lower layer, with the top layer using the services defined by the bottom layer in the stack, with the bottom layer being unaware of any higher levels. An example of this is the modern network stack; here an HTTP service runs on top of TCP/IP which in turn can be transmitted through a variety of physical networks. Layering has its downsides, for example modifying the bottom data layer by adding a new database row, leads to modifications needing to be made at the higher application and presentation layers.

The Three Layer Architecture is an attempt to solve the domain logic issues raised by the older 2 Layer client/server model, an example of one is a client connecting and manipulating a database directly. It consists of three primary levels; data, logic and presentation. This architecture lends itself well to designing web based applications, for example;

- Presentation Layer consisting of HTML documents and JavaScript in order to display data to user and provide interactivity and comfortable UI experience.
- Domain Logic Layer consisting of PHP or JAVA code in order to; handle specific domain logic and input validation, providing calculations on the inputs and stored data, handling inputs from presentation layer and dispatching outputs.

- Data Layer, most often a relational database system whose concern is to provide data integrity, transactions, redundancy and storage.

Thomas Shelford et al. (59) provide an example of a Three Layer architecture, this sample application is also illustrated in **FIGURE 10 – SAMPLE 3 LAYER APPLICATION:**

1. User Requests an article at the web based presentation level.
2. The request is processed by the business logic, which queries the database.
3. Data is converted into an XML document.
4. This in turn is transformed and returned to the user.

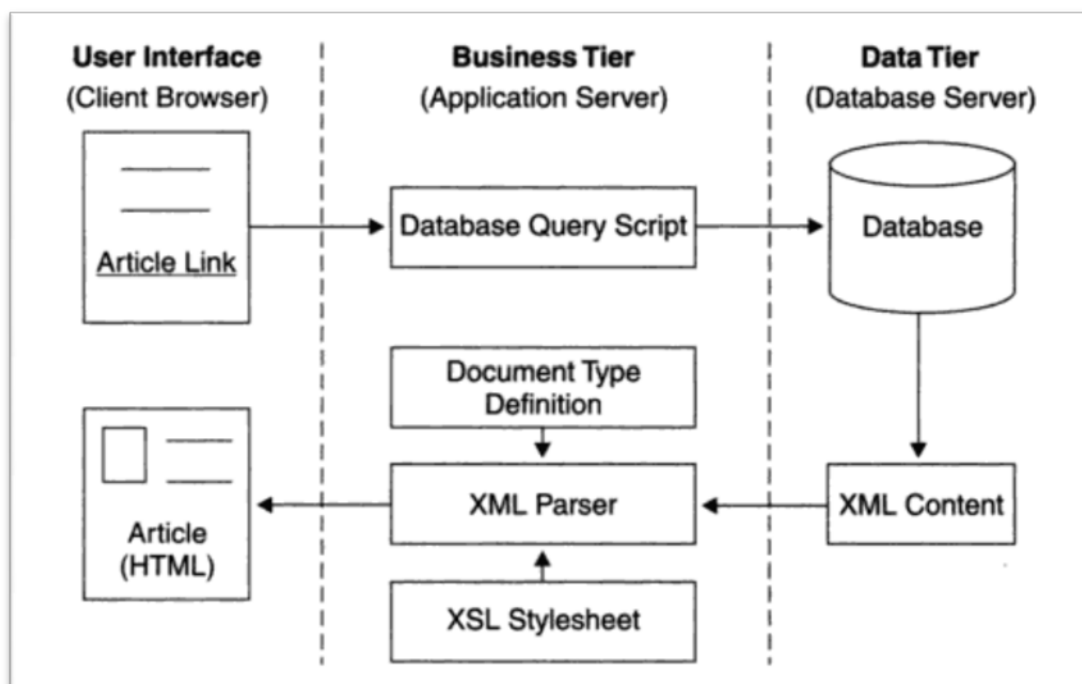


Figure 10 - Sample 3 Layer Application

This architecture is not without its own issues, for example questions usually arise as to where to place data validation and filtering or how to map relational data to object oriented models. Nevertheless Multi-Tier Architecture and its variations are important when it comes to designing web applications, these usually consist of:

- Web servers such as Nginx serving documents and browsers such as Internet Explorer presenting this information to the user, this is the presentation layer

- Application logic involved in processing and generating dynamic content, for example PHP or ASP.net scripts.
- Back-end database managing and storing the data, for example MySQL database.

Some of the above components can be identified within the Juno system and will be described in further detail in the Technology section of the next chapter.

Model View Controller

N-Tier architecture concerns itself with separating Data Access, Business Logic and UI but, difficulties arise in trying to separate the UI from Business Logic. Model View Controller (MVC) design pattern tries to address this issue, by introducing a Controller, which communicates with the View and the Model. The main components of MVC are:

- Model is an object or a set of objects, which represent domain logic and functionality to access and work with the relevant data. There is no Data Layer in MVC as it is encapsulated by the model.
- View displays the data generated by the Model. It also concerns itself with UI aspects such as presenting human readable information to users.
- Controller handles the information received from the View, for example: an HTML form with new information being submitted. It processes and filters this information, updates the Model accordingly and then updates the View.

According to Fowler (58), the Model View Controller pattern splits user interface interaction into three distinct roles; as illustrated in **FIGURE 11 – CLASSIC MODEL VIEW CONTROLLER**. He also identifies two principal separations; separating the View from the Model and separating the Controller from the View. Separation of the View from the Model is a good software engineering practice as it allows for:

- Easier testing of visual and non visual components.

- Separation of concerns. The View deals mostly with UI issues and how to present data, while the Model involves thinking about business processes and data interactions.
- Creation of multiple Views, this facilitates the output from the Model to be shown in different ways, for example data from the Model can be displayed as an HTML page or exposed via an API.

The separation between View and Controller is often harder to distinguish, but it does help in application development and keeping the code clean. An example would be *ASP.net code behind* where the HTML view is separate from the C# code, which handles events. Also a View could potentially interact with multiple Controllers or the controller can be replaced with a dummy test Controller in order to facilitate testing, hence this separation is important.

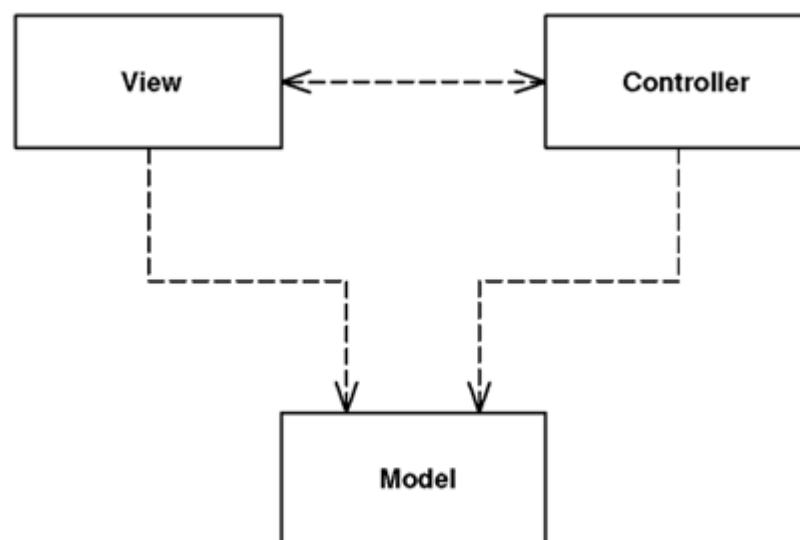


Figure 11 – Classic Model View Controller relations

MVC pattern is used widely in web applications, mainly because it lends itself so well to the request/response communication cycle between the client and the server. It is considered to be an evolution of the legacy Input, Processing, Output (IPO) pattern used in old text-only applications (60). The following example describes how MVC fits into the context of a web application:

- The Model consists of a set of classes representing the domain logic objects. An example could be a User class, which has methods to interact with an external database to add or remove users in the system; it also could have optional methods to check data validity such as whether a username is of a correct format.
- The View is rendered by the browser, for example in response to an HTTP request to the server; the output from the above User Model could be a list of users in the system rendered as an HTML table.
- The Controller is the glue that links the View and the Model, in this example it could be a PHP script running on the server, which processes a received HTTP POST request from the clients browser, which has the rendered View, it would process this request and pass the data to the Model, a sample request could be to delete a user from the system.

Jason Sweat provides another way of looking at the MVC pattern (61), as used in HTTP based web applications within the context of the request/response cycle; this is illustrated in **FIGURE 13 – MVC IN HTTP REQUEST/RESPONSE CONTEXT**.

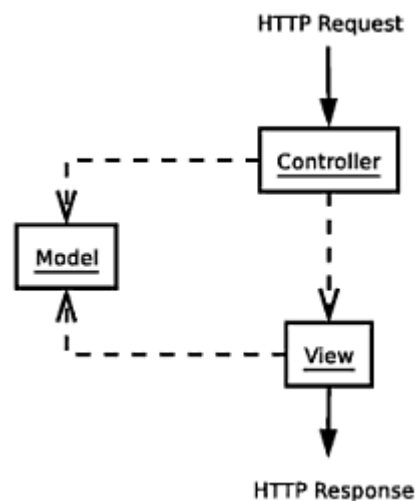


Figure 13 - MVC in HTTP request/response context

There are many implementations of the MVC pattern in multiple programming languages, with frameworks such as ASP.net MVC or Zend Framework (62) offering the infrastructure required to build an MVC class of applications.

Front Controller

The Front Controller pattern attempts to solve several issues inherent in web applications, by providing a single point of entry for all requests to an application. Matt Zandstra identifies some of these issues (63):

- If incoming HTTP requests are handled at multiple locations, this inherently leads to code duplication.
- Any updates may need to be deployed across several application entry points, this makes it difficult to test or maintain a system.

A solution for these issues is to configure the web server to forward all HTTP requests to one front controller entry point script; this allows the application developer to consolidate common functionality into the front controller, but a method needs to be created in order to route the requests to appropriate command objects (also referred to as Page Controllers, more on them later).

FIGURE 12 – FRONT CONTROLLER SEQUENCE DIAGRAM shows a sample execution sequence involving the Front Controller pattern, also displayed is the Intercepting Filter pattern, which is often used alongside Front Controller; this pattern allows the developer to add a chain of filters to perform a variety of common tasks such as authentication, logging or localisation (64). Here we also see the handler analyse the URL, determine the course of action and then dispatch control to an action command. This action command object could be a variation of the Page Controller pattern.

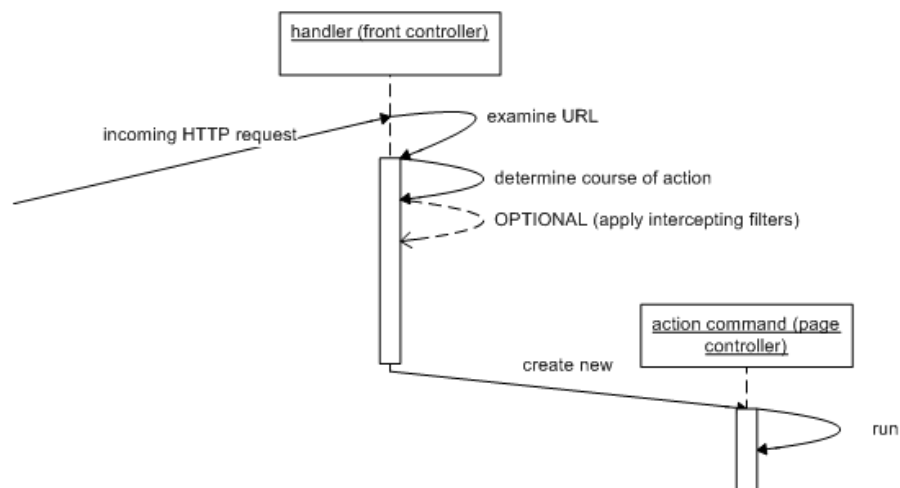


Figure 12 - Front Controller Sequence Diagram

Registry Pattern

As described earlier a common way of designing applications is to use several layers with defined communication interfaces, this done in order to gain flexibility. In Object Oriented designs this raises issues when it comes to accessing global data, parameters and shared objects; a common example in web applications would be access to the shared database connection class. The use of global variables is often a bad design choice and causes issues; such as a component modifying a global variable, which leads to unexpected outcomes elsewhere.

A solution to referencing well known objects without resorting to the use of global variables is the Registry pattern, which has been described as (61):

“The Registry pattern is like an object phone book – a directory – that stores and retrieves references to objects, making it the definite source of information for your entire application.”

| Registry |
|--|
| -object_store[] |
| +init() +get() +set() +get_instance() |

Figure 13 - Simple Registry class

The Registry class allows one controlled access to objects and eliminates the need for global variables; this can be useful in Unit Testing of components and helps reduce dependency bugs in software. As can be seen in Figure 13 - Simple Registry class the Registry consists of:

- An associative array often called the “object store”; here the references to objects are stored.
- A small set of getter and setter methods, which allow access to the object store.

Chapter 3 – System Requirements and Technologies

In Chapter 2, RFID and Middleware technologies were reviewed. These technologies and architectures provide a foundation for the Juno system. This chapter begins by identifying the gathered functional and non-functional requirements. Following on is an overview of the various software technologies chosen for this project and used in the development of the solution.

Requirements

The final outcome of a completed software system is heavily dependent on the steps taken and the decisions made early on in the software development process; the same applies to any engineered system comprising of hardware and software components. Requirements gathering and analysis is a very important first step in the development of any project, since any mistakes made or errors introduced early on in the development of a system, are amplified and cause large and costly issues closer to the completion of a project.

The goal of software development is to develop quality software that meets the customer's needs, and is produced on time and on budget (65). In order to meet this goal, the system requirements must be gathered and analysed. Requirements' gathering is a communication intensive process; in the case of this project this involved meeting with the "customers", which were the employees of Efast Teo (the industrial partner for this project), in order to record, clarify and discuss the system requirements.

The following 5 step approach was taken in order to gather clear requirements, and compile them into a useful requirements document:

1. Firstly an agreement was reached on the definition of the problem to be solved, this involved creating a problem statement document, and discussing it

with the “customer”. The problem statement document consists of; description of the problem, stakeholders affected by this problem and the impact on business activity.

2. An understanding of root causes of the problem was gained by listing them.
3. Stakeholders involved where identified, and the users of the system.
4. The boundaries of the proposed system where defined.
5. And finally system constraints such as technical and financial issues identified.

The requirements were also influenced by previous asset tracking system in place at Efast. This piece of proprietary software called A-Track is a Java based application, whose main features include:

- Importing a list of assets from a comma separated file into the database.
- Listing the assets and viewing details of each one.
- Synchronising with XML files containing tag reads produced by WaveTrend RFID reader.

Unfortunately this system did not work too well, due to a wide range of bugs and ever changing requirements. Some of the bigger issues with the A-Track system that were noted are:

- Inconsistent and unintuitive User Interface, with most of the work being done in Excel instead by the users in order to save time.
- Import of assets via a comma separated data in a text file being buggy and producing inconsistent results or failing without any errors.
- Windows Server containing the Java application crashing and becoming unresponsive after few days of running, a memory leak is suspected to be the cause.
- Synchronization issues with the RFID reader resulting in corrupt data or none at all being recorded.
- Deployment and maintenance issues requiring attention from technicians and increased labour costs.

These largely negative experiences with the A-Track system have helped in the requirements gathering process, since these issues could be examined and new solutions proposed.

The final requirements for the Juno RFID asset tracking system can be broken into 2 sections, consisting of functional and non-functional requirements. Functional requirements describe what a system should do and how it behaves (66). Non-functional requirements specify constraints on the tasks of the system.

The top level requirements can be listed and described as follows:

- Data is to be read by various RFID readers from tags attached to assets.
- A common interface and messaging data format is required in order to simplify working with the following readers; Sirit, WaveTrend and Feig.
- Flexibility has to exist within the design in order to be able to easily add other reader types to the system.
- Tag read data has to be collected from the networked readers by the RFID middleware server and various filters applied, such as duplicate removal.
- The processed information is to be matched and reconciled with the asset database.
- Asset tracking logs are to be kept, in order to provide functionality such as asset movement history report.
- Any missing assets outside the RFID interrogation area that are unaccounted for several minutes are noted, and an exception is raised notifying the users, so manual reconciliation can be undertaken.
- The system has to be web based for easy deployment and use via a browser.
- Password based authentication system is required in order for the user to be able to access the interface.
- A simple, consistent and clean user interface to be presented to the user.
- Asset database needs to be accessible with a web based asset registry browser.

- Detailed item view is required, providing the users with; detailed asset info, user comments, asset and tag relation, asset photograph and finally asset accounting information.
- The interface needs to be easily customisable for branding reasons, and hooks in place in order to provide translated versions.
- Administration panels are required in order to provide a range of functionality such as editing users, assets, tags and dealing with exceptions.
- Reader overview and live status display dashboard in order to simplify system management.
- Asset import facility via upload of a comma separated values text file.
- Asset import via a third party middleware system from an ERP system.
- Easy deployment and configuration of the application.
- Use of open source technologies in order to cut costs and aid any future maintenance and extension works on the software.

The above are the main system requirements and some of the constraints. As can be seen these requirements are numerous, and can be further broken down into sub requirements and tasks. The completed system would contain several sub systems and would need to span a range of hardware and software systems as well as require the creation of own custom software and integration of the all the parts.

System Usage Scenario

During the course of the requirements gathering process an example system usage scenario was agreed on with the industrial partner. This helped focus the attention towards the key requirements of the system, and also helped highlight the main actors, inputs and expected outputs of the Juno system. The scenario was generated via the use of a use case driven approach and techniques. Doug Rosenberg writes the following about use cases (67):

“Use cases give you a structured way of capturing the behavioural requirements of the system, so that you can reasonably create a design from them. They help you to understand two fundamental questions; what are the users trying to do? And what’s the user’s experience?”

There are two types of uses cases, business and system. The difference between these two is the scope; the business use case describes in non-technical language the business processes and actors involved and what goals are achieved, while on the other hand the system use case describes the functionality offered to the user and what the actor achieves.

Use cases are represented using UML Use Case diagrams, UML and other UML techniques are discussed in more detail in the subsequent chapter.

The three typical system usage scenarios are:

1. Continuous asset monitoring. In this scenario tagged assets within the read space are continuously monitored and reconciled with the asset register. RFID readers and antennae are setup at strategic points within a large area such as store room. Assets are located throughout this area and are tagged with RFID tags. The readers then periodically interrogate the tags within the read range. Tag read data is collected and processed from all readers before being reconciled with the asset register, with any exceptions being recorded.
2. Working with the asset registry. This scenario involves the system users performing various tasks such working with assets, handling exceptions and exporting system reports.
3. Integration with a third party ERP system. Here the system user would initiate a series of events, which would cause a connection to be made to a third party ERP system, allowing for assets to be imported from the ERP system.

Technologies

Developers usually have a variety of choices when it comes to choosing platforms, tools and even hardware when it comes to designing, developing and testing a system. In this section the infrastructure and technologies used during the development of the Juno system are presented. Also described are the reasons for the choices made in selecting the hardware and software components of the system.

Linux OpenSUSE

The very foundation of the system is built on top of the Linux operating system. This allows access to a wide range of free and open source software, and helps meeting the requirement goals of: low cost, easy maintenance and deployment, as well as flexibility and extensibility. Linux is a UNIX like operating system that is highly modular and very flexible, it adheres to POSIX standards and current development is overseen by the Linux Foundation (68). The name Linux refers to the core of the operating system, the GNU/Linux Kernel, this provides the drivers and the interface between hardware and software, as well as other core system tasks such as CPU scheduling and memory management (69). The rest of “Linux” operating system consists of third party open source modular components such as; file systems, desktop environments, command shells, compilers etc. These components and programs built on top of the kernel are commonly referred to “Linux” as well, but this is not strictly correct.

The large mixture in the way components and libraries are build and configured on top of the kernel has spawned a wide range of Linux “flavours”, each flavour is constructed to fulfil different software needs, examples of which include:

- Edubuntu follows the philosophy that an operating system has to be accessible to everyone (70), and its main aim is to aid in education. Hence it includes components such as educational games and scientific tools.
- BusyBox is a popular distribution (distribution is another name for a Linux “flavour”) used widely in embedded systems such as TV set top boxes (71). BusyBox project describes itself as the “Swiss Army Knife of embedded Linux

Systems”. One of the better examples of this operating system environment running in an embedded system is the FEIG RFID reader used in this project.

- CentOS is a Linux distribution developed in parallel to the commercial Red Hat Enterprise version of Linux, and does not include the same commercial support. Its main use is on the server, mainly running web applications, and is widely used due to support for various web hosting packages (72).

With so many available Linux distributions of various shapes and sizes, it is hard to make a choice, but after careful consideration it was decided to use OpenSUSE Linux distribution for the purposes of this project. The main reasons and advantages of using OpenSUSE are:

- OpenSUSE is one of the major and most popular Linux distributions (73), with the project being actively developed, maintained and upgraded. It is also the most popular distribution in Europe at this time, with plenty of support available online.
- The YAST system administration interface is unique to SUSE and puts all of the common system administration tools under the same well organised set of control panels. This greatly helps reduce maintenance and aids in system configuration tasks such as; disk management, software package management and updates, network management and network services configuration, user and authentication management. All of the other Linux distributions have these settings in various locations, hence making the system administration tasks harder to reach and work with.
- The install package can be highly customised and only the components required installed, this allows to; reduce dependencies, cut down the system size, install and use only what’s needed which in turn helps system performance and efficiency. For this project a bare minimum of an operating system was customised with only the required modules and their dependencies. The achieved size was in the region of 400 MB with further reduction in size possible, but a point of diminishing returns is passed below this size.

- OpenSUSE is the community developed fork of Novell SUSE Enterprise Linux, which comes with enterprise level support. This allows the Juno system to be easily migrated to SUSE Enterprise if such level of support is required.

VMware

Ease of deployment is one of the main requirements for this system; with VMware this aim can be achieved. VMware is a virtualisation server which can run multiple unrelated operating systems on top of a Windows or Linux host. A virtualized operating system is packaged and installed in a virtual machine, this provides several advantages:

- Ease of management and deployment, as the whole virtual machine exists as a set of files in a folder. These can be easily transferred from host to host, and several software systems are deployed in this manner, such as OpenBravo ERP discussed later on this chapter.
- “Cloning” allows for an exact replica of a virtual machine to be created, this helps greatly in the testing process where a clone can be easily created and new configuration tested.
- Backups, snapshot and screen capture functionality allows creating backups, or rolling the system up to a point in time, or replaying a series of events.
- Isolation between host and guest and between guests provides better security.
- Sub-division of resources allows several guest operating systems to run on one server, thus better utilising server resources.
- Being able to change virtualized hardware specification of a virtual machine, helps in the testing process. For example; the RAM memory usage of a machine can be gradually decreased to test performance in a low memory environment and to define the minimum system requirements.

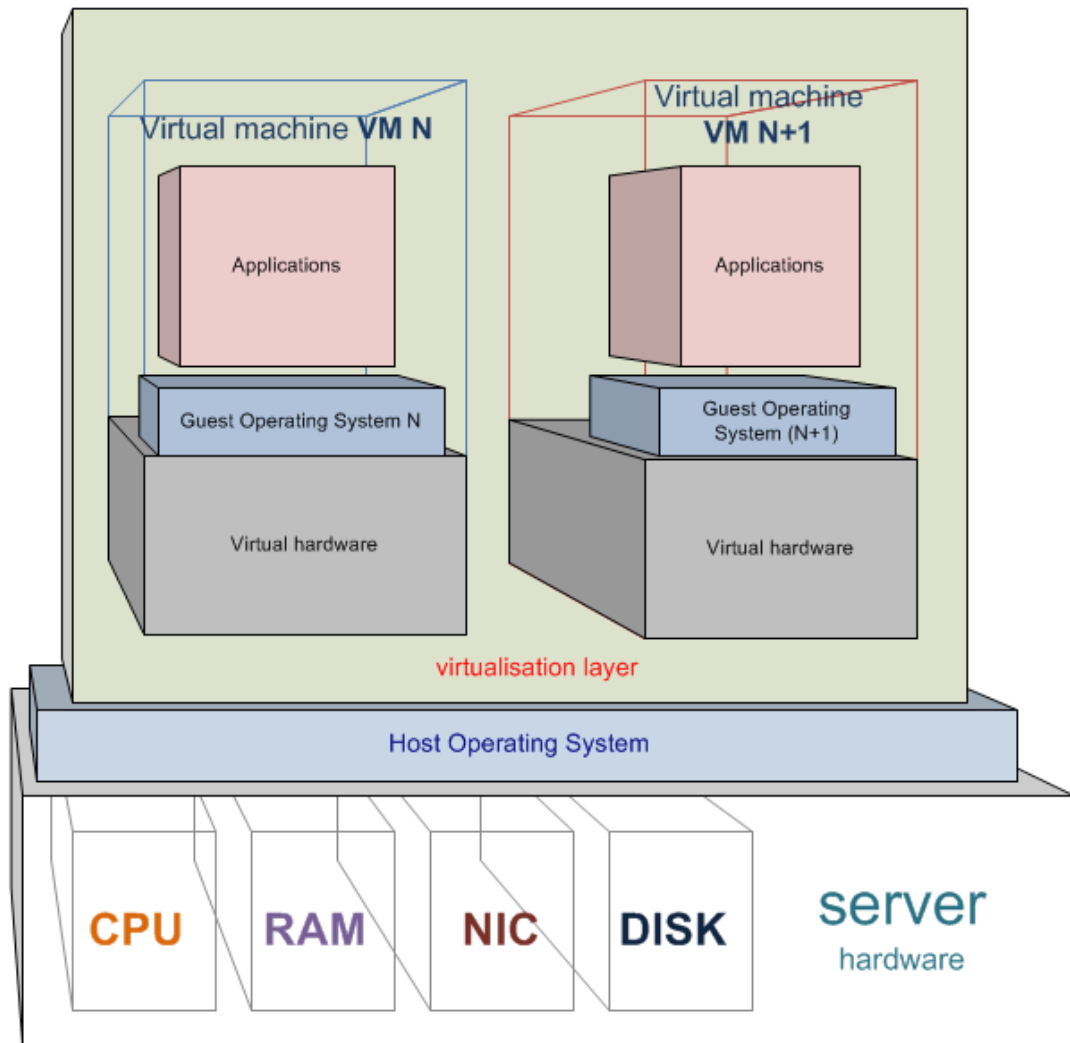


Figure 14 - Virtualized System overview, hosted architecture

There are two virtualisation architectures; hosted and hypervisor. FIGURE 14 - VIRTUALIZED SYSTEM OVERVIEW shows an example of a hosted architecture; here the virtualisation software such as VMware Workstation runs on top of full operating system. This means that the virtualisation layer relies on the host operating system for device and resource management (74), and this incurs extra operating overhead.

Hypervisor architecture is the second approach to virtualisation; this is a thin and bare-metal software virtualisation layer on top of the server hardware with no underlying operating system host. There are several examples of a hypervisor architecture; Xen

open source project and VMware ESX. In this project the Hosted virtualisation method was chosen as it provided more flexibility, but the Juno system and the virtual appliance it runs in can be easily deployed on a VMware hypervisor server, in order to make better use of the server hardware.

MySQL

Some of the key requirements of this project include; the use of low cost open source software and the need to store large amounts of data generated and filtered by the RFID asset tracking processes. At the core of the Juno system sits the data store, with the MySQL relational database system being picked in order to fulfil the data storage needs and requirements.

MySQL is currently one of the most widely used open source database systems, thanks to; speed, stability and extensive interoperability with multiple programming languages (75). MySQL supports multiple table types, each with different set or properties and characteristics, this unique aspect of the MySQL database system has proved very useful within the Juno system, for example:

- Memory – Memory table type stores the data in memory, this allows for very fast data retrieval and updates. This table type is useful for non important data such as temporary caches and was used within the Juno project to cache system parameters and to provide in memory locking of forms in order to prevent CSRF attacks (discussed later in detail in the security chapter). One downside of this table type is that data is destroyed when server is restarted; hence it's important not to store any important data in this table type.
- MyISAM – This table type provides very fast retrieval and indexing, also a light compression scheme is applied on the data. This is the default table type in MySQL, and was used on several tables within this project, such as the *temp_tags* table which contains the RFID tag reads before they are further processed. This table type does not provide transaction safety; this means it

cannot be trusted to maintain data consistency, as there is a chance that data can go missing.

- InnoDB – InnoDB is latest alternative to MyISAM which offers extra functionality for full ACID support (76), such as; transactions, foreign key constraints, row level locking and crash recovery. ACID stands for Atomicity, Consistency, Isolation and Durability, these properties are highly important to a database system which guarantees that transactions are completed reliably and safely. This table type was used extensively within the project in order to provide data integrity and transaction support. These extra features come at a price of reduced read speeds, but the difference is negligible.

During the design and planning stage of the project other database systems were evaluated, the most notable include PostgreSQL, which is a newer open source database project. PostgreSQL is indirectly used within the project, as its one of the two database back ends used by OpenBravo ERP system. Decision was made to use MySQL due to better documentation available, which proved useful during the system development stage.

Nginx

One of the core project requirements is the availability and access to the system via a web based application. Web applications run on the server and the client, where the web server responds to HTTP requests with mainly HTML documents, which render and execute on the clients browser. A web server is one of the core components in the server/client communication process, whose features generally include:

- Quickly serving static files in response to HTTP requests, such as images and style sheets.
- Serving dynamic content generate by a program, script, or API called by the web server. Examples would include dynamically generate content such as bulletin boards and auction systems; these are usually generated by

programming and scripting languages such as PHP, ASP.net and Java Server Pages.

- HTTPS support, this allows secure and encrypted SSL or TLS connections between the client and the server.
- Logging detailed information based on the request/response cycle, these logs can be data mined in order to extract useful traffic statistics and information.
- Authentication features to prevent unauthorised access to sensitive information.
- Compression to reduce the size of responses to lower bandwidth usage and speedup web application performance.

Nginx web server supports all of the above features and more, via a modular architecture. Nginx is a free and open-source HTTP server, known for its stability, high-performance and rich feature set (77). Due to the highly modular structure and open source nature it was possible to compile a customised Nginx web server containing only the components required for to fulfil the requirements of the project. This flexibility and customisation feature of Nginx allows for the creation of a small executable web server to fulfil the requirements, also reducing execution overheads and helping security by reducing the attack surface for this internet facing server.

PHP

PHP is a high level programming language, similar in syntax to C, and highly adapted for the purposes of web application development. PHP stands for “PHP: Hypertext Pre-processor”, and is by far the most popular language for use on web servers (78). The open source PHP project is managed by the PHP Group, and the platform is available free of charge.

PHP is used throughout this project, due to the following language and platform characteristics; Object Oriented Programming (OOP) features, web application specific functionality, command line support, clear documentation, open source license and rapid application development (RAD) facilities.

Object Oriented Programming (OOP) features allow for greater code reuse, more modular components and designs, cleaner designs and higher code reusability. The object is a combination of data and related methods that work on this data, represented in many Object Oriented (OO) languages by a class structure. Rasmus Lerdorf, who is the creator of the PHP programming language, writes (79) “OOP acknowledges the fundamental connection between data and the code that works on the data, and lets you design and implement programs around this connection”. PHP does not include full set of OO features compared to languages such as Java, but the language is continually evolving and being improved thanks to its open source nature, current version of PHP is 5.2 with version 5.3 now being in the final testing stages (PHP 5.3 will bring new features such as namespaces which are currently missing). Inheritance, interfaces, abstract methods, constructors and destructors are all present and available in the current version of PHP, allowing the users to use many object oriented design patterns.

PHP’s main aim is to allow for fast and rapid web application development. The platform includes useful functionality in order to achieve this, such as;

- Easy access to HTTP request information such as the GET and the POST parameters.
- Sockets support to allow for messaging over TCP connections.
- Delimiters allow for easy construction of HTML output, this output is then sent to the client.
- A variety of frameworks such as the Zend Framework (62), which have highly granular components in order to accomplish tasks and solve issues common to web applications.

The language is widely used especially for web applications, with excellent online support and printed literature. PHP.net website (80) contains extensive API documentation as well as examples about all of the language functions and features.

Unlike Java, PHP is a dynamic language with dynamically typed variables; this means a variables type can change and does not need to be defined, the determination of the type is left to the language software interpreter. This property allows the programmer to be more productive by allowing the code to be written in a higher abstraction level, which is further removed from the hardware and facilitates in rapid prototyping and testing. PHP code is pre-processed and compiled at runtime, extensions such as APC which allow for the dynamically compiled code to be cached in shared memory greatly increasing performance in high load applications; this also helps avoid the constant parsing and compiling overheads.

PHP scripts are usually executed when a web server (in the case of this project Nginx) passes the contents of a HTTP request sent by the client to the PHP processor. In this project several PHP Fast CGI processes are started and they keep running on the server in the background. The maintenance, upkeep and rotation of these processed falls on the PHP FPM extension, which is compiled in with PHP, the purpose of this patch is to maintain PHP Fast CGI instances and to ensure they don't turn into zombie processes or leak system memory. To summarise the web server hands the HTTP request to the PHP instance running on the server, PHP processes the request and replies, and the output is passed back to the server which in turn sends it to the client. This process can be scaled horizontally in order to increase system throughput and availability by adding more PHP instances on same or other hardware servers, with Nginx having the ability to load balance requests between multiple Fast CGI back-ends.

JavaScript

JavaScript is an interpreted programming language with Object Oriented capabilities (81). JavaScript is most commonly used and run client side, within the end-users web browser; here it can interact with the user, the html document and the browser. In the case of this project, these client side abilities are important at providing enhanced user interaction with the Juno system.

Most notable features of JavaScript and their usefulness with respect to the Juno system are: event processing, client side validation, document manipulation and asynchronous client/server communication.

Being able to catch browser events such as button clicks and allows the software to react to these events. This form of software architecture is referred to as event driven programming. In a traditional web application, an event such as submitting a form usually results in triggering a full HTTP request/response cycle to the web server, with JavaScript one can intercept this event and perform a variety of operations such as double checking the data is safe or altering the user interface to notify the end user that an operation is in progress.

Client side validation ties with the above ability to interact and react to user initiated events. Client side validation such as checking whether an input from a text field is of a correct format (example: date formatting), is quite important in providing a user with a more richer experience with the application, as any issues can be detected before communication with the server occurs. This allows for a smoother user experience, often seen in desktop applications.

Document manipulation is another important feature of JavaScript. Web servers usually reply to HTTP requests with HTML documents in the response, HTML allows for JavaScript code to be embedded inline within the document or to be linked to in a form of a separate file containing JavaScript code. As the document is loaded and parsed by the web browser, any inline or linked JavaScript is executed in a top down fashion. Listening to and using browser events, script execution can be deferred until

the browser fully downloads, renders the document and builds up a complete Document Object Model (DOM). David Flannigan describes DOM as “an API that defines how to access the objects that compose a document” (81). This means that parts of the document can be manipulated, such as adding, removing or updating portions of text, hyperlinks or images. An example of this would be showing the user an error message on the page if an exception is detected.

One of the most useful features of JavaScript is the ability to perform asynchronous HTTP requests. HTTP requests are traditionally triggered by a browser when a link is followed to a new page or an image is loaded onto the page or when a form is submitted, this is the basic request/response cycle. Asynchronous communication allows one to develop richer web applications that are very similar to desktop applications. This process works by JavaScript making use of the XMLHttpRequest object, this is a key component of a web architecture known as AJAX. AJAX stands for Asynchronous JavaScript and XML, despite its name it is not limited to XML requests, and in fact most of the asynchronous calls within the Juno web application sub system are JavaScript Object Notation (JSON) encoded data structures (more on this shortly). AJAX requests can be performed and handled in the background by JavaScript with no page reloads in browser, requests can also be set to poll the server at time intervals allowing for more functionality and giving the application a more responsive feel. Data received in an asynchronous response can be encoded with XML or JSON. JSON is light weight, text based and human readable data interchange format (82), it has several advantages over XML in this form of client/server communication such as; small data size, faster processing and parsing as the format is native to JavaScript.

An example of all of the above JavaScript features in use within Juno would be the user inputting a new asset to the system. Client side validation would check the fields for correctness once a submit event is triggered, the inputted information is serialized into a JSON structure and sent as background asynchronous request, the response JSON structure is received by a call-back event which decodes the response, which in turn is processed and the user is notified via page manipulation of a successful or otherwise outcome.

Despite all of the positive functionality provided by client side JavaScript, some issues exist, such as the cross-browser and cross-platform inconsistencies. These differences sometimes exist within the different versions of the same browser, for example the XMLHttpRequest object and its methods have different names and other nuances. All of these issues make designing and testing client side applications rather difficult compared to server side programming, where one can write a test case to unit test code and ensure that the outputs are consistent. Fortunately several JavaScript frameworks have recently come into existence with jQuery (16) and the Yahoo User Interface Library (YUI) (83) being used within this project, in order to gain API consistency, component modularity and data encapsulation with respect to JavaScript code. As outlined by Richard York (84), jQuery allows a programmer to:

- Significantly reduce the amount of JavaScript programming due to a powerful syntax.
- Cut down on the quality assurance testing, since jQuery replaces cross-browser inconsistencies with a common API.
- Make the code more intuitive and easier to understand, hence aiding future maintenance and feature extensions.

OpenBravo

One of the main requirements of this project is to integrate the RFID middleware with an ERP system. Due to the prohibitively high cost of an ERP system and not being able to gain access to a research licensed version of an ERP system such as SAP, a decision was reached to use an open source ERP system, in order to display the ability to bridge an ERP solution with the developed Juno RFID middleware.

OpenBravo is an open source, web based ERP system. Its main aim is to cater to small and medium enterprises by providing a wide range of features. It provides a portal

where business information about products, employees, assets and much more is stored, in order to help managers to track the state of the business and aid decision making (85). Main features of OpenBravo include:

- JAVA API allows access to the data and can be used to synchronise with other applications.
- Web based interface to allow a user to view and enter data relating to; product information, order tracking, customer, company and workflow information.
- Procurement and warehouse management features that deal with the operational part of the business.
- Asset management and overview.
- Simple deployment using a downloadable VMware image.

Some of these features are highly relevant and useful to this project, adding to the reasons for selecting OpenBravo ERP.

Customisation and control are key requirements for enterprises when it comes to choosing an ERP system. OpenBravo excels when it comes to offering this functionality, for example Galencium, who a supplier of raw materials to the pharmaceutical industry compared SAP Business One, Microsoft Dynamics and OpenBravo with OpenBravo being selected. The COO Erich Buchen had the following observation (86), “The most important factor was that it is easier to customise OpenBravo than the other two. SAP and Microsoft Dynamics are much more rigid in what they can do, or at least in what their consultants say they can do”.

OpenBravo system architecture is comparable to the Juno system design, with many structural similarities such as:

- VMware virtual computer image is used to distribute OpenBravo, with the virtual images being available for free on SourceForge (15).

- The foundation of this virtual machine consists of a stripped down and a highly tuned Linux operating system.
- The data is stored in a bundled open source PostgreSQL database, Oracle can also be used as a backend database.
- OpenBravo ERP web application is executed inside the included JAVA Platform and served by Apache Tomcat web server.

OpenBravo uses the above components and the Model View Controller pattern in order to keep the presentation logic and the business logic separated. Model Driven Development is also employed; this allows programmers to describe the application in terms of models rather than code. Paulo Juvara who is the CTO at OpenBravo describes the used of these methods as (87), “The use of both proven development frameworks in one seamlessly integrated ERP is an innovative approach enabling a better way to build and maintain software coding”.

One of the more unique aspects of OpenBravo is the Wizard for Application Development which generates human readable MVC code from the metadata in the dictionary. WAD is executed every time the system administrator updates the application configuration; the required changes are recompiled in order to suit the user’s needs (88).

Other Open Source ERP systems such as Compiere and ERP5 where initially explored, with OpenBravo being picked due to the features listed earlier and better facilities available for further integration.

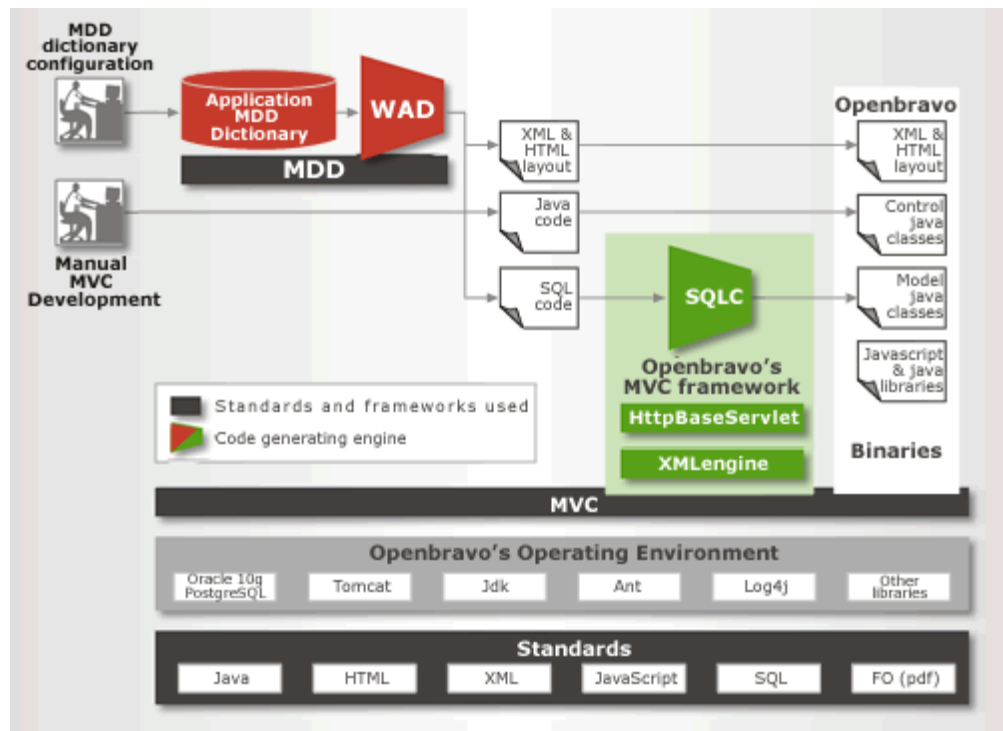


Figure 15 - OpenBravo Architecture (source: openbravo.com)

Chapter 4 – System Design, Development and Implementation

In the previous chapter the system requirements were listed and an overview of relevant software technologies provided. This chapter outlines the design decisions made in order to fulfil the system requirements and project objectives, along with the Software Engineering methods used to aid in the design and development process, and finally this chapter describes how the specifics of the solution were implemented.

Unified Modelling Language

Unified Modelling Language (UML) is a vital software engineering toolbox, comprised of a set of graphical notation techniques, used for modelling and to aid communications during system design. The official definition of UML according to the Object Management Group is (89):

“A standardized graphical notation for expressing the structure and behaviour of object oriented systems.”

As noted by Fowler (90), the main aim of UML is to help in the communication process by allowing developers to model complex systems. Natural language is too vague when it comes to describing these difficult concepts while code on the other hand is precise but far too detailed.

Multiple types of structural, behavioural and interaction diagrams are available in UML, for example;

- Component diagrams help show how a system is split into components and what are the dependencies between these components.
- Class diagrams describe the structure of a class, its properties and methods as well as the relationships between classes. Highly useful in Object Oriented design. An example already used in this thesis can be seen in **FIGURE 13 – SIMPLE REGISTRY CLASS**

- Sequence diagrams illustrate the messages passed between objects as well as lifespan of these objects. An example of such a diagram has been encountered in an earlier chapter, see **FIGURE 12 – FRONT CONTROLLER SEQUENCE DIAGRAM**
- Activity diagrams display the workflows of components in a system.

UML was already used in this thesis to, and will be used in this chapter to convey issues and design concepts.

General System Architecture

As discussed earlier the aims and objectives of this research are extensive, and the previously outlined requirements are numerous and detailed. A systematic approach was taken early on in the design process, in order to subdivide the project into smaller better defined sub systems and their related interfaces. This divide and conquer approach was taken in order to meet the original requirements, deliver the core components of the system in stages as per specification, and to keep the project on schedule.

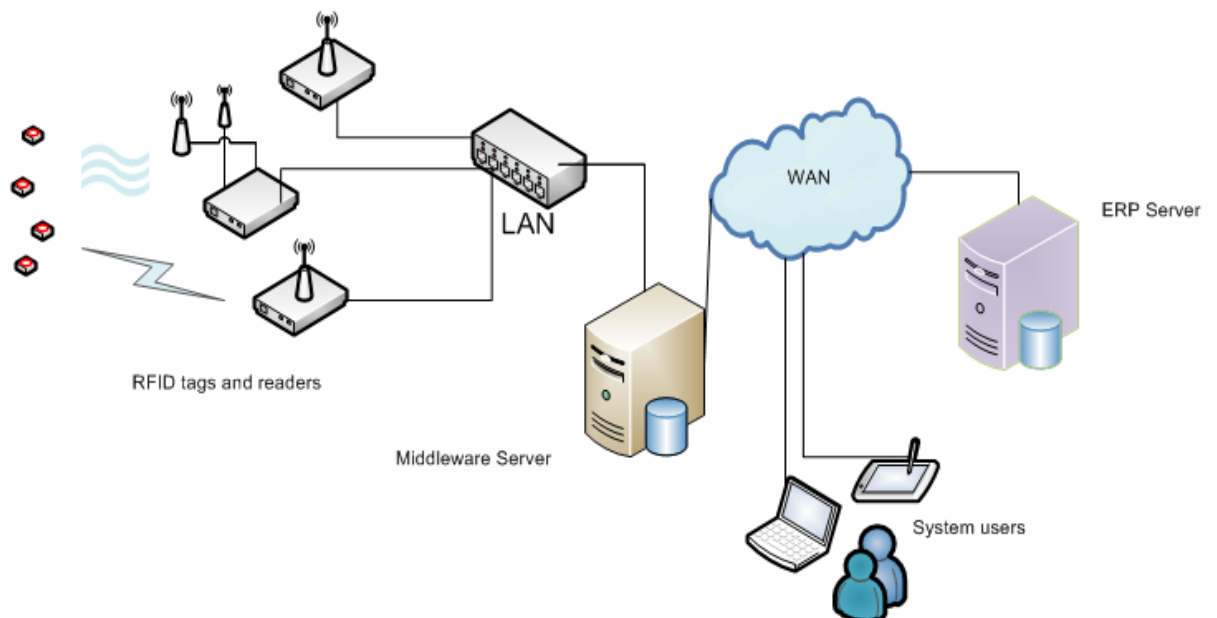
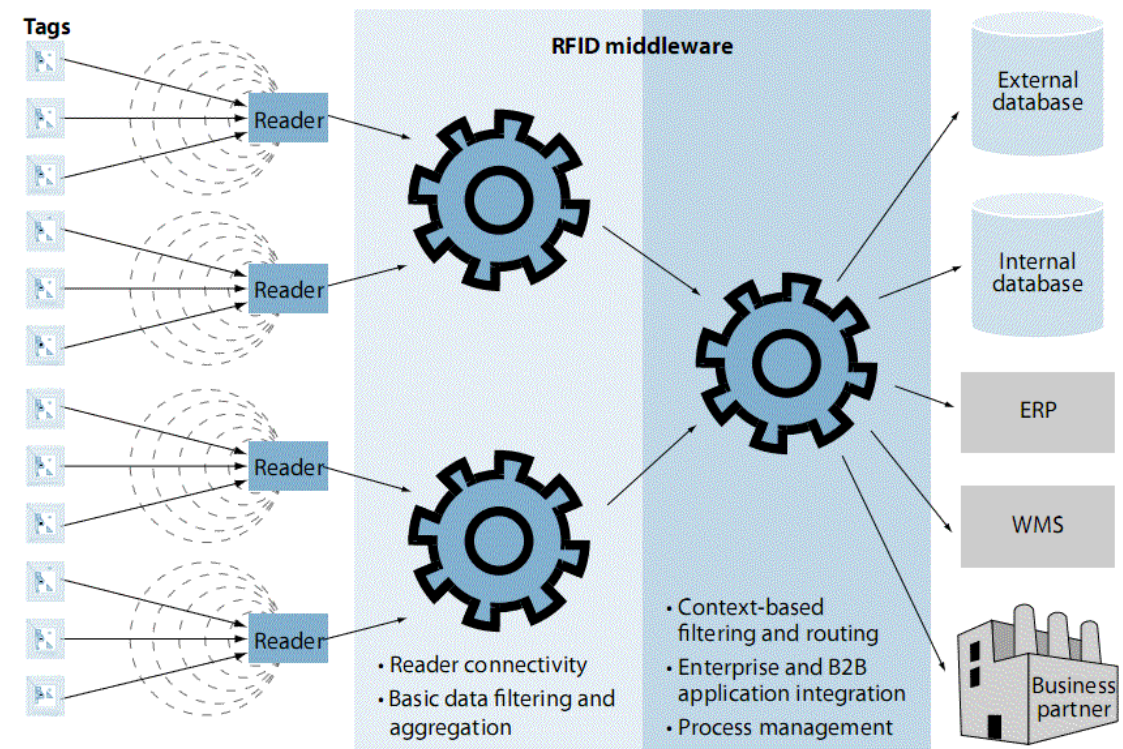


Figure 16 - Main Hardware and Networking components

The project involves extensive integration of hardware and software components, and development of custom software to meet the rest of the requirements. **FIGURE 16 – MAIN HARDWARE AND NETWORKING COMPONENTS** shows the key components of the system, here we have:

- RFID readers interrogating and communicating with tagged assets. These readers are connected via Ethernet network cables to a LAN router.
- A Middleware server sits at the core of the system; its main aims include processing data received from readers, storing this data and providing secure access to the filtered information to System users.
- Final component is an ERP server available either on the local network or via the wider area network.



Source: Forrester Research, Inc.

Figure 17 - Multitier RFID Architecture

This structure is roughly similar to the Multitier middleware architecture as described by Forrester (91) which can be observed in **FIGURE 17 – MULTITIER RFID ARCHITECTURE**

The overall system is subdivided into 3 sub-systems, which reflect the above structure:

- Data from RFID Readers and tags being collected via a Reader Driver Abstraction layer, then filtered and put in a data store.
- The Juno web application which displays data to relevant authorized users via a web based interface, and also allow these users to interact with the data using a range of exposed functionality related to the asset tracking processes.
- Bridging and integration with an Enterprise Information system via Microsoft BizTalk middleware to allow for flow of data from various sources.

Each of these is explored in detail in subsequent sections.

RFID Driver and Abstraction Layer Design

Several different passive and active hardware RFID readers and tags were acquired for this project, these are:

- Sirit Infinity 510 passive reader
- WaveTrend RX900 active reader
- FEIG LRU 2000a passive reader

Initial examination and tests showed that these readers have widely differing ways of retrieving tag read data and controlling said readers, also no common data exchange format is in use, for example; the FEIG reader returns data in proprietary hexadecimal encoded messages while the Sirit reader returns data in simple XML message format. Despite calls for standardisation of RFID reader communications from industry leaders such as EPC Global with their Reader Protocol Standard 1.1 (92) or the proposed

RFIDPROBUS protocol (93), there is still no widely accepted or used standard interface provided by manufacturers of RFID readers.

A solution devised to handle these inconsistencies between the readers, was to use an approach similar to the Layering pattern previously described in this thesis:

1. Find and list the data fields and required functionality common to all of the readers.
2. Create an Abstraction Layer which hides the reader differences from the application requiring the data from these readers by:
 - a. Exposing a common API interface to the application
 - b. Allowing each specific reader interface to be handled using a plug-in architecture.

With the help of the plug-in architecture, tag read data is polled from each of the readers used in this project and added to a temporary “*temp_tags*” table in the database. These read queries are initiated at regular intervals using an event manager; in this case the Linux system scheduler *crontab* sends a signal every minute to the abstraction layer in order to gather data from the readers. The readers themselves physically communicate with any tags within the read space at set intervals, this interval varies reader to reader but can be changed and often occurs every few seconds, these reads are buffered within the readers on board memory, this buffer in turn is read and emptied by the reader plug-in, where the data is passed through the abstraction layer and used by the application.

The “tag read” data returned by the abstraction layer can be described by this tuple:

(reader_id, tag_id, timestamp, rssi, antennae_number)

- **reader_id** – Unique identifier for the reader.
- **tag_id** – Unique identifier for the tag, usually a unique UPC number of the tag, but these identifiers can be edited.
- **timestamp** – A Unix timestamp for when the event occurred.

- **rsssi** – Received signal strength indicator, a measurement of power present in a received radio signal (in this case between tag and reader antenna)
- **antennae_number** - Some readers provide the information on which antenna the signal was received and this data can be used for location purposes.

The reader driver plug-ins and the system abstraction layer are coded in two different programming languages, and are executed via command line and system calls. The reader drivers are implemented as Java .jar files; they accept input parameters via command line arguments and output response in a plain text data format. The reasons for choosing Java in this situation were; familiarity by the researcher (James O’Shaughnessy) with the language and cross platform availability.

A command line (CLI) PHP wrapper scripts, wrap around, execute and provide a bridge between the Juno system and the drivers. The reason for choosing PHP at this stage of the abstraction layer (by the author) is mainly due to code reuse, such as Database connectivity and interaction functionality within the Juno system. Also due to the dynamic nature of the language as previously discussed changes can be made rapidly and tests performed quickly.

The temporary “*temp_tags*” table is a Memory type MySQL table discussed earlier; the data is stored within the memory of the serve, this allows for operations on this table to be performed quickly and efficiently. The main operations performed at this stage are:

- Filtering operations performed on the temp data, the main filtering function is the duplicate removal process, which helps in removing excess and redundant data before any further processing. This process is common to all RFID middleware systems as previously discussed in Chapter 2 – Middleware section; it helps in refining large amounts of data into smaller quantities of more relevant information.
- This filtered data is then reconciled with the asset inventory data table and analyzed for any exceptions. An exception is issued and stored for display to

the system users, exceptions notify the users that an asset is missing from the read area and provide a date and time to when the asset went missing.

Juno Design

As previously discussed at the core of the overall system sits the Juno web application, which allows authorised users to interact with the system and work with the information. In order to meet the requirements set out by the industrial partner and to construct a web application which possesses the properties of; extensibility, modularity and ease of use, this sub-system was designed and coded using some of the design patterns examined in Chapter 2.

Initial Prototype Design

Due to the importance placed on usability and UI design by the industrial partner, a decision was made by the author very early in the design stage to dedicate some time to creating a mock-up prototype of what the completed interface might look like. The prototype had some functionality attached to it, such as being able to login and view sample asset register. It took several weeks of work to create the prototype, at that stage of the project the industrial partner were still on board, and hence it was possible to receive feedback from end users as the prototype was developed. This is a form of a top down approach to design and helped in the developing of and working out some of the finer details of the system requirements. The response from the industrial partner was positive as it allowed the employees to compare side by side the user interface of the old A-Track system against the prototype of the new Juno system.

Soren Lauesen defines prototypes in the context of user interface design as (94) :

“A prototype is a primitive version of a system. It is not intended for real use, but for experiments to resolve difficult questions such as: Is it fast enough? Will the stakeholders like something of this kind? Can the users figure out how to use the system?”

Developing the prototype and getting early feedback from the end users, was of great help in answering several key questions and changing the overall system design at an early stage.

Database Design

At the core sits the relational database which is used to store the required data, with MySQL being chosen for the task. The following approach was taken in order to design the database:

1. Entities and relations were extracted from the system requirements and use case documents, and then listed.
2. An Entity-Relation (ER) diagram was constructed using a visual database design tool called MySQL Workbench, a free and Open Source application.
3. The ER diagram was then mapped onto tables and a database schema created.
4. Several of the tables were configured further in order to accommodate some of the system requirements, such as setting the type of some of the tables to Memory table type.

In this manner the database schema was modelled and created in a top down fashion; see APPENDIX 1 – JUNO ENTITY RELATION DIAGRAM AND SCHEMA.

Model View Controller Framework

As previously discussed a very common design pattern in web application is the Three Tier Design; here the data layer which includes the database schema detailed in previous section sits at the bottom. Connected to the data layer is the business logic and in turn interface layer of the Juno web application, which is composed of a framework that supports; models, controllers and views coded in PHP 5.

The typical execution flow follows the request/response cycle of a HTTP application.

An example can be illustrated with the use of a common task performed within Juno, a task such as adding a new asset to the system:

1. User completes a web form in their browser and submits it.
2. The data is send along a HTTP request via network to the Nginx web server.
3. The web server performs various checks such as HTTPS authentication, and decides where to forward the request as per web server configuration file; in this case the request is for a dynamic resource and is forwarded to one of the many instances of PHP FastCGI processes running on the server.
4. The requests are received by the front controller, the Front Controller design pattern was discussed in length in Chapter 2 – Design Patterns. This is a single point of entry into the application, which performs several actions common to all requests, such as:
 - a. Loading application wide settings.
 - b. Calling intercepting filters, these are common actions that filter and work with the request stream, for example filtering the request parameters in order to avoid security issues such as XSS, which is discussed in more details in the next chapter. This is a similar concept to the Intercepting Filters design pattern discussed earlier.
 - c. Deciding and routing to the correct page controller, also known as module. In the case of our example the control and data is delegated to the *“process_add_item”* action.
5. The Page controller (also can be referred to as Action controller) is invoked by the Front controller, this fits under the Controller part of the Model-View Controller design pattern. Each page controller has a very specific task to perform and several other actions which are common across them all:
 - a. Like the front controller, the page controller loads its specific settings, this allows for fine grained control of each module/action.

- b. Various common filters as specified in the settings are executed, for example filters such as establishing a database connection, maintaining a session or setting up the variables used localisation.
 - c. The main job of each controller is to create one or many instances of required Models, and calling and working with the required methods of these Models. In the example of the new item being created, this would involve creating an Item object and passing in the parameters to the *add()* method, the model would perform object to relational data mapping and insert an entry into the database table.
 - d. Once the controller successfully finishes working with the model, and assuming no exceptions arise, then a response is constructed. This involves passing response parameters to an output template and parsing this template in order to build up the response, which is usually a HTML page.
6. The response is outputted by the PHP instance and is received by the web server, which at this stage might perform post processing actions such as output compression or logging, before finally sending the HTTP response to the client.
 7. The response is rendered by the browser and the user interface is updated with a success message.

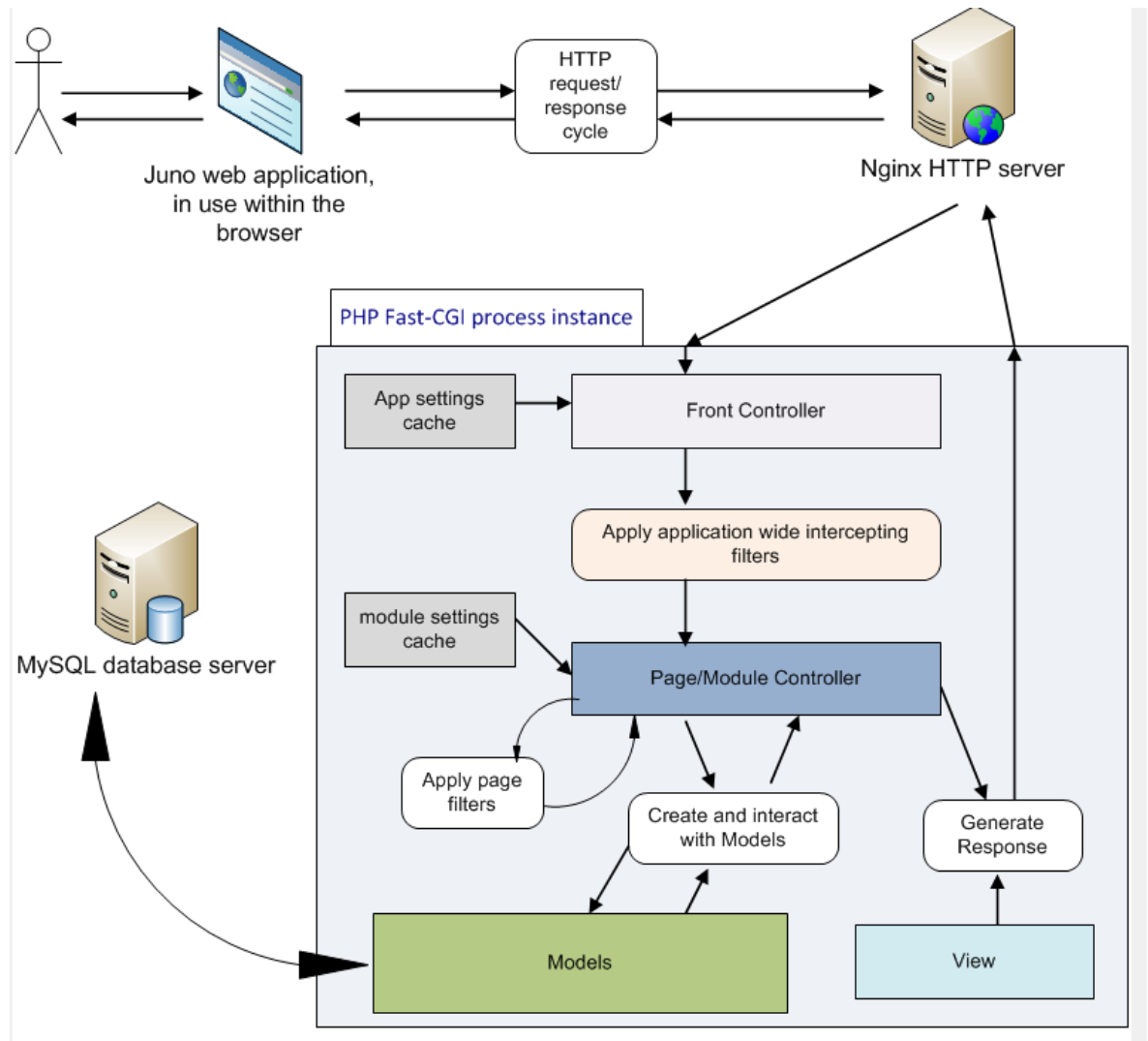


Figure 18 - Juno web application, system overview and dataflow

This whole process is illustrated in **FIGURE 18 - JUNO WEB APPLICATION, SYSTEM OVERVIEW AND DATAFLOW** and is similar across all of the other function points of the application, with the main differences being in the page controller, the models they work with and the views that are used. For example if “remove asset” functionality is required to be added to the system, this would involve creating:

- Page Controller class which extends the base Action Controller class, this uses object oriented technique of inheritance.
- Specifying the settings for this action, such as what filters to load.

- Completing the *main()* method of this action class with code to:
 - Create an Item model
 - Double-check the input parameters
 - Call the remove method of the model
 - Creating a View and passing in output parameters such as the name of the item review
 - Outputting the result from the View which contains the rendered response page as a HTTP response.
- The View template which contains placeholders for output parameters and allows rendering of the response which is often a HTML page.
- Creating a series of unit tests to test how this Page controller performs with a range of inputs. This also helps in quickly finding bugs and issues, instead of waiting for them to surface at a later time when it might be harder to find any issues.

Authentication, Session control and Secure HTTP

Since HTTP is a stateless protocol where each request is separate from the next, providing session security and system access control is rather complicated in comparison to a typical event driven desktop application. All users by default have the lowest permissions and belong to a “guest” user group this allows them to only access the login authentication screen and nothing else. The user is authenticated via a Secure HTTP (HTTPS) connection, which makes use of industry standard Transport Layer Security (TLS) cryptographic protocol in order to encrypt a connection end-to-end between client and server, this method of 2-way encryption is familiar to anyone who ever used online banking services. Once the secure connection is established and user’s login credentials are verified against the database, a session is created with session information being kept server side in memory for fast access and authentication of every single request, while a HTTPS cookies is sent in response and is stored by the user’s browser, the cookie contains a session hash, valid only for that

user and that session. The user's browser would subsequently store this cookie and send it along side every other request and can be used server-side in order to authenticate these requests. This session cookie is valid for a set time period or until a user closes the browser window or tab, whichever comes first. The use of HTTPS helps avoid any severe security breaches that can be caused by man-in-the middle or replay attacks which can be performed at the network level. The use of session cookies provides an authenticated session on top of a stateless protocol. Storing the session information server-side allows to keep control of the session data. User permission and user group membership data is stored within the database in a many-to-many relationship between the users and user group tables.

Required Software and configuration

The main third party software components which Juno uses and builds on top are:

- VMware virtualisation server
- OpenSUSE Linux
- MySQL relational database
- Nginx HTTP server
- PHP5

All of the above are described in detail in the previous chapter, with configuration details in the appendix.

Since the web applications main area of use would be within a local network, the virtual machine containing the OpenSUSE instance and all of the required software was given a local IP address of 192.168.1.130 with the application being available on <http://192.168.1.130/>, but of course it is possible to deploy the application over the public internet by changing the configuration of the virtual machine and the changing the Nginx configuration file to bind to the required IP address and port. Also the DNS system could be used to give the application a more memorable name, for example a

DNS server was setup on the local network to point the domain *juno.loc* to the required IP address, and from a usability perspective this makes it easier for the users to access the system instead of remembering an IP address.

The MySQL database configuration file was configured to run on a server with small amounts of memory, but this can be changed of course. The PHP configuration files were set to make use of small amounts of system memory as well since the deployment virtual machine was only allocated 256MB RAM in order to see how the system performs in a low resource environment. Another thing to note is the restriction placed on the PHP instances which only allow access to a small portion of the overall file system; this was done in order to increase system security.

User Interface Design

Nielsen notes that the most important aspects of web application usability are: content organisation, navigation and usability (95). As already mentioned in this chapter a prototype was developed which helped gather feedback about the UI early. A great amount of emphasis was placed on the User Interface for the application, due to the documented shortcomings of the legacy A-Track system in use by the industrial partner. Appendix 4 contains screenshots of the old A-Track interface which has many usability problems such as large amounts of wasted on screen space, harsh colour gradients making it impossible to read some text, large amounts of horizontal scrolling and inconsistent navigation. These are only some of the UI related issues of A-Track there were many functional issues recorded as well and taken into account during requirement gathering process.

The main objectives of the UI for Juno were: ease of use, clean design, clear navigation and more responsive interface. These objectives were achieved with the help of several web design techniques such as separation of content from style by using Cascading Style Sheets (CSS) and separation of behaviour from content by using JavaScript.



Figure 19 - Juno login screen (non HTTPS)

A relatively simple, clean and consistent interface was developed. The first screen any user is presented with is login, which authenticates and starts the user session. The rest of the application is split into clear sections by using a simple *tabview* list. In order to provide cross-browser compatibility and consistency the interface makes use of YUI (83) framework which contains a set of cross browser tested UI elements such as fonts, buttons, tabs and overlays. jQuery (16) JavaScript library was also on the client side of the application, in order to provide interactivity and handle user initiated events such as submitting a form, and most importantly allowing data to be send from the client by using asynchronous HTTP as previously discussed.

A minimum operating screen resolution is set at 800x600 pixels with all content resizable and able to make most of any screen size, unlike the older A-Track system which wasted a lot of screen space for no apparent reasons, causing discomfort to users.

Appendix 5 contains more key screenshots of the interface and all of the main functionality can be seen from these.

ERP Integration via BizTalk Server

The third major subsystem and aspect of this project is the integration of the Juno web application with a third party ERP system. The main requirement here revolves around flexibility and being able to connect and retrieve asset related information from a number of ERP systems, these software systems are used within enterprises in order to more efficiently manage resources and company information. The purpose of this section is to describe the design and implementation of a connection between OpenBravo and Juno, accomplished with the help of Microsoft BizTalk messaging middleware. Whereas the Driver Abstraction Layer discussed early concerns itself with retrieving data and filtering it into useful information to be used by Juno asset tracking application, the middleware integration allows Juno to retrieve asset and asset related financial information from existing ERP systems.

As discussed earlier in Chapter 2, Microsoft BizTalk is a message oriented middleware which delivers messages between multiple components. For this project Microsoft BizTalk 2006 R2 was chosen, BizTalk 2009 was also explored but decision was made not to use it due to lack of documentation and tools at the time, due to it being so new. BizTalk can be configured to receive messages on from various locations using “ports”, then process and map these messages, also perform any number of business rules on the messages and finally make the data available via a variety of formats such as web services, files and databases.

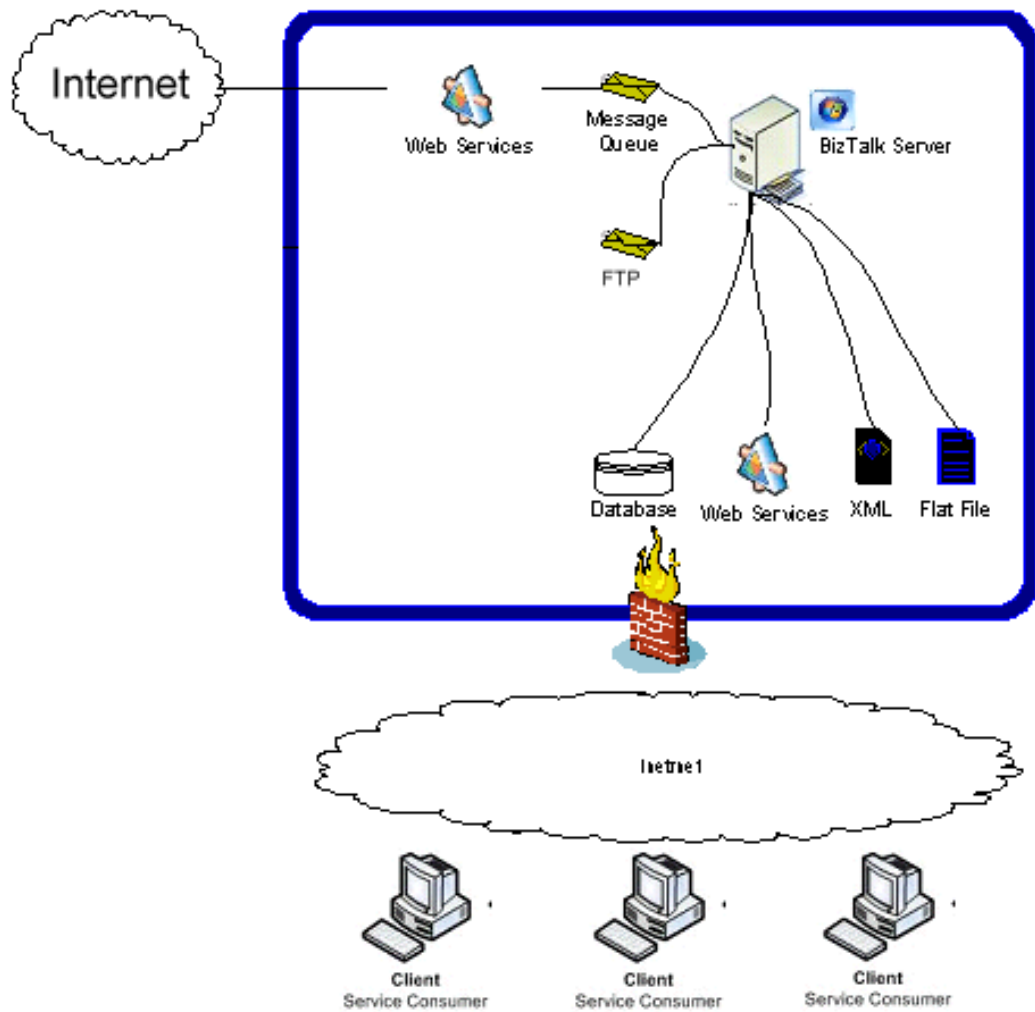


Figure 20 - Typical BizTalk middleware system overview

A BizTalk solution can be created and tested within Microsoft Visual Studio integrated development environment (IDE). BizTalk Server 2006 also provides a development and run-time environment for business process management (BPM) and automation. The system mainly concerns around sending and working with messages, which are usually triggered by any number of events. In the case of this project BizTalk automates the business process that begins when an initial request is sent from Juno, triggering an event which ends with the required information being retrieved from an ERP system, mapped and returned in a required format. The following series of steps occurs in order to retrieve asset information from the sample ERP system chosen:

1. An information request message is received via web service call from Juno.
2. The orchestration schedule (whose main job is to define the business rules to be applied) within BizTalk evaluates whether a message is valid, using the company’s business policies and requirements for approval criteria.
3. Once the request is approved by the orchestration, asset details are extracted from the OpenBravo’s PostgreSQL database using BizTalk 2006 SOAP adapter.
4. The message is then parsed using the correct “*mapper*” components, see screenshot **FIGURE 21 – MAPPER COMPONENT CONFIGURATION FOR OPENBRAVO ASSETS WITHIN BIZTALK SERVER 2006**
5. And stored within the “*MessageBox*” database within BizTalk
6. The resultant XML file containing the required and transformed asset and related asset financial information is then served within the response to a web service request.

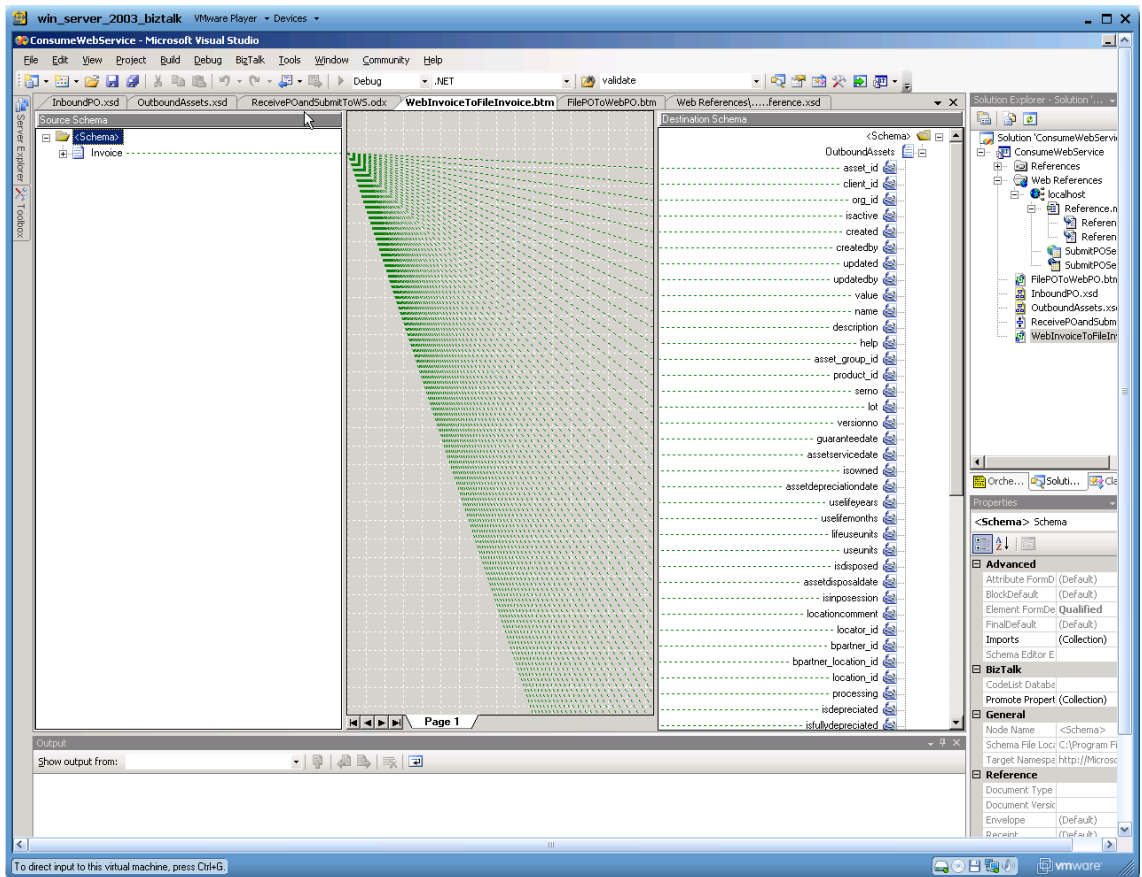


Figure 21 - Mapper component configuration for OpenBravo assets within BizTalk Server 2006

The orchestration schedule can be seen in Appendix 3 – BizTalk Orchestration Schedule.

Extra functionality was added to Juno as well in order to facilitate and initiate this process and to complete integration. One of the completed features of Juno is the asset registry import from comma separated text file; the functionality was based on the requirements provided by the industrial partner. A controller and view was created based on the file import functionality, except instead of an upload file box a simple button is added to the interface. Once this “Import from ERP system” button is clicked an event is fired, this event leads to a web service call to BizTalk which initiates the asset register request process as described in previous paragraph. The response from this web service call received from BizTalk contains an XML file which is then parsed and added to Juno. The difference between import from ERP system and import from file is minimal from the perspective of Juno web application. Thanks to the loose coupling of these systems, the details of the actual ERP asset retrieval implementation are hidden from Juno. Therefore it is possible to change and adapt the BizTalk implementation to retrieve asset information from other ERP system such as SAP and this would not affect the design of Juno.

The detailed iterative design process used in the creation of the BizTalk implementation is documented in Appendix 2 – BizTalk Integration

Development Techniques and Tools

During the development of the Juno software several important software engineering techniques were employed in order to keep the project on track and aid testing and development.

Version control software is an important tool commonly used in software development projects in order to keep track of the changes to the code and grant permissions to make changes to project members (96). Popular and commonly used open source

version control server software called Subversion (97) was used to provide version control for the project. Subversion was setup and configured to run as a service on a version control server (a virtualized server running within VMware). The development client machines used for coding would connect to this version control server and update or commit code to the project repository. Windows based clients had a GUI for Subversion installed called TortoiseSVN (97), this provides the client with a user interface to make it easier to work with Subversion, this interface exposes common functionality such as “commit”, “update”, “browse repository” and “compare versions”. This functionality has been of great help during the development stage.

In order to help avert any potential disasters, several backup scripts were coded and put in place in order to regularly backup the work in progress. These backup scripts would collect project files, data files and various configurations, then compress these, and finally upload via a secure connection to an offsite backup server. A proper backup procedure in place helps insure any large software development project against unforeseen events or user error.

Two free and open source integrated development environments (IDE's) were used for the majority of the coding and debugging tasks. These are:

- Eclipse for JAVA development
- Eclipse PDT, which is a variation of the Eclipse IDE specifically aimed at PHP development

Visual Studio 2005 from Microsoft was also used for BizTalk Server integration, since it has BizTalk solution templates.

Teamwork Breakdown

As already mentioned due to the scope of this project, the project was broken down to split the workload and tasks between the two main researchers, with one being the

author and the other being James O’Shaughnessy. Beside the two researchers the project was budgeted to include a third fulltime System Software Architect position, but no suitable candidate was found, instead Mr. Andrew Shields was brought in as consultant to help design and work on the Juno to OpenBravo integration via BizTalk middleware. The work which involved the two researchers and the consultant was more or less split along the lines of the 3 main project subsystems with some overlaps. The research and work performed by Researcher 1 - James O’Shaughnessy is outlined in detail within his thesis (*DESIGN OF A DISTRIBUTED RFID SYSTEM MODEL FOR ASSET TRACKING AND DEVELOPMENT OF THE REQUIRED SOFTWARE ABSTRACTION LAYER*), with James being on the project from the very start, his main tasks included the research into and testing RFID hardware, and also the design and development of the Reader Abstraction Layer used in retrieving data from readers. The design and iterative development performed by Consultant 1 – Andrew Shields is described in Appendix 2, and was exclusively within the BizTalk middleware integration side of the project, where asset and financial information is retrieved from an ERP system via flexible middleware solution.

The work performed on this project, by the author of this thesis include:

- The complete design and development of the Juno web application.
- Participating in the initial design of the Reader Abstraction layer.
- Participating in the initial design of the BizTalk integration sub-system.
- Integration of the completed Reader Abstraction and BizTalk integration sub-systems into the final completed implementation.
- Testing of the Juno web application sub-system
- Testing of the overall system, consisting of the three main components.

Chapter 5 – Testing and Security

Testing and Validation

Unit Testing

The development of Juno web application was done in a test driven manner. This involves creating unit tests and running components against the unit tests in order to detect any bugs early at the lowest level of granularity, such as a Class or a controller in the case of this project. The IEEE Standard for Software Unit Testing has the following to say about unit testing (98):

“Software unit testing is a process that includes the performance of test planning, the acquisition of a test set, and the measurement of a test unit against its requirements. Measuring entails the use of sample data to exercise the unit and the comparison of the unit's actual behaviour with its required behaviour as specified in the unit's requirements documentation.”

Another definition is provided by Elfriede Dustin (99):

“Unit testing is the process of exercising an individual portion of code, a component, to determine whether it functions properly.”

Unit tests allow to the smallest functional software components against a test case or a series of tests, this more than often helps in identifying and finding any bugs early, another advantage of unit tests is being able to retest components and verify that any changes elsewhere in the system don't result in a change to the expected or required output. Unit testing is a form of a bottom up approach to testing, where the smallest components are tested and verified, this makes it much easier to perform Integration tests and Regression Tests. Integration involves testing the result of a sum of several components while Regression Tests are used to refactor code, with Unit tests it's easier to identify issues and ensure the output remains the same when it comes to these two types of testing as well.

Within Juno unit testing involved writing and testing all of the Model classes and each controller was unit testing to ensure the outputs are as expected and exceptions are handled correctly and bubble up through to the main Exception handler. Unit testing the controllers is more of an Integration type of testing, since each controller could be working with multiple Models, but the concept and procedure involved in testing these is the same; write a test case, test against the required code, move on and repeat. Time and time again this technique has helped in finding and fixing software bugs early in the development process. The unit tests are bundled with the source code so any future work or addition to the code can be checked against the provided test cases.

Integration and Usability Testing

Usability testing was one of the more important tasks undertaken, since such an emphasis was placed by the industrial partner on the usability of such a system, and in the light of the bad UI design decisions made with the legacy asset tracking web application in use by the partner. Since the partner unfortunately pulled out of the project it was not possible to deploy and test the system with the end customers, this meant that usability testing was performed by asking several other postgraduate students to use the system for a short period of time, and collect the feedback, impressions and opinions from this test group. This technique is referred to as Hallway Usability Testing; the theory is that a small number of random people is indicative of cross-section of end users and was first described by Jakob Nielsen, who claims that 95% of usability problems can be discovered by this technique (100).

Integration testing was also carried out when putting together the three main sub-systems of the overall project. A number of issues were identified and fixed, mostly involving incorrect and inconsistent data formats being used.

Client-Side Testing and Debugging

While the server side code can be easily unit tested, and debugged using the appropriate IDE, debugging client side JavaScript and asynchronous HTTP requests is rather more complicated. Since a sizable proportion of the UI makes use of JavaScript and asynchronous requests, the testing and debugging of this code was performed with the help two free software tools; Microsoft Fiddler web debugger and Firebug plug-in for the Mozilla Firefox program. These provide several important features that were helpful in the development and testing process:

- Debugger allows for the stopping of the execution of a client side script at a breakpoint, and stepping through the code line by line, while examining variables.
- A HTTP proxy, which intercepts and records all HTTP requests and responses, this allows one to examine both the headers and the payload data.

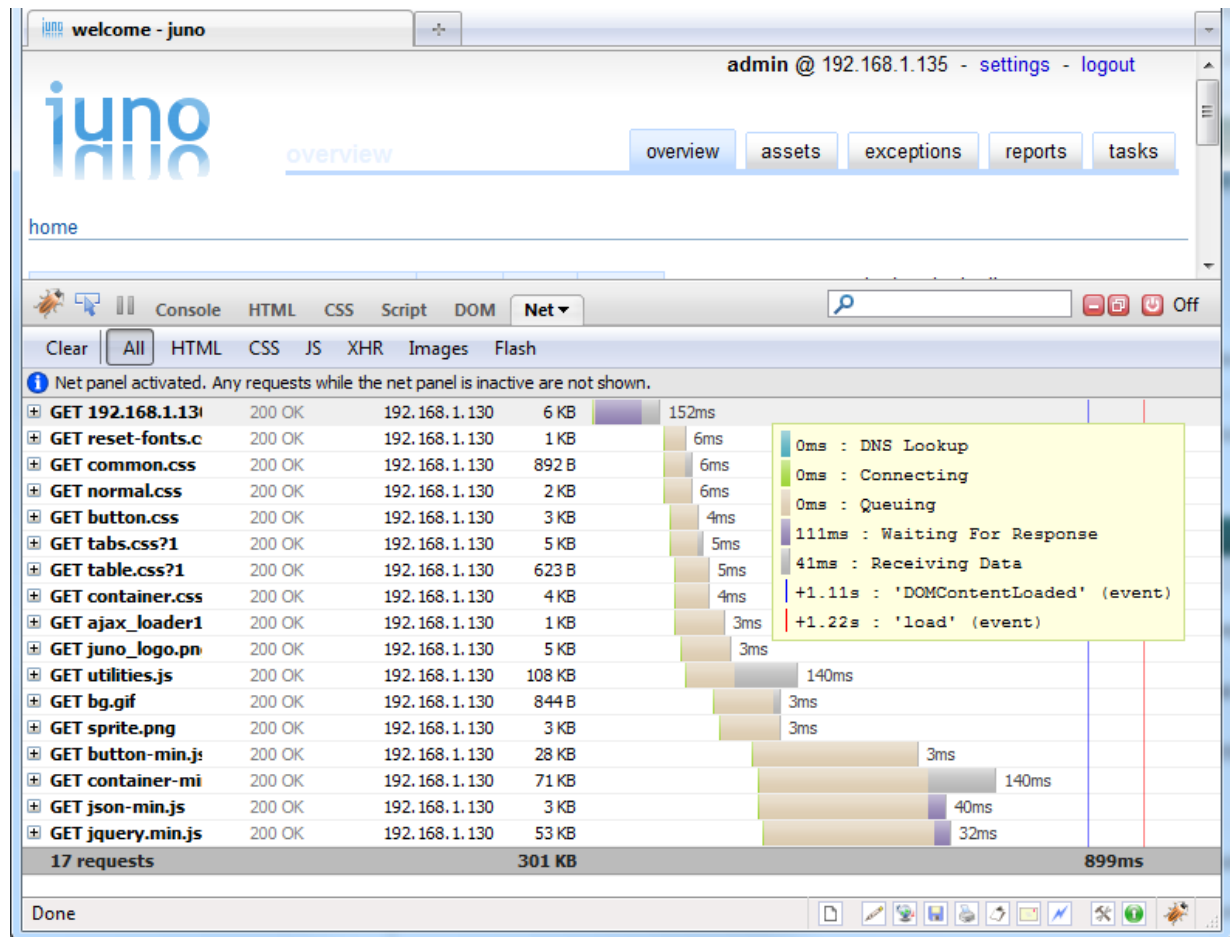


Figure 22 - Firebug Debugger, network connections overview of Juno main dashboard

Security

Since Juno is mostly a web based application that works with a company's asset registry security is paramount. Unlike desktop applications the client/server nature of web applications and use of HTTP protocol allows for a range of possible exploits, which the developers need to be aware of and be able to counteract. This section outlines some of the dangers and explores and describes the solutions as they were implemented within this project.

Web attacks are split into two types; application and infrastructure attacks.

Infrastructure attacks are aimed at the network infrastructure or the web server

software such as Apache web server, these include man-in-the middle and denial of service attacks. Application attacks are aimed at the web application itself, most common types include cross-site scripting and request forgery attacks. These attacks are important to be aware of as they are quite common when it comes to web applications and are preventable.

XSS

Cross-site scripting (XSS) is a common attack vector against web applications; it involves a user inputting malicious code into an input area in an application which in turn can be served to other users where this code (more than often client-side JavaScript) then executes. Dave Kleiman et al. write the following about XSS (101):

“The hacker uses the trusted Web site only as a conduit to perform the attack. The user is the intended victim, not the server. The server is merely the host, while the attack executes within the Web browser. Once an attacker has the thread of control in a user’s Web browser, he can do many nefarious acts such as account hijacking, keystroke recording, intranet hacking, history theft, and so on.”

XSS attacks sometimes are coupled with browser exploits to gain further access to a victim’s machine, with more modern and up to date browsers providing better security. All XSS attacks are possible due to a web application not properly filtering all its inputs or in some cases not filtering input at all, this is usually due to inexperienced developers who do not yet know that all inputs into the application especially arriving via HTTP requests must be filtered, validated and never trusted.

Every module in the Juno web application part is based on the principle of not trusting the user inputs and filtering and validating them thoroughly. For example numeric parameters received such as an “*asset_id*” are verified to be an integer and then verified that such an asset actually exists and accessible to the requesting user. Text input fields such as asset descriptions are filtered with special characters escaped, html and script cleaned out and special care is taken to ensure no malicious code can be encoded using the extended Unicode character set.

CSRF

Cross-site request forgery (CSRF) relies on a victim to initiate the attack rather than the attacker using an XSS exploit, in this manner the trust granted by the site to the user is abused. Chris Shiflett defines CSRF as (102):

“Attacks that attempt to forge HTTP requests, using the authorisation of the victim to bypass security checks”.

Since HTTP is a stateless protocol, applications track users via the use of session cookies, these session cookies are usually issued on authentication. CSRF attacks exploit this trust of the user’s session, usually by embedding an image or object in a page, when the user’s browser loads this object (which may also be hidden to the user) it causes the browser to fire a HTTP GET request to the server. A web application that accepts GET requests and then changes the application state or edits some data based on session contained in a cookie would be vulnerable to CSRF attacks. Hence it is important that GET requests are only used to return read-only data, while POST requests are used for operations that change the state of the application or work with any of the models.

Four techniques were used within the Juno web application to prevent CSRF, these are:

1. Using HTTP POST requests for all operations that involve a change of state or updating the database.
2. “Locking” the forms by using a randomly generated shared secret hash, with one half stored on the server and the other half embedded with the form, this hashed string would be sent along with any form submission and compared to the value stored in database.
3. The third method used in conjunction with above two, involves encoding the requests as JSON string before sending to the server, with the replies also being JSON encoded.

4. And finally having the sessions expire once the user closes the page, this means the user doesn't not persistently stay logged into the application and hence no way to perform a CSRF attack

SQL Injection

Yet another common attack vector against web applications tries to exploit the lack of filtering of the incoming input received via GET or POST requests, and then using the parameters of the request within a database query. These can be easily avoided by filtering input parameters, ensuring that all SQL queries use prepared statements. Prepared statements are where a programmer uses variable placeholders instead of using variables directly to construct a query. These statements also define the data type of the variable to be used, and allows for faster queries since the database engine can pre-compile and security check the queries that make use of prepared statements. Within Juno prepared statements were used by ensuring all database queries are handled by the PHP Data Objects (PDO) library.

Chapter 6 – Conclusions and Future Work

Conclusions

This principal aim of this research was to investigate and develop a software model of integration between networked RFID systems, associated middleware and existing enterprise information management systems. The final outcome of this research is the successful development of the Juno fixed asset tracking system, that utilises a wide range of technologies such as; RFID, databases, web-application and enterprise messaging middleware in order to achieve the original objectives of the project and meet the requirements of the industrial partner. The development of this software system has benefited greatly from structured development methodologies, and the use of open source technologies to address the system requirements.

The industrial sponsors fell on hard times during an economic recession, and had to abandon the project, but fortunately there were sufficient funds to complete the research. This development meant that the facilities were not present to test the system in a real world working environment, such as a warehouse, and all system testing had to be confined to laboratory.

From testing and validation of the Juno asset tracking system the following conclusions can be drawn:

- The use of the client-server paradigm allows for the application to be accessed via a web browser, this brings several advantages:
 - The clients don't have to update the software with every change as any updates get done on the server.
 - No installation is required by end user; just knowing the URL and appropriate authentication details is enough to access and work with the application.

- The application can be accessed on a wide range of platforms and devices within the intranet, with the possibility of (security considerations aside) of access via the internet.
- This helps to make it easier to maintain and configure the application at the server.
- The time taken to train end users to be able to use the Juno system is reduced; since the asset tracking process can be performed via the browser anyone who is computer literate will be familiar with it with a little training. This also reduces the costs of running the system for the customer.
- A clear, clean and consistent User Interface allows for more efficient use of the application and helps perform tasks faster.
- Using a modular and component based structure which follows the Model View Controller design pattern allows for rapid addition of new features and helps to easily maintain and extend existing functionality.
- The use of Microsoft BizTalk server allows Juno to interact with a wide range of enterprise information systems such as OpenBravo and SAP, BizTalk provides connectivity and messaging allowing data to be retrieved and then transformed between an extensive list of formats such as web services, plain text, file based, SQL access , etc. This allows the integration of the RFID asset tracking system with a range of financial systems that are in common use in organisations.
- The system is designed with the RFID tags as being the main data source for the asset tracking register, but the code has been designed to be modular and flexible enough to use with other wireless communication technologies available now or in future, some possible alternatives include Bluetooth, ZigBee, GSM and GPRS. Some of these technologies are already employed in various business scenarios; for example tracking delivery Trucks via GSM network.

All these features are achieved at reduced development costs via the use of open source software and modern agile software development techniques. The project was not 100% based on open source software, with BizTalk software from Microsoft being used to assist in the middleware integration process, the comparison of the proprietary versus open software models helped in gaining an understanding of the advantages and disadvantages of each.

The overall system was designed and developed to contain 3 loosely coupled subsystems, these have previously described in Chapter 4, and consist of:

- Connectivity with RFID readers, gathering tag read data from various active and passive readers which communicate in a variety of data formats, then filtering and processing this data into useful information.
- Web based asset tracking application, for system users to interface and interact with the asset register and be able to work with related information.
- Connectivity via BizTalk middleware to an ERP system in order to retrieve asset information.

Conclusions and observations can be made regarding each one of the above subsystems.

The reader driver abstraction software can be used to provide connectivity to various RFID readers. As discussed and illustrated earlier on in this thesis, the range of passive RFID technology is too short to be of much practical use within the asset tracking scenario proposed by the industrial partner. Active RFID technology shows more promise and the system works relatively well with these tags, but several cons of this technology need to be highlighted again in the conclusion:

- Current high prices for active tags make it only cost-effective to tag high value assets. But prices are expected to fall as new miniaturization and circuit printing techniques are developed.

- Relative current large size of the active tags themselves, mainly due to onboard battery size, makes it impractical to attach the tags to some small assets. But as mentioned in Chapter 2, new battery printing technology could lead to smaller and cheaper active tags.

Since the abstraction layer abstracts away the interaction and information flow down to the most common parameters, other asset tracking technologies can be “plugged in”. With the help of plug-ins it then becomes possible to use other technologies such as barcode scanners or any new radio tracking technologies such as RuBee, provided these technologies allow for identification of assets via a unique identifier and a software driver can be written.

The Juno asset tracking web application constitutes the second main subsystem for this project, and allows the users to work with the asset register. This part of the overall project allows for the most flexibility and can be easily expanded and built upon to fulfil other tracking requirement that may arise, beside the requirements gathered from the industrial partner. The code itself is highly modular and makes use of design patterns discussed in Chapter 2 in order to provide much coveted aspects of any software system; these being readability, testability, modularity, extensibility, low coupling and high cohesion. All of the requirements from the industrial partner have been met and in some cases exceeded, and the new Juno system overall performs more reliably, offers more features and addresses many of the shortcomings of the legacy A-Track software in use by the industrial partner.

The third main aspect of the overall project is the integration via BizTalk to OpenBravo (a sample ERP system chosen for this project). This subsystem of the of overall project allows for integration with third party ERP software in order to retrieve and synchronise the asset information contained in these management information systems. Using the methods learned, such as knowing how to orchestrate and map dataflow from the sample ERP system to Juno, it is possible to modify this portion of the project to integrate in a similar manner with other ERP systems such as SAP. Unfortunately due to excessive costs of software such as SAP ERP and limited project

budget, OpenBravo which is an open source ERP system was chosen to demonstrate the ability to bridge these software systems with Juno database.

There is one important final observation and conclusion to be made. The industrial partner's use case and requirements, call for RFID readers and antennae to be positioned in fixed locations, in order to scan a volume of space, which contains tagged assets at large installations such as a warehouse or manufacturing facility. The author believes that in practice monitoring a fixed area for exceptions, which would occur if a tagged item leaves this RFID illuminated area, is not an optimal procedure for asset tracking. Several issues arise in such a setup containing fixed readers, these are:

- Black spots occur due to interference or presence of metallic or liquid bodies.
- As explained earlier the read range of RFID readers is rather short, hence in order to read a large area a number of expensive readers would be required.

Therefore, it is proposed that in a manner similar to other existing RFID asset tracking systems, emphasis is placed on the use of mobile, handheld readers in order to scan an area. Or alternatively the system would make use of a mixture of fixed readers in high traffic areas, and handheld readers elsewhere. This proposal would require changes to the way exceptions are recorded by Juno and in the way these exceptions are presented to the user. No changes would need to be made to the Driver Abstraction layer or the BizTalk integration sub system.

Future Work

With the completion of this project and with the knowledge gathered it is possible to list and make recommendations on any future work and research.

On a commercial side GMIT and Enterprise Ireland can continue on with commercialisation of the developed system, and build on the knowledge gathered in

order to deliver a successful product based on a blend of hardware and software. This product could be also backed by services such as support and training.

Commercialisation would involve carrying out mainly non engineering tasks such as:

- Market research and marketing.
- Product packaging and service delivery.
- Drafting business and training plans.
- Providing detailed documentation.
- Customising the interface to suit the needs of any customers, the Juno web application is designed and is capable of being extended and the user interface is highly customisable due to the modular nature of the application.

A business plan centred on the developed asset tracking product can be drawn up, with profit opportunities arising from various services:

- Installation of the tracking system
- Training on how to use such a system
- Service and maintenance contracts
- Providing customisation and custom extensions.

Another interesting alternative commercial idea with possibility for rapid growth, involves opening up this system and releasing it as an open source project, then earning profits from consulting services. An example of this could be seen in some of the software used to illustrate the concept in this project such as OpenBravo.

OpenBravo is released as an open source ERP system, this has allowed OpenBravo S.L, the company behind OpenBravo and who performed the original research and development of the system, to:

- Rapidly gain market share
- Compete with software system giants such as SAP and Microsoft in a crowded sector of the software industry.
- Incorporate ideas and feedback from users.

- Leverage the Open Source development model in order to improve the product.
- Gain wide product recognition.
- Provide consultancy services for specialised product extensions and support.

OpenBravo S.L has recently received \$12 million in venture capital, and has grown to over 100 employees in 3 years (103). OpenBravo is not the only company to follow such a business practice of using open source to gain market share and generate profits, MySQL database used in this project was originally developed by MySQL AB and then sold to Sun Microsystems for \$1 billion (104), who recently have been acquired by Oracle Corporation.

On the technical and engineering aspects of the project some areas for any future work and research are:

- Explore the use of BizTalk RFID in place of the Reader Abstraction layer and plug-ins developed for this project. BizTalk RFID promises the potential to achieve similar aim of providing abstraction for multiple readers, but this Microsoft software is not Open Source and is costly.
- Provide an API for Juno; hence other systems can interact with the asset register data.
- The use of new technologies that show promise such as RuBee in order to replace active RFID.
- Location triangulation, using data collected from 3 or more readers it should be possible to identify the location of tagged item and then display the coordinates visually to the end user using in browser mapping technologies such as Google Maps, which can be modified to work with custom maps such as business premises. Unfortunately due to the short range of passive RFID, the 4 passive readers acquired for this project were useless in a triangulation scenario, and only 2 active readers were acquired of which one reader became defective.

- The user interface could be easily adapted, by creating separate View files on top of the same Controllers, in order to work on latest generation of lower resolution mobile devices such as iPhone's and ultra-portable tablet computers.

The web application framework developed for this project is flexible enough to be used in any other projects within GMIT that require a web application interface. This can help in code reuse and prevent someone else reinventing the wheel. Some of the research projects at GMIT such as the Data Acquisition, Web Based Learning (DAQ-WBL) project at GMIT's Centre for Integration of Sustainable Energy Technologies could make use of the web application framework developed for Juno. Their project involves collecting and processing data from a network of environmental sensors and displaying the information in order to facilitate learning. DQL-WBL project has similarities with this project, in the way that data is collected from sensors (RFID readers and tags), the data is processed and stored, and finally a database backed web application is used to interact with the information.

Bibliography

1. **Dierkes, E. Fleisch and M.** Ubiquitous Computing: Why Auto-ID is the Logical Next Step in Enterprise Automation . *autoidlabs.org*. [Online] 1 October 2003. [Cited: 7 March 2009.] <http://www.autoidlabs.org/uploads/media/STG-AUTOID-WH004.pdf>.
2. **Oliver Günther, Wolfhard Kletti, Uwe Kubach.** *RFID in Manufacturing*. Berlin : Springer, 2008. 3540764534.
3. **Bill Clover, Himanshu Bhatt.** *RFID Essentials*. Sebastopol, CA : O'Reilly Media, 2006. 0596009445.
4. The Healthcare Equation - RFID Journal. *rfidjournal.com*. [Online] 2 May 2005. [Cited: 31 March 2009.] <http://www.rfidjournal.com/article/view/1563/1/2>.
5. *A Prescription for Pharmaceuticals*. **Wasserman, Elizabeth.** s.l. : RFID Journal , 2005, Vol. May/June 2005.
6. **Charles C. Poirier, Duncan McCollum.** *RFID strategic implementation and ROI: a practical roadmap to success*. s.l. : J. Ross Publishing, 2006. 1932159479.
7. **IBM Ireland.** Asset management and the benefits of RFID technology. *ibm.com*. [Online] November 2005. [Cited: 09 March 2009.] <http://www-03.ibm.com/solutions/sensors/us/detail/resource/P122350H75637X99.html>.
8. **US Federal Law.** Sarbanes-Oxley Act, also known as the Public Company Accounting Reform and Investor Protection Act of 2002. *Pub. L. No. 107-204, 116 Stat. 745*. July : 24, 24 July 2002.
9. **Tarantino, Anthony.** *Manager's guide to compliance*. s.l. : John Wiley and Sons, 2006. 0471792578.
10. **Jackson, Peggy M.** *Sarbanes-Oxley for nonprofit boards*. s.l. : John Wiley and Son, 2006. 0471790370.

11. **Walker, Joshua.** *Vendors Race To Fill A New Void: RFID Middleware.* s.l. : Forrester, 26 January 2004.
12. **Free Software Foundation, Inc.** The GNU General Public License - GNU Project - Free Software Foundation (FSF). *gnu.org.* [Online] 29 June 2007. [Cited: 11 March 2009.] <http://www.gnu.org/copyleft/gpl.html>.
13. **Board, OSI.** The BSD License. *opensource.org.* [Online] 21 October 2006. [Cited: 9 March 2009.] <http://www.opensource.org/licenses/bsd-license.php>.
14. **Google, Inc.** Google and Open Source - Google Code. *code.google.com.* [Online] <http://code.google.com/opensource/>.
15. **SourceForge, Inc.** SourceForge.net: Open Source Software. *sourceforge.net.* [Online] <http://sourceforge.net/>.
16. jQuery: The Write Less, Do More, JavaScript Library. *jquery.com.* [Online] <http://jquery.com/>.
17. **James D. Foley, Andries Van Dam, Steven K. Feiner, John F. Hughes.** *Computer graphics: principles and practice.* s.l. : Addison-Wesley, 1995. 0201848406.
18. **Power, Daniel J.** *Decision support systems: concepts and resources for managers.* s.l. : Greenwood Publishing Group, 2002. 156720497X.
19. **Reilly, Edwin D.** *Concise encyclopedia of computer science.* s.l. : John Wiley and Sons, 2004. 0470090952.
20. **Frederick P. Brooks, Jr.** *The Mythical Man-Month, Essays on Software Engineering, Anniversary Edition.* MA : Addison-Wesley Pub. Co, 1995. 0201835959.
21. *Communication by Means of Reflected Power.* **Stockman, Harry.** s.l. : Proceedings of the IRE 35, 1948. pp. 1196–1204.
22. **V. Daniel Hunt, Albert Puglia, Mike Puglia.** *RFID: a guide to radio frequency identification.* Hoboken, N.J. : Wiley-Interscience, 2007. 0470107642.

23. **Hosang Jung, F. Frank Chen, Bongju Jeong.** *Trends in supply chain design and management*. s.l. : Springer, 2007. 1846286069.
24. **Roussos, George.** *Networked RFID: Systems, Software and Services*. London : Springer-Verlag, 2008. 9781848001527.
25. **Philips Semiconductors.** *Item-Level Visibility in the Pharmaceutical Supply Chain, A Comparison of HF and UHF*. July 2004.
26. **Ari Juels, Ravikanth Pappu.** Squealing Euros: Privacy Protection in RFID-Enabled Banknotes. [book auth.] Rebecca N. Wright. *Financial Cryptography: 7th International Conference, FC 2003, Guadeloupe, French West Indies, January 27-30, 2003 : Revised Papers*. s.l. : Springer, 2003.
27. **Finkenzeller, Klaus.** *RFID Handbook, Fundamentals and Applications in Contactless Smart Cards and Identification, 2nd Edition*. Chichester : John Wiley & Sons Ltd, 2003. 0470844027.
28. **Lu Yan, Yan Zhang, Laurence T. Yang, Huansheng Ning.** *The Internet of Things: From RFID to the Next-Generation Pervasive Networked Systems*. Boca Raton, FL : Auerbach Publications, 2008. 1420052810 .
29. **E.Sarma, Sanjay.** *Towards the five-cent Tag*. s.l. : MIT Auto ID Center, 2001. MIT-AUTOID-WH-006.
30. **Erick C. Jones, Christopher A. Chung.** *RFID In Logistics, A Practical Introduction*. Boca Raton, FL : Taylor & Francis Group, 2008. 0849385261.
31. **Obal, Philip.** *Glossary of supply chain terminology*. s.l. : Industrial Data & Information Inc, 2003 . 0966934539.
32. **Poslad, Stefan.** *Ubiquitous Computing: Smart Devices, Environments and Interactions*. s.l. : John Wiley & Sons Ltd, 2009. 0470035609.
33. **Want, Roy.** An Introduction to RFID Technology. *IEEE Pervasive Computing*. January, 2006, Vol. 5, 1.

34. —. *RFID Explained: A Primer on Radio Frequency Identification Technologies*. s.l. : Morgan & Claypool, 2006. 1598291084.
35. **Connie K. Haley, Lynne A. Jacobsen, Shai Robkin**. *Radio Frequency Identification Handbook for Librarians*. s.l. : Libraries Unlimited, 2007. 1591583713.
36. **Dobkin, Daniel Mark**. *The RF in RFID: Passive UHF RFID in Practice*. s.l. : Newnes, 2007. 0750682094.
37. **Fraunhofer Press**. Research News July 2009 - Printable batteries. *fraunhofer.de*. [Online] July 2009. [Cited: 1 September 2009.]
http://www.fraunhofer.de/en/Images/rn7_FERTIG_tcm63-13052.pdf.
38. **ScienceDaily LLC**. Inexpensive Thin Printable Batteries Developed. *sciencedaily.com*. [Online] ScienceDaily LLC, 5 July 2009. [Cited: 2 September 2009.]
<http://www.sciencedaily.com/releases/2009/07/090702080358.htm>.
39. **Kekre, Sunder**. Forging New Links for a 21st-Century Supply Network. *Tepper Magazine*. Fall, 2004.
40. **Keunwoo Rhee, Jin Kwak, Wan S. Yi, Chanho Park**. Efficient RFID Authentication Protocol for Minimizing RFID Tag Computation. [book auth.] Daniel Howard, Dominik Slezak Marcin S. Szczuka. *Advances in Hybrid Information Technology*. Berlin : Springer, 2007, Vol. 4413.
41. **Hans-Dietrich Haasis, Hans-Jörg Kreowski, Bernd Scholz-Reiter**. Dynamics in Logistics: First International Conference, LDIC 2007, Bremen, Germany, August 2007. Proceedings. [book auth.] Bonghee Hong, Wooseok Ryu, Sungwoo Ahn Ashad Kabir. *LIT Middleware: Design and Implementation of RFID Middleware Based on the EPC Network Architecture*. Berlin : Springer, 2008.
42. **EPCglobal**. EPCglobal ALE. *epcglobalinc.org*. [Online] GS1, 13 March 2009. [Cited: 3 May 2009.] <http://www.epcglobalinc.org/standards/ale>.
43. *A Mobile Agent Based InTransit Goods Tracking System*. **Feng Li, Ying Wei**. Qingdao, China : Springer, 2007. Advanced Intelligent Computing Theories and Applications - With Aspects of

Theoretical and Methodological Issues. Vols. Lecture Notes in Computer Science, Vol. 4681 .
9783540741701.

44. **Weiwei Sun, Xuhong Tian, Minjie Jiang.** Research of RFID Middleware in Precision Feeding System of Breeder Swine. [book auth.] Daoliang Li. *Computer and Computing Technologies in Agriculture: First Ifip Wg 12.5 International Conference on Computer and Computing Technologies in Agriculture*. Wuyishan, China : Springer, 2007.

45. **Mark Beckner, Mark Simms, Ram Ventatesh.** *Pro Rfid in Biztalk Server 2006 R3*. Berlin : Springer, 2009. 1430218371.

46. **IBM Redbooks.** *Developing Php Applications for IBM Data Servers*. s.l. : Vervante, 2006. 0738497398.

47. *Asset Tracking on the International Space Station Using Global SAW Tag RFID Technology.* **Paul Brown, Paul Hartmann, Amy Schellhase, Annie Powers, Tim Brown.** 31, Houston, Texas : IEEE, 2007, Vol. 28. 10.1109/ULTSYM.2007.31.

48. **Dimitrios Buhalis, Carlos Costa.** *Tourism business frontiers*. s.l. : Butterworth-Heinemann, 2006. 0750663774.

49. **Polishuk, Paul.** *RFID Monthly Newsletter*. 2007, Vol. 4, 3.

50. *Bokode: Imperceptible Visual tags for Camera Based Interaction from a Distance.* **Ankit Mohan, Grace Woo, Shinsaku Hiuray, Quinn Smithwick, Ramesh Raskar.** New Orleans : MIT, 2009. http://web.media.mit.edu/~ankit/bokode/bokode_sig09.pdf.

51. **John K. Stevens.** First Meeting Of Working Ggroup for IEEE RuBee. *standards.ieee.org*. [Online] IEEE, 27 January 2007. [Cited: 28 August 2009.]
http://standards.ieee.org/announcements/pr_P1902.1_wgmtg.html.

52. **Roberti, Mark.** Perfect Alternatives to RFID? *rfidjournal.com*. [Online] RFID Journal LLC, 24 July 2006. [Cited: 28 August 2009.]
<http://www.rfidjournal.com/article/articleview/2505/1/128/>.

53. **Malik, Ajay.** *RTLS for Dummies*. s.l. : Wiley, 2009. 047039868X.

54. *An Introduction to RuBee (IEEE P1902.1) and Its Use in Real-Time Visibility Networks*. **John K. Stevens**. Atlanta : s.n., 2006. Active RFID Summit.
http://www.idtechex.com/events/presentations/an_introduction_to_rubee_ieee_p1902_1_and_its_use_in_real_time_visibility_networks_000636.asp.
55. *An overview of passive RFID*. **Chawla, V. Dong Sam Ha**. 9, Toronto : IEEE, 2007, IEEE Communications Magazine, Vol. 45. 10.1109/MCOM.2007.4342873.
56. *Application-oriented wireless sensor network communication protocols and hardware platforms: A survey*. **Zhongmin Pei, Zhidong Deng, Bo Yang, Xiaoliang Cheng**. Chengdu : IEE, 2008. ICIT 2008. IEEE International Conference on Industrial Technology.
10.1109/ICIT.2008.4608532.
57. **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal**. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. s.l. : John Wiley & Sons, 1996. 0471958697.
58. **Fowler, Martin**. *Patterns of Enterprise Application Architecture*. Addison Wesley : s.n., 2002. 0321127420.
59. **Thomas J. Shelford, Gregory A. Remillard**. *Real Web project management*. s.l. : Addison-Wesley, 2002. 0321112555.
60. **Lecky-Thompson, Heow Eide-Goodman, Steven D. Nowicki, Alec Cove**. *Professional PHP5*. s.l. : John Wiley and Sons, 2004. 0764572822.
61. **Sweat, Jason E**. *PHP Architect's Guide to PHP Design Patterns*. s.l. : Marco Tabini & Associates, 2005. 0973589825.
62. Zend Framework. *framework.zend.com*. [Online] Zend Technologies Ltd. [Cited: 15 June 2009.] <http://framework.zend.com/>.
63. **Zandstra, Matt**. *PHP Objects, Patterns, and Practice, Second Edition*. s.l. : Apress, 2007. 1590599098.

64. **Deepak Alur, John Crupi, Dan Malks.** *Core J2EE Patterns: Best Practices and Design Strategies.* s.l. : Pearson Education, 2001. 0130648841.
65. **Dean Leffingwell, Don Widrig.** *Managing software requirements: a use case approach.* s.l. : Addison-Wesley, 2003. 032112247X.
66. **Sommerville, Ian.** *Software engineering, 8th edition.* s.l. : Pearson Education, 2007. 0321313798.
67. **Doug Rosenberg, Matt Stephens.** *Use Case Driven Object Modeling with UML: Theory and Practice.* s.l. : Apress, 2007. 1590597745.
68. **Petersen, Richard.** *Linux: the complete reference.* s.l. : McGraw-Hill Professional, 2007. 007149247X.
69. **Thomas, Keir.** *Beginning SUSE Linux: from novice to professional.* s.l. : Apress, 2005. 1590594584.
70. —. *Beginning Ubuntu Linux; From Novice to Professional.* s.l. : Apress, 2006. 1590596277.
71. **Yaghmour, Karim.** *Building embedded Linux systems.* s.l. : O'Reilly, 2003. 059600222X.
72. **Petersen, Richard.** *Fedora Core 7 & Red Hat Enterprise Linux.* s.l. : McGraw-Hill Professional, 2007. 0071486429.
73. **Michael McCallister, Mike McCallister.** *SUSE Linux 10 unleashed.* s.l. : Sams Publishing, 2005. 0672327260.
74. **Hammersley, Eric.** *Professional VMware server.* s.l. : John Wiley and Sons, 2006. 0470079886.
75. **Kofler, Michael.** *The Definitive Guide to MySQL5, Third Edition.* s.l. : Apress, 2005. 1590595351.
76. **Luke Welling, Laura Thomson.** *MySQL Tutorial.* s.l. : Sams Publishing, 2003. 0672325845.

77. **Nginx Community.** nginx, mainpage. *wiki.nginx.org*. [Online] 8 April 2009. [Cited: 6 June 2009.] <http://wiki.nginx.org/Main>.
78. **Holzner, Steven.** *PHP, The Complete Reference*. s.l. : McGraw Hill Professional, 2007. 0071508546.
79. **Lerdorf, Rasmus.** *Programming PHP*. s.l. : O'Reilly, 2006. 0596006810.
80. PHP: Hypertext Preprocessor. *php.net*. [Online] PHP Group. [Cited: 15 June 2009.] <http://php.net/>.
81. **Flanagan, David.** *JavaScript: The Definitive Guide, Fifth Edition*. s.l. : O'Reilly, 2006. 0596101996.
82. **Zakas, Nicholas C.** *Professional Javascript for Web Developers*. s.l. : John Wiley and Sons, 2009. 047022780X.
83. **Yahoo! Inc.** The Yahoo! User Interface Library (YUI). *developer.yahoo.com*. [Online] Yahoo!, 2009. [Cited: 19 June 2009.] <http://developer.yahoo.com/yui/>.
84. **York, Richard.** *Beginning Javascript Development With JQuery*. s.l. : John Wiley and Sons, 2009. 0470227796.
85. **Glenn, Godfrey.** *Enterprise Resource Planning 100 Success Secrets - 100 Most Asked Questions*. s.l. : Lulu, 2008. 0980497183.
86. **Gruman, Galen.** Is Open Source the Answer to ERP? *CIO*. February, 2007, Vol. 20, 9.
87. **McConnachie, Dahna.** Openbravo executives open up on the ERP solution. *pcworld.idg.com.au*. [Online] PC World, 25 October 2007. [Cited: 22 June 2009.] http://www.pcworld.idg.com.au/article/203541/openbravo_executives_open_up_erp_solution.
88. **Nicolás Serrano, José María Sarriegi.** Open Source Software ERPs: A New Alternative for an Old Need. *IEEE Software*. May/June, 2006, Vol. 23, 3.

89. **Object Management Group, Inc.** Object Management Group - UML. *uml.org*. [Online] OMG, 08 January 2009. [Cited: 2 July 2009.] <http://www.uml.org/>.
90. **Fowler, Martin.** *UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition*. s.l. : Addison-Wesley, 2004. 0321193687.
91. **Leaver, Sharyn.** Evaluating RFID Middleware, Picking The Right Solution For Integrating RFID Data Into Business Applications. *forrester.com*. [Online] 13 August 2004. <http://www.forrester.com/Research/Document/Excerpt/0,7211,34390,00.html>.
92. **EPC Global.** Reader Protocol Standard, Version 1.1. *epcglobalus.org*. [Online] 21 June 2006. [Cited: 4 July 2009.] http://www.epcglobalus.org/dnn_epcus/KnowledgeBase/Browse/tabid/277/DMXModule/706/Command/Core_Download/Default.aspx?EntryId=836.
93. *RFIDPROBUS: A universal RFID reader communication protocol.* **Yuanxin Ouyang, Qiao Ren, Ting Zhang, Jiuyue Hao, Zhang Xiong.** s.l. : IEEE, 2008. 10.1109/ICDEW.2008.4498294.
94. **Lauesen, Soren.** *User Interface Design: a Software Engineering Perspective*. s.l. : Pearson Education, 2005. 0321181433.
95. **Nielsen, Jakob.** *Designing Web Usability*. s.l. : New Riders, 2000. 156205810X.
96. *Software Support Tools and Experimental Work.* **Mockus, Audris.** Dagstuhl Castle, Germany : Springer, 26-30 June 2006, Lecture notes in Computer Science, Vol. 4336, p. 2006. 354071300X.
97. Subversion. *subversion.tigris.org*. [Online] CollabNet, Inc. [Cited: 24 August 2009.] <http://subversion.tigris.org/>.
98. **Software Engineering Technical Committee of the IEEE Computer Society, USA.** *IEEE Standard for Software Unit Testing (ANSI / IEEE Std 1008-1987)*. s.l. : IEEE , 1986. 0738104000.
99. **Dustin, Elfriede.** *Effective Software Testing: 50 Specific Ways to Improve Your Testing*. s.l. : Addison-Wesley Professional, 2002. 0201794292.

100. **Nielsen, Jakob.** *Usability Engineering*. s.l. : Morgan Kaufmann, 1993. 0125184069.

101. **Dave Kleiman, Timothy Clinton.** *The official CHFI study guide (Exam 312-49): for computer hacking forensic investigator*. s.l. : Syngress, 2007. 1597491977.

102. **Shiflett, Chris.** *HTTP Developer's Handbook*. s.l. : Sams, 2003. 0672324547.

103. **O'Gara, Maureen.** Openbravo Picks Up \$12m Second Round. *sys-con.com*. [Online] SYS-CON Media, 23 May 2008. [Cited: 3 September 2009.] <http://opensource.sys-con.com/node/575090>.

104. **Richards, Jonathan.** Sun buys MySQL for \$1 billion. *business.timesonline.co.uk*. [Online] Times Newspapers Ltd, 16 January 2008. [Cited: 3 September 2009.] http://business.timesonline.co.uk/tol/business/industry_sectors/technology/article3199272.ece.

Glossary

ACID

ACID is feature of database systems which provide; Atomicity, Consistency, Isolation and Durability that guarantee that transactions complete correctly.

AJAX

Asynchronous JavaScript and XML, a group of interrelated client side web application techniques which includes asynchronous HTTP communication via JavaScript, used to provide interactive rich web applications.

ALE

Application Level Events are a specification from GS1 EPCglobal designed to provide abstraction and facilitate interaction with RFID middleware systems.

API

Application Programming Interface is a collection of methods, hooks and data structures within software libraries and systems that allow for integration with other components.

BSD License

Berkeley Software Distribution, a form of permissive open source license

ERP

Enterprise Resource Planning is an enterprise wide information system used to aid the completion of business processes such as financial, supply chain and human resource management.

Fast CGI

Fast CGI is a newer protocol superseding the older Common Gateway Protocol (CGI) which is used as an interface between interactive applications and web servers.

Fixed Asset Tracking

This is an accounting process of tracking assets within the company for the purposes of financial accounting, theft prevention and preventative maintenance.

GNU

Self-referentially, short for "GNU's not UNIX", a UNIX-compatible software system developed by the Free Software Foundation (FSF). The philosophy behind GNU is to produce software that is non-proprietary. Anyone can download, modify and redistribute GNU software. The only restriction is that they cannot limit further redistribution. The GNU project was started in 1983 by Richard Stallman at the Massachusetts Institute of Technology.

GPL

General Public License. This license accompanies some open source software that details how the software and its accompanying source code can be freely copied, distributed and modified. The most widespread use of GPL in reference to the GNU GPL, which is commonly abbreviated simply as GPL when it is understood that the term refers to the GNU GPL. One of the basic tenets of the GPL is that anyone who acquires the material must make it available to anyone else under the same licensing agreement.

HTTP

Hyper text transfer protocol, an application level protocol which forms the foundation and allows access to inter linked documents on the World Wide Web. It specifies how web browsers request and update documents.

JSON

JavaScript Object Notation is a text based, human readable computer data interchange format (RFC 4627) for representing simple data structures and associative arrays.

OOP

Object Oriented Programming is a programming paradigm that uses object data structures that consist of data and methods to manipulate this data in order to implement programs in a more modular fashion.

PHP FPM

PHP FPM is a BSD Licensed patch for managing PHP Fast CGI processes.

RFID

Radio Frequency Identification is a system of tracking objects via electronic tags.

SQL

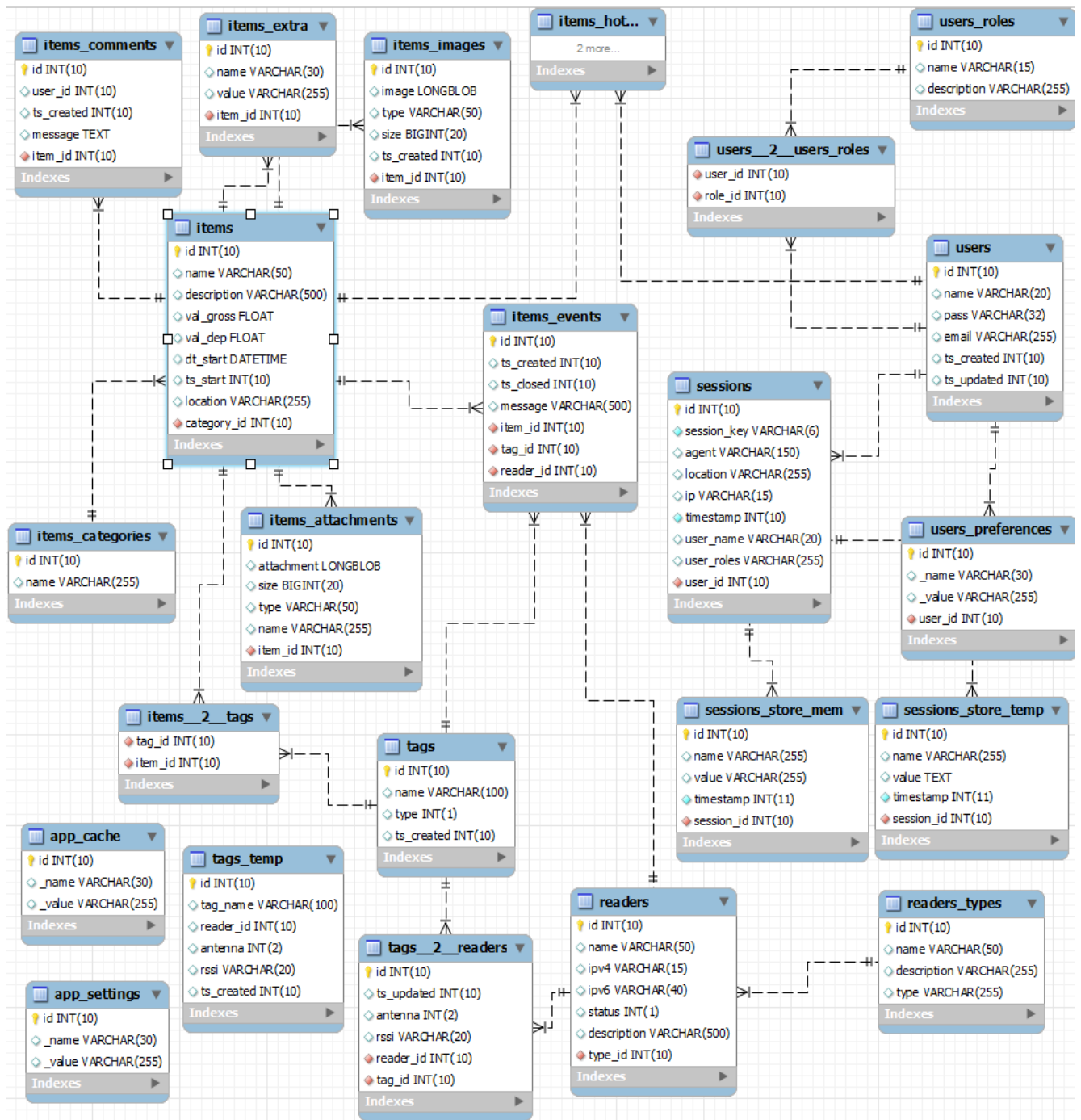
Structured Query Language is high level language designed for retrieval and manipulation of data in a relational database system.

YUI

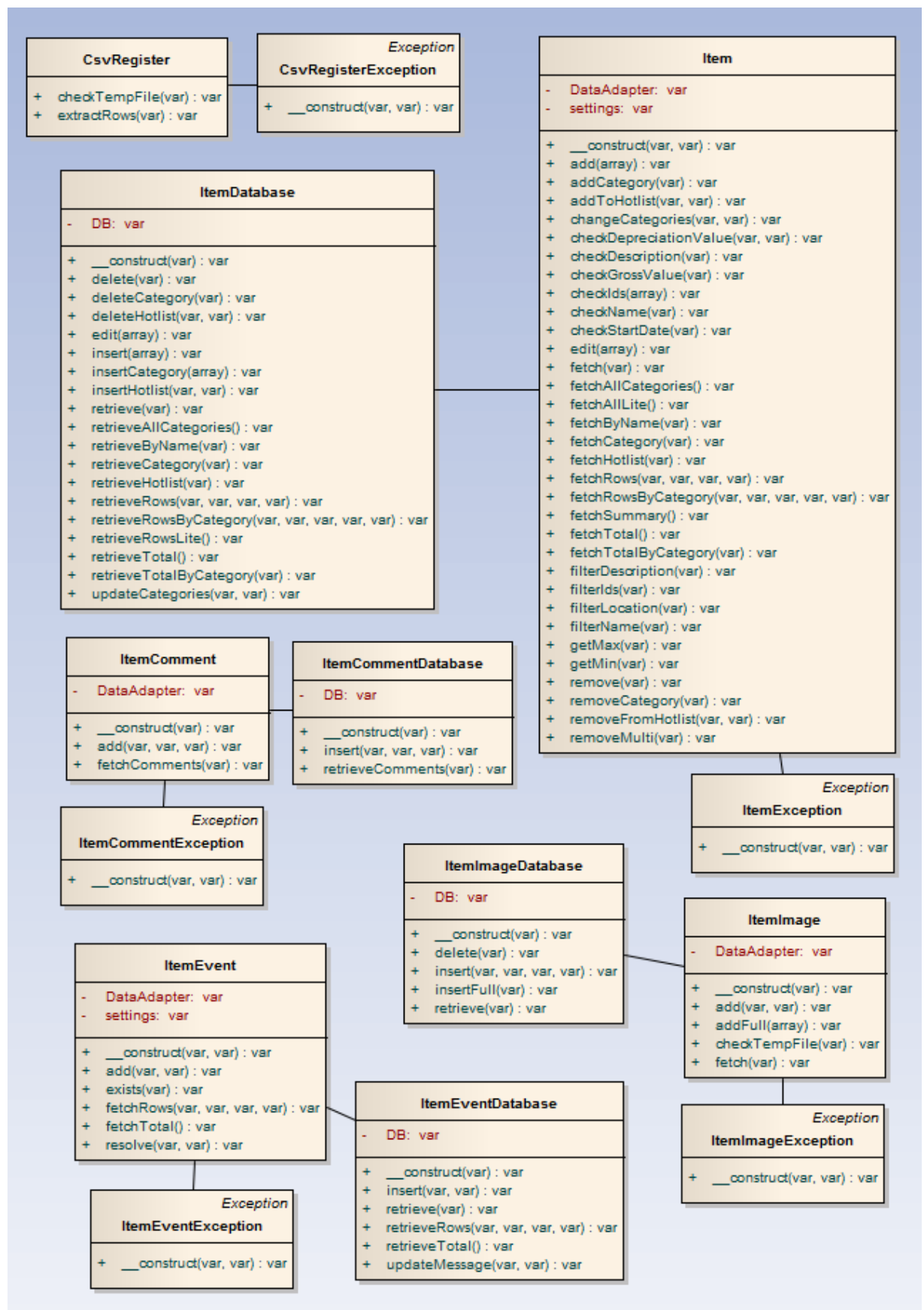
The Yahoo User Interface library is an open source, BSD licensed JavaScript framework and set of libraries which can be used to build rich and interactive web applications.

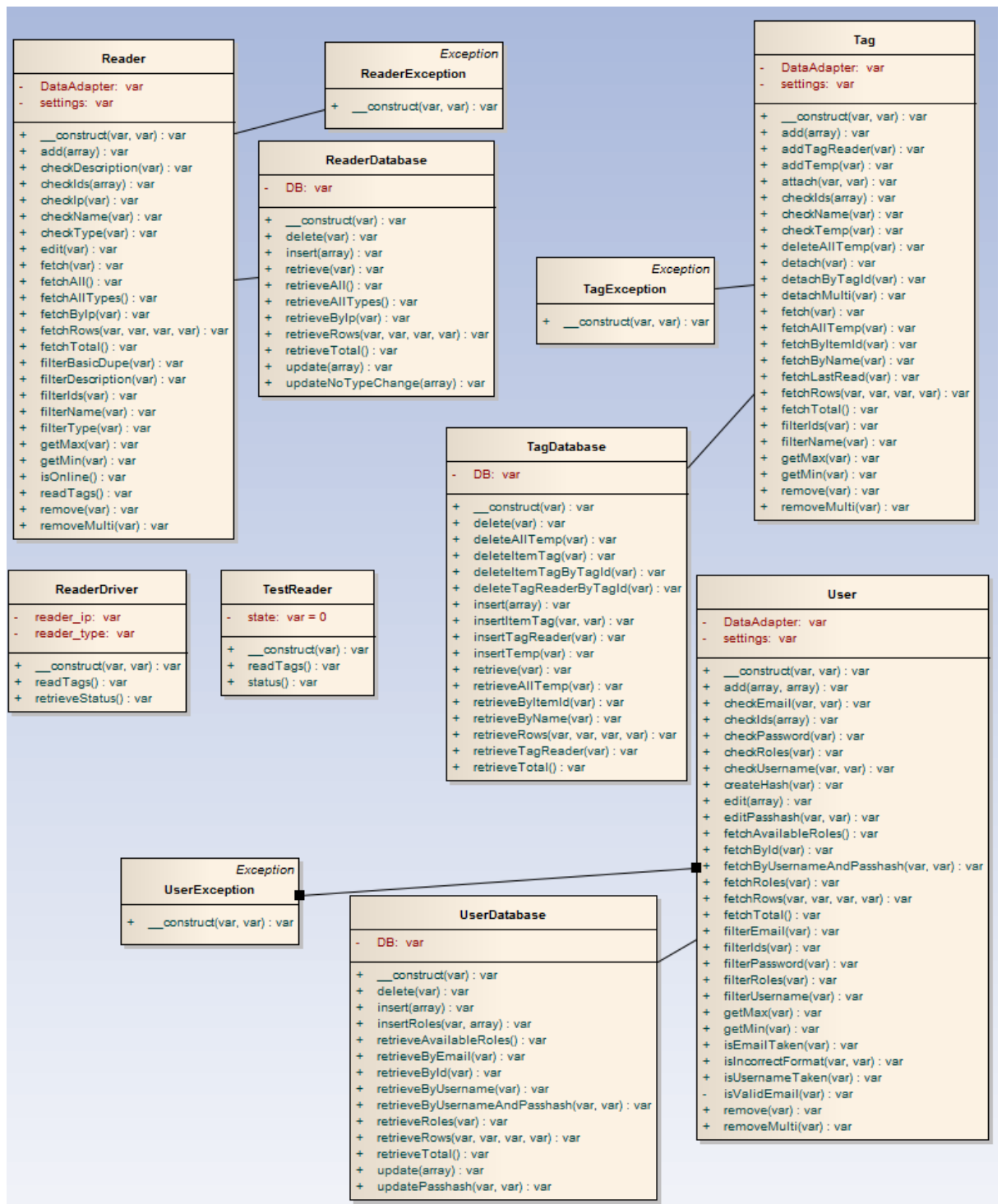
Appendix

1. Juno Entity Relation Diagram and schema



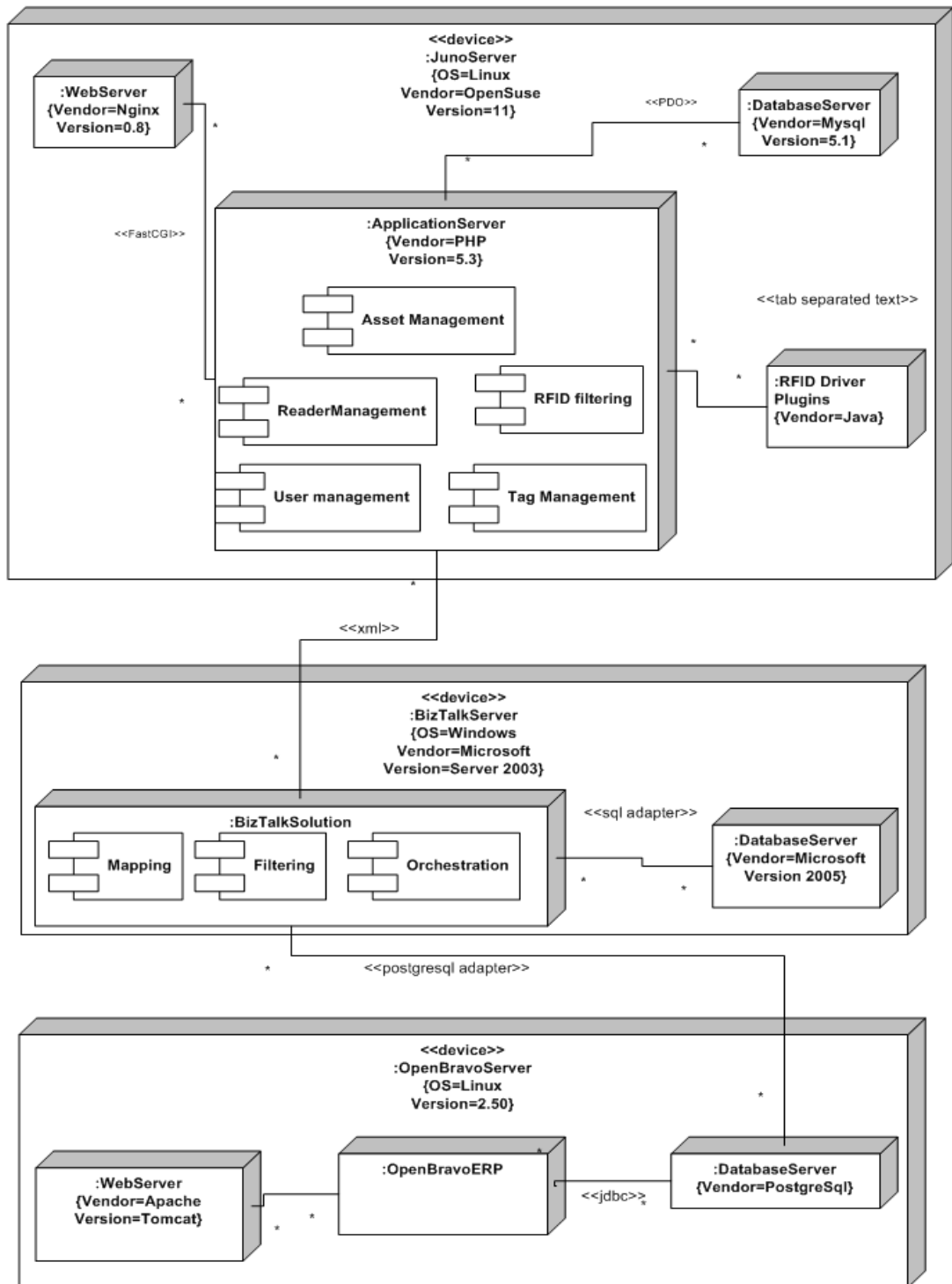
2. Juno UML Class Diagram for the main Model Classes





At the core of the Juno system sit the main Model classes (the M in MVC pattern). Their main functionality mostly involves checking and filtering data and working with the appropriate database tables. With the exceptions of the ReaderDriver class which communicates with the RFID readers via each reader driver and the Reader class which also has a method to filter tag data.

3. UML Deployment Diagram

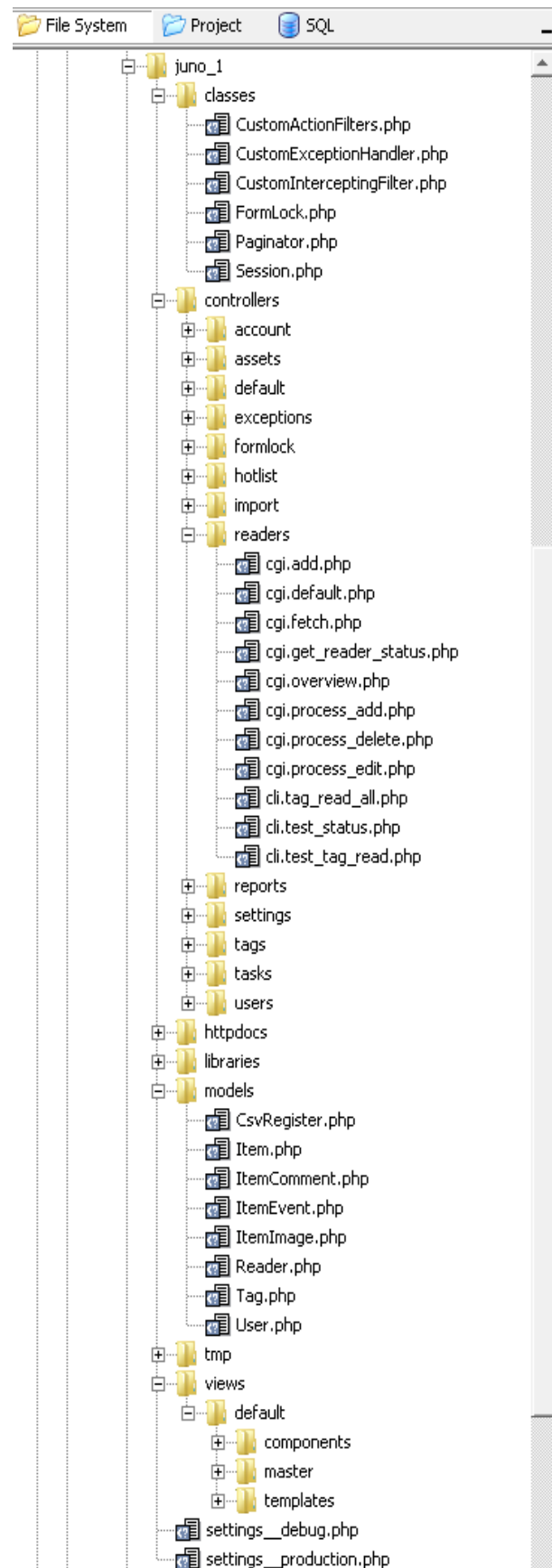


The above UML Deployment diagram shows each of the 3 devices used within this project and its main components. Each of the devices is a virtual machine instance within the VMware server, with communication over a local network. This allowed for easier testing, development and deployment of the overall system.

4. Juno source code outline and structure

As discussed earlier, the Juno web application part of the project was mostly coded in PHP5. The project structure as illustrated on the right follows the MVC architecture with a clear separation between controllers, models and view templates. This helped achieve the project aims of maintainability and extensibility of the code. Any new functionality can be added by extending a new controller, coding its interactions with the model(s) and constructing an appropriate response for the end user. The project also contains several filter classes which hook into the HTTP request/response cycle and third party open source JavaScript libraries.

Thanks to the separation afforded by the MVC pattern some of the controllers (such as reader driver testing and diagnostic) are not exposed via the CGI interface in the normal HTTP request/response cycle but are usable from the command line. This proved to be helpful in rapid development and testing of the models, such as the Reader class.



5. RFID tag data filtering and processing

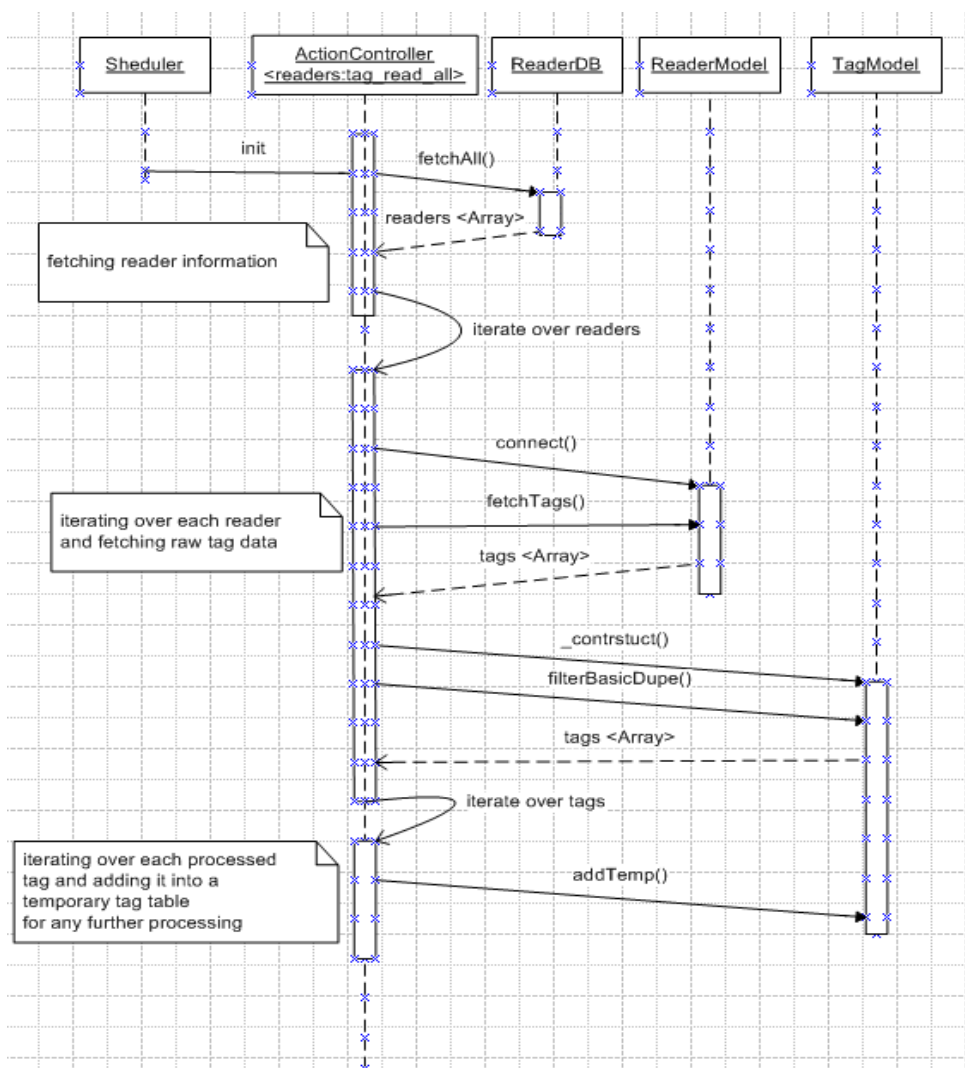
The main functionality of the Juno system revolves around capturing, filtering RFID tag data (sequence 1) and then processing this data into useful information and storing it (sequence 2).

The outline of sequence 1 is as follows:

- Scheduler initiates the script every minute. One minute intervals is the smallest period that the Linux crontab scheduler can do, alternatively the events can be triggered via the web application by a button click or by using a custom scheduling daemon which can poll the readers at smaller of intervals if required. For simplicity and since the business requirements of the partner did not require a near real time system, polling the readers at one minute intervals is sufficient.
- The controller fetches a list of readers and its driver and connection settings.
- And then proceeds to iterate over this list contacting each reader in turn asking for raw tag data, in the event of a Reader problem such as communication issues or the Reader taking too long to respond an exception is raised. An alternative approach was also tried, this involved forking of a separate thread for each Reader and reading the tags within its own thread, this alternate approach offers some safety against the occurrence of I/O blocking and would be better suited for a system with a large number of readers where a simple linear iteration might exceed one minute intervals. This alternative method of forking each reader polling thread is recommended for systems with large numbers of readers as discovered after stress testing this part of the code with a mixture of real and virtual dummy test readers.
- The returned tag data is then passed through a basic duplicate removal filter in order to eliminate redundant tag noise (As mentioned earlier some readers physically poll the tags at faster rates than other readers). For example some readers might return 10 entries for a particular tag within the minute interval but we only need the most recent entry in the process cutting down on

redundant (for the purpose of this project) data, of course the filter method could be easily modified with any evolving business requirements.

- The next step involves iterating over this returned and filtered list of tag data (the format is discussed earlier in thesis on page 58) and adding each temp_tag tuple into a temporary table. The reason for this approach as compared to performing further filtering within this thread is a simple case of refactoring a large process into a focused method in order to follow the basic software engineering principle of achieving strong cohesion. Storing the partially filtered tag data in a fast Memory type table allows this data to be further filtered by a separate processing script(s) (See sequence 2) and leaves the initial capture and basic filtering sequence performing fast. This approach also allows the system to scale as filtering is split into two steps with the possibility of farming out the further filtering and processing steps onto other servers.



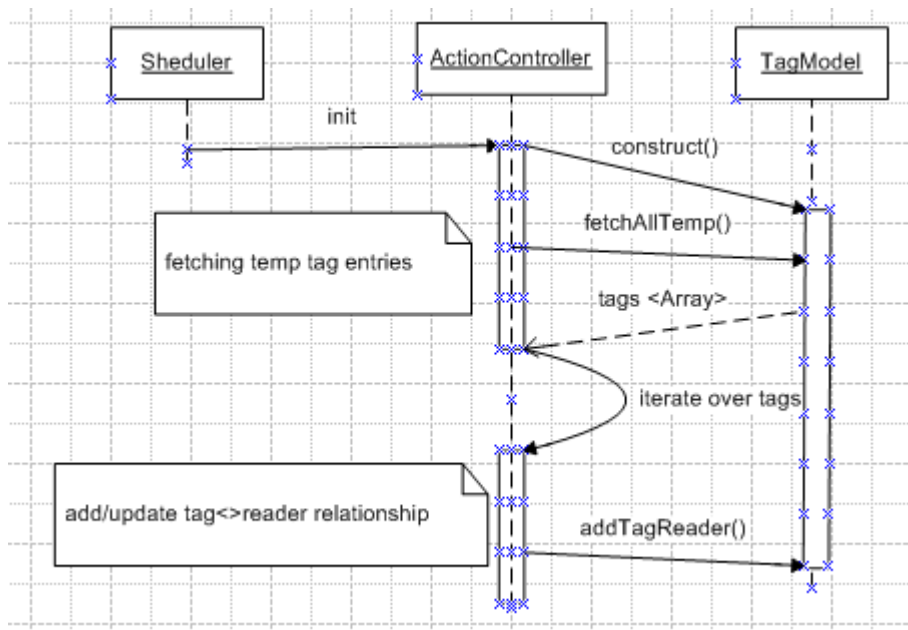
As mentioned already the basic functionality described in “sequence 1” on previous pages and implemented in the ActionController called “readers:_tag_read_all” inserts data into a temporary Memory table, data which has been retrieved and then filtered to remove unneeded duplicate entries.

This allows for the implementation of any further filtering or processing steps that might be required in their own separate controllers running on their own schedule. Based on the requirements gathered from the industrial partner, there are three separate filtering/processing actions being performed on the gathered data, their aim is to derive useful information to fulfil business needs, these are:

- Processing the tag data and updating the appropriate Reader<>Tag relation in the database, hence associating each tag (and indirectly each tagged asset) with a reader and also updating needed attributes such as timestamp, antennae number and RSSI.
- Inserting log entries of the above derived relation into a logging table, this provides us with a history of the asset location and an audit trail.
- Raising exceptions if any of the assets leave the read range of all the readers after a period of time (1-2 minutes). This alerts the system users that an asset might be missing.

Further filtering options can be easily added if required, for example triangulating the assets position and displaying the location on a room map.

The outline of one of these filtering sequences above is represented in the following UML sequence diagram.



6. BizTalk Integration, Development Iterations

BizTalk Server 2006 is a stand-alone system that operates separately from the various other organisations systems. BizTalk Server 2006 ensures that the remote systems and the organisations tracking application systems interact smoothly and efficiently. This includes applying business rules to the messaging process and routing the messages appropriately. The message flow for this scenario looks similar to the following illustration.

Message Flow

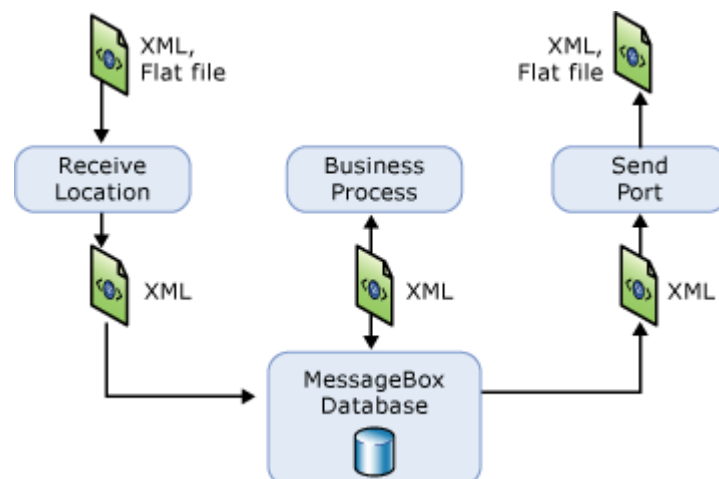
To fully implement this system an agile development methodology was adopted as much as possible. The main influence of this was an iterative approach to development. The following sections discuss each of the major iteration involved in this development.

Development Iteration One – Flat File

For this basic section of the research a visual studio BizTalk Server project was created. The project contains two message schemas, a pipeline, and a map. In the solution, a remote system sends a message to BizTalk Server for processing. In this iteration the following items were developed:

- The solution, to hold the project.
- The project, to hold the schemas, pipeline, and map.
- The schema for the message the remote system sends to BizTalk Server.
- The schema for the message BizTalk Server sends to the client application if the message is invalid.
- A pipeline, which transforms both messages so BizTalk Server can use the FILE adapter to send and receive the messages.
- A map, which connects the schemas of the two messages.

By mapping the schemas, data from the remote systems message can be included in the request denied message. The business process routes the messages and evaluates the contents of the inventory replacement request message against approval criteria. The following illustration shows the flow of data through BizTalk Server in the Iteration 1 scenario.



Middleware development - Iteration 1

Iteration two – Message Queues MSMQ

Writing Messages to a Message Queue (C#)

For this project Message Queues were investigated in regard to transferring asset data between a data base through middleware to a web service and file system. The research avenues investigated are outlined below.

With the ability to read write and delete messages from a web service established it is necessary transfer the received messages to the messages queue.² Microsoft Windows Message Queuing is the easy way to communicate with application programs quickly and reliably by sending and receiving messages. Messaging provides a flexible and powerful mechanism for inter-process communication between the components of server-based applications.

Message queuing provides a guaranteed message delivery. These messages can be prioritised according user requirements. Messages can be sent and remain in the queue the same way as it sent till the message is delivered. In other words, it is possible to implement offline capabilities. Several related messages can be coupled into a single transaction to ensure that they are sent in order, delivered only once and successfully retrieved in the destination queue. If any error occurs in this transaction, the transaction is cancelled. Windows security can also be used to secure access control, authenticate and encrypt the messages sent and received.

The following section explains the how MSMQ has been implemented between the web service and middleware server and the basics of message queuing.

² <http://www.codeproject.com/KB/cs/mgpmyqueue.aspx>

Queue Creation

The process starts with the queue creation. In the example a private queue called 'MyQueue' is created. Public queues can be created on your machine or any other machine with Message Queuing. For that however, it will be necessary to have domain or enterprise administrative access rights. There is a difference in creating a new queue and creating *an instance* of Message Queuing component, which refers to an already existing queue in the operating system. This code snippet states that if the private queue called MyQueue already exists; create an instance of MessageQueue to point to that queue. If it doesn't exist then create a private queue called MyQueue.

```
if( MessageQueue.Exists(@".\Private$\MyQueue"))  
  
    //creates an instance MessageQueue, which points to the already  
    existing MyQueue  
  
    mq = new System.Messaging.MessageQueue(@".\Private$\MyQueue");  
else  
  
    //creates a new private queue called MyQueue  
  
    mq = MessageQueue.Create(@".\Private$\MyQueue");
```

It is also possible to verify whether the queue is created or not with the Computer Management Console. In the Computer Management Console, expand the *Services and Applications* section. Then expand the *Message Queuing* section and select *Private Queue*. The newly created queue named *MyQueue* can be viewed here.

The next step is to send a new message.

Sending a Message

```
System.Messaging.Message mm = new System.Messaging.Message();

mm.Body = txtMsg.Text;

mm.Label = "Msg" + j.ToString();

j++;

mq.Send(mm);

//Use Message object to send messages. This will give you more control
over your messages.

//To receive,

try
{
    mes = mq.Receive(new TimeSpan(0, 0, 3));

    mes.Formatter = new XmlMessageFormatter(
        new String[] {"System.String,mscorlib"});

    m = mes.Body.ToString();
}

catch
{
    m = "No Message";
}

MsgBox.Items.Add(m.ToString());
```

There are several considerations in retrieving and reading messages from the queue. These are i) locking access, ii) properties and iii) format for reading messages. To read

messages from a queue, a formatter is necessary to serialise and desterialise the message before manipulating it. In this example `XmlMessageFormatter` is used.

It is possible to select properties when retrieving a message from a queue. These properties are in a class called `MessagePropertyFilter` and correspond to actual properties on the `Message` class. When the value for one of these properties is set to `true`, the component will retrieve the corresponding property each time a message is removed from the queue. If any properties are not required the `MessagePropertyFilter` is set to `false`.

Access locking to the queue is performed by setting the `DenySharedReceive` property to `false`. This will temporarily prevent other users from reading message from the queue and also prevents unauthorised removing of messages.

Put the creation logic in the constructor of the form and the send and receive in appropriate button clicks, you'll get the simple inter-process communication between the two forms

This added the ability to receive messages via a Message Queue as described previously described, this was to allow messages to be received by BizTalk server 2006 in a completely decoupled manner.



Iteration Three – Database

The Receive port was configured to extract the asset details from an open bravo PostgreSQL database. Because BizTalk server 2006 has no default adapter for this type of database a web service was written to act a proxy between BizTalk and open bravo. This was achieved using the BizTalk 2006 SOAP adapter. This database is a repository for all of the assets and can be queried by the various user applications (Juno in this case). The flow of data through the system now looks as follows.



Middleware development - Iteration 3

Iteration Four – Web Services

BizTalk server 2006 comes complete with Web Service SOAP adapters which allow data to be accessed. To maintain compatibility with other middleware products we decided to develop a bespoke web service to interface with the messages in the SQL Server 2005 database described previously.

For this section a new web service was developed. A new namespace will be defined, and within this namespace will be a set of classes that define the Web Service. By default the following classes will be created:

- Global (in global.asax) - Derived from `HttpApplication`. This file is the ASP.NET equivalent of a standard ASP `global.asa` file.
- Service (Service.cs) - Derived from `System.Web.Services.WebService`. This is the `WebService` class that allows you to expose methods that can be called as Web Services.

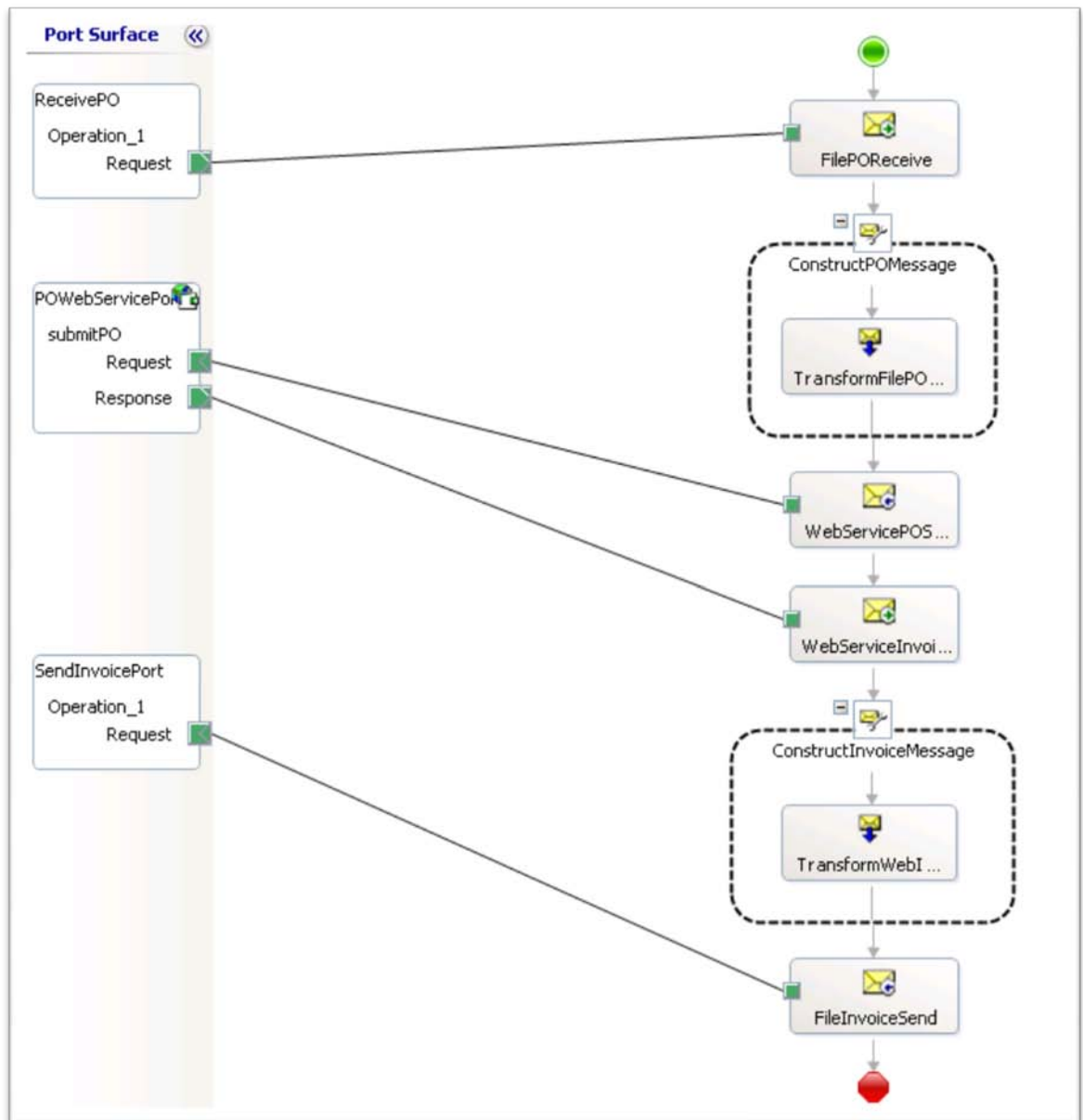
There are also a number of files created:

- `AssemblyInfo.cs` - Contains version and configuration information for the assembly.
- `web.config` - Defines how the application will run (debug options, the use of cookies etc).
- `Service.disco` - Discovery information the web service.

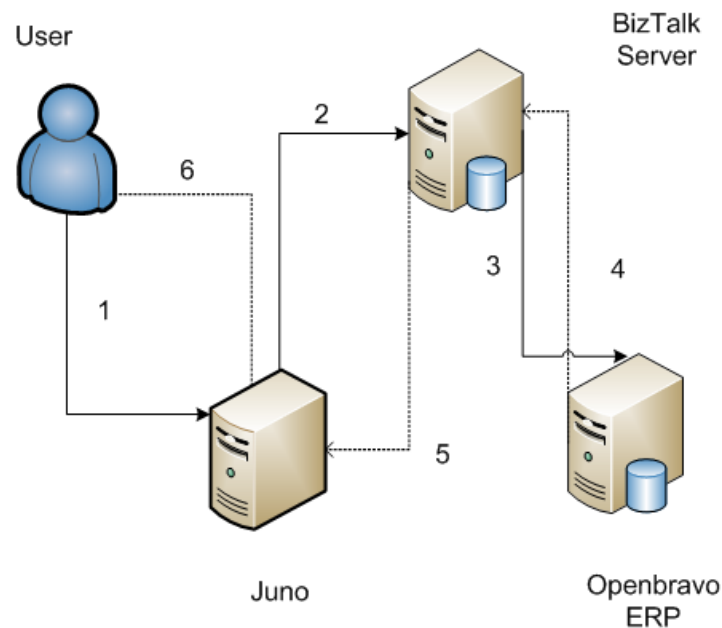
Navigate to `Service.asmx` file in a browser and you will get back a user-friendly page showing the methods available, the parameters required and the return values. Forms are even provided allowing you to test the services through the web page.

`Bin\Service.dll` is the actual Web Service component. This is created when you build the service. The class for the service is called `Service.asmx`. To test the service you can navigate to `Service.asmx` from your web browser.

7. BizTalk Orchestration Schedule



8. ERP interaction events and processing



As discussed earlier this project uses Microsoft BizTalk server in order to connect to an ERP system (Openbravo ERP in the case of this project) and then retrieve asset information from the ERP system. The main events in this process are as follows:

1. User initiates the retrieval process by clicking on a button in the Juno UI
2. A web service request message is send from Juno to the BizTalk application. Here the BizTalk orchestration schedule checks whether the message is valid.
3. Once the request is verified by the orchestration BizTalk connects to the Openbravo database and retrieves asset details.
4. This asset information is then “mapped” within the BizTalk server and only the relevant fields as required by the business rules are kept in a local database.
5. An XML file consisting of the required asset information fields is returned as a response to the webservice request in 2).
6. This response is parsed by relevant Juno controller and the asset information added to the item database within Juno, the User receives a success or failure response in the UI.

7. Legacy A-Track application, user interface

efAST *a-track*
Main page Asset Register Exceptions Admin Reports

Information

Logged in as: **admin**
[Logout](#)

Passive - these are the tagged items that require a user to proactively gather the information and record in a-track.

Active - these are the tagged items that will automatically update the a-track system when there is a read event (i.e. entry or exit of the building).

Search

This allows users to search the a-track system for assets based on the criteria specified below:

Asset Code:

Asset Register Summary

| Fixed Asset | €M GBV | €M DPN | €M NBV |
|-------------------------------|--------|--------|--------|
| Foirgrinmh | 7.42 | 1.47 | 5.95 |
| Talamh | 0.02 | 0 | 0.02 |
| Fearas & Treallamh Ginearalta | 6.1 | 5.4 | 0.7 |
| Trealamh Tarcur | 0.22 | 0.15 | 0.07 |
| Feithicili | 0.72 | 0.29 | 0.43 |
| Crua Earrai Riomhaire | 5.16 | 2.88 | 2.28 |
| | 19.64 | 10.19 | 9.45 |

Latest Exceptions

| Asset ID | Description | Status |
|------------|----------------------------|--------|
| SCN0000514 | NEW ATOM FOR TG4 INTELIFAX | ● |
| X02A020003 | XtenDD plus DVE | ● |

Hot List

List and current status of high value assets

| Asset ID | Description | Status |
|------------|--------------------------------|--------|
| 01A0500025 | Fujitsu notebook | ● |
| 01A0500022 | Dell pedge 750 JVVPG1J | ● |
| 01A0300016 | DELL PC | ● |
| 01A0500015 | DELL LATITUDE L04811 D810 | ● |
| 01A0500006 | DELL DIM3000 3PCTG1J | ● |
| 01A0400012 | DellPwredge750 4V7BC1J | ● |
| 01A0300006 | INSPIRON Laptop PF04L | ● |
| 01A0300008 | Laptop latitude Pentium M D400 | ● |
| 01A0400002 | FUJITSUSCENIS P-300 CEL 20 | ● |
| 01A0500021 | Pc HP XW800 | ● |
| 01A0400013 | DellPwredge750 8V7BC1J | ● |
| 01A0300005 | INSPIRON Laptop PF04L | ● |
| 01A0300007 | INSPIRON Laptop - Peabar 333 | ● |
| 01A0500012 | DELL DIM3000 FPCTG1J | ● |

8. Juno screenshots

Juno – Main Dashboard


The screenshot shows the Juno Main Dashboard. At the top, there is a navigation bar with the Juno logo and the text "welcome - juno". Below this, there is a breadcrumb trail: "admin @ 192.168.1.250 - settings - logout". A set of navigation tabs is visible, including "overview" (which is active), "assets", "exceptions", "reports", and "tasks".

The main content area displays the "Asset Register Summary" table. Above the table, it states "no assets attached to the hotlist". The table has the following data:

| | € GBV | € DPN | € NBV |
|--------------|----------------|---------------|----------------|
| computers | 2000.00 | 299.00 | 1701.00 |
| equipment | 0.00 | 0.00 | 0.00 |
| virtual | 344.00 | 25.30 | 318.70 |
| total | 2344.00 | 324.30 | 2019.70 |

At the bottom of the dashboard, there is a "home" link and a copyright notice: "© 2008 - GMIT".

Juno – Asset Register View



assets

admin @ 192.168.1.250 - settings - logout

overview
assets
exceptions
reports
tasks

home > assets

Filter By Category: --choose--

| | id | name | category | location | € GBV | € DPN | € NBV |
|--------------------------|----|-----------------|-----------|-----------|---------|--------|--------|
| <input type="checkbox"/> | 1 | DELL CS2465 | computers | gd142 | 1000.00 | 99.00 | 901.00 |
| <input type="checkbox"/> | 2 | Gateway CS2465 | computers | gd142 | 1000.00 | 200.00 | 800.00 |
| <input type="checkbox"/> | 3 | virtual asset 1 | virtual | gd142 | 1.00 | 0.10 | 0.90 |
| <input type="checkbox"/> | 4 | virtual asset 2 | virtual | gd142 | 1.00 | 0.20 | 0.80 |
| <input type="checkbox"/> | 5 | test1 test | virtual | top shelf | 10.00 | 1.00 | 9.00 |
| <input type="checkbox"/> | 12 | test123 | virtual | shelf | 111.00 | 12.00 | 99.00 |
| <input type="checkbox"/> | 13 | test1234 | virtual | shelf | 221.00 | 12.00 | 209.00 |

|< first
<< previous
1-7 out of 7
next >>
last >|

-> check at least one asset<-


|< first
<< previous
1-7 out of 7
next >>
last >|

add an asset

[manage asset categories](#)

Juno –Tagged Item Detailed View

admin @ 192.168.1.250 - [settings](#) - [logout](#)



assetsoverviewassetsexceptionsreportstasks

[home](#) > [assets](#) > [view asset - 24" Samsung LCD](#)

asset information

name: 24" Samsung LCD
description: test monitor
category: computers
location: top shelf

€ GBV: 250.00
€ DPN: 50.00
€ NBV: 200.00

start date: 2008-01-01 00:00:00


Edit AssetDelete Asset

rfid information

tag id: 5
name: 97
type: active
last read: 2008-12-17 10:10:46 @ reader #20 (wavetrend reader) - antenna #1

Detach RFID Tag

asset image



Upload Image

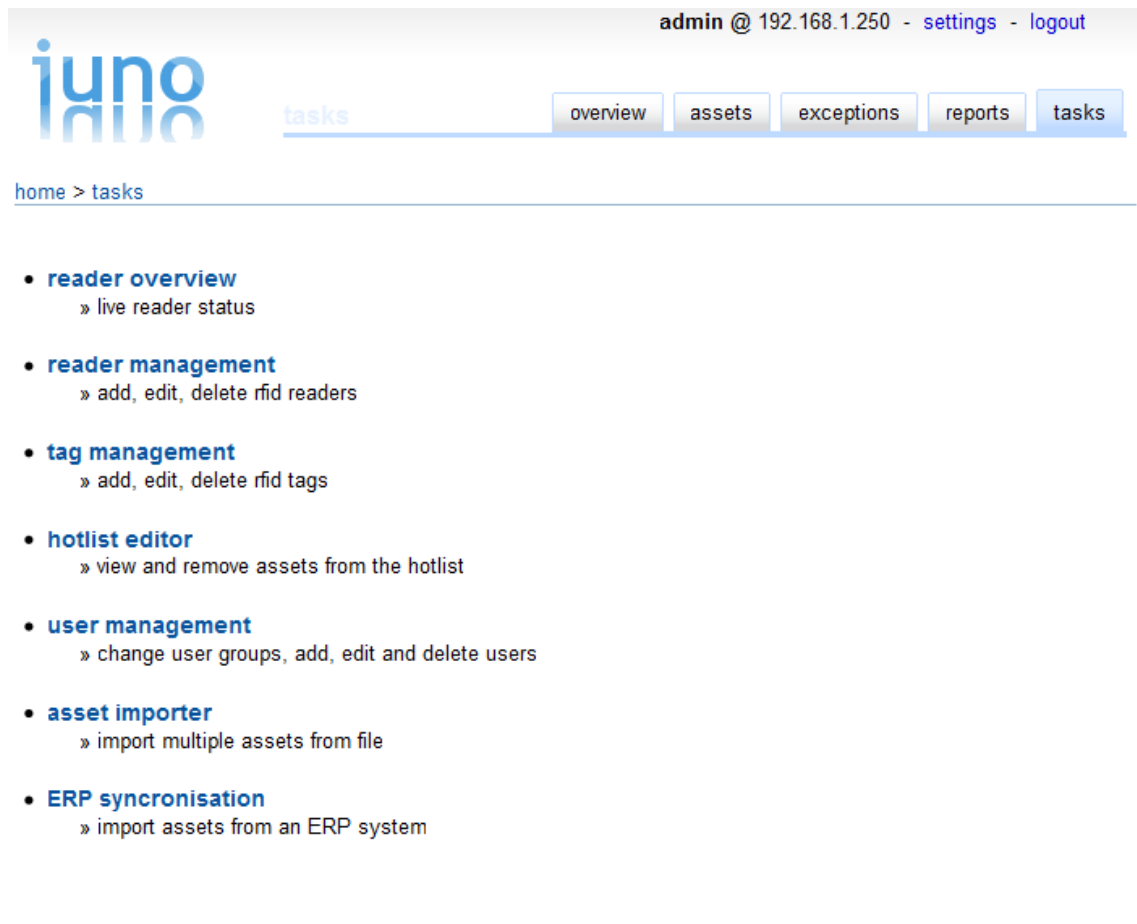
asset comments

admin @ 2009-01-28 10:49

test

Leave a Comment

Juno – Common User Tasks



The screenshot shows the Juno web application interface. At the top right, the user is logged in as 'admin @ 192.168.1.250' with links for 'settings' and 'logout'. The Juno logo is on the left. A navigation bar contains 'tasks', 'overview', 'assets', 'exceptions', 'reports', and 'tasks'. Below the navigation bar, the breadcrumb 'home > tasks' is shown. A list of tasks is displayed:

- **reader overview**
 - » live reader status
- **reader management**
 - » add, edit, delete rfid readers
- **tag management**
 - » add, edit, delete rfid tags
- **hotlist editor**
 - » view and remove assets from the hotlist
- **user management**
 - » change user groups, add, edit and delete users
- **asset importer**
 - » import multiple assets from file
- **ERP synchronisation**
 - » import assets from an ERP system

© 2008 - GMIT