Applicability of Agent Technology for Software Release Management

B.A.D.A.S. Bogoda
08/10001

Faculty of Information Technology

University of Moratuwa

September 2010

Applicability of Agent Technology for Software Release Management

B.A.D.A.S. Bogoda
08/10001

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Degree of M.Sc. in Artificial Intelligence

September 2010

Declaration

I declare that this dissertation does not incorporate, without acknowledgment, any material previously submitted for a Degree or a Diploma in any University and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organization.

B.A.D.A.S. Bogoda

Name of Student Signature of Student

Date

Supervised by

Prof. Asoka S. Karunananda

Name of Supervisor(s) Signature of Supervisor(s)

Date

Dedication

To My Parents

Acknowledgements

I am heartily thankful to my supervisor, Asoka S. Karunananda, whose encouragement, guidance and support form the initial to the final level enabled me to develop this solution.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

Abstract

Information Technology industry is one of the most widely spreading industry around the world in recent past as its applicability and adaptability nature for various streams. Influence created by IT industry, on various streams help to accelerate their development in large portion with in a short period of time. Because of quick reaction in problem solving and easy way of storing and retrieving information people tend to replace existing manual systems by computerized systems.

Enterprise level applications are developed as a combination of several components. Modification done in a one component is used to fulfill some functionality of other component. When preparing those modification to deliver for the customer it is required to know which component's modification is highly depend on each other and which is not depend on other. Currently this handle by human where developers go through the modification and identify which component is highly depending on the modification done on other components. Developers need to have good communication among each other to identify correct order, the components should arrange. Once the order is identified they start compiling and building jar files. This process is highly time consuming task as there are frequent updates done for the code when fixing issues. This subject to reduce developer's effective time he can work. It's a big burden for the company as well.

In this thesis it will discuss how to solve above problem by automating the software release management process using Multi-agent technology. In the literature review chapter it will discuss about different researches conducted related to software release management domain. In the design chapter it will give high level picture about the design. Then gradually it will explain how agents solve this by communicating with each other, under implementation chapter. At the evaluation chapter it discuss how implemented solution has been tested and what are the other advantages it can gain by overcoming limitations of the existing release management tools. Under conclusion chapter it discusses whether each objective has achieved by providing appropriate test samples.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Information Technology industry is one of the most widely spreading domain around the world in recent past. Because of quick reaction in problem solving and easy way of storing and retrieving information people tend to replace existing manual systems by computerized systems. Once system is computerized it is required to maintain the system while it is in production environment. Almost all of the software development companies used to maintain software product fully or partially by manually and which require more time and human resources. In this chapter it is going to discuss about background and motivation factors that influence to have an automated release management system in software development domain in order to save time and human resources. After discussing background and motivation factors it will discuss about aim and objectives with reference to this project. At the end of this chapter it will discuss about set of resources which were used to implement this system.

1.2 Background and motivation

Software development process in Information Technology industry has become one of the leading jobs providing area to the world market in recent years. According to the Software Marketing Research [16] it earned around $451.8 billion in the year 2008 in world wide. Not only fast development of software stream provide huge number of jobs to the world job market but also it assists considerable growth of areas like education, entertainment, healthcare, traveling and banking. It is true that contribution of IT industry is highly benefited to each and every area to walk towards a new era. In this each area it can see that there are large numbers of enterprise level software tools available in different flavors. When using these tools in production environment various unknown failures can arise. These failures should identify and fix in order to have a quality software tool. Once particular failure is fixed the new product should hand over to the customers who are using it. Process of identifying failures, fixing them and handover to the customer is known as software release management. This process is highly time consuming process and it is currently handle by manually with support of software developers. As a result developers have to spent considerable amount of their time for release management activities. When there are

many clients using the product the frequency of identifying new issues will increase. Result will be frequent releases through out the week for each and every issue fix. This is a huge draw back for the IT companies since valuable time of the software developers can not be fully allocated to developing the software product. Result will be the reduction of productivity and the motivation factors of the developers. In order to improve the productivity and motivation, involvement of developers for release management process should minimize. It can only be done by automating the software release management process.

1.3 Aim

Develop a system to automate the software release management process using multi agent systems technology.

1.4 Objectives

- Reduce human interaction in release management process
- Reduce network traffic
- Minimize the time spent on release management task
- Reduce exceptions occur in compilation and build process
- Optimize CPU memory usage.

1.5 Resource Requirements

In this section it will give description about the software it used to develop the solution. As this is very much based on software tool no special hardware is required. Develop the system to run on Windows XP operating system. Java Development Toolkit [12] (JDK Version: 1.6) was used as the programming language to develop the system. For source repository management it used TortoiseSVN [20] (Version: 1.5.9). SVNKIT [15] (Version: 1.3.0) used as the communication bridge between TortoiseSVN and Java programming language. SVNKIT used as the communication bridge as SVNKIT support to execute almost all TortoiseSVN commands using Java programming language. JADE Toolkit [10] (Version: 3.6.1) used as the agent development tool kit which is also developed by using Java programming language. To compile the Java source code and build jar files it use the Apache Ant [2] tool as it is highly configurable and support for commands used in Java programming language. Eclipse IDE [6] (Version: 3.4.2) used to develop Java code for this system.

1.6 Summary

In this chapter it was discussed about surrounding areas of the software development process in IT industry. Further more it has discussed importance of the problem it is going to address in line with the rapidly growing IT industry. Consequently it has pointed out aim and objectives of this project with respect to software development domain. As the end it has provided set of software tools and their versions which were used to develop this system.

Rest of the document arranged as follows. In the second chapter it will discuss about each steps in waterfall model and set of researches conducted on software release management domain. In the third chapter it will discuss about technology behind in software release management. In the fourth chapter it will discuss about multi agent approach for software release management. In the fifth chapter it will discuss about the design of the proposed solution. In the sixth chapter it will discuss about implementation of the solution by giving appropriate example of the source codes. Under seventh chapter it will discuss on evaluation of the developed system and in the eighth chapter it will discuss about conclusion and further works with reference to this project.

Chapter 2

Current trends in Software Release Management

2.1 Introduction

In the previous chapter it has discussed about one of the major problems face by developers in the software development process when maintaining the software product. Also it has discussed about aim and objectives it is going to achieve by doing this project. At the beginning of this chapter it is going to discuss about major stages in software development process by considering the waterfall model. After that it will discuss about different researches conducted on software release management domain under three main categories.

2.2 Waterfall Model

The waterfall model is a sequential software development process, in which progress is seen as a waterfall through the phases of Requirement Analysis, Design, Implementation, Verification and Maintenance. It is known as waterfall model, as its progress flow from the top to the bottom like a waterfall. This model was originated from the manufacturing and construction industries as there was no formal software development methodology exists at the beginning. Simplest form of the waterfall model is shown in Figure 2.1.

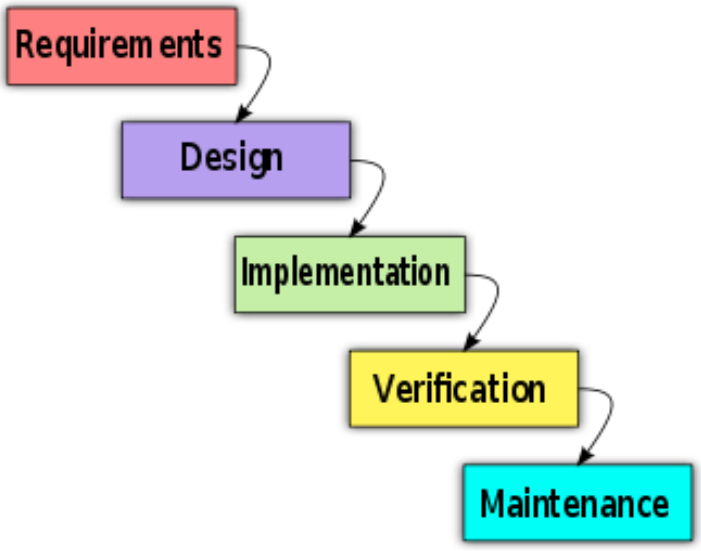

Figure 2.1: Waterfall Model

As it shown in the Figure 2.1 the waterfall model proceeds from one phase to next in a sequential manner from top to bottom, once particular phase is fully completed. Detail of the each phase and its functionality describe separately as follows,

2.2.1 Requirements

Requirement analysis is the initial stage of the waterfall model as shown in the above Figure 2.1. One of the major out come of this phase is the requirement specification document. Requirement specification is a well stranded document which normally calls as Software Requirement Specification (SRS). This document contains the complete description of the behavior of a system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. Use cases are also known as functional requirements. In addition to use cases, the SRS also contains non-functional requirements. Non-functional requirements are requirements which impose constraints on the design or implementation such as performance engineering requirements, quality standards, or design constraints. Once it sign-off it can not be modify throughout the life cycle.

2.2.2 Design

Software design is a process of problem-solving and planning for a software solution. After the purpose and specifications of software are finalized, software developers start designing a plan for a solution. It includes low-level component and algorithm implementation issues as well as the architectural view. If the software is semi automated or user centered, software design may involve user experience design yielding a story board to help determine those specifications. If the software is completely automated, a software design may be as simple as a flow chart or text describing a planned sequence of events. There are also semi-standard methods like Unified Modeling Language and Fundamental modeling concepts. In either case some documentation of the plan is usually the product of the design. Software design documentation may be reviewed or presented to allow constraints, specifications and even requirements to be adjusted prior to programming. Redesign may occur after review of a programmed simulation or prototype. It is possible to design software in the process of programming, without a plan or requirement analysis, but for more complex projects this would not be considered a professional approach. A software designer or architect may identify a design problem which has been solved by others

before. A template or pattern describing a solution to a common problem is known as a design pattern. The reuse of such patterns can speed up the software development process, having been tested and proved in the past.

2.2.3 Implementation

Once design of the system is done then developers are starting to implement the requested system step by step. In computer science, an implementation is a realization of a technical specification or algorithm as a program, software component, or other computer system through programming and deployment. Many implementations may exist for a given specification or standard. For example, web browsers contain implementations of World Wide Web Consortium-recommended specifications, and software development tools contain implementations of programming languages. Software Implementations involve several professionals that are relatively new to the knowledge based economy such as Business Analysts, Technical Analysts, Solutions Architect, and Project Managers. Once the implementation is done the flow will move to verification stage.

2.2.4 Verification

Once the system is implemented that should be verify. As there is a special team to develop the system, there is a special team to do the verification as well. They knew as quality assurance team. They are supposed to test the quality of the software which is developed by the development team. When testing the system it required to test whether requirement specified in the requirement specification is met. It is normally known as functional testing. In the quality assurance domain there are different kinds of testing strategies they follow in order to make sure that developed system is working on a particular standard. If the developed system is not working as expected or it deviate from the content included in the requirement specification, quality assurance will report an issue for that. Development team need to go thorough those issues and should implement a solution for that. When number of issue count get reduces and if system is in acceptable state the implemented system will handover to maintenance team.

2.2.5 Maintenance

Maintenance stage the implemented system will be fine tuned. Functionality miss matches, performance problems, new modification, new features are identified and re-implement them again. Actual implemented system will be handover to the customer with in this stage. It is well-known thing that unexpected issues are identified when the system is running on production environment opposed to the local environment. This can happen as in customer environments they are using different versions of the tools with different way data base configurations. Even performance issues can arise as too many users are using the system at the same time in real environment. In internal test environment these kinds of issues are rarely identified as small numbers of users are used to test the system. In addition to customer reported issues local quality assurance team also will identify issues which are not yet identified by customers. When any particular serious issue is identified by the customer or local quality assurance team it should be fixed immediately as it will be a black mark for the system. Some customers may reject to use the system if system contains lot of serious issues. Therefore once that kind of issue happen developers in the maintenance team should get immediate action on it.

2.3 Maintaining software product

Even though waterfall model is one of the oldest software development model it contains same set of phases available in the other development models such as V-model, Spiral model, Incremental model. In general those models are extended models of waterfall model. Currently different software development companies are using their own customized version of software development process according to their requirements. What ever the model they use at the end of this each model it will end up with a developed system which intended to use by a customer. Once it handover to the customer maintaining the software with good quality and in high performance is an essential factor in any model to keep the relationship between client and the company. As discussed earlier when maintaining the software product it is required to do continuous releases for every improvement or modification done to the code base. Therefore it is true that software release management is essential activity in any kind of software development model when maintaining a software product.

2.4 Movements in Software Release Management

Under software release management domain author has gone through several researches conducted by different universities and institutes around the world. In this section it is going present those researches under three main categories as Standardized Release Management System, Cost Effective Release Management System, and Flexible Release Management Systems for Component Based Products.

2.4.1 Standardized Release Management System

Martin Michlmayr who is a member of Technology Management center in University of Cambridge UK conduct a research with the idea of identifying problems with current release practices, verifying possible advantages of an increasingly popular release model and developing interventions to improve release management in free software projects [11]. In his research researcher has focus to conduct the research based on free software and open source projects. As he mention in his research, release management process identified as problematic area for open source developers where volunteers are continually modify the code in an unpredictable fashion. According to the author release management in distributed domain often associated with problems, as programmers who maintain software projects do not have management and coordination skills since they are basically capable on developing the code according to the requirements. Essentiality of extra commitment from project participants during a release to meet deadlines also become a problem as volunteers are not agree to put more effort on release management in their working hours. Finally, mismatch between the requirements of users and developers become a problem since developers mainly use development releases, where they might not see the need for well tested and stable releases aimed at less technical and corporate users.

On the existence of above problems release management standard can reduce the quality of it. As mentioned in the research low quality standard of release management can lead to number of problems such as software which is out of date, breaks compatibility, or does not meet the quality standards or the requirements of users. In this research researcher has interviewed twenty experienced free software and open source developers from variety of projects, release strategies and processes, along with number of problems with current release cycles. Those interviews were base on two major release strategies as time based release and feature based release.

In time based releases, it follows a clear schedule and release is made according to a fix time plan. In feature based releases, releases are done according to completion of certain features in the product. Time-based releases are particularly suited for large and modular projects because they allow individual developers to independently follow the schedule which has been set. This will reduce the amount of coordination required. Since developers know when their features must be ready on time possibility to have steady releases becomes high in time-based release. Because of the predictability, development of the system leads to more features and better code.

Five members of the Utrecht University conduct a literature study about software product management process in IT industry with the idea of developing a reference framework [22] avoiding the limitation of existing product management practices. As mentioned by the researchers the role of product manager has emerged over the last years and appears to be of strategic value, but complex one to execute as product manager is responsible for managing requirements, defining releases, and defining products in a context where many internal and external stakeholders are involved. According to them software product management did not have proper attention since the industrial revolution in the 19th century. Only relatively recently it has received attention in product software companies like Microsoft and Alcatel. In their research they have mentioned about some of the existing tools like ReleasePlanner, ReqSimile which are currently using in the software release management domain. But those tools are very much focus on release planning and delivery management. There are more areas yet not supported by those tools.

Hyrum K. Wright who is a member of Department of Electrical and Computer Engineering in University of Texas at Austin conduct a research with the idea of modeling and quantifying existing release processes and an effort to prescribe improvements to those processes [9]. According to the research software release is the most prominent aspect of a software deployment [18]. This is common for web service, an open source project, a commercial system, or internally developed application. According to researcher success of a quality software is depend on successful release process. Therefore development teams must guarantee that they have a satisfactory high quality release process to create low fault and high frequency releases. As mentioned in the article while developers and architects focus their

engines on the design of the software, release process also should start gradually design its release plan. As Researcher claims that because of the ubiquitous nature of the release process in the software development cycle, very little amount of researches were conducted on the release process area. As a result of that no significant improvements can be seen in tools and processes which are used in software release management area. In this research researchers are trying to define very flexible, common and useful process which can be use by different organizations, although these every organization has their own art of release management process. By investigating the previous researches conducted on similar domain for open source software, researchers are trying to expand their work further. At the moment they are working on crating a uniform database to keep release information for a variety of open source projects. In this process researches are planning to create rough methods of normalizing release information to allow inter-project comparison. In addition to that they are planning develop tools to support the maintenance and improvement of the collected data. This data is used to return prescribe improvements to the projects' release process with help of their proposed model and metrics. They are also plan to study current tooling infrastructure with relevant to release engineering and define a common standardize tool infrastructure to improve and streamline the release engineering process. As researchers mention in their 'conclusion' section, release engineering for most teams are ad hoc and homegrown over the past few years. Further they added that tools support for this domain is often hit-or-miss.

Kim Pries and Jon Quigley who works at product development training and cost improvement firm conduct a research with the idea of identifying, whether sound configuration management is fundamental to deliver a capable product [14]. As they claim an optimally running configuration management program reduces fault reports and makes testing easier, whereas incompatible system components subject to end up with more failures and much work wasted. The impact of good configuration management becomes even more meaningful if the product under test is actually a single system or a collection of components. It does not matter how well developers designed the product if they are unable to manufacture and deliver that design. Successful product development, in general, is thus completely dependent on robust configuration management. It is useless if we deliver wrong version to the customer

even we have implement the functionality correctly. In this article they pointed out that there are four requirements for any configuration management system which is define by Military standard 973 and military handbook 61. Configuration identification, Configuration control, Configuration status accounting, Configuration auditing are the four requirements. Under Configuration identification it define that correct versioning should follow when maintaining the product through out its' life cycle. Configuration control can consider as part of configuration management in which we manage changes to the product. Configuration status accounting also called as CSA is used primarily for the reporting feature, although its implication is much broader. Under CSA, it record changes, update configurations when items change, and issue reports. In Configuration auditing it compares the configuration expected to what is delivered. Configuration auditing can also divided in to two parts as physical configuration and functional configuration. In physical configuration audit it compares existing documents to contracted or required documents, and a functional configuration audit it verifies functionality against requirements. After defining about those configuration requirements researchers discuss about how software release products should manage. They suggest that to manage the complexities of software, the configuration management tool should allow branching, merging, developmental release numbers, and released product release numbers. In this article they have allocate a separate heading call 'Insufficient Configuration Management' which describe about side effects by having a bad configuration management system. As they describe inability to predict the impact of changes on the project is one of the subsequent failure which result to fail the whole configuration management system. They argued that it is required to have good configuration management system as issues found in configuration management are not easy to identify quickly. That will result to waste of both time and resources. Good configuration management system result to avoid both time and resources when investigating issues. Duration overruns on software and hardware development already impact time available for testing. The spending of additional time and resources to solve problems that are not valid is a waste and reduces the probability of launching on time and within budget. When the end product to the customer is configurable, these problems are transferred on to the customer, sometimes with no notice. If the product in question is a high-volume item, the issue may not be found until a number of customers have been involved and the product delivery pipeline is full of the errant product. As they claim without proper

configuration management, it is useless to fix the wrong version of the software, introducing yet more errors into the product. Finally they suggest that good configuration management always leaves a competent audit trail and never is the recipient of customer complaints.

2.4.2 Cost Effective Release Management System

Two members of Information and Computing Sciences Institute in Utrecht University conduct a research on finding on cost/value functions for product software vendors to support their release package planning method [13]. In this research researchers have define 'software product release management' as storing, publication, identification, and packaging of the elements of a product. According to researches 'release package planning' is a part of the release planning process which defines what features and bug fixes are included in a release package. In their research it mention about several types of packages that are used in the release process. First one is known as "update package" which promotes a customers configuration to a newer configuration. Update package basically divided in to three main parts as bug fix update package, feature update package, minor and major update package. Bug fix update package is an update package which contains only bug fixes where feature update package is an update package which contains only new features introduced to the system. Minor and major update package is an update package that contains both bug fixes and new features. In this research they are discussing the real world scenarios where customer make a decision to take new features from the software company or remain in the same version depending on the cost they have to allocate for it. They have define two major functions by considering customers and vendors as "customer cost/value functions" and "vendor cost/value functions" with required equations to calculate particular cost. Researches have provided ten misconceptions that software vendors can easily have by considering Dutch software organizations which provide occupations for around 1500 people. At the end of the research paper they have discuss about how cost can reduce in vendor side and also in customer side. Researches claim that process of "release package creation must be automated" as much as possible to eliminate simple manual task. They say that if a release package is checked for completeness automatically each time a release package is created, it does not need to be checked extensively by quality assurance, eliminating a large part of this process. Also they mention that cost of software delivery is greatly minimized

if “all delivery is done through a network” instead of expensive media like CDs or DVDs. Another reason is that releases stored on these media are never as up to date as the vendor’s release package repository. According to researches in the customer software development costs can be reduced for the customer by automating the update process. This requires the software vendors to seriously invest for an update tool and to develop its architecture. These customizations should remain functional after an update. Also the vendors should inform about this update to their customers in more practical manner using news letters, customer days, and e-mails.

2.4.3 Flexible Release Management Systems for Component Based Products

Two members from University of California at Irvine and University of Colorado at Boulder conduct a research with the idea of defining flexible release management process and implementing a built specific tool in the context of distributed component-based software development [1]. According to them in the component based software development domain, software systems are created by integrating independently developed, pre-existing components. These common components are developed by various companies located all over the world. Continuously they make improvements to their components and release new versions. As they mentioned in their research, heart of the software release management is the notion of dependence. In order to proper functioning of a particular component rest of other dependent component should be up-to-date. According to researches understanding a component’s dependencies is complicated by the fact that components may be developed by different organizations, that those organizations autonomously control the release of new versions of their components, that each version of a component may have different dependencies, and that dependent components may themselves be complex component-based systems. Therefore researches claim that documenting dependencies accurately and then using the documentation to both drive and constrain software release management, is critical for supporting developers and users of component-based software. As a solution they introduce their Software Release Manager (SRM) tool in their research based on two key notations. First one is while components can be released from physically separate sites; the actual location of each component is transparent to those using the SRM. Second one is dependencies among components are explicitly recorded so that they can be understood and exploited by the tool and its users. From their tool they help developers automatically document

and track transitive dependencies. They develop this tool with the assumption of while SRM is responsible for storing and reasoning over dependencies, it itself is not responsible for generating or otherwise deriving the dependencies. Instead, when a developer releases a component, they are expected to specify the dependencies of that component. As they mention SRM tool is focused specifically on the activities that take place between the time when components are developed and when they are installed. SRM intentionally does not support traditional configuration management of source code, nor does it support the installation, configuration, activation, or run-time reconfiguration of a component-based application. As researches suggest instead, SRM forms a bridge from the organizations where components are authored and released, to the organizations where the components are assembled into an application. In this research they have implement SRM tool as a web base application. In their tool they keep central data base to store update versions and provide graphical user interface to show the dependencies among the different component as a diagram.

Gerco Ballintijn who is a member of Centrum voor Wiskunde en Informatica in Amsterdam conduct a research on release management by selecting a health-care information system developed by the Dutch software vendor ChipSoft [8]. In this research he has interview different customers who are using this particular health-care information system and identified that almost all customers require their own version of the application. According to his research these applications frequently consist of many components that depend on each other. Researcher claims that the components evolve over time to answer the changing needs of customers. Consequently, releasing these applications takes a significant amount of effort and is frequently error-prone. In their existing system they are using Intelligent Software Knowledge Base (ISKB) which stores information about all the artifacts that are part of an application's life cycle. As different customers require different version of application they keep local software knowledge base for each customer. And all those local software knowledge bases are linked to central knowledge base through internet. According to researcher in versioning ChipSoft uses Microsoft's Visual SourceSafe (VSS). In its source code repositories, ChipSoft uses mostly a file locking policy. As they mention locking source files is not perceived as a bottleneck since the development work is distributed over the developers in disjoint subsets.

Tijs van der Storm who is a member of Centrum voor Wiskunde en Informatica in Amsterdam conduct a research with the idea of automating release and delivery of software updates in the context of component based systems [19]. By doing this implementation researchers plan to deliver particular fix or the feature quickly to the customers. Normally software vendors are interested in delivering bug-free software to their customers as soon as possible. Most challenging problem is that how to deliver updates in component based faction, to the customers as they should get only features what they require. Researchers are going to present and analyze a technique to automatically produce updates for component-based systems from the knowledge extracted from build and testing processes. Updates are produced on a per-component basis. They contain fine-grained bills of materials, recording version information and dependency information. Users are free to choose whether they accept an upgrade or not within the bounds of consistency. They can be up-to-date at any time without additional overhead from development. Component-based releasing presumes that a component can be released only if its dependencies are released [21]. Often, the version number of a released component and its dependencies are specified in some file. If a component is released, the declaration of its version number is updated, as well as the declaration of its dependencies, since such dependencies always refer to released components as well. This makes component-based releasing a recursive process. As mentioned by researchers it is a significant cost associated with this way of releasing. The more often a dependent component is released, the more often components depending on it should be released to take advantage of the additional quality of functionality contained in it. Furthermore, on every release of a dependency, all components that use it should be integration tested with it, before they can be released themselves. As a solution for this researchers suggest in practice the tendency is to not release components in a component-based way, but instead release all components at once when the largest composition is scheduled to be released. So instead of releasing each component independently, as suggested by the independent evolution history of each component, there implicitly exists a practice of big-bang releasing. In their solution it uses version control system which is polled changes done by the continuous release system. Every time there is a change, it builds and tests the components that are affected by the change. Every component revision that passes integration is released. Its version is simply its revision number in the

version control system. The dependencies of a released component are also released revisions. The system explicitly keeps track of against which revisions of its declared dependencies it passed the integration. This knowledge is stored in a release knowledge base. When handling updates researchers suggest that, they are using revision identifiers while existing package development system like NIX [5] use cryptographic hashes to identify state of the component which is more aggressive than their approach.

2.5 Summary

Under this chapter author has discussed detail about the each and every phase defines in the waterfall model as a common template for software development models. And it was proven why it is important to pay more attention on release management process by considering the similarities in other available software development models. Subsequently it has discussed about several researches conduct on software release management domain by breaking them in to three different sections. By critically reviewing above researches author has identified existing problems in software release management domain and what are the actions researchers have been taken to over come this. In several researches, researchers claim that the less number of researches conducted in release management area result to have less improvement in this domain over the past few years. Also they have mentioned that the time has arrived to pay more attention on software release management process because of rapidly growing IT industry. In addition to that following are some key points author has identified form those researches. Absence of the proper release management standard, dependencies among component based systems are complex and difficult to handle, continuous releases take significant amount of efforts in terms of time and resources. Among these issues author has identified that spending considerable time and resources on release process is becoming a vital problem in software development process over several years.

Chapter 3

Technology behind in Software Release Management

3.1 Introduction

In the previous chapter it has discussed in detail about waterfall model which was commonly used as software development process over the years. Subsequently it discussed about different kind of researches conducted in software release management domain under three main topics. Under that it has discussed different approaches, technologies they have taken and their limitations. In this chapter it is going to discuss about existing tools and technologies available for software release management with their advantages and disadvantages. At the end of this chapter it will discuss about suitability of using multi agent technology for software release management automation.

3.2 Technologies already in use

In this section it is going to discuss about various tools available in the software release management domain which are currently used by different software companies. Under each heading it will be describing the some specific features those tools are having over the other tools. In addition to that it is going to discuss about the Multi-agent systems technology with reference to this problem identified in software release management process. It also critically reviews the specific characteristics of Multi-agent technology over the other conventional software tools available in the software release management domain.

3.2.1 CruiseControl

CruiseControl [4] is one of the leading software tool available in software release management locale that can be use to simplify release process. It is both a continuous integration tool and an extensible framework for creating a custom continuous build process. It includes dozens of plug-ins for a variety of source controls, build technologies, and notifications schemes including email and instant messaging. A web interface provides details of the current and previous builds in CruiseControl. CruiseControl is written in Java but is used on a wide variety of projects. CruiseControl is available for download in three distributions as binary distribution, windows installer and source distribution. Binary distribution is a zip file containing

CruiseControl with a sample project which is prepared for run. This considers as the most popular distribution and recommended starting point when using the CruiseControl for the first time. Windows installer has the same content as the binary distribution, additionally which add a service to the windows service. In source distribution it contains number of contributed elements that aren't part of the precompiled distributions. These include the distributed builder, plug-ins that requires external libraries.

CruiseControl is composed with three main modules as build loop, jsp reporting and the dashboard. The build loop is the core of the system which triggers build cycles and notifies to various users by various publishing techniques. Configured xml file is used to map these build cycles to various tasks and depending on the configuration it may produce build artifacts. The jsp reporting application allows the user to browse the results of the builds and access the artifacts. The dashboard, which is a web interface that provides a visual presentation of all project build statuses. Architecture diagram for CruiseControl is as shown in Figure 3.1 below.

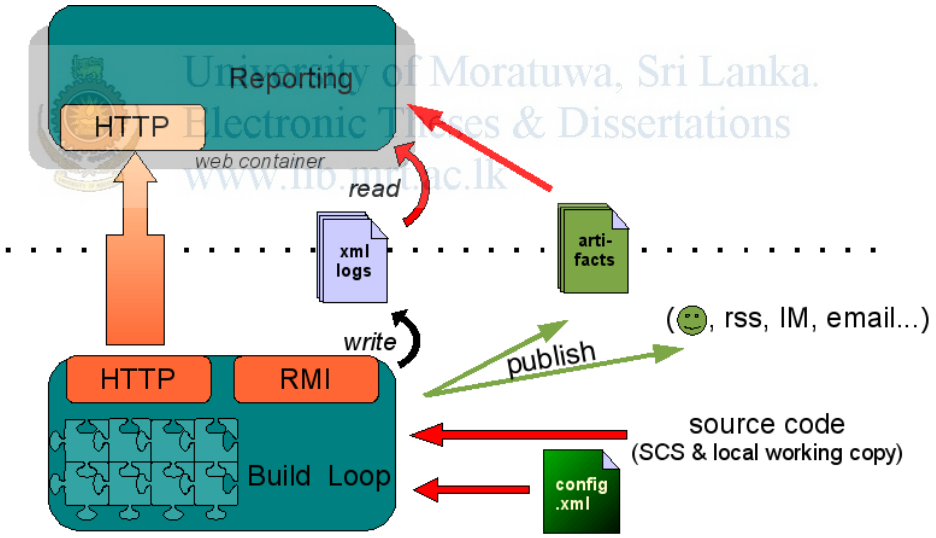

Figure 3.1: Architecture of CruiseControl

In order to work ‘CruiseControl’ correctly it require build tool like ‘Apache ant’ and version controlling system like TortoiseSVN. Currently it available various types of plug-in which can simply integrate to ‘CruiseControl’ and customize its functionality according to the requirement. It is one of the key points for ‘CruiseControl’ to be popular among software industry. Release completion e-mail alert service, copping required files to customized locations, meaning full log messaging functionality are some of the additional features that subject to improve its popularity. But it is

required human interaction to identify dependency among projects. As upon the modification done to the projects dependency can change. So it can not be a fixed value. It should adjust according the particular modification.

As ‘CruiseControl’ is conventional software it is not possible to expect that it will behave according to the dynamic situation. When it request to compile and build the project it blindly execute the instructions what are given. If it forgets to give updated version of a project, it will continue with available previous version and at some point build process get fail. Then person who is responsible should look at the error given by the ‘CruiseControl’ and should take necessary action for it. Sometime he may need to update depend properties and do the release from the beginning. Another downside of ‘CruiseControl’ is that it does not have the intelligence to identify in which order projects should be built. In practical scenario this is done by manually. Responsible developer will communicate with other parties in other projects and come in to an agreement in which order they have to build their projects. On the other hand when printing the error message ‘CruiseControl’ use the web console. This error message can not be read and understand by a machine. To understand what is the error is human interaction is required. Sometime the error what is showing in the console is not the actual error. At particular situation human experience is essential to identify what actually it means.

3.2.2 Maven

Maven [3] is also another popular release management tool which has some advanced features over ‘CruiseControl’. As oppose to ‘CruiseControl’ ‘Maven’ is capable to figure out the latest release version from the common location. ‘Maven’ has additional feature as opposed to ‘CruiseControl’ where ‘Maven’ is capable of working with multiple repositories. Using ‘Maven’ it can define global repository location which resides over in different location other than local repository. Most probably this can be web URL. Developer who execute the release is supposed to update a property with the information what is the latest version other developers have to use when compiling their projects. Once ‘Maven’ starts the building process it will retrieve latest .jar file from the common location. But ‘Maven’ tool also unable to find the dependency among projects when they are in compilation [17] and building. In the middle of the process ‘Maven’ also get fail and some kind of human interaction is required to get rid from it.

3.2.3 Multi Agent Systems Technology

In the Multi-agent systems it contains several agents that work to achieve a common goal. These agents are autonomous and decentralized. There is no central control location available to handle the behavior of the agents. All the agents are work to achieve the common goal by passing messages. Multi-agent systems can be used to solve problems which are difficult or impossible to solve by conventional systems. Multi-agent systems also referred to as self-organized systems as agents are capable of adjust their behavior to comply with current status of the environment. When modeling any real word problem using Multi-agent technology it can consider on four main aspects as shown in Figure 3.2.

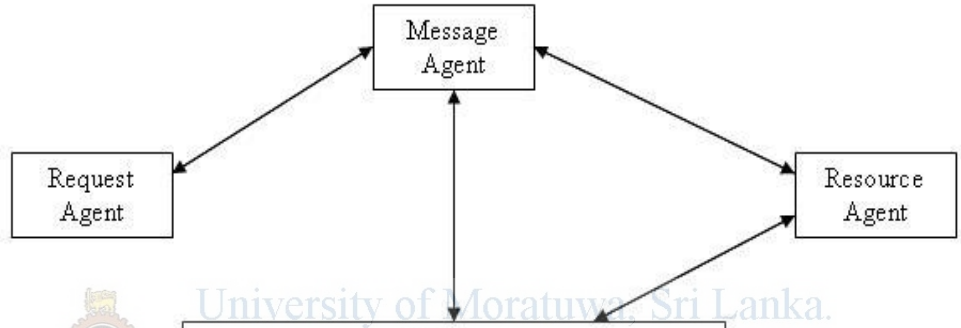

Figure 3.2: Architecture of Multi-agent system

The message agent and the ontology can be up and running through out the lifecycle of the system. Up on the request resources are allocated and once they done those resources will be released. In the next subsections it will discuss about agent communication, ontology and their lifecycle which are the key areas in Multi-agent system technology.

3.2.3.1 Agent Communication

In Multi-agent systems communication among agents are done by message passing. These messages consist with simple texts and simple keywords which can be understand by using simple rules. In most cases the intended recipient of the message is unknown at the time of sending message from one agent. In rare cases messages might send directly from one agent to another. To over come the situation where recipient is unknown Multi-agent systems technology suggests to keep common message board where the entire agents send their messages to it. As a result of that

agents can get to know what is happing in the system by going through the messages in the common message board. If the context of the message in the message board is match with particular agent's context it will start to take suitable action on it. In addition to that agent will update the common message board by saying that I have taken this request for execution. It helps to avoid the processing of same message, by another agent. Agents communicate by using a common agreed language and a common protocol which helpful to understand by every in the same context. Agent Communication Language normally known as ACL is one of the most commonly used language which is define by the FIPA [7] organization.

3.2.3.2 Ontology

Ontology is a formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts. It is used to reason about the entities within that domain, and may be used to describe the domain. In general ontologies can be any data or knowledge structure such as database, xml file, web page or a text file. The concept of ontology is becoming increasingly important factor for agent based systems as it can use to define the behavior of the agent. Normally ontology comes as complex network. These networks can be modified, updated, combine with other ontologies by using different toolkits. As ontologies are accessed by different parities it is required to keep them in an understandable format for external parties also. In general ontologies can be classified in to two major parts as domain ontologies and upper ontologies. Domain ontology models a specific domain, or part of the world. It represents the particular meanings of terms as they apply to that domain. An upper ontology is a model of the common objects that are generally applicable across a wide range of domain ontologies. It employs a core vocabulary that contains, the terms, and associated object descriptions, as they are used in various, relevant domain sets.

3.2.3.3 Lifecycle of Agent

In typical Multi-agent systems agents are created up on the request. At the initial stage there are no agents in the system. Once the system starts agents are created. Agents communicate among each others by message passing and do the appropriate action depending on the situation. If particular agent fails some operation in the middle of it another agent can start doing the reset of it as all the agents are capable of

doing any task in the particular domain. In the conventional software this can not be achieved as each components of the system are specifically developed to do a specific task only. Conventional software tools are developed to work in a sequential manner where out put of one task required to continuation of the next task. In conventional software if one task get fail then whole system get fail as all the tasks are depend on each other. As opposed to that Multi-agent systems are capable of continuing its work either one of its agent fail to execute particular task as they are not heavily depend on each other. They are only depend with each other with simple rules that are used when massage passing. In general terms it can be said that Multi-agent systems are capable of working with out any problem in single point of failure while conventional systems are not.

3.3 Summary

Under this chapter it has discussed about most commonly available software tools like CruiseControl and Maven. When discussing about those tools it explain about the architecture of release management tools by providing diagrams when it is applicable. Subsequently it has discussed about advantages and disadvantages of those tools by comparing over the other. Also it discuss about the main factor that those tools can not support for fully automated system with the way those are currently implemented. Finally it discussed about Multi-agent systems technology in detail by considering major areas like agent communication, ontology and agent lifecycle. When describing agent's lifecycle it has discussed why it is possible for agent to work with dynamically changing and complex environments. In the next chapter it will discuss detail about the approach to implement the system.

Chapter 4

Multi Agent Approach for Release Management

4.1 Introduction

In previous chapters it has discussed about different technologies currently use in software release management process. Also it has identified advantages and disadvantages of those tools by comparing with each other. Subsequently it discussed about Multi-agent systems technology and its features under three main sub headings. In this chapter it is going to propose the approach to automate software release process using the Multi-agent base technology mentioned in the previous chapter.

4.2 Proposed Solution

In this section it is going to discuss about the proposed solution in terms of Input, Output, Process, Users and Features. It will discuss about what are the information insert to the system when starting. Then it discusses about the outputs, expected to have from that input data. Next it will discuss about the process which are useful to convert that given input as outputs which are meaning for the system. Users and some highlighted features of the system are discussed in the later part of this chapter.

4.2.1 Inputs

Date of the previous release and the URL of the repository will be the inputs for the system. These values saved in a .properties file and system will read them when release process starts. Once release successfully completed system will save that date in .properties file and which will be useful when running the system next time. To do this it has used Java programming language with the help of toolkit classes given by the SVNKIT. Java programming language provides a support to read/write data from the file system where SVNKIT is useful to create a link to the SVNRepository.

4.2.2 Outputs

Set of build .jar files of the modified projects will be the out puts of the system. For each project there will be a separate .jar file. When building depends projects previously build .jar file is used by following project to generate its .jar file. When generating .jar files sequence they are building is important. To finalize the order they have to build, it uses the JADE tool kit which is developed by using Java

programming language. To execute build instruction commands programmatically it use Apache Ant tool which originally develop to support for Java base classes. Apache Ant tool is scripting language which writes in .XML format that support plenty of Java commands normally executed from command prompt.

4.2.3 Process

When user initiates the system it will read the last release date and repository URL which is saved in .properties file. Then it will connect to repository and fetch all the changed projects from that specified date to today's date. Separate agent will be created to each modified project. Those agents will communicate and come to an agreement in which sequence they have to build. Once it finalize respective project will generate .jar file according to the agreed sequence. In the process section it uses Java programming language to initiate the application. Creation of different agent is done through JADE toolkit with the support from Java programming language. Each agent will execute Apache Ant build scripts to compile and build .jar files.

4.2.4 Users

Users for this system can be software developers who are responsible on release process. Administrators or Team leads also can consider as users of this system.

4.2.5 Features

By considering with existing release management tools this application is developed to work with out any human interaction. Changed modules and sequence they have to build is identified by the application itself. To do that features in the agent technology was useful. This system can be used to reduce the usage of network bandwidth as it sends one single request to the resource repository. While in the existing system each person in the project sends a request to the resource repository. That will lead to have high network bandwidth problems. Performance of this application will be more with compared to existing tools as in existing tools they checkout large number of files from the repository while in new solution it only checkout the modified files.

4.3 Functional Specification

- Once system started there should be Graphical User Interface which gives some idea about the progress of the system to the end user.

- System should continue if there are modified files exists to compile and build only.
- If no files are modified with the time duration system should terminate by giving meaningful message to the end user.
- Agents should create up-on the request and details of the created agent should show to the end users.
- Once agent created small dialog should display with communication message details.
- Once agents' assigned task are done agent should terminate by giving meaningful message to the end user.
- Each message pass between different agents publish on message board and agent itself, which will be helpful for the user to identify progress of the system.
- Modified files should only check out to the local machine from the remote repository.
- There should be progress message in the message board after each communication happens in between agents. That help user to keep track how dependency modified after each message passing.
- Compile and build the jar files should generate with the name of the particular module, which is helpful to identify correct .jar file for the specific project.

4.4 Summary

In this chapter it has discussed about the proposed solution in terms of Inputs, Outputs, Processes, Users and Features. When discussing about Inputs, Outputs, Processes it has also provide the relevant software tools it is intended to use when implementing the system. Also it discuss about key functional specifications are expected to have for this system. In the next chapter it is going to discuss about analysis and design part to the Software Release Management tool.

Chapter 5

Design of Release Management Tool

5.1 Introduction

In the previous chapter it discussed about set of Inputs, Outputs, Process, Users and Features by providing detail information about the tools it is going to use. In addition to that it has provided list of functional specifications which supposed to have when implementing this system. In this chapter it is going to give top level design diagrams with set of modules and describe how each module work. By taking the particular design diagram in to considering it will discuss about how overall system work with the collaboration of each other.

5.2 Analysis and Design

In this section it is going to describe about the key information author has gathered by analyzing the system and the design it come up with. In the existing process there can be two or more release per week. In this each release only modified modules will build and release to the customer. If there are dependency exists among the modules responsible person will communicate with other parties and identify suitable release order. There can be scenarios where developer blindly starts the release process with out considering about other parties. When release get fail at the middle of the process they identify that there is a dependency need to update. Then it has to start from the beginning. By analyzing existing system author has design a solution as shown in Figure 5.1.

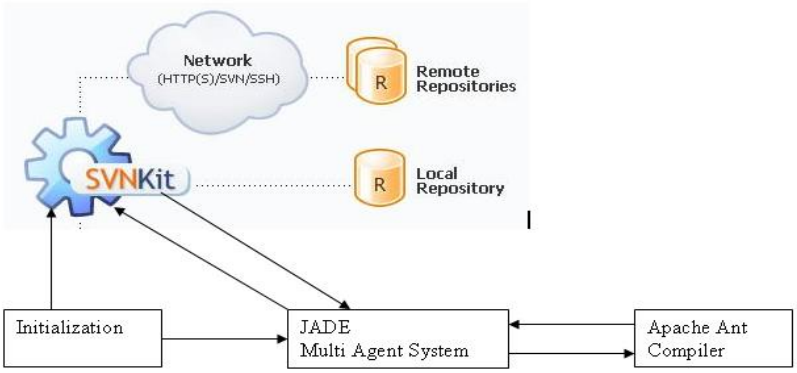

Figure 5.1: Top Level Architecture of the Proposed Multi Agent System

5.2.1 Initialization

In the initialization stage system will gather all relevant information to create agents in the next stage. In the initialization system will read the last release date and SVN repository URL from .properties file and retrieve the history data from the repository. This history data contains the file path user has change, comment which is added when committing the modification, version of the revision and the person who done the change. Set of major operations happen in initialization can be categorized as follows.

5.2.1.1 Retrieving release dates

When retrieving release date it is required to have two values. One is the last release date and the other one is current date. It is design to store last release date once the particular release successfully executed. This will save as in a property file. When it is requested by giving the key of that value system will go thorough the file system and return the saved value in the property file.

5.2.1.2 Setup repository libraries

When dealing with SVN repository it is required to have initialized different protocols available. Most commonly available protocols are Hyper Text Transfer Protocol, subversion repository factory protocol and file system protocol. In order to communicate with SVN repository these protocol libraries should be setup by the system. Hyper Text Transfer Protocol which commonly known as http is the most frequently used and popular protocol among other as it is the protocol for World Wide Web accessing.

5.2.1.3 Handling authentication

When dealing with SVN Repository it is required to have authentication as the repository is secured by providing passwords to avoid unauthorized access. SVNKIT provide an interface to get authentication from SVN repository by providing set of predefined inbuilt classes. In addition to that it provides a support to create SVN repository instance by providing the URL it is intended to have with anonymous access. Detail of how these design parts are implemented is described in the implementation with appropriate source code examples.

5.2.1.4 Retrieving history data

In SVN repository version of the data can be retrieved by given the version number or the date of the modification. When some one update some information in a file and upload to SVN, it will keep the date of the modification, author of the modification, version of the modification, detail modification and the comment value inserted by the user. With the help of previously retrieved data such as release dates, repository protocols and authentication system will identify history data available in the SVN repository.

5.2.2 JADE Multi Agent System

As discussed in the Initialization section as an out put it gives the modified file details between last release date and today. Under this heading it is going to discuss the design of the multi agent system on several major sub sections.

5.2.2.1 Agent creation

Once it identifies modified file information it is required to create separate agent for each modified project. When creating agents unique name is used for each agent to identify them separately. As only this program is running on the JADE platform it is decided to create agents on the default container. Containers provide by the JADE is used to perform agents operations. For a single agent there can be only one container where it can live. When design the agents features this agent is extend by Agent class provided by JADE toolkit. This design of the classes for proposed system is shown in Appendix C.1. In the Appendix C.2 it shows sequence of creating agents by operation number 7.

5.2.2.2 Update local files

Once the system is done with creating agents it is needed to update local machine with the latest files from the SVN repository. In this system it design to implement that scenario with support provide by the SVNKIT. This system first call to the SVNKIT to checkout specified file set from the SVN repository. SVNKIT will communicate with SVN repository and execute the operation request by the system. It is design to execute this operation once agent creation is done as each agent need to get the latest changes from repository in order to compile and build the jar files. In the Appendix C.2 it shows sequence of updating local repository by operation number 8, 9 and 10.

5.2.2.3 Agent communication

Under this section JADE toolkit is responsible for handle communication among the agents created in previously. As all the agents are in the same container it is easy to communicate among agents with simple algorithms. As shown in Appendix C.2 once the updating of local machine get completed agent start to communicate with each other to identify dependency among them. It is a cyclic flow which happens only among agent until they found an appropriate solution.

5.2.2.4 Agent deletion

When creating an agent system will allocate memory space for the agent to perform its task. When there are plenty of agents created, collection of the memory allocated get increase. Once the agent is done with its work it is better to delete the agent from the system as it will help to utilize memory usage. That will result to improve the performance of the system. To do this it is design to make use of the inbuilt support which is provided by the JADE toolkit. Once system request to delete the particular agent JADE will delete the agent and release its used memory as well.

5.2.3 Apache Ant Compiler

When designing the system it is plan to separate compilation and building jar file section from other components as it is required to get support more on Apache ant software. On the other hand it is required to have compiling and building support for all the agents as all of them have their own responsible project to build. As shown in Appendix C.2 compilation and building will execute under operation 12 (CompileAndBuild) once the operation 11 (CommunicateAndIdentifyBuildOrder) is completed. Design can be divided in to two main sub sections as Invoke ant targets and Build jar files.

5.2.3.1 Invoke ant targets

When compile and building the source code it has design to have one default target on the build script and invoke rest of the targets from the Java programming language itself. By doing so it was able to reduce the coupling between ant script and the implementation.

5.2.3.2 Building jar files

When building jar file it is needed to write a common script for all the projects. Otherwise it has to write separate script for each and every project. It is not practical and it is difficult to maintain. Some time it is not possible to create separate scripts as the numbers of projects are high.

5.3 Agent's Lifecycle

When considering design of the agent's life cycle in this system agents are created upon the request. As it is mentioned in the initialization state system will identify what are the modules which are modified during that time period. Number of agents is equal to number of projects which were modified. These agents will communicate with each other to execute its task. This is done through a message board. Created agents will directly link with resources (projects, SVNRepository) and execute its task. Status of the agents will update in the message board. When updating the message board it is used to define the type for the message. For example if it is just information from an agent then the type of the message is information. If it is a service requesting by one agent then the type of the message is request. Once particular agent's task complete it will terminate by it self. In Figure 5.2 life cycle of the agent is shown.

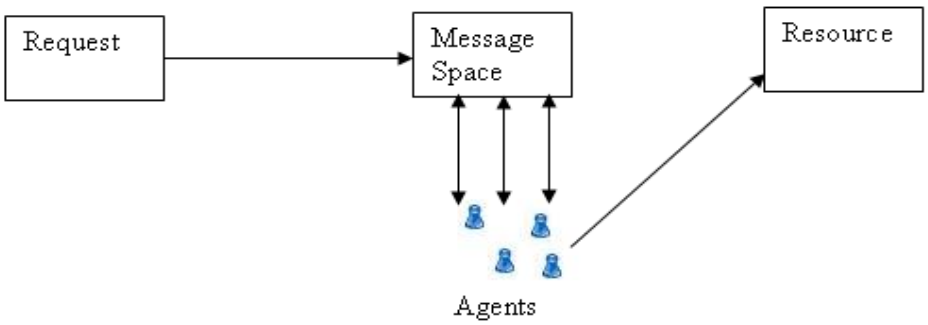

Figure 5.2: Lifecycle of the Agent

5.4 Class Diagram and Activity Diagram

Class diagram for the implemented system is shown in Appendix C.1. As this system is implemented using JADE toolkit all agent related classes are extended by them. By doing so it was easy to reuse the inbuilt features provided by JADE tool for agents.

As shown in the class diagram StartUp class is the main class which starts the system. StartUp class will initiate MessageAgent instance which is a sub class of RMA class. RMA is extended from ToolAgent where ToolAgent is extended from Agent class. RMA, ToolAgent and Agent classes are built in classes which are provided by JADE toolkit which is used to inherit multi agent features. SVNHistory, SVNRepository, SVNUpdateClient, SVNReviosion are the key classes that are used to communicate with SVNRepository using Java programming language. In addition to that there are some ToolKit classes which support to fulfill special functionalities. Sequence diagram for the implemented system is shown in Appendix C.2.

5.5 Summary

In this chapter it has discussed about key modules like initialization, JADE multi agent system, Apache ant compiler by considering the top level architecture provided to the Multi-agent system. Under initialization section it has discussed design of the system to retrieve last successful release date, setting up the repository libraries and authentication. Under JADE multi agent system section it has discussed about the design by taken in to consideration on agent creation, local file update, agent communication and agent deletion. It has provided appropriate appendixes when ever possible under each heading. Under Apache ant compiler section it has discussed about how author design to invoke ant targets using Java programming language and ant build scripts. In addition to that author has discussed about the life cycle of the agent which relevant to this implementation. Class diagram and Sequence diagram for the implemented system also included to have better idea about the system implementation. In this next chapter it is going to describe how these designs have implemented in more technical level.

Chapter 6

Implementation

6.1 Introduction

In the previous chapter it is presented the top level architecture of the design in a graphical representation. Under that chapter it has discussed about detail on each component mention in the top level architecture diagram. Subsequently it has discussed about lifecycle of the agents with relevant to this solution. In addition to that it is presented the design of the Class and Sequence diagrams in a different subsection by providing some brief description about each class available in the class diagram. In this chapter it is going to discuss how each module which is mentioned in the design chapter is implemented in a detail manner by providing appropriate source code examples.

6.2 Initialization

In the initialization stage system will gather all relevant information to create agents in the next stage. Under this stage it will create SVN repository instance with appropriate authentication and protocols. Finally it collects all the log information which is required in the provided time period. Details of each of those functionalities are discussed in the following sub sections with appropriate source code examples.

6.2.1 Utility classes

When implementing this system it was required to perform common operations in the source code under different classes. If those operations are implemented in each and every location there can be code redundancy occur. Size of the source files get larger unexpectedly. Maintainability of the code get reduce as if it need to change some behavior of that common operation, have to update it in all the places. Some places may neglect and time taken for the operation gets increase as number of locations get increase. Debugging the source code get complex and time consuming as different places has duplicate codes. Identifying an issue in one place also get difficult as each and every location should deeply inspect.

These utility classes are most commonly useful when handling operations with String, DateTime, and Files. Those classes are commonly known as ToolKit class when writing the source code. For example when reading a file content by giving the file

name can be consider as common method as the only thing it change there is the file name. This is same for writing to a file. FileToolKit is such a utility class which contains set of following methods,

```
public static String [] readContent(String path)
public static void writeToFile(String content, String path)
public static void updateToFile(String key, String content, String
path)
public static String getValueByKey(String key, String path)
public static void deleteFiles(String path)
```

In the above examples first method is used to read file content by giving the file path, second method is used to write to a file by giving the content need to write and the file path, third method is used to update the file by giving the key, content and the file path, forth method is used to get the specified value contain in the file by giving the key and the file path, fifth method is used to delete the set of files and sub folders by giving the folder path. In StringToolKit, DateTimeToolKit classes' similar kind of utility methods were implemented depending on the requirement to handle String and DateTime values.

6.2.2 Retrieving release dates

In this implementation it is require current date and last release date to get modification details from the SVN Repository. Information of the last release date is saved as a property file at the end of the each success build. FileToolKit utility class is used to read that value from the file system as mentioned in the previous section. This value is passed to DateTimeToolKit utility class to get a DateTime object which can understand by Java programming language. Current datetime also requested by using the DateTimeToolKit class by giving the specific format which is supported by Java.

6.2.3 Setup repository libraries

When connecting to the SVN Repository it is required to identify in which protocol file modifications are saved. In general there are three different protocols under SVN Repository that used to save information of file modifications. The first important and most frequently used protocol is about hyper text transfer protocol indicates by http://

and https://. This initialization is set up by executing DAVRepositoryFactory.setup(). The second protocol is about SVN Repository factory indicate by svn:// and svn+xxx://. Initializing this protocol is done by executing SVNRepositoryFactoryImpl.setup(). Third protocol is about file system which is indicates by file:////. Initializing this protocol is done by executing FSRepositoryFactory.setup(). In this implementation as it is keeping local SVN repository file system protocol can be used. But when it is dealing with remote repository over the internet http or https should be used.

6.2.4 Handling authentication

When dealing with SVN Repository it is required to have authentication as the repository is secured by providing passwords to avoid unauthorized access. To do this SVNKIT provide special set of classes, those can call thorough Java. Example code for create default authentication manager is as follows,

```
ISVNAuthenticationManager authManager = SVNWCUtil.CreateDefault
AuthenticationManager(name,password);
```


By executing above it request to create a default authentication manager by giving the user name and password. In this scenario it provides anonyms user name and password as it is connecting to a local repository.

Once it creates the authenticationManger it should be set to the SVNRepository instance, system is going to use. Example code for creating SVN Repository instance and setting authenticationManger are as follows,

```
repository = SVNRepositoryFactory.create(SVNURL.ParseURLEncoded(url)
);
```

Here it pass SVNURL instance to create() in the SVNRepositoryFactory class to get and repository instance for the provided URL.

```
repository.setauthenticationManger(authManger);
```

By setting the authenticationManger it provide privilege to do any operation with SVN Repository.

6.2.5 Retrieving history data

In this step it is going to retrieve the repository history data with the help of previously processed values. Release dates which were identified in previous steps using utility classes will be used under this section to get relevant revision numbers. In SVNReposiory it keeps version information against the date where it generates unique revision number for each update done for the repository. Further it keeps the author, names of the files modified and some comment insert by the developer. Example code for retrieving startRevision and endRevision values are as follows,

```
endRevision = repository.getDatedRevision(this.m_toDate);
startRevision = repository.getDatedRevision(this.m_fromDate);
```

In addition to release dates, here it is using the repository which is created in previous step by giving default authentication privileges. After system gets this revision interval it passes it as follows to get log entries. In the log entries it contains paths of the modified projects, person who done the modification, revision number and comment user has added. Code example is as follows,

```
logEntries = repository.log(new String[] {""}, null, startRevision,
endRevision, true, true);
```


As it shows there are six arguments which have to give, to retrieve log entries for the given revision period. SVNKIT is return log entries as a collection data set. In that data set there can be set of different entries which refers to same file but as different log entries. In the processing stage it is only need to know what the files which are modified are. Therefore duplicate entries for the same file should be removed. To do this StringToolKit utility class is used as it can check equality of the file paths which are modified.

6.3 JADE Multi Agent System

As discussed in the Initialization section as an out put it gives the modified file details between last release date and today. In this section it will explain how lifecycle of the agent precede with appropriate source code examples.

6.3.1 Agent creation

In this system agent are created up on the request. If there are no modification found no agent will be crated and system will terminate as there is no work to execute.

Separate agent will be created for each modified module with the details of modified files with in that module. Example source code implemented for creation of agent as follows,

```
super.newAgent(modifiedProjectName,"se.agents.util.ModuleAgent",new
Object[]{entriesList.toArray(new String[entriesList.size()])},"");
```

In the above code example it passes four arguments to create a new agent instance with in the JADE toolkit. As the first argument it passes the name of the agent which wants to create. In this implementation it create separate agent for each modified project. Therefore system will pass the name of the modified project as the first argument. As the second argument it passes the package path of the agent class name. This class contains set of inbuilt methods provided by JADE toolkit to handle agent behavior. Under those methods it implements the own specific functionalities to be aligned with the project requirements. As shown in the class diagram in Appendix C.1 ModuleAgent is extended from the Agent class which is provided by JADE toolkit. By doing so it will automatically inherit the Agent features to the customized ModuleAgent class. As the third argument it passes the object array which contains the information about the modified class names with respect to that modified project. As the fourth argument it passes empty String value as the container name where it is used to identify in which container, this particular agent needs to create when there are multiple containers available. As there is no specific container, JADE toolkit will assign those agents to default container. In the JADE toolkit it does not provide any support to keep any information about created agents in particular container. As a result of that system keep a map data collection to keep information about each created agent for this system.

6.3.2 Update local files

After agent creates each agent start to check out modified files from SVN Repository to the local machine. It is required to have same SVN repository instance, created by initialization phase to checkout the repository files. As it is keeping static values it is possible to retrieve same object in two different stages. Before updating the local files it is required to delete all the files which are already exists in the local machine. FileToolKit utility method is used to delete all required files by giving the required

path. Once it is done methods provided by the SVNKIT is used to checkout the modified files to local machine. Example source code is as follow,

```
SVNUpdateClient svnUpdateClient = new SVNUpdateClient(
authentication Manger, null);
svnUpdateClient.doCheckout(svnRepository.getLocation(),new File(
sourceFileDestination), svnFromRevision, svnToRevision,
SVNDepth.INFINITY,false);
```

In the above code example it will first create and instance of SVNUpdate client by providing the authenticationManger as an argument to the system. SVNUpdateClient is inbuilt class which is provided by SVNKIT to handle update operations available in TortoiseSVN. After creating the instance it call the doCheckout() with six arguments. As the first argument it passes location of the SVN repository which is useful to identify the location where repository is. As the second argument it passes an instance of File class by giving the path for source file destination. Particular path is used by the system to copy source files from the repository. Third and fourth arguments are about from and to revision values which is useful to check out modified files with in that range. SVNDepth.INFINITY argument is used to say that check out should be done up to lowest level in the repository. Once the execution gets success all the relevant modules that are modified get updated in local machine. These latest files will be used when compiling and building jar files in later stage.

6.3.3 Agent communication

Once agent check out latest files from the repository they start communicating in order to identify dependency among them selves. Example source code for sending message from an agent is as follows,

```
ACLMessage aclmsg = new ACLMessage(ACLMessage.REQUEST);
aclmsg.addReceiver(agentsAID.get(i));
aclmsg.setContentObject(modification);
this.send(aclmsg);
```

In the above example it creates an instance of ACLMessage class which is provided by the JADE toolkit. When creating the message it define the type of the message as it is useful when processing the received message by receiver. Before sending the

message it assigns who are the receivers of the message and what is the content of the message which is going to send. As the final step system sends the message using this.send(aclmsg).

When message passing started each agent will start updating the message board about their modified files. Then particular agent (AgentA) who wants to build its project will first check what the classes in other projects I am using are. If classes it is using, in another project (AgentB) is also modified then AgentA will wait till AgentB's build complete. Once AgentB completes AgentA will start building its project using the build files generated by AgentB. If there are several parties involved on it then all of them come in to common agreement by message passing with each other. Following code example is useful to execute above mentioned behavior.

```
Class changedClass = Class.forName(classPath);
Method [] methods = changedClass.getDeclaredMethods();
Field [] fields = changedClass.getDeclaredFields();
```

From the above code segment system can identify what are the methods and fields contains in a particular class. By identifying fields system can check that this particular class is referring to classes in other projects. If so then system checks whether particular class in other project also modified. If it is modified then it can say that modified project should build before to the other project. But if particular class of the other project not modified it can build the project independently.

6.3.4 Agent deletion

In JADE toolkit it provide special functionality to delete a particular by invoking delete(). This method is originally implemented in Agent class which is an inbuilt class of JADE toolkit. In this implementation it just calls the available method provide by JADE toolkit. Toolkit itself does the reset by deleting the agent and releasing its memories. To implement agent specific features it has use JADE tool as it contains plenty of support to communicate with agents using Java programming language. Another reason to use JADE is that it is FIPA [7] standard tool.

6.4 Apache Ant Compiler

After identifying dependency order next task is to compile and generate .jar files. This can be easily done by using Apache Ant tool which is fully develop to compile,

assemble, test and run Java applications. To generate jar files through Java programming language it is required to invoke different ant targets. Code modification done to invoke ant targets is discussed under 'Invoke ant targets' sub section. Script modification done to build jar file discussed under 'Building jar files' sub section.

6.4.1 Invoke ant targets

As this system is developed using Java programming it is also required to invoke ant targets using programming language itself. Sample code to invoke ant targets is as follows,

```
Project proj = new Project();
proj.init();
proj.executeTarget("clean");
proj.executeTarget(proj.getDefaultTarget());
proj.executeTarget("jar");
```


In the above example it creates a new project instance from the inbuilt class provided by Apache ant. As the first step it initializes the project by calling init(). In order to compile and build jar files already saved class files should be removed. It is done by executing the proj.executeTarget("clean"). After that it will request to run the default ant target which is define in the build.xml file. In this scenario default target is compiling the source code. When executing that line compilation of source file get started. As the last target it will run the proj.executeTarget("jar") where it will generate jar files for the compiled classes.

6.4.2 Building jar files

In this solution what author has done is executing the build.xml file which is located inside each project, programmatically. Sample of the build.xml file as follows,

```
<target name="clean">
        <delete dir="build"/>
</target>

<target name="compile">
      <mkdir dir="build/classes"/>
```

```
      <javac       srcdir="SourceCode"       destdir="build/classes"
classpathref="classpath" />
</target>

<target name="jar">
     <mkdir dir="build/jar"/>
     <jar                destfile="build/jar/${ant.project.name}.jar"
basedir="build/classes">
      <manifest>
          <attribute                                    name="Main-Class"
value="se.agents.util.LoadAgent"/>
      </manifest>
      </jar>
</target>
```

Sample of build.xml file is shown in the Appendix B.2.

In the above example there are three major sectors it can clearly identify. When system runs the build.xml file programmatically, it is possible to specify which target ant should execute. Code examples are as follows when executing an ant target.

```
project.executeTarget("clean");
```


By executing above code system request ant tool to execute the instruction given in the <target name="clean">. Then it will delete the build directory.

```
project.executeTarget("compile");
```

By executing above code system request ant tool to create a directory call “classes” under build directory. Then compile the java files located in SourceCode folder and copy the class files to previously created “classes” under build directory.

```
project.executeTarget("jar");
```

By executing above code system request ant tool to create a directory call “jar” under build directory. Then it will generate .jar file using the compiled class files located under build/classes folder. Example of generated jar files is shown in Appendix B.1.

6.5 Summary

In this chapter it is discussed how each module’s functionality implemented by combining different technologies. Under initialization section it has discussed about

set of utility classes which were introduced by this implementation. As a common example author has taken FileToolKit class with some utility methods implemented on it. In the initialization section itself it has discussed how release date is retrieved from the file system using Java programming language. Under that section author has discussed in detail how authentication and data retrieving is handled by providing appropriate source code examples. Under JADE multi-agent system section author discussed about agent creation, local file update, agent communication and agent deletion in detail manner by providing source code examples. Under Apache ant compiler section author have discussed about set of ant script examples which are used to compile and build jar files. In the next chapter it is going to discuss about evaluation criteria for the implemented system in terms of aim and objectives.

Chapter 7

Evaluation

7.1 Introduction

In the previous chapter it has discussed detail how system is implemented using JADE, SVNKIT, Java, Apache ant, and TortoiseSVN. In the implementation chapter it divides the system in to three major sections as Initialization, JADE multi agent system and Apache Ant compiler. Under each subsection it discussed how required functionalities are implemented by giving appropriate source code examples. In this section it is going to evaluate the implemented system by going through each objective which has defined in the introduction chapter.

7.2 Evaluate Reduction of Human Interaction

One of the main objectives of this project is to reduce the human interaction to the system at its maximum. As dependency identification is replaced with multi agent technology functionality this task has been fulfilled. Now the tool itself can identify correct dependency with out human support. This functionality is implemented and tested. Evaluate it as follows;

Create four projects as ProjectA, ProjectB and ProjectC. First user does a modification on ProjectA. This modification uses two classes in ProjectC. On the other hand user modifies same classes in ProjectC. As a result of that now ProjectA depend on ProjectC. In another word ProjectC should build before ProjectA. Then user modifies several classes in ProjectB. These classes are actually use reference to set of classes in ProjectA and ProjectC. Then user modifies several classes in ProjectA and ProjectC including the classes which are referred by ProjectB. Now ProjectB is depend on ProjectA and ProjectC. Therefore ProjectA and ProjectC should build before Project B. After all these modification sequence of the projects that need to build rearranged to ProjectC, ProjectA and ProjectB. System first completed the building of ProjectC. Then it completed the building of ProjectA and finally it completed the building of ProjectB. Example for this scenario is attached in Appendix A.3, Appendix A.4, and Appendix A.5.

7.3 Evaluate Reduction of Network Traffic

Author's next objective was to reduce the next work traffic when users are dealing with building process. When users are using CruiseContrller to build projects sometimes all the team members are trying to access same build server at the same time. This will case to reduce network speed. In the implemented solution there is only one person to initiate the release process. When particular person initiate the release process relevant agents will be created automatically for each changed modules. Created agents will only communicate with each other internally. Means agents are created in a single machine and communication happen among them inside the machine only. This certainly reduces the network traffic with compared to existing system.

7.4 Minimize the time spent on Release Management task

In current system dependency identify at the release time when particular build is failed. Once build fails developer he has to go through the code modification done during previous release to now. By examine all those modifications developer has to identify what are the dependencies and what are not. Once the dependency order identified developer has to do the complete release process again. This subject to waste more time on release process as it has to do same task repeatedly. From this new solution all the dependencies are identified by tool it self at the beginning and relevant order will create by tool it self. That helps to complete release in one action. That reduce considerable amount of time as apposed to existing system.

7.5 Reduce Exceptions occur in Compilation and Build process

This objective is fulfilled along with the comments mentioned in the previous objective. As it mention system will go though the all modification and identify what are the dependencies for each module. In existing scenario developers might fail to notice some required modification done for one of their dependent modules. That will result to do the release again by considering that modification also.

7.6 Optimize CPU memory usage

When initially start the system there are no agents. Once any modification identified by the system agents will create. And they use computer memory to execute their task. Once it complete those agents get terminate from the system. Then those used

memory is released. That subject to utilize the computer memory allocate for agents. An example for release the memory is shown in the Appendix A.1 and Appendix A.2. In Appendix A.1 it shows memory usage taken to implemented tool in red color and in Appendix A.2 it shows that particular memory allocation is removed from the system.

7.7 Summary

In this chapter it evaluates the implemented solution on five major accepts. As the first objective it evaluates the reduction of human interaction by implementing this system. To evaluate it author gave simple example with three projects which are modified by making them dependent on each other. Once system executed, order of the projects are identified by the tool it self without any human interaction. Once build order is identified by the tool it self it automatically start the building jar files without any compilation problem. By doing so author was able to say that the first objective is achieved. As the same manners author has evaluate reset of the objectives as follows. Reduce network traffic is achieved by sending one request to processing place and start execution internally using several agents. Minimize the time spent on release management task is achieved by identifying dependencies in advance. Reduce Exceptions occur in Compilation and Build process is achieved by identifying dependencies in advance. Optimize the computer resources are achieved by managing the computer memory usage. In the next chapter it is going to discuss about conclusion on this solution with respect to discussed evaluation criteria. Also it is going to discuss about further improvements which can be done for this system to get the maximum use of it.

Chapter 8

Conclusion and Further work

8.1 Introduction

In the previous chapter it has discussed each and every objective in a separate topic to make sure that it has been achieved. When going through each objective author has given appropriate screen shots as appendixes to show that particular objective is achieved. In this chapter it will discuss about the final conclusion about the implemented system, using Multi-agent systems technology. Subsequently it will discuss about set of problems which are identified while implementing the system in a separate heading call Problem encountered. After that it will discuss about limitations and further works of the implementation as separate sections.

8.2 Conclusion

Author's major aim was to develop a solution to automate software release management tool with some intelligence in order to reduce human interaction. In addition to that author has identified some major limitation of the existing release management tools by going through the several researches conducted in that area. Following are the objectives it was planed to fulfill when developing this system.

8.2.1 Reduce Human Interaction

One of the main objectives of this project is to reduce the human interaction to the system at its maximum. As dependency identification is replaced with multi agent technology functionality this task has been fulfilled. As discussed in the evaluation chapter it was able to 'reduce the human interaction' by introducing the intelligence tool with the help of multi agent technology.

8.2.2 Reduce Network traffic

When it uses CruiseContrller to build projects sometimes all the team members are trying to access same build server at the same time. This will case to reduce network speed. In the implemented solution there is only one person to initiate the release process. It was able to 'reduce network traffic' by implementing this solution as it reduce number of communication link that have to be done with outside parties.

8.2.3 Minimizing the time spent on release management task

'Minimizing the time spent on release management task' also been achieved as tool itself handle the release management task without human interaction. In practical scenario more time is wasted for release process as developers have to do the same operation over and over again because, sometimes they might forgot to update or check for dependencies. Once the compilations get failed they realize there is a problem on dependencies. In the new implementation it goes through all the modules with in the specified time duration and identifies dependencies in advance. This help to reduce the time spent on release management task.

8.2.4 Reduce exceptions occur in compilation and build process

'Exceptions occur in compilation and build process' can avoid as system will go thorough all the modification and identify exact dependencies for them before start the compilation and building. When doing the release process manually correct dependencies are identified if the communication among developers is in high level. System which is implemented is able to identify dependencies correctly if agents do the communication accurately. This will result to minimize the time taken for release management.

8.2.5 Optimize CPU memory usage

'Optimize CPU memory usage' have been achieved as agents are only creating up on the request and once they are done system will delete them from the system. The memory and other resources that have been used for agent is released by doing so. This support is provided by the JADE toolkit to simplify the agent lifecycle managing. This objective was able to achieve as it implement the system using Multi-agent systems technology. In agent technology agents are not highly depend on each other. They are independent entities where they interact with each other by message passing. Therefore once agent functionality is done it can be easily remove from the system as there is no hard link of that agent with other entities in the system.

8.3 Problem Encountered

Under this section it will discuss about different problems that have encountered while implementing this system and those are as follows. In this system it has used JADE toolkit as the agent development framework which is written using Java

programming language. Although it provides various features to implement Multi-agent behavior, there are several problems encountered while implementing the fix using JADE. In JADE toolkit once agents are registered to the container it doesn't provide a support to retrieve details of the agents register for that particular container by giving agent's name or the ID. As a solution for that it uses a common data collection to store details of each agent at the time system creates them.

When updating local files from repository it check out complete project even though there are only few files are modified. For large projects this is a performance issue as it uses network resources heavily. But in current version controlling systems it is not allowed to keep sub directories on different versions. It expected to have them in same version. As a result of that system had to delete complete set of local files before checking out the new project to the local machine.

8.4 Limitations

By evaluating the system author has identified that there are some limitation exists in the system as follows. This system is developed to work on a local repository location which is provided by the system administrator at the time of execution. But in practical scenarios repository is located on a remote location where every one can access. This system only supports to compile client side projects. But it required to extend the system to compile and build server side projects as well. There can be situations where developers update the repository with compilation errors by mistake. By the time system compiling only those is identified. It will be good to have meaningful notification for the user about the problem and the location by going through the compilation exception provided by Java compiler. Once release process started it is required to avoid committing files to the repository until release process completes. If release process starts at the middle of the file update, compilation will fail as some of the file modifications are yet to commit. To avoid that it is good if it can implement a temporary repository location to save the modification until release process completes. Once release completes, modification in the temporary will copy back to the original repository by the system itself.

8.5 Further works

This system is not strictly test with heavy work load with a slow network. It should be test in that kind of environment and should fine tune the system to work in heavy

work load. System is basically implemented using a local SVNRepository where in real scenario it can reside in different network with different access rights. Accessibility of remote repository should be implemented to make real use of this implementation. In the current system dependency among modified projects are handle by passing messages among agents. But it is good if system can keep dependency information from the previous successful release in an understandable format to use in next releases. In this system compressed jar files are copied to a common location. When new compressed files are available, system replaces them with the existing. But in release management it is needed to keep information about history files also. As a solution, it will be good if system can copy the new compressed to a well standard folder structure which can be easily identified later on.

8.6 Summary

In this chapter it has discussed about how each objectives been achieved from this implemented system. Subsequently it has discussed about several technical limitations encountered (under problem encountered section) when implementing this system. One limitation is regarding JADE toolkit and other limitation is with TortoiseSVN version controlling tool. After the problem encountered section author has discussed about set of limitation which identified when evaluating the system. Unavailability of server side compilation process, Unavailability of temporary repository location to save updates can be consider as key limitations, identified in that section. As the last section it has discussed about set of future enhancement that should be done in order to make this system more user friendly and reliable. Making the system to work with remote repositories, keeping standard folder structure to copy build jar files are some of the key points which were discussed under the further works.

References

[1] Andr´e van der H, Alexander L. (2003) *Software release management for component-based software.* University of California at Irvine, Irvine. University of Colorado at Boulder, Boulder.

[2] Apache Ant (2010), *The Apache Ant Project*, Available at: http://ant.apache. org/

[3] Apache Maven (2010), *The Apache Maven Project*, Available at: http://maven. apache.org/

[4] CruiseControl Home (2008), *CruiseControl Getting Started*, Available at: http ://cruisecontrol.sourceforge.net/gettingstarted.html

[5] Dolstra E, De Jonge M, Visser E. (2004) NIX: *A safe and policy-free system for software*. 18th Large Installation System Administration Conference, Atlanta, Georgia, USA.

[6] Eclipse Platform (2010), *Eclipse Platform*, Available at: http://www.eclipse. org/platform

[7] Foundation for Intelligent Physical Agent (2010), *FIPA specifications*, Available at: http://www.fipa.org/specifications/index.html

[8] Gerco B. (2006) *A Case Study of the Release Management of a Health-care Information System.* Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands.

[9] Hyrum K. (2008) *Release Engineering Processes, Models, and Metrics*, University of Texas at Austin.

[10] Java Agent Development Framework (2010), *Open source platform for peer to peer agent based applications,* Available at: http://www.jade.com

[11] Martin M. (2005) *Quality Improvement in Volunteer Free Software Projects: Exploring the Impact of Release Management.* Centre for Technology Management, University of Cambridge, Cambridge, UK.

[12] Service-Oriented Architecture (SOA) technology resources (2010), *Java Development Kit*, Available at: http://searchsoa.techtarget.com/sDefinition/0,, sid26_gci868274,00.html.

[13] Slinger J, Sjaak B. (2006) *Ten Misconceptions about Product Software Release Management explained using Update Cost/Value Functions.* Information and Computing Sciences Institute, Utrecht University.

[14] Software Magazine (2010) - *Configuration Management Is Key to Robust Software,* Available at: http://www.softwaremag.com.

[15] SVNKIT (2009), *Subversion clients and IDE integrations*, Available at: http:// www.svnkit.com

[16] Systems Infrastructure Software Marketing Research - ResearchWikis (2006), *Systems Infrastructure Software Industry*, Available at: http://researchwikis.com/Systems_Infrastructure_Software_Marketing_Research

[17] The Tech Terms Computer Dictionary(2010), *Compile Definition-Compile*, Available at: http://www.techterms.com/definition/compile

[18] The Trusted Technology Source for IT Pros and Developers (2002), *A Cyclic Model for Software Deployment*, Available at: http://www.informit.com/ articles/article.aspx?p=25026

[19] Tijs V. (2007) *Continuous Release and Upgrade of Component-Based Software,* Centrum voor Wiskunde en Informatica, Amsterdam.

[20] TortoiseSVN (2010), *The coolest Interface to (Sub)Version Control*, Available at: http://tortoisesvn.net/

[21] Van der H, A. L. Wolf. (2003) *Software release management for component based software.* Software—Practice and Experience, 33(1), 77–98.

[22] Weerd I, Brinkkemper S, Nieuwenhuis R, Versendaal J, Bijlsma L. (2006) *A Reference Framework for Software Product Management,* Utrecht University.

Appendix A

Appendix A.1 Memory usage in initialization

In the Figure A.1 it shows the usage of the memory when initializing the tool with red color margin by using the Windows Task Manager.

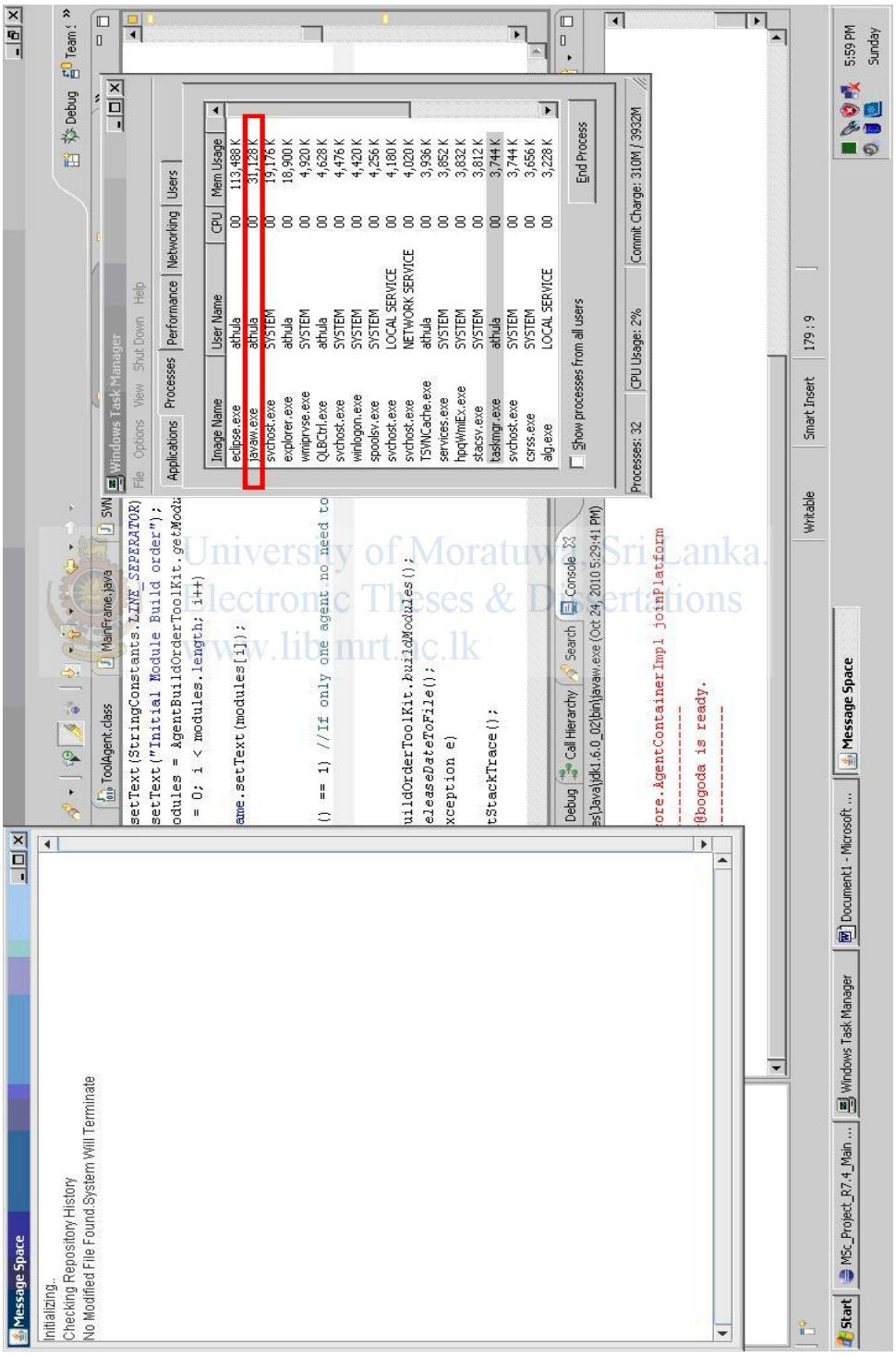

Figure A.1: Memory usage in initialization

Appendix A.2 Memory usage after termination

In the Figure A.2 it shows the usage of the memory after terminating the tool. Memory used in initialization step gets cleared from the system once system get terminate.

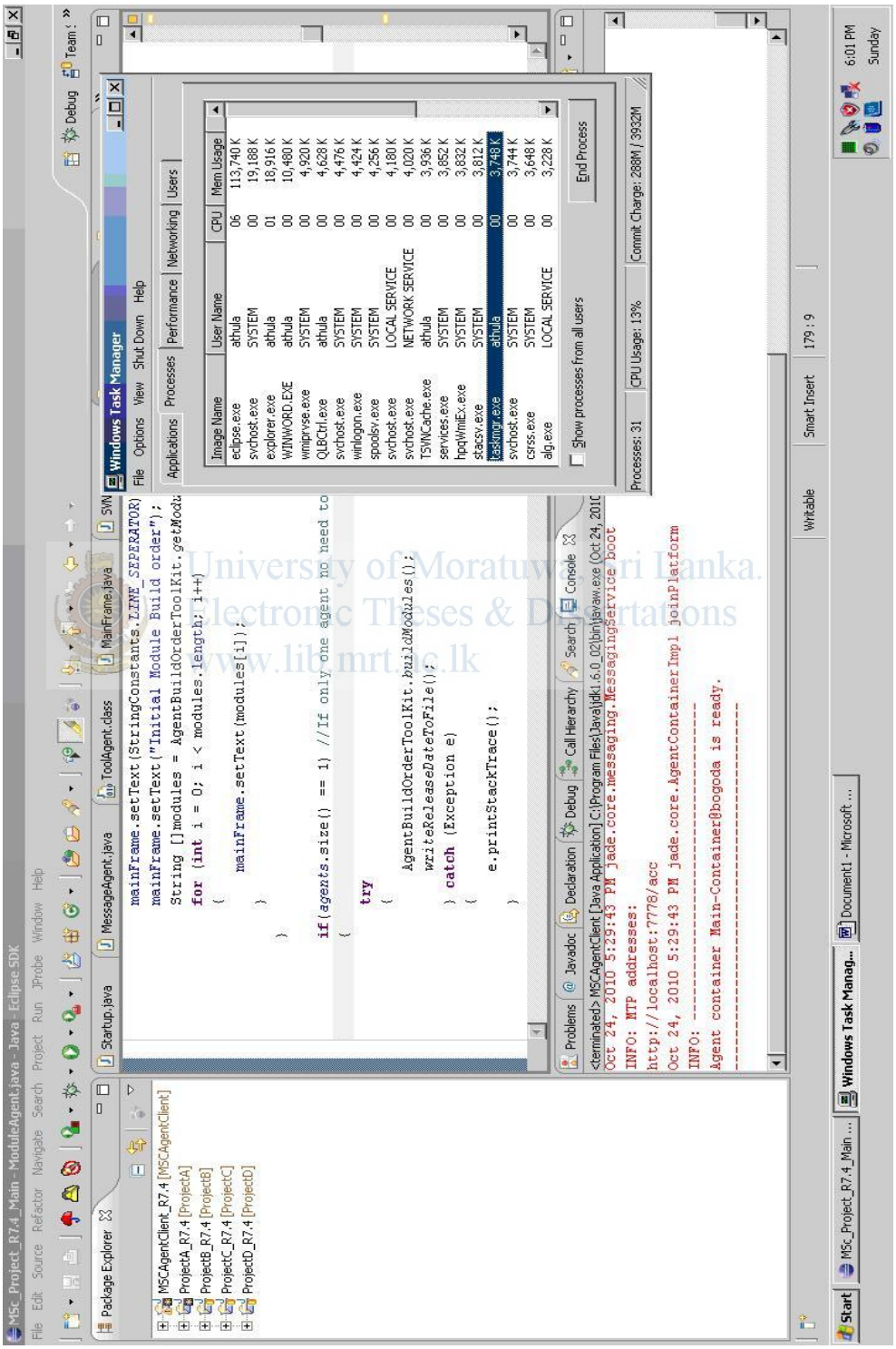

Figure A.2: Memory usage after termination

Appendix A.3 Initial state of agent creation

In the Figure A.3 it shows the initial stage of creating agents when initializing the system. As it shows, agents do not start their communication with other agents at the initial stage.

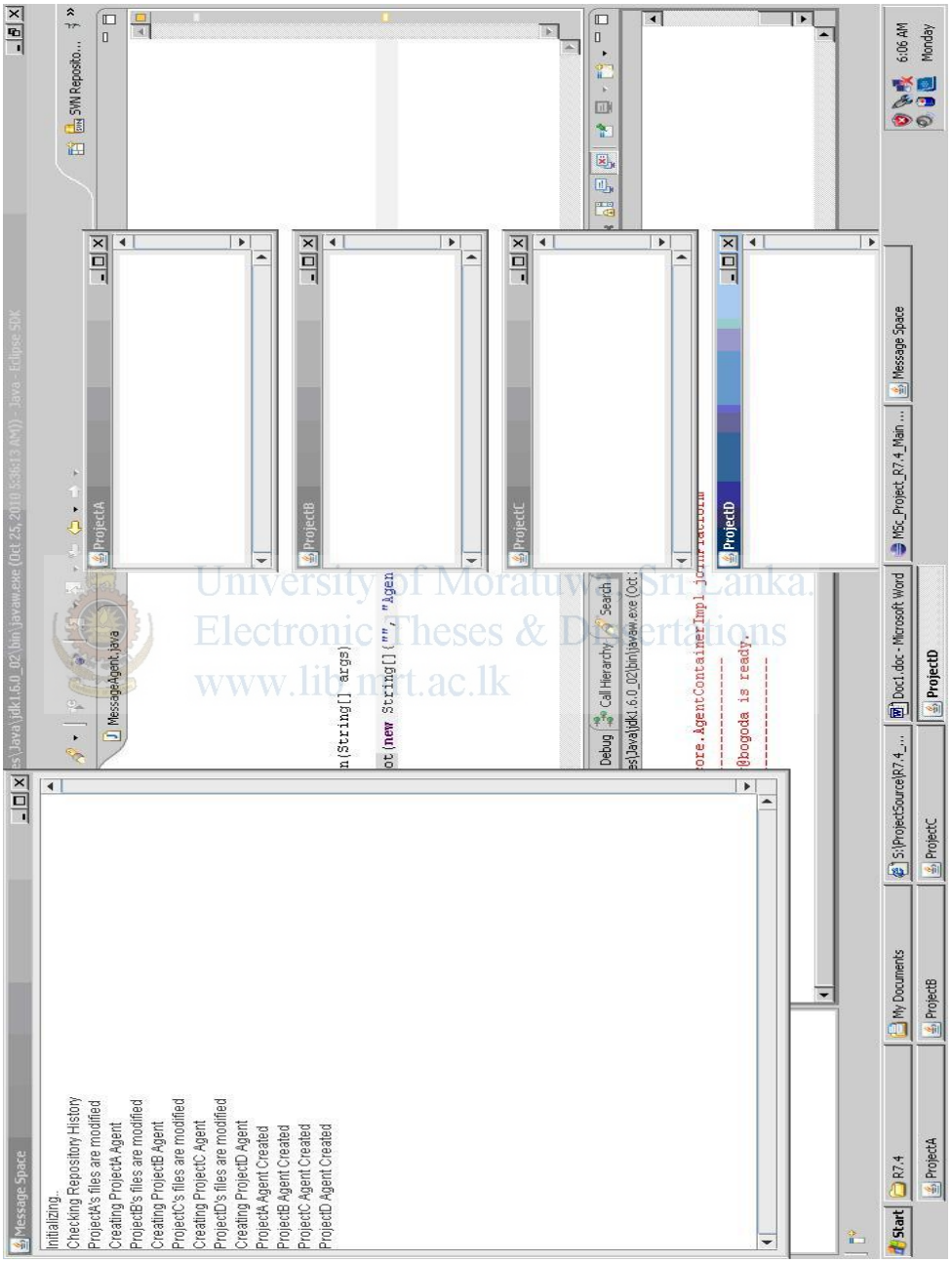

Figure A.3: Initial state of agent creation

Appendix A.4 Agents state after communication start

In the Figure A.4 it shows the progress of each agent when their communication started. For each agent separate message dialog is displaying to show the progress. Message board is also displaying to show the progress of the whole agents.

Figure A.4: Agents' communication progress

Appendix A.5 End state of agent communication

In Figure A.5 it shows the module build order which has identified by communicating among agents. Once their communication is over all the agents are terminated from the system.

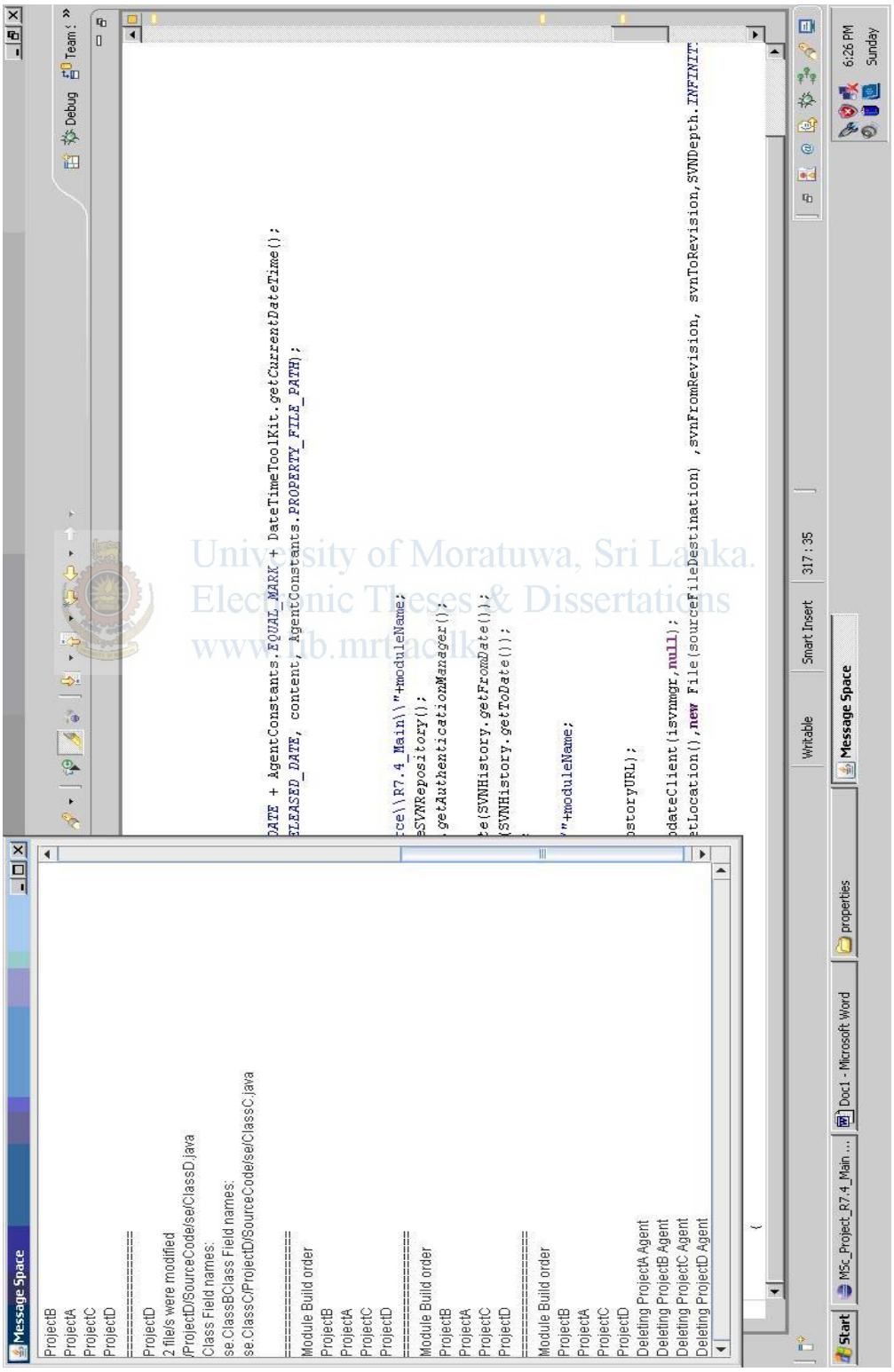

Figure A.5: End of agents' communication

Appendix B

Appendix B.1 Generated .Jar files

In the Figure B.1 it shows set of generated .jar files after executing the implemented system. Only modified project's jar files will be generated once the system is executed.

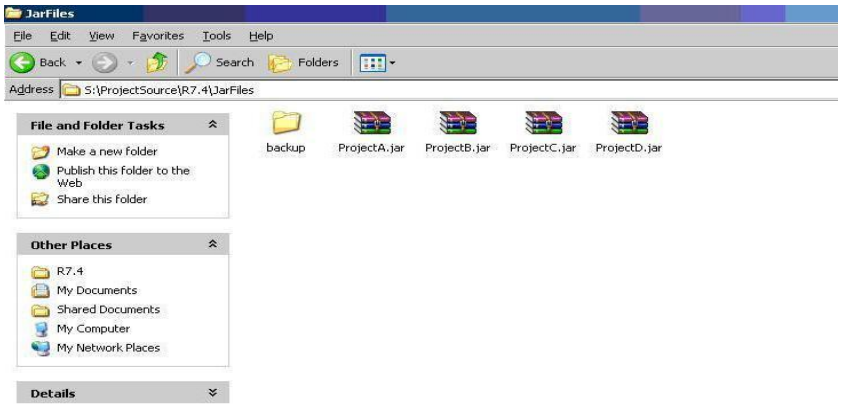

Figure B.1: Generated jar files

Appendix B.2 Apache Ant script file

In the Figure B.2 it shows the content of the Apache ant script file which is used to generate .jar files by compiling and building the modified project's source code.

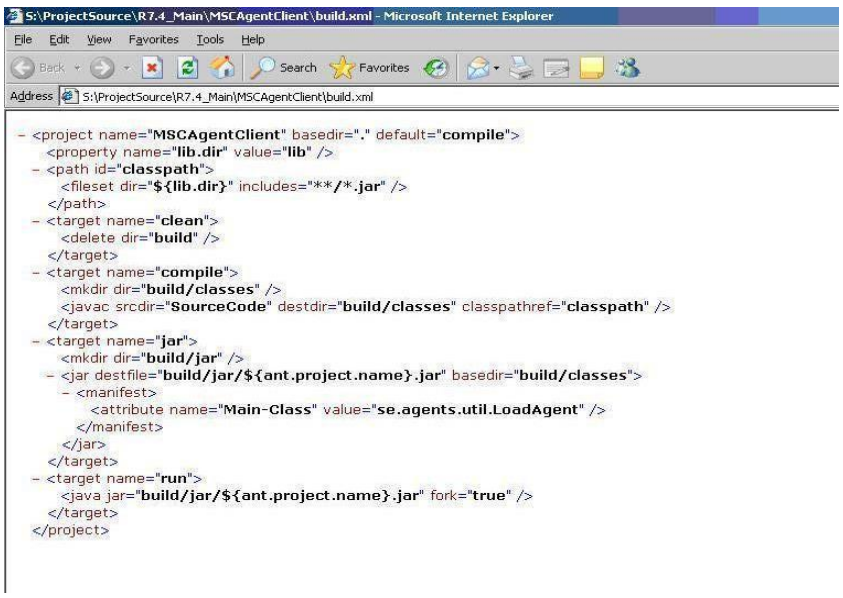

Figure B.2: Apache Ant script file

Appendix C

Appendix C.1 Class Diagram for implemented Multi Agent System

In the Figure C.1 it shows the class diagram for implemented system. Agent class is the super class which is located in JADE toolkit. ModuleAgent and MessageAgent are the classes which are written with customized features to fulfill requirements of agents.

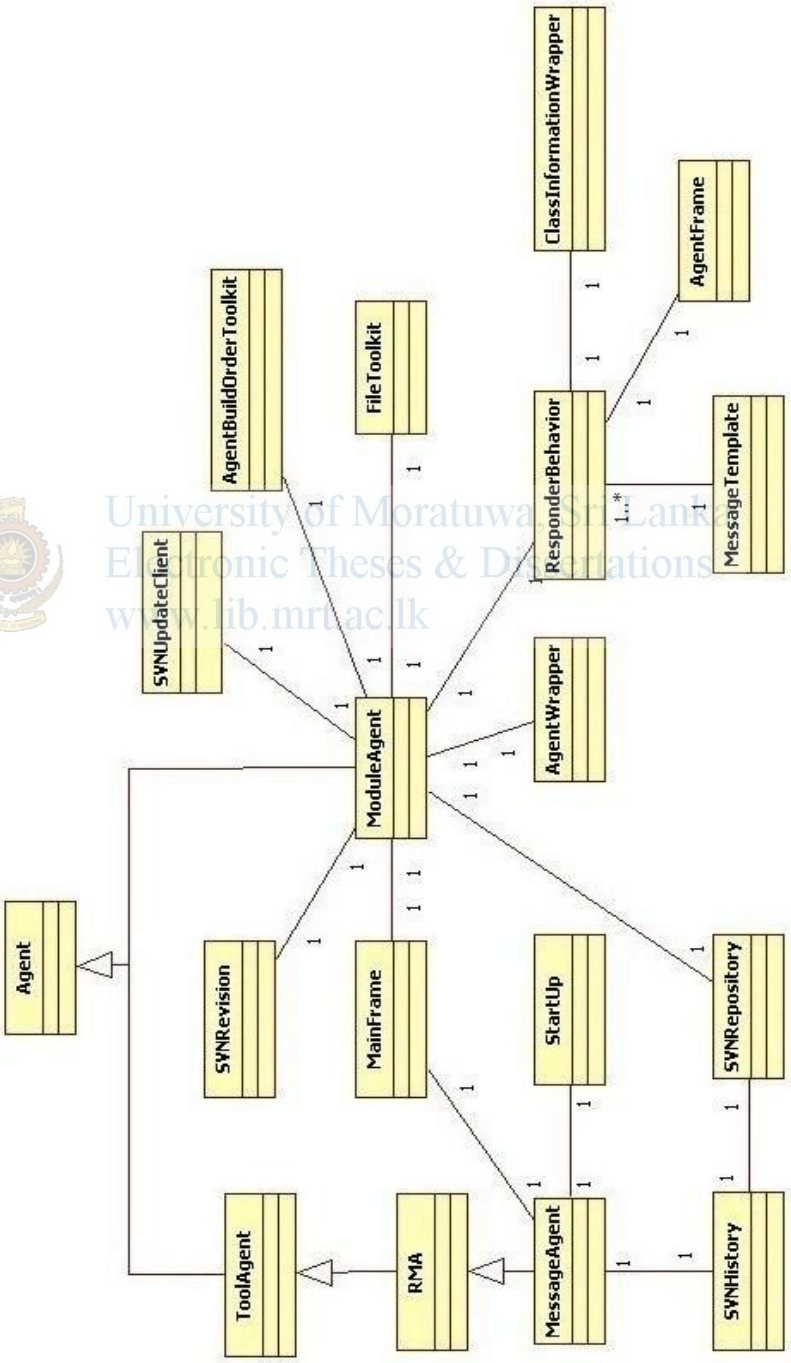

Figure C.1: Class Diagram

Appendix C.2 Sequence Diagram for implemented Multi Agent System

In the Figure C.2 it shows the sequence diagram for the implemented Multi Agent System. It shows the sequence of each operation among each component by using the arrows.

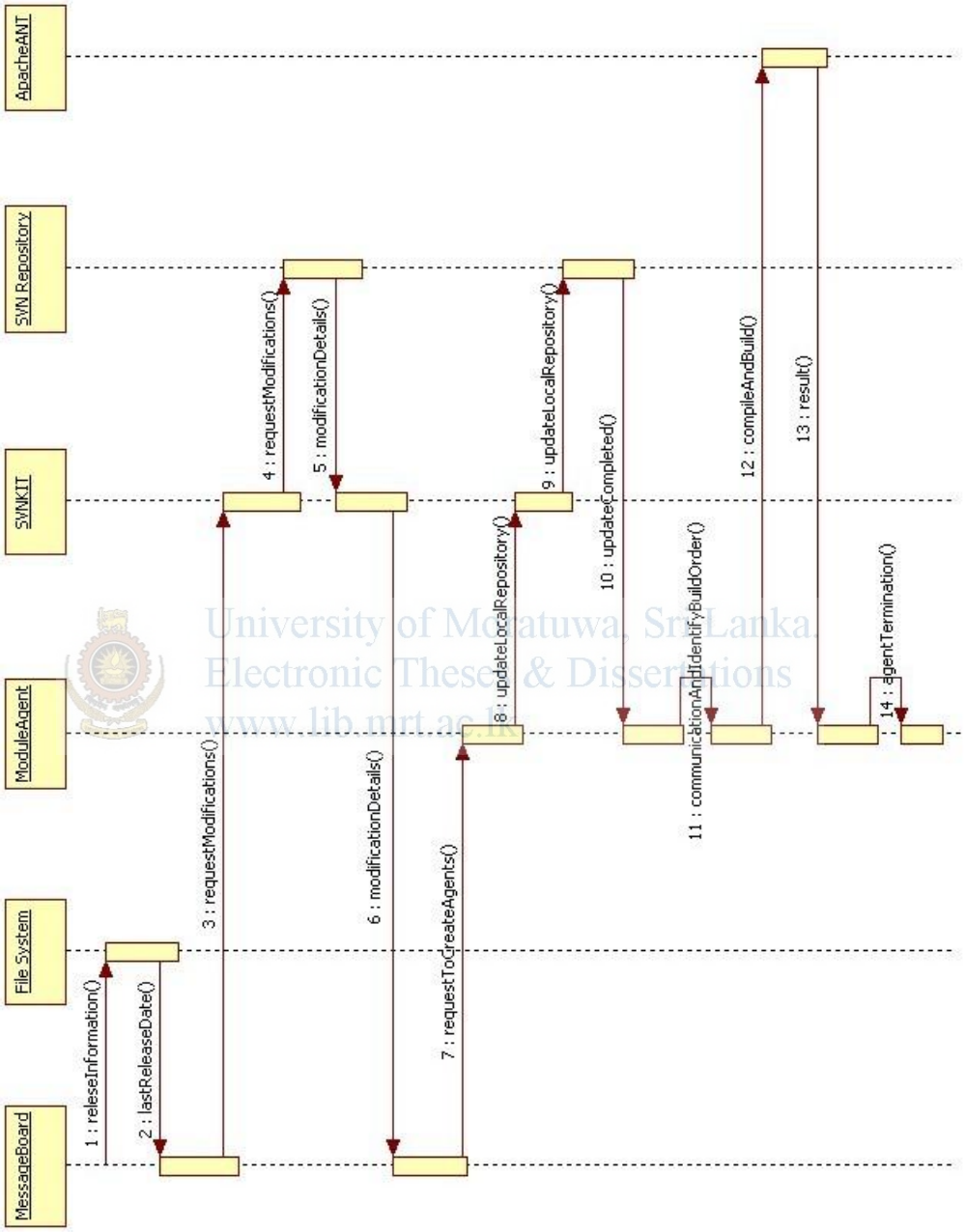

Figure C.2: Sequence Diagram