

**PROCESS CONTROL AND CONFIGURATION OF A
RECONFIGURABLE PRODUCTION SYSTEM USING A MULTI-
AGENT SOFTWARE SYSTEM**

By:

JEAN JANSE VAN RENSBURG

Thesis submitted in fulfilment of the requirements for the degree:

MASTER TECHNOLOGIAE: INFORMATION TECHNOLOGY

in the

School of Information and Communication Technology

of the

Faculty of Engineering, Information and Communication Technology

at the

Central University of Technology, Free State

Supervisor:

Prof. H.J. Vermaak

Co-supervisor:


Mr B. Haskins

Bloemfontein

December 2011

Declaration of Independent Work

I, JEAN JANSE VAN RENSBURG (Identity number: ██████████ Student number: 20444303) do hereby declare that this research project which has been submitted to the Central University of Technology for the degree MASTER TECHNOLOGIAE: INFORMATION TECHNOLOGY, is my own independent work; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology; and has not been submitted before by any person in fulfilment (or partial fulfilment) of the requirements for the attainment of any qualification.

Student Signature: 

Date: 2012 – 04 – 12

Acknowledgements

I would like to acknowledge the following individuals or organisations for their support without which the completion of this project would not have been possible:

- The Research Group in Evolvable Manufacturing Systems (RGEMS) within the Central University of Technology, Free State (CUT) for giving me the opportunity and support to conduct this project.
- The Advanced Manufacturing Technology Strategy (AMTS) within the National Research Foundation (NRF) for their monetary support of this project (project number AMTS-07-11-P).
- Prof Herman Vermaak and Mr B Haskins, for their support and guidance.
- Various fellow research students within RGEMS working on related projects.

Abstract

Traditional designs for component-handling platforms are rigidly linked to the product being produced. Control and monitoring methods for these platforms consist of various proprietary hardware controllers containing the control logic for the production process. Should the configuration of the component handling platform change, the controllers need to be taken offline and reprogrammed to take the changes into account.

The current thinking in component-handling system design is the notion of re-configurability. Re-configurability means that with minimum or no downtime the system can be adapted to produce another product type or overcome a device failure. The re-configurable component handling platform is built-up from groups of independent devices. These groups or cells are each responsible for some aspect of the overall production process. By moving or swapping different versions of these cells within the component-handling platform, re-configurability is achieved. Such a dynamic system requires a flexible communications platform and high-level software control architecture to accommodate the re-configurable nature of the system.

This work represents the design and testing of the core of a re-configurable production control software platform. Multiple software components work together to control and monitor a re-configurable component handling platform.

The design and implementation of a production database, production ontology, communications architecture and the core multi-agent control application linking all these components together is presented.

Abstrak

Tradisionele ontwerpe vir komponenthanteringsplatforms is rigied gekoppel aan die produk wat vervaardig moet word. Beheer- en moniteringmetodes van dié soort platform bestaan uit verskeie hardewarebeheerders wat die kontrole-logika van die proses bevat. As die konfigurasie van die komponenthanteringsplatform verander, moet die beheerders gestop word en nuwe kode in hulle gelaai word om die verandering in ag te neem.

Hedendaagse denke oor komponenthanteringsplatformontwerp berus op herkonfigurasie. Herkonfigurasie behels dat met die minimum of sonder onderbreking in produksie, die komponenthanteringsplatform aangepas word om 'n ander tipe produk te vervaardig of 'n fout in 'n komponent uit te skakel. Dié soort komponenthanteringsplatform bestaan uit groepe onafhanklike komponente. Die groepe of selle is elkeen verantwoordelik vir 'n bepaalde aspek van die produksieproses. Deur die selle of ander weergawes van die selle binne die platform rond te beweeg word die sisteem herkonfigureerbaar. 'n Dinamiese sisteem soos dié benodig 'n buigsame kommunikasieplatform en hoëvlak sagtewarebeheerargitektuur.

Hierdie studie verteenwoordig die ontwerp en toetsing van die kern van 'n herkonfigureerbare sagtewareproduksiebeheerplatform. Sagtewarekomponente werk saam om 'n herkonfigureerbare komponenthanteringsplatform te beheer en te monitor.

Die ontwerp en implementering van 'n produksiedatabasis, produksie-ontologie, kommunikasieargitektuur en kern, multi-agent kontroletoeëpassing wat al die komponente verbind, word voorgelê.

Table of Contents

<i>Declaration of Independent Work</i>	<i>ii</i>
<i>Acknowledgements</i>	<i>iii</i>
<i>Abstract</i>	<i>iv</i>
<i>Abstrak</i>	<i>v</i>
<i>List of Acronyms</i>	<i>ix</i>
<i>List of Figures</i>	<i>xi</i>
<i>List of Tables</i>	<i>xiii</i>
<i>Chapter 1: Introduction</i>	<i>1-14</i>
1.1 Introduction and Background Information	1-14
1.2 Problem Statement	1-14
1.3 Research Goals and Objectives	1-14
1.4 Project Limitations	1-15
1.5 Research Methodology	1-15
1.6 Dissertation Overview	1-15
<i>Chapter 2: The Production Environment</i>	<i>2-16</i>
2.1 Introduction	2-16
2.2 Current Production Control	2-17
2.3 IT Infrastructure	2-17
2.3.1 Servers	2-17
2.3.2 Ethernet Network	2-19
2.3.3 Other IT Devices and Services	2-19
2.4 OPC	2-19
2.4.1 Introduction	2-19
2.4.2 Configuration	2-20
2.5 Conclusion	2-22
<i>Chapter 3: Literature Reviews</i>	<i>3-23</i>
3.1 Introduction	3-23
3.1.1 Software Licensing Terms	3-23
3.1.2 List of Literature Reviews.....	3-24
3.2 An Review of Software Agent Implementations	3-24
3.2.1 An Extended Process Automation System: An Approach Based on a Multi-Agent System [29]	3-25
3.2.2 Agent-Based Approach to Supervisory Information Services in Process Automation [30]	3-27
3.2.3 Information Agents in Process Automation [31].....	3-28
3.2.4 Contributions to Conception, Design and Development of Collaborative Multi-Agent Systems [32]	3-29
3.2.5 SYROCO: A Novel Multi-Agent Shop-Floor Control System [33]	3-30
3.2.6 Industrial Deployment of Multi-Agent Technologies: Review and Selected Case Studies [34].....	3-33

3.2.7	RFID-Enhanced Multi-Agent Based Control for a Machining System [35]	3-35
3.2.8	ManufAg: A Multi-Agent-System Framework for Production Control of Complex Manufacturing Systems [36].....	3-37
3.2.9	A Multi-Agent Based Cell Controller [37].....	3-38
3.2.10	A Multi-Agent Architecture for Dynamic Scheduling of Steel Hot Rolling [38].....	3-40
3.3	Conclusion.....	3-41
3.4	A Comparative Study of Agent Oriented Software Engineering (AOSE) Design Methodologies	3-41
3.5	Discussion of the TROPOS Methodology	3-49
3.6	A Study of Ontologies and Methods of Creating and Employing them within the System .	3-57
3.6.1	Introduction	3-57
3.6.2	Ontology Definition.....	3-57
3.6.3	Ontology Languages.....	3-58
3.6.4	Ontologies.....	3-58
3.6.5	Ontology Tools.....	3-60
3.7	An Evaluation of Tools and Methodologies to Plan, Execute and Monitor Production	3-63
3.7.1	Introduction	3-64
3.7.2	Planning and Scheduling API's	3-64
3.7.3	Production Execution.....	3-65
3.7.4	Production Monitoring	3-70
3.8	An Overview of Methods to Implement Security within the System	3-70
3.8.1	Role Based Access Control (RBAC)	3-71
3.8.2	The RBAC Database Schema	3-72
3.8.3	Integration of RBAC into the MAS Design.....	3-72
3.9	Conclusion.....	3-73
3.9.1	Chosen Ontology Language	3-73
3.9.2	Chosen Ontology Design Tool	3-74
3.9.3	Chosen Upper Ontology.....	3-74
3.9.4	Chosen Ontology API	3-74
3.9.5	Chosen Inference Engine	3-74
3.9.6	Chosen Production Planner	3-74
3.9.7	Chosen Production Executor.....	3-74
Chapter 4: Implementation		4-75
4.1	Introduction	4-75
4.2	Java to OPC Communication Method	4-75
4.2.1	Introduction	4-75
4.2.2	Java to OPC Framework Design	4-76
4.2.3	Chosen OPC Access API.....	4-81
4.3	Design of the Multi-Agent System According to the TROPOS Methodology	4-82
4.3.1	Introduction	4-82
4.4	System Database and Ontology Design	4-90

4.4.1	Database Introduction	4-90
4.4.2	System Ontology	4-92
4.5	Combination of Components into a Functioning MAS	4-95
4.5.1	Introduction	4-95
4.5.2	Pre-Implementation Issues	4-95
4.5.3	Test MAS	4-96
4.5.4	System Implementation.....	4-99
Chapter 5: Results	5-108	
5.1	Test Scenario	5-108
5.1.1	System Start-up	5-108
5.1.2	Operator Interaction	5-109
5.1.3	Production Execution of the Test Process.....	5-111
5.1.4	Conclusion.....	5-115
Chapter 6: Conclusion.....	6-116	
6.1	Introduction	6-116
6.2	Summary.....	6-116
6.3	Research Goals and Objectives.....	6-116
6.4	Contributions	6-116
6.4.1	Production Ontology.....	6-116
6.4.2	System Database.....	6-117
6.4.3	Web Service to ACL Translation Procedure.....	6-117
6.4.4	Integration of RBAC into the Production Ontology.....	6-117
6.4.5	IT Infrastructure	6-117
6.5	Critical Analysis of Work.....	6-117
6.5.1	MAS, Ontology and Database	6-117
6.5.2	OPC Access.....	6-118
6.6	Future Work	6-118
6.6.1	Production Planner Implementation	6-118
6.6.2	Product Tracking.....	6-118
6.6.3	Operator Functionality.....	6-119
References.....	6-120	
Appendices	6-128	
Appendix A	Example of Operator Web service Endpoint Public Function.....	6-128
Appendix B	Web service Service Plan	6-130

List of Acronyms

ADACOR	—	ADaptive holonic COntrol aRchitecture
ADSL	—	Asymmetric Digital Subscriber Line
AOSE	—	Agent Oriented Software Engineering
API	—	Application Programming Interfaces
AVG	—	Autonomously Guided Vehicle
BDI	—	Belief-Desire-Intention
BFO	—	Basic Formal Ontology
CCTV	—	Closed Circuit Television
COTS	—	Commercial Of The Shelf
CUT	—	Central University of Technology
DAML	—	DARPA Agent Markup Language
DCOM	—	Distributed Component Object Model
DNS	—	Domain Name Service
DNS	—	Domain Name Service
DRL	—	DROOLS Rule Language
FAA	—	FIPA Abstract Architecture
FIFO	—	First In First Out
FIPA	—	Foundation for Intelligent Physical Agents
GFO	—	General Formal Ontology
GPL	—	General Public Licence
GUI	—	Graphical User Interface
IIS	—	Internet Information Services
IO	—	Input\Output
ISP	—	Internet Service Provider
IT	—	Information Technolgy
KIF	—	Knowledge Interchange Format
LGPL	—	Lesser General Public Licence
MAS	—	Multi-Agent System
NAS	—	Network Attached Storage
NAT	—	Network Address Translation

NAT	—	Network Address Translation
OIL	—	Ontology Inference Layer
OWL-DL	—	Web Ontology Language - Description Logics
PLC	—	Programmable Logic Controller
PNML	—	Petri Net Markup Language
PPPoE	—	Point-to-Point over Ethernet
PSU	—	Power Supply Unit
RAID	—	Redundant Array of Independent Drives
RBAC	—	Role Based Access Control
RGEMS	—	Research Group in Evolvable Manufacturing Systems
SU	—	Stellenbosch University
SUMO	—	Suggested Upper Merged Ontology
TCP	—	Transmission Control Protocol
UPS	—	Uninterruptable Power Supply
VPN	—	Virtual Private Network

List of Figures

Figure 2.1: Component Handling System _____	2-16
Figure 2.2: Allen-Bradley Module Backplane _____	2-16
Figure 2.3: Server Enclosure _____	2-17
Figure 2.4: RGEMS OPC Configuration _____	2-20
Figure 3.1: Water Tank Test Process [29] _____	3-26
Figure 3.2: PROAGE Framework [30] _____	3-28
Figure 3.3: AgentSearch Overview [32] _____	3-30
Figure 3.4: SYROCO System [33] _____	3-32
Figure 3.5: Agent Planning [34] _____	3-34
Figure 3.6: MAST GUI [34] _____	3-35
Figure 3.7: Agents and Agent Classes [35] _____	3-36
Figure 3.8: ADACOR Architecture [37] _____	3-39
Figure 3.9: Modelling Relationship Combinations in the Diminished Tropos Model [66] _____	3-54
Figure 3.10: Design Pattern Sequence [69] _____	3-55
Figure 3.11: Access Controller Software Design Pattern [73] _____	3-56
Figure 3.12: MASON Ontology _____	3-59
Figure 3.13: ADACOR Structure [89] _____	3-60
Figure 3.14: Protégé Ontology Editor [91] _____	3-60
Figure 3.15: Parallel Manufacturing Tasks [131] _____	3-66
Figure 3.16: Overall Production Tree Example _____	3-69
Figure 4.1: Composite Design Pattern [151] _____	4-76
Figure 4.2: Modified Composite Design Pattern _____	4-76
Figure 4.3: Adapter Design Pattern [152] _____	4-77
Figure 4.4: Other Framework Classes _____	4-77
Figure 4.5: Observer Software Design Pattern [153] _____	4-78
Figure 4.6: Proxy Software Design Pattern _____	4-79
Figure 4.7: Proxy Design Pattern in Action Step 1 _____	4-79
Figure 4.8: Proxy Design Pattern in action Step 2 _____	4-79
Figure 4.9: Iterator Software Design Pattern [154] _____	4-80
Figure 4.10: OPC testing GUI _____	4-81
Figure 4.11: XML-DA Response Timing [156] _____	4-82
Figure 4.12: TROPOS ER Diagram _____	4-89
Figure 4.13: Late Requirements Diagram _____	4-89
Figure 4.14: TROPOS Architectural Design Diagram _____	4-90
Figure 4.15: Production Ontology _____	4-94
Figure 4.16: Test MAS Architectural Design Diagram _____	4-97

Figure 4.17: Test MAS ACL Message Passing _____	4-98
Figure 4.18: Test MAS ACL Message Example _____	4-99
Figure 5.1: MAS Debug Output Window _____	5-108
Figure 5.2: JADE Agent Management GUI _____	5-109
Figure 5.3: JADEX Control Centre _____	5-109
Figure 5.4: Web Service Demo Client Application Admin Screen _____	5-110
Figure 5.5: Web Service Demo Client Application Device Screen _____	5-110
Figure 5.6: The Production Execution Window _____	5-111
Figure 5.7: Test Process Start _____	5-112
Figure 5.8: Pallet _____	5-112
Figure 5.9: Test Process Step 2 _____	5-113
Figure 5.10: Test Process Step 3 _____	5-114
Figure 5.11: Test Process Step 4 _____	5-114
Figure 5.12: Test Process Step 5 _____	5-115

List of Tables

Table 3.1: List of Methodologies _____	3-48
Table 3.2: Steps in the Tropos Methodology [68] _____	3-53

Chapter 1: Introduction

1.1 Introduction and Background Information

This software project was designed and tested on a small industrial component handling system within Lab 120 in the BHP-Billiton Building on the campus of the Central University of Technology (CUT), Free State [1] as a project of the Research Group in Evolvable Manufacturing Systems (RGEMS) [2]. RGEMS specialises in the development of innovations in the manufacturing environment.

The component handling system is composed of various devices and software packages that together are to facilitate the assembly, inspection and tracking of various simple product lines by utilising the reconfigurable nature of the component handling system. It is expected that the software developed in this project be applied to a similar component handling system under development at Stellenbosch University (SU) [3].

1.2 Problem Statement

Current component handling systems are controlled and monitored through various proprietary software packages that are written by device manufacturers for their particular devices. Planning and control of the production process is not an integrated whole and is tightly coupled to the physical layout of the component handling system. This results in a rigid set of products that can be assembled on any particular system. A proposed solution to these limitations is the concept of a reconfigurable component handling system that can be relatively easily reconfigured to adapt to various changes. Unfortunately these reconfigurable systems are complex, and require appropriate flexible control and monitoring software.

1.3 Research Goals and Objectives

The main goal of the project is to develop a flexible software system that is capable of controlling and monitoring a physically reconfigurable production system. Control in the context of this project means the software system issues commands to the devices in the component handling system. These commands enable the devices to work together and produce a certain product. Re-configurability of the system means that the system can, with minimal adjustment, produce another product, or automatically attempt to overcome a failed hardware device in the factory such as a robotic arm. Monitoring means the system will monitor aspects of the hardware devices present in the factory.

The research objectives of the project include the following components:

- The evaluation and testing of the OPC [4] communication architecture for real time device control and monitoring.
- The evaluation and testing of a Multi-Agent System (MAS) as the core application architecture for the production control application.

1.4 Project Limitations

The project will have certain limitations, but will be designed to be as extensible as possible. The limitations of the project include the following:

- Only production devices that can communicate through OPC will be monitored and controlled.
- The Graphical User Interface (GUI) to the system will initially only support the selection and execution of production plans, not the configuration of low-level process configuration properties.

1.5 Research Methodology

The research methodology to be followed in this study includes the following steps that will be followed from top to bottom, although it is anticipated that some steps will be executed more than once as new information becomes available and problems are encountered:

- The evaluation of the current situation and the collection of project parameters.
- The identification of aspects that need further study and investigation.
- The execution of literature studies about identified aspects that need further study.
- The selection of technologies that will be used in the implementation of the project.
- The implementation of the project according to the selected technologies.

1.6 Dissertation Overview

This dissertation consists of the following six chapters:

- **Chapter 1: Introduction**
A general introduction to the project and background information is given.
- **Chapter 2: The Production Environment**
The production environment in which the project is implemented is introduced and described in this chapter.
- **Chapter 3: Literature Reviews**
Literature reviews about research areas that the author has insufficient knowledge about is done in this chapter.
- **Chapter 4: Implementation**
The project implementation process is documented in this chapter.
- **Chapter 5: Results**
The results obtained during the execution of test scenarios are documented in this chapter.
- **Chapter 6: Conclusion**
Conclusions reached in the project are listed in this chapter.

Chapter 2: The Production Environment

2.1 Introduction

The component handling system consists of various hardware devices (Figure 2.1) such as commercial and non-commercial conveyor systems, robotic arms, computer vision systems, controllers, sensors and actuators. The specific production devices utilised in this project is described in more detail in section 5.1.3.

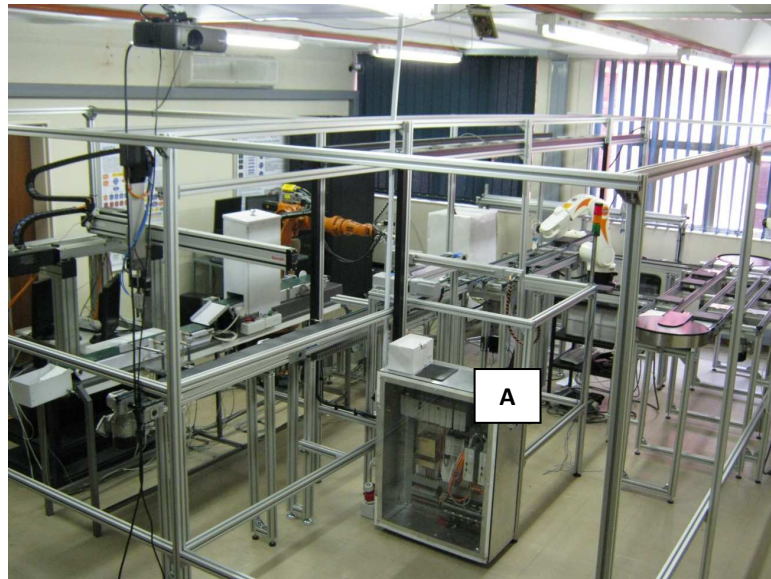


Figure 2.1: Component Handling System

The main wiring enclosure can be seen in Figure 2.1 at position “A.” This is the place within the system where most of the lower-level control hardware is situated.



Figure 2.2: Allen-Bradley Module Backplane

The most important hardware device within the enclosure is the Allen-Bradley [5] module backplane (Figure 2.2). This device contains modules, each with specialised functionality. Input/Output modules connect to sensors, motors and actuators on the conveyors. An Allen-Bradley 1756-ENBT/ Ethernet/IP module connects the module backplane to the Ethernet network, and an Allen-Bradley 1756-L63/B LOGIX5563 Programmable Logic Controller (PLC) controls the various modules in the backplane. A PLC performs various actions on its outputs based on its internal programming. Other modules are also installed on the backplane that provide connectivity to the DeviceNet [6] and Profibus [7] industrial networks and their compatible devices. Through the Ethernet/IP module, the other modules on the backplane can be accessed over the Ethernet network.



Figure 2.3: Server Enclosure

The Information Technology (IT) enclosure containing the Ethernet switches can be seen in **Error! Reference source not found.** At the beginning of the project, virtually no IT infrastructure was in place. The IT infrastructure was built up and refined during the course of the project and is described briefly in the next section.

2.2 Current Production Control

Currently the production sequence is controlled by various PLC's. PLC programs are written by programmers on workstations using programming software packages and uploaded into the PLC's. Programs are uploaded to PLC's via the Ethernet network or by directly connecting the PLC to a workstation with a serial cable. If the production sequence has to be changed, the PLC has to be stopped and an updated program loaded.

2.3 IT Infrastructure

2.3.1 Servers

The base of the IT core consists of three servers, each one dedicated to a particular range of functions.

The servers that were configured offer various other services that are not directly concerned with the production process such as controlled web proxy services, Closed Circuit Television (CCTV) recording, software license activation services, routing between networks and Subversion [8] version control services. However, these services help the RGEMS research students to perform their research more efficiently.

2.3.1.1 Server: RGEMS0

The specifications for RGEMS0 are as follows:

- Quad core Intel Q6600 processor at 2.4GHz.
- 8Gb DDR-2-800 Ram.
- 800W Power Supply Unit (PSU).
- 1x250Gb hard drive for the operating system, 2x250Gb hard drives to store daily image backups of the operating system disk and one 500Gb hard drive to store weekly image backups.
- 1.5TB drive to store CCTV footage of the production environment.
- Windows Server 2008R2 64Bit Standard operating system.
- 2200KVA Uninterruptable Power Supply (UPS).

This server runs a SharePoint 2007 [9] platform that contains the RGEMS website. The server also provides controlled Internet access to RGEMS members through Squid [10] as a caching proxy server combined with Papercut [11] for bandwidth accounting. The server sends and receives mail from the RGEMS email accounts, manages the network anti-virus server from Kaspersky Labs [12], provides accurate time for network devices and records the lab CCTV cameras.

2.3.1.2 Server: RGEMS1

RGEMS1 has the same specifications as RGEMS0, except for the absence of the 1.5TB hard drive and the addition of a 250Gb drive used to store several virtual machines. Several Microsoft Hyper-V [13] virtual machines are hosted on this server, with the most important being the OPC server virtual machine. The server also hosts various software licensing servers that manage the floating licenses of several software packages. RGEMS1 is also a Network Address Translation (NAT) device, providing controlled access between the RGEMS network and the CUT student network for certain services. NAT functionality is achieved by employing the Routing and Remote Access Service (RRAS) [14].

2.3.1.3 Server: RGEMS2

This server contains the same components as RGEMS0, but has an additional Adaptec [15] Redundant Array of Independent Drives (RAID) controller in RAID 10 with 4x250Gb drives for database storage space and a 1200W PSU. This server is also protected by a stronger 3000KVA UPS. RGEMS2 hosts MySQL [16] and Microsoft SQL Server 2008 [17]. The server is also the RGEMS Domain Name Service (DNS) server.

2.3.2 Ethernet Network

The network is of a commercial grade Ethernet type. Two gigabit speed switches connect the Ethernet devices, creating a high bandwidth network.

2.3.3 Other IT Devices and Services

2.3.3.1 Internet Modem

The Internet modem or Asymmetric Digital Subscriber Line (ADSL) bridge was installed in addition to the Internet router and Virtual Private Network) VPN server (described below) and is connected to both. This is done because multiple Point-to-Point over Ethernet (PPPoE) connections can be made through a single ADSL bridge, allowing devices to share the physical ADSL line. Each device uses its own internet account, receives its own public IP address, but shares the overall physical ADSL line bandwidth.

2.3.3.2 Internet Router

The internet router is connected to the internet modem and uses a normal ADSL account with a built-in dynamic DNS service from the Internet Service Provider (ISP). This allows the RGEMS domain (www.rgems.co.za) to be automatically linked to a dynamically changing IP address and thus always be accessible from the Internet.

2.3.3.3 Wi-Fi Access Point

The Wi-Fi access point allows secure access to the network for wireless clients.

2.3.3.4 DHCP Server and VPN Server

A D-Link [18] DIR300 router was flashed with the VPN version of the free DD-WRT [19] firmware for routers. This allows the router much more functionality than the factory firmware. The two features identified in the DD-WRT firmware as necessary for the network in this project are a very configurable DHCP server (to manage the ever growing number of Ethernet devices) and an OpenVPN [20] VPN server (to administer the network securely from the Internet). The D-Link DIR300 is a small hardware device with no moving parts, low power consumption and is highly reliable. The device is connected to the Internet Modem and uses a local-only Internet account with a built-in dynamic DNS service from an ISP. This allows the changing IP address of the device to be linked to a domain making the device always accessible from the internet.

2.3.3.5 Network Attached Storage (NAS)

A 4TB NAS device was installed for RGEMS members to store files. This device is configured in a RAID5 configuration, giving 3TB of useable space. The device also keeps monthly image backups of all three servers in addition to the monthly off site server image backups.

2.4 OPC

2.4.1 Introduction

OPC is communication architecture that allows standardised access to a compatible hardware device. OPC simplifies access to hardware devices as the client does not have to implement a proprietary communication protocol for each hardware device it wants to access.

It only has to implement the OPC protocol. OPC has different flavours and versions of those flavours. In this project only OPC-DA (Data Access) is utilised. OPC-DA is concerned with reading and writing live data from hardware devices and is the chosen communication method for this project. OPC DA is available in the following versions: OPC-DA1, DA2, DA3 and XML-DA

OPC is a client-server-based architecture. The OPC server connects to one or more hardware devices using appropriate proprietary communication mechanisms and then presents the hardware devices in a treelike structure of groups containing items. Each item is a variable, representing some aspect within the underlying hardware device. Each item has a name, data type, read-write access specifier and current data value. An OPC client connects to the OPC server and is then able to browse the item tree. Clients can subscribe to changes in the data values of one or more items as well as being able to write new data values for items. A considerable amount of time was spent configuring, testing and troubleshooting various OPC configurations in the production environment. The goals that had to be met concerning OPC were:

- Compatibility
- Security
- Simplicity

All these goals have been met. As can be seen in Figure 2.4, the OPC configuration in the production environment is constructed of components from various software vendors and uses various technologies.

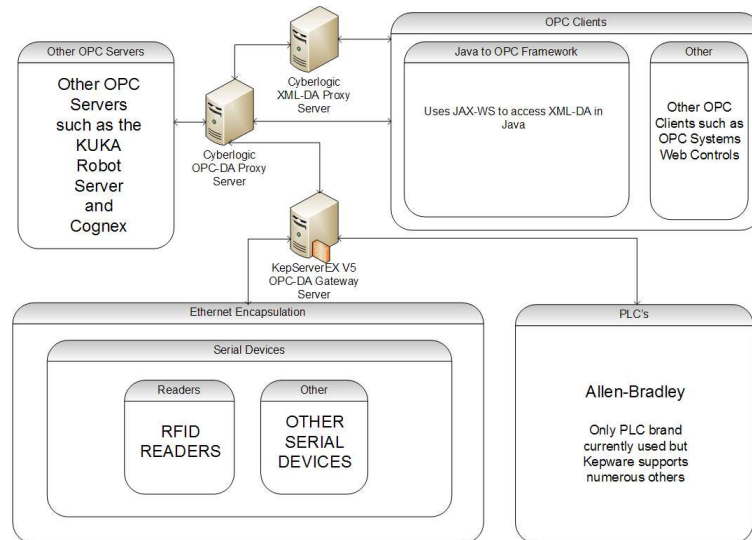


Figure 2.4: RGEMS OPC Configuration

2.4.2 Configuration

2.4.2.1 Compatibility and Simplicity

The core of the OPC configuration implemented in the production environment consists of the Cyberlogic OPC Crosslink Suite [21]. The software provides the core functionality of being able to aggregate various OPC-DA servers and represent them as one unified OPC-DA and XML-DA server.

Another advantage of this product is that it is provided free of charge to development environments. As can be seen in Figure 2.4, all OPC clients connect to the Cyberlogic OPC server and not to the underlying OPC servers. This simplifies the OPC configuration on clients. The Cyberlogic OPC server is OPC DA 3.0 and XML-DA certified, ensuring compatibility with a wide range of clients and servers.

Another software package, KepServerEX v5 OPC and Communications Server from Kepware [22], was acquired to enable communication with PLC's and other hardware devices using the Ethernet network. KepServerEX 5 communicates with a multitude of industrial devices over an Ethernet network and allows access to them through OPC. As can be seen in Figure 2.4, KepServerEX 5 is mainly used to allow OPC to PLC communication.

2.4.2.2 Security

Securing OPC presented numerous challenges. OPC security was achieved by implementing the following strategies:

- A username and password is required by OPC clients to connect to the Cyberlogic OPC server. A password-protected, limited Windows account was configured on the OPC server and the server's Distributed Component Object Model (DCOM) [23] and Internet Information Services (IIS) [24] settings set appropriately. OPC-DA and XML-DA access is thus password protected. Clients connecting to the OPC server need to authenticate before they can access OPC. XML-DA on the OPC server was secured by requiring XML-DA clients to authenticate using the same Windows user account as was used to secure OPC-DA, but in this instance using an HTTP authentication challenge configured in IIS.
- OPC servers on the network have been configured to only communicate with the Cyberlogic OPC Server through firewall settings.

2.4.2.3 XML-DA Performance Adjustments

Due to the way XML-DA clients connect to a XML-DA server and the configuration of the Windows line of operating systems and some parameters specified in the Transmission Control Protocol (TCP) specification, two modifications need to be made to the Operating system running the XML-DA server to enable reliable functioning. If these settings are not adjusted, the following problem occurs: The XML-DA server's operating system runs out of network ports to assign to new XML-DA client connection attempts under heavy XML-DA client loads. The modifications are well documented by IBM [25] in [26] and require two Windows registry changes. These changes are applicable to any high load Windows server environment where the server runs out ports to assign to new connection requests:

- **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\TcpTimedWaitDelay**

This key has to be modified or created if it does not exist. A favourable value for this entry is 30. This means that Windows will only wait 30 seconds before making a used port that has been used in a terminated connection available again for use by the system. The default value is 240 seconds.

- **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tcpip\Parameters\MaxUser Port**

This key has to be modified or created if it does not exist. A favourable value for this entry is 65534. This means that Windows will make available all ports to applications needing them. The default value is 5000.

2.5 Conclusion

In this chapter the production environment and its configuration was introduced. OPC was described and how it is implemented within the production environment. An important observation can be made about the importance of a solid IT infrastructure in current production environments: IT and the production environment are closely interlinked and dependent on one another. A solid IT infrastructure is needed to run and configure a successful production environment.

Chapter 3: Literature Reviews

3.1 Introduction

Before the development of the software system can begin certain decisions need to be made about which one of many similar technologies to use. Also, information about unknown subjects must be gathered. In order to decide between the different but similar technologies and collect useful information, literature reviews need to be done. The literature reviews that are presented in this chapter should not be seen as just summaries of other people's work. The aim of the following literature reviews is to extract information and then to select optimal technologies relevant to the functionality required in this research project.

3.1.1 Software Licensing Terms

This project relies on many freely available software packages or Application Programming Interfaces (API) that are integrated into the system. The licence terms on which the software packages are released greatly determine what can be done with the particular software package. It is a common mistake to assume that Open-Source software is completely free and can be incorporated into closed-source projects without any charge.

The most common Open-Source licence under which software is released is the GNU General Public Licence (GPL) [27]. Software licensed under this licence may not be incorporated into closed source software. To use software licensed under the GPL in closed-source systems, the system code should either be made available under a GPL licence (which is unacceptable for closed-sourced commercial systems such as the current project), or a commercial licence to use the software should be bought from the original licence holders of the GPL licensed software.

A more flexible Open-Source licence was developed to overcome the GPL license difficulties in commercial software. This license is called the Lesser General Public Licence (LGPL) [28]. This licence allows commercial systems to use LGPL licensed software free of charge and still maintain a closed source or commercial licence, if certain conditions are met. Other customised licences also need to be taken into consideration when evaluating a particular piece of software. Many authors choose to licence software on their own terms.

For a "free" piece of software to be compatible with this project it has to be released under a licence that at least allows the inclusion of the software into a commercial, closed source system at no charge and allows the project to be released under a closed source license. The LGPL licence is a good example of such a license.

The LGPL Licence

In summary, the LGPL licence allows LGPL licensed software to be used in closed source commercial software if the following conditions are met:

- The LGPL software is not modified and then used in the project; the software is only linked to. This means making API calls to the original, unmodified LGPL licensed software component is permitted.
- The LGPL licensed software components and licence is supplied with the combined software package and notice is given that LGPL licensed software is used in the system.

3.1.2 List of Literature Reviews

The following literature reviews have been completed:

- **An Review of Software Agent Implementations**
The main reason for the review is to get an idea of how multi-agent technology has been implemented in especially process-automation projects. The second reason for this review is to assemble a list of technologies for further review.
- **A Comparative Study of Agent Oriented Software Engineering (AOSE) Design Methodologies**
The choice of an AOSE design methodology is an extremely important decision that has to be made early in a multi-agent software system project.
- **A Study of Ontologies and Methods of Creating and Employing them within the System**
In order for the software agents to communicate with each other within the system, they need a common view of the system and the concepts within it. To create this common view, an ontology has to be designed.
- **An Evaluation of Tools and Methodologies to Plan, Execute and Monitor Production**
Various approaches exist to plan and execute the actual production process. An evaluation of these various approaches is necessary in order to select an approach that can be integrated with the multi-agent software system this project is based on.
- **An Overview of Methods to Implement Security within the System**
Security is very important in any software system, especially one where physical devices are controlled by the system. Possible security implementations are evaluated in this study.

3.2 An Review of Software Agent Implementations

Studying currently implemented multi-agent software projects gave valuable insights into the following aspects:

- AOSE Design Methodologies.
- Software Agent Platforms.
- Agent Knowledge Representation and Ontologies.
- Agent Reasoning and Deliberation.
- Agent Technology in the Context of Process Automation.
- Controlling and Configuring the Process Automation Process.

- Scheduling Production in the Process Automation System.
- Re-configurability in a Process Automation System.
- Security Model of the Software System.

The following projects, systems or studies are reviewed:

- An Extended Process Automation System: An Approach based on a Multi-Agent System[29].
- Agent-Based Approach to Supervisory Information Services in Process Automation[30].
- Information Agents in Process Automation[31].
- Contributions to Conception, Design and Development of Collaborative Multi-Agent Systems[32].
- SYROCO: A Novel Multi-Agent Shop-Floor Control System[33].
- Industrial Deployment of Multi-Agent Technologies: Review and Selected Case Studies[34].
- RFID-Enhanced Multi-Agent Based Control for a Machining System[35].
- ManufAg: A Multi-Agent-System Framework for Production Control of Complex Manufacturing Systems[36].
- A Multi-Agent Based Cell Controller[37].
- A Multi-Agent Architecture for Dynamic Scheduling of Steel Hot Rolling[38].

3.2.1 An Extended Process Automation System: An Approach Based on a Multi-Agent System [29]

3.2.1.1 Introduction

The literature in this review is of a doctoral dissertation by Ilkka Seilonen of the Helsinki University of Technology, Information and Computer Systems in Automation in Helsinki, Finland. This project can be seen as an extension of the project by Shanku Chakraborty in [39] and was implemented on the same hardware and in the same test environment. This project extends the project in [39] by utilizing deliberative agents instead of just reactive agents to enhance a process automation system. The information contained in the reviewed literature promises to be very relevant to the current research project as it introduces a functioning multi-agent system using deliberative agents.

3.2.1.2 Project Description

The project is implemented in a lab environment. The aim of the project is to test the implementation of a relatively simple multi-agent system. The multi-agent system is to control a Test Process. The Test Process hardware is a small scale water temperature control experiment consisting of the following components as can be partially seen in Figure 3.1:

- A water tank, pump and pipes.
- Electronically controllable magnetic valves.
- Temperature and pressure sensors.

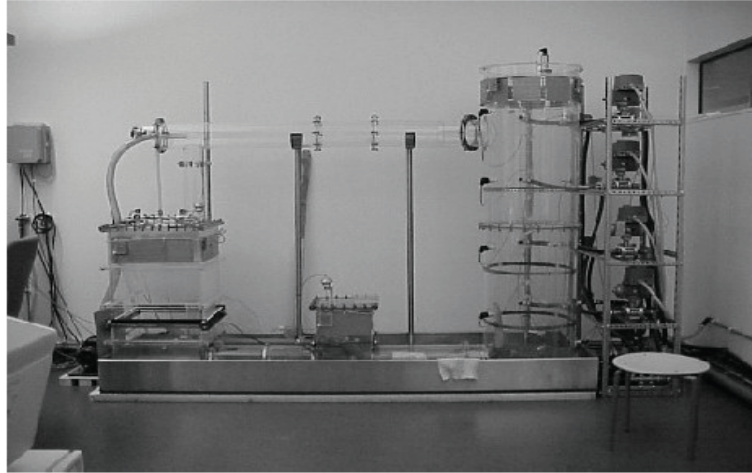


Figure 3.1: Water Tank Test Process [29]

Three low level control loops exist in the Test Process:

- **Loop 1**
Maintains a certain water level inside the tank.
- **Loop 2**
Controls the water temperature in the upper half of the tank between set two points.
- **Loop 3**
Controls the water temperature in the lower half of the tank between two set points.

The system's function is to keep the water tank at a certain water level, while maintaining set temperatures in the upper and lower parts of the water tank:

“These values might be different which creates a simple temperature profile. The temperature is calculated as the average of two separate measurements at the both parts.” “The water temperature at the upper part of the tank has a strong effect on the temperature at the lower part through the water flow in the tank. A weaker reverse effect is caused by the water flow through the pipes. The temperature control as a whole can be regarded as a MIMO control problem, in which the inputs of the controller are the temperature set points and the outputs of the controlled system are the temperature measurements. The set points of the pump and the valves are the control variables” [29].

The Agent Society implemented in the Test Process has five Process Automation Agents and a Directory Facilitator. The topmost higher level agent is called the Process Agent and is responsible for controlling the whole Test Process via the subordinate Agents. The Tank Agent and Pump Agent are responsible for their respective parts of the Test Process. The Upper Part Agent and Lower Part Agent control their respective temperature control loops.

3.2.1.3 Results and Conclusions

The most valuable knowledge gained from this review includes the following two techniques:

- **Sequential Control Based on Distributed Planning**

The author introduces an implementation of distributed planning to enable sequential control in the automation system. The role of sequential control is to coordinate control sequences of the lower level automation system through deliberation between agents. This functionality is an integral part of the current research project and will be included for further study as a possible solution to plan activities in the production system.

- **Supervisory Control Based on Distributed Search**

The author introduces an implementation of distributed search to enable supervisory control in the automation system. The role of supervisory control is to monitor the controlled process and initiate control procedures. These control procedures can include negotiations between agents to keep the processes they are responsible for within specifications. The other reason for negotiations between agents can be because of external commands sent to the system by an operator. This technique is of particular interest to the current project as this type of supervisory control is essential not only to control the production system (such as initiating production sequences), but also for the system to automatically manage its components during production. This technique will be included for further study.

3.2.2 Agent-Based Approach to Supervisory Information Services in Process Automation [30]

3.2.2.1 Introduction

The literature in this review is of a Master's thesis by Janne Jussila of the Helsinki University of Technology, Information and Computer Systems in Automation in Helsinki, Finland. This project is mainly concerned with knowledge representation (using semantic web technologies), transfer and storage within a complex system with heterogeneous data sources using a multi-agent system to facilitate the operator decision making process. The use of an ontology as agent knowledge base to enhance information representation, retrieval and the deduction of dependencies between concepts is tested.

3.2.2.2 Project Description

As can be seen in Figure 3.2, the system operates as a mediator between various information sources and clients in the plant. A common ontology is used to relate diverse information across the plant. The plant ontology has the following characteristics [30]:

- A global schema to facilitate query and task formulation.
- The representation of information using triplet patterns.
- The representation of the agent's knowledge-base.
- The communication vocabulary between agents.
- The relation of information from different domains.

- A local information source such as maintenance history is represented by local ontology dedicated to the source.

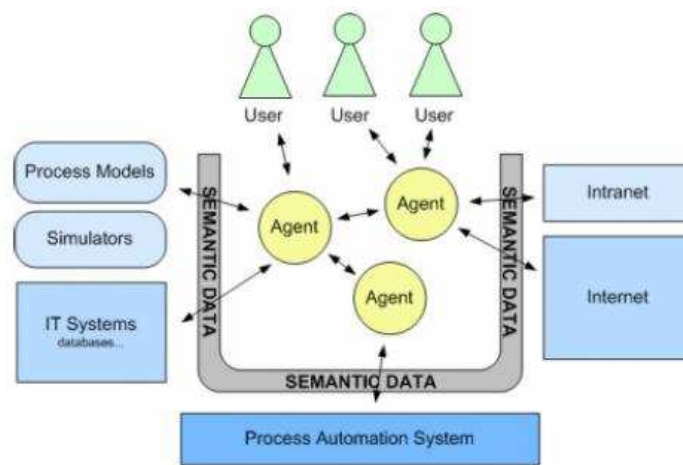


Figure 3.2: PROAGE Framework [30]

The local ontologies are merged into a global domain ontology. The domain ontology integrates information in the domain “spatially, functionally and temporally” [30]. The implemented application in the project demonstrates information retrieval using agents. Information can be retrieved from different databases and integrated using ontological relations [30].

The project was implemented using the JADEX [40] multi-agent platform to execute agents on, the JENA [41] rule based inference engine to enable inferences, the Protégé ontology editor to create and edit ontologies and Apache Tomcat[42] to host a user interface. A user builds up a query through the user interface. A Client Agent takes care of this interaction. An Information Agent receives the query from the Client Agent, breaks the query up into smaller queries that are delegated to Wrapper Agents.

The Wrapper Agents are able to query their respective data sources Structured Query Language (SQL) [43] databases for the appropriate information. The information is sent back to the Information Agent. The Information Agent combines all information received from the Wrapper Agents. The information is then passed to the Client Agent that in turn updates the user interface.

3.2.2.3 Results and Conclusions

The project highlighted valuable information relating to ontologies, information queries and technologies applicable to a multi-agent system. The author implemented a system on the JADEX agent platform, used JENA as an inference engine and OWL [44] to represent the plant ontology. This particular combination of technologies is very interesting as the technologies used are still relatively new. An actual implementation of multiple agents working together on an information query using multiple ontologies as information bases was demonstrated. Also, a range of new terms and technologies was extracted from the documentation and will be of great use to the current project.

3.2.3 Information Agents in Process Automation [31]

3.2.3.1 Introduction

The literature in this review is of a Master's thesis by Milan Fajt of the Helsinki University of Technology, Information and Computer Systems in Automation in Helsinki, Finland. This particular project deals with accessing and monitoring information about a running process automation system by enhancing OPC data access with information agents. As with the project in [29] already reviewed, the same test process is used to implement the project.

3.2.3.2 Project Description

Information agents are connected to hardware devices in the system. These agents read data from devices and also provide the collected information to other agents and eventually human operators through user agents that communicate with the process agent. Information agents form a hierarchical structure that mimics the hierarchy present in the physical system with the process agent at the top of the hierarchy.

As variables that are subscribed to in the physical test process change, the information agents inform the process agent of these changes. The process agent informs the GUI agent that then updates the GUI so that a human operator is aware of the change.

3.2.3.3 Results and Conclusions

The reviewed project, although not so much different from other reviewed material, did provide a good overview of OPC technology. Also, a detailed implementation of a small multi-agent system was presented where agents subscribe to physical process variables and receive updates when these variables change by information agents. The functionality demonstrated in the reviewed project is functionality that is definitely required in the current project. The information gathered from the reviewed project gave a good overview of how this part (information agents) of the current project could be implemented.

3.2.4 Contributions to Conception, Design and Development of Collaborative Multi-Agent Systems [32]

3.2.4.1 Introduction

The reviewed project is a Doctoral thesis by Gheorghe Cosmin Silaghi of Babes-Bolyai University, Faculty of Economics, Department of Business Information Systems in Romania. The documentation is a large and extremely detailed collection of information about numerous multi-agent-related concepts and technologies. A conceptual design of a multi-agent system is presented in the project, but not actually implemented. Due to the extremely detailed conceptual design, the project is included as an example of a multi-agent system implementation. The author has the following objectives with his study:

- An investigation into agent collaboration and interaction and tools to achieve these.
- An investigation into the maturity of agent technology.

3.2.4.2 Project Description

The project contains a vast literature study and then a conceptual design of a multi-agent system.

The conceptually designed system is called “AgentSearch” and serves as a sophisticated system that will search for catalogue web pages on the internet. It will also allow user queries to be made to its webpage database. In addition to describing the technologies that can be used to implement the system, the author describes various software design patterns used when the system was designed. The system is presented in Figure 3.3: AgentSearch Overview

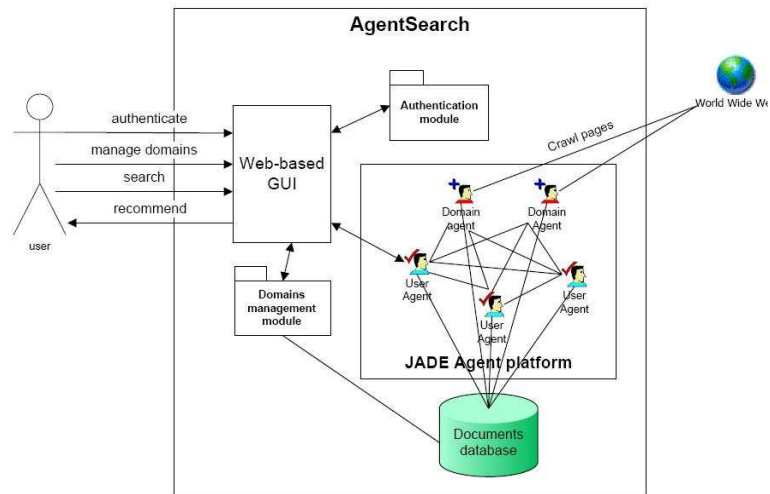


Figure 3.3: AgentSearch Overview [32]

3.2.4.3 Results and Conclusions

The authors reached the conclusion that the agent software paradigm is suitable to model social realities as well as approach the software design and development process. They remark that agent collaboration is more successful if soft rules are used that allow the agents to operate more freely. [32]. A large amount of knowledge about multi-agent systems was gained from studying the reviewed thesis. The author’s personal opinion and experience on various technologies is very helpful in deciding which technologies to use in the current project. The sound literature review gave a great deal of information about numerous agent technologies. The author, however, left out a few key technologies and gave no reason for doing so.

3.2.5 SYROCO: A Novel Multi-Agent Shop-Floor Control System [33]

3.2.5.1 Introduction

The reviewed project is a shop-floor automation scheduling and control system by D. Roy, D. Anciaux and F. Vernadat of the University Of Metz, France. The objective of the project is to propose and develop a new control approach that addresses the problem of the dynamic management of real-time production control, the automation of the control process and the handling of unexpected changes to the production plan [33].

3.2.5.2 Project Description

Although the reviewed project is not implemented on a known, standards-based Foundation for Intelligent Physical Agents (FIPA) [45] compatible agent platform, the project discusses functionality very similar to that required in the current project that is being designed.

An overview of the system architecture can be seen in Figure 3.4. The developed system is called SYROCO and is built from the following agents [33]:

- **The Meta-Object Agent**

This agent uses a path-finding technique in conjunction with a parts process plan prior to production to generate a product routing plan. The agent then generates Product agents that represent each product to be manufactured.

- **The Supervisor Agent**

This agent supervises the whole production process by keeping track of production objectives and the status of physical shop-floor devices. It also manages changes to the system. These changes include re-routing products to another device should a failure occur or rescheduling the production sequence by inserting Product agents should a more important order arrive. The Supervisor agent also coordinates communication between Cell agents.

- **The Cell Agents**

A Cell agent represents a collection of shop-floor devices (resources) that is needed to produce a certain product.

- **The Product Agents**

A Product agent represents a product to be produced. It stores information about what action need to be performed on a product to produce it, as well as what resources are necessary in the production process.

- **The Resource Agents**

A Resource agent represents a physical shop-floor device. This agent acts as a bridge between software and the physical process by sending production commands to the hardware and monitoring the status of the device it represents.

Communication in the SYROCO system is composed of the following messages:

- **Orders**

An order is a message sent from a superior agent to a subordinate agent that cannot be ignored.

- **Consultations**

A consultation occurs when one agent request and receives information from another agent.

- **Reports**

Reports are information sent from subordinate agents up to superior agents when some problem is detected.

- **Request**

Requests are messages sent to and from peer agents. The only peer message sent in the system is from the Supervisor agent to the Meta-object agent to generate new Product agents.

The Meta-Object agent receives Product agent creation requests from the Supervisor agent before production commences. The Meta-Object agent then consults with a path-finding utility in combination with a product process database. After the consultation, the new product request can be either accommodated without problem, or if it is impossible to find open slots for the requested products, a human operator can authorise that production can begin even if it will be delayed. After this phase, Product agents are created and the Supervisor agent is notified.

The Supervisor agent then instructs a Cell agent to start production. The Cell agent then sends production orders to the relevant Resource agents, and gathers information from Product agents as production proceeds. This information is sent to the Supervisor agent.

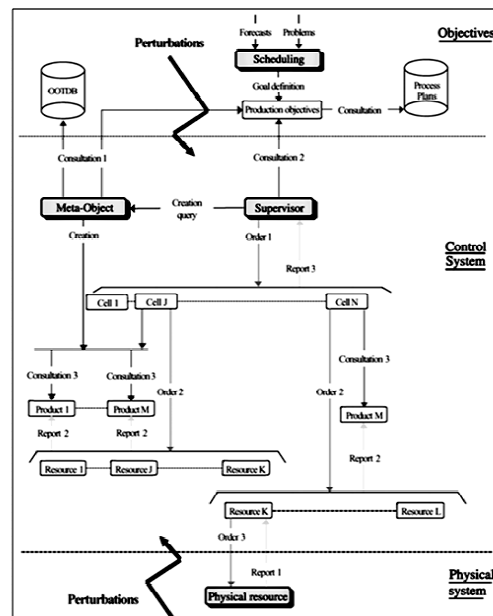


Figure 3.4: SYROCO System [33]

The Supervisor agent can send two commands to a cell agent. The first command notifies the cell of a change in product routing. This happens when a new product order is introduced into the system. The Cell agent will consult its Product agents to inform them of the change in resource availability. The second command is an interrogation message that determines which production sequences have been scheduled on a certain Cell. This is used by the Supervisor agent to re-route a future production phase that is affected by a failure of a shop-floor device.

3.2.5.3 Results and Conclusions

The reviewed project gave valuable insights about how the current system could be implemented.

The author highlights the fact that agent actions in a process automation system should be very quick, and uses reactive agents for controlling the process as it runs, while using a deliberative agent to do planning before production starts and when a disruption in the system occurs. Computer processing power when the project was implemented was much less than at the time of writing, and a move to more deliberative agents could be possible without negative effects if the system is implemented today. A hierarchical agent society is used, which fits well to the hierarchical nature of production systems. Of great interest is the route-finding system used that can determine possible product flows through the physical process. Also, the notion of Cell agents is introduced, which has been overlooked so far.

3.2.6 Industrial Deployment of Multi-Agent Technologies: Review and Selected Case Studies [34]

3.2.6.1 Introduction

The study was done by Michal Pechoucek of Gerstner Laboratory, Agent Technology Group, Department of Cybernetics, Czech Technical University, Prague, Czech Republic and Vladimir Marík of Rockwell Automation Research Centre, Prague, Czech Republic. It is a literature study of four industrial implementations of multi-agent technology. Only the first three projects are included for review:

Agents in Shipboard Automation Distributed Control and Diagnostics

The project aims to demonstrate the use of a multi-agent system in the control of equipment on a ship using Commercial Of The Shelf (COTS) products.

Agent-Based Production Planning of Engine Assembling

A car company needed a multi-agent system to facilitate in the planning of the mass manufacturing process of their car engines.

Agent Deployment in RFID Enabled Material Handling Control

MAST is a simulation tool used to simulate material handling in flexible manufacturing systems. It uses agent modelling to include components such as conveyors, manufacturing cells and Autonomously Guided Vehicles (AVG).

3.2.6.2 Project Description

The objective of the study is to give an overview of the successful deployment of multi-agent based software in industry by reviewing four projects. The description for each project is listed below:

Agents in Shipboard Automation Distributed Control and Diagnostics

The multi-agent architecture in the project is organised into three hierarchically organised levels. The Ship-level communicates with the crew of the ship and is concerned with the overall goals of the ship. The Process-level tries to optimise the automation components to ensure service availability. The Machine-level is responsible for real-time control. The first version of the system was focussed on controlling a chilled water system (CWS) [34].

An agent in this system has four main components: the planner, device model, execution control and diagnostic modules. The components are described below [34]:

- The planner represents the core of the agent. It contains plan templates that are modified at runtime with data obtained in the actual controlled process.

- The device model represents the actual process environment and its current state.
- The execution control module translates plans into actions that are sent to the appropriate controller.
- The diagnostic module gathers diagnostic information that is accessible by any agent in the community.

Agent-Based Production Planning of Engine Assembling

This project is implemented in an engine manufacturing plant producing about 1200 engines a day. The system is required to be open for integration with third-party monitoring and management tools as well as to produce a six-week, detailed production plan so that the following objectives can be met: [34]

- Minimisation of storage costs, tooling changes and product handling between production steps.
- An increase in production uniformity.

A coarse six-week production plan is submitted to the system. The plan is a result of high-level planning. The plan does, however, not include knowledge of many unforeseen factors that can influence production adversely. The system initiates low level planning using simulation to detect conflicts and inconsistencies in the high-level plan by sending the high-level plan to agents on the shop floor. These agents communicate with one another. When inconsistencies in plans are detected between agents, local re-planning is initiated. The agent with the highest priority requests changes in the plans of other agents so that all its constraints are satisfied. The same process is repeated for lower level agents. [34] The long-term planner used in the project is called LP PLANNER.

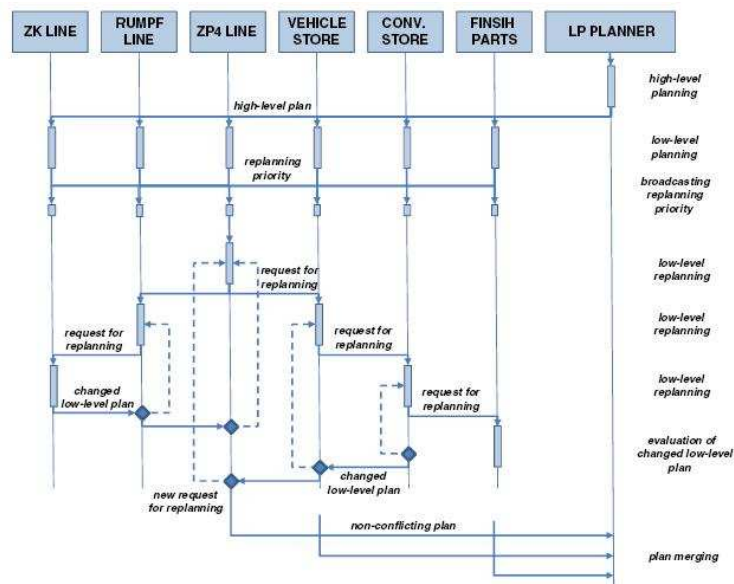


Figure 3.5: Agent Planning [34]

Agent Deployment in RFID Enabled Material Handling Control

The goal of transportation processes within the MAST system is to find the optimal transportation route within the system by planning around component failure and the addition of new components to the system. One of the first use-cases for the MAST system was the assembly of customised gift boxes [34].

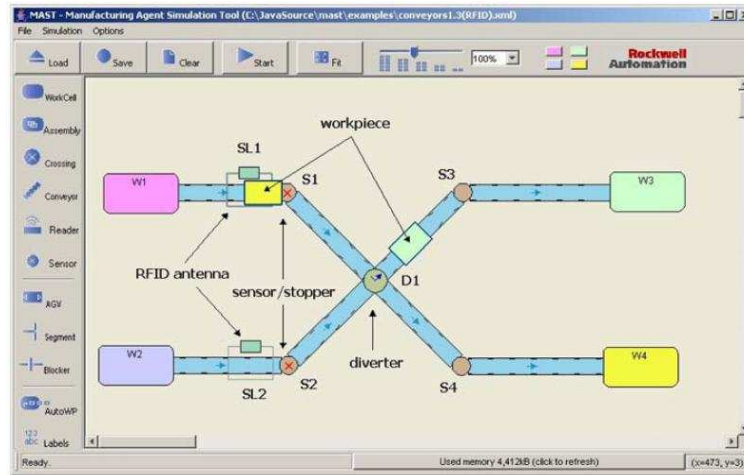


Figure 3.6: MAST GUI [34]

The agents communicated between one another about issues such as routing, product storage and picking and placing by robots. RFID agents notified other agents about the location of products in the system [34].

3.2.6.3 Results and Conclusions

Agents in Shipboard Automation Distributed Control and Diagnostics

The project demonstrated the advantage of a decentralised approach to control. The project is also of particular interest because of the involvement of Rockwell Automation, Inc [46] in the development and the use of Rockwell Automation hardware components.

Agent-Based Production Planning of Engine Assembling

The reviewed project highlights agent planning on the process (high) level and how it interacts with low level planning on agents representing hardware devices. It also highlights utilities that can be used to plan, such as LP PLANNER.

Agent Deployment in RFID Enabled Material Handling Control

The project again introduced the notion that individual products in the production process can be represented by an individual software agent each. The notion of dedicated Radio Frequency Identification (RFID) [47] agents is introduced. Due to the requirements of reconfigurability and modular design, dedicated RFID agents might be too restrictive in the project currently being designed and will probably be designed as tracking event agents.

3.2.7 RFID-Enhanced Multi-Agent Based Control for a Machining System [35]

3.2.7.1 Introduction

The reviewed study was done by Andre Garcia Higuera from the Engineering School, University of Castilla-La Mancha, Spain and Adolfo Cenfor Montalvo from the Manufacturing Department, Tecno Security, Spain. The project aims to show that a multi-agent system can be used to control an RFID enhanced machining system.

3.2.7.2 Project Description

The reviewed study is a conceptual design of a machining system incorporating a multi-agent system to schedule machining operations and RFID tracking of products as they move through the process.

The first step in the design process was the identification of agent classes and then defining concrete agents. The agent classes and implementations are illustrated in Figure 3.7. The second step in the design process was to define how agents can communicate with one another.

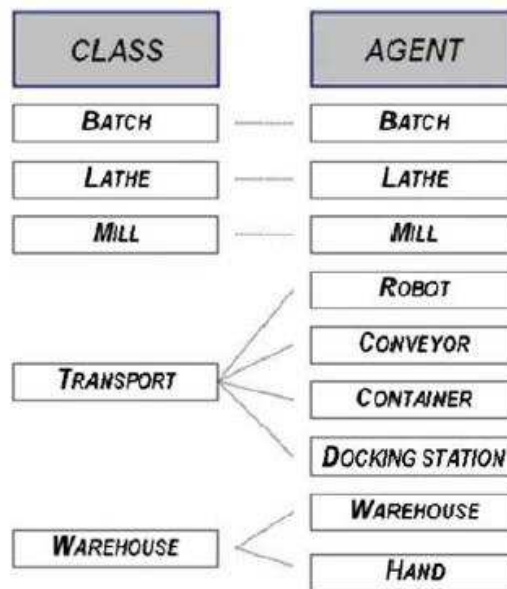


Figure 3.7: Agents and Agent Classes [35]

“Thus, information on the manufacturing order is sent to the batch agent. The resource agent has to negotiate with the batch agent attending to the status of the system (internal perception) and to the external perceptions to fulfil its intentions. All other agents have a similar behaviour: connecting to the batch agent to get information emanating directly from the manufacturing order. Along the process the agents must also listen to the perceptions both internal and external so that they can react to perturbations and negotiate in accordance to the status of the environment” [35].

“The agents “lot” represents here the different products to be manufactured and behave just like people in the market with a certain amount of money to spend. The products purchase the services they require for their manufacturing process to be completed. Their initial allowance of money is established according to the agent beliefs (as the priority increases so does the capital)” [35].

“...RFID readers are placed in strategic points of the manufacturing system. Every significant movement of a product is recorded. When a product reaches a workstation, all required information for the process to be performed on it is there. A proper negotiation process can take place between the machine and the product. Every delay or perturbation in the manufacturing process is recorded, and the information becomes available for the subsequent negotiations” [35].

A priority-based scheduling system is used instead of the usual First In First Out (FIFO) approach. If an order arrives with higher priority than the one currently being produced, the higher priority order will be attended to first, while the current lower priority order is rescheduled. In the case of multi orders being active in the system, the highest priority order will be attended to first, but other orders can still run concurrently by utilising resources unused by the highest priority order. The scheduling system can also handle perturbations such as device failure. Agents can reroute their products to other functioning devices that have the needed capabilities.

3.2.7.3 Results and Conclusions

Important features such as maximising production by scheduling work on unused devices and priority based scheduling were highlighted and is sure to be incorporated into the project being designed. Another interesting feature of the project is that products in the production process that cannot have an RFID imbedded due to their nature are transported in carriers. The carriers contain the product and the RFID tag.

3.2.8 ManufAg: A Multi-Agent-System Framework for Production Control of Complex Manufacturing Systems [36]

3.2.8.1 Introduction

The aim of the reviewed project is to develop a hierarchical multi-agent platform to run a manufacturing system. The authors present a new FIPA compliant multi-agent platform built- on the Microsoft.NET [48] called ManufAg.

3.2.8.2 Project Description

The authors evaluated current multi-agent platforms for compatibility with their idea of a hierarchical organisation of agents. Current platforms such as JADE [49] and ZEUS [50] did not meet their requirements, or were no longer being supported. The ManufAg multi-agent platform was developed.

The platform is FIPA compliant by implementing the FIPA Abstract Architecture (FAA) [45] and runs on the Microsoft.NET [48] platform. .NET remoting is used for inter-platform communication. The platform introduces native hierarchical agent organisation as well as rights based communication between agents. In addition to the implementation of the FAA, the PROSA [51] reference architecture for holonic manufacturing systems is also implemented.

3.2.8.3 Results and Conclusions

The PROSA reference architecture for holonic manufacturing control was highlighted. An argument for a hierarchical arrangement of agents was made.

The combination of heterarchical and hierarchical arrangement of agents is noted. Further attention will be given to the PROSA architecture in the project being designed.

3.2.9 A Multi-Agent Based Cell Controller [37]

3.2.9.1 Introduction

This is a study about multi-agent systems in distributed manufacturing systems. It was done by Paulo Leitão of the Polytechnic Institute of Bragança, Bragança, Portugal, Francisco Restivo of the Faculty of Engineering-University of Porto, Porto, Portugal and Goran Putnik of the Department of Production and Systems Engineering, School of Engineering, University of Minho, Guimarães, Portugal.

3.2.9.2 Project Description

The researchers propose an initial design for a multi-agent system. The goal is to upgrade an existing non-multi agent control system to a multi-agent system. A description of the different cells present on the shop-floor is given as well as the components present in each one. The following shortcomings with the current non-multi-agent system are identified:

- Reconfiguration.
- Learning and disturbance management.
- Distribution and decentralisation.
- Code re-usability.

A multi-agent architecture named ADACOR (Adaptive and Cooperative Control Architecture for Distributed Manufacturing Systems) is proposed. The following agent types are present in the system [37]:

- **Operational**
The Operational agent communicates directly with production devices.
- **Supervisor**
The Supervisor agent mediates communication between Operational agents.
- **Product**
The Product agent stores product information and the process plan.
- **Task Agent**
The task agent executes the production plan.
- **System Management**
The System Management agent allows for the administration of the system by operators.

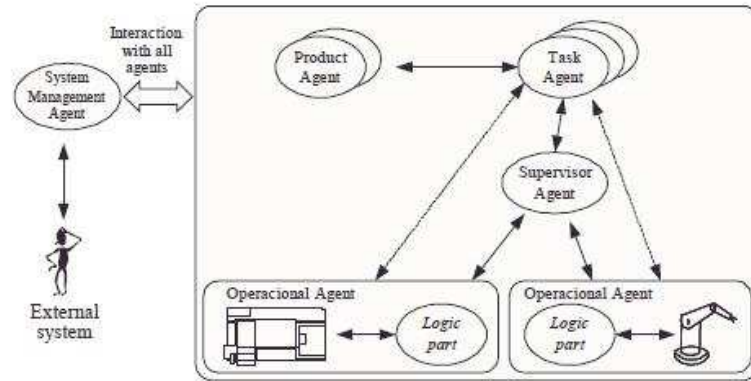


Figure 3.8: ADACOR Architecture [37]

The system includes a central database that stores information about:

- Behaviour of the agent.
- Community where the agent belongs.
- Objectives.
- Constraints.
- Experience.
- Decision rules.
- Procedures.
- Dynamic information.
- Organisational-related information.

Each agent contains three modules:

- **Decision-Making Module**

The agent will use the local database for information when solving a problem using a decision engine. If the problem cannot be solved, deliberation with other agents can be initiated using cooperation mechanisms, learning mechanisms and dynamic organisational techniques.

- **Communication Module**

This module manages communication between agents.

- **Local Control and Supervision Module**

“...intends to control and to supervise the operational execution of the agent. An important feature of this module is the interaction with the physical devices, which is supported by a platform based in CORBA objects that defines the basic services for the control and supervision of the physical devices: write and read variables, and programs manipulation (download, start, stop, etc)” [37].

A prototype system was implemented in Java [52] on a Microsoft Windows [53] 2000 workstation. Unfortunately this system uses no known and available agent runtime environment. The authors mention that the system will be moved to a FIPA compliant architecture.

“In this system, each product launches a task agent to supervise its execution. If that final product comprises other sub-product, the task agent will launch other task agents to supervise the execution of each sub-product. Each task agent should announce all operations to the available operational and supervisor agents, allocate them to the best proposals and wait for the end of the execution of the operation” [37].

The authors then go on to describe the differences in heterarchical and hierarchical control and how the system should be able to accommodate both approaches. The following advantages of the new agent based approach are also listed:

- Platform independency.
- Application development.
- Code re-usability.
- Distribution and Autonomy.
- Plugging Intelligence.

3.2.9.3 Results and Conclusions

The authors conclude that an agent-based approach offer many advantages over the approached employed in the system in question. They concede that their system is not complete and must be implemented on an existing FIPA compliant architecture.

They advise that metrics be developed to compare traditional approaches against the agent based approach. The project provided descriptions of different agent types and their roles in the proposed system. Also, the concept of cells in a manufacturing environment (a group of resources) is clarified and will be incorporated into the design of the current project. The concept of a transport cell that transports products and raw materials within the manufacturing environment and a palletising and de-palletising cell are especially interesting.

3.2.10 A Multi-Agent Architecture for Dynamic Scheduling of Steel Hot Rolling [38]

3.2.10.1 Introduction

This is a study completed by Peteri I. Cowling from MOSAIC Research Centre, Department of Computing, University of Bradford, Bradford, UK, Djamila Ouelhadj and Sanja Petrovic from 2ASAP Research Group, School of Computer Science and IT, University of Nottingham, Jubilee Campus, Nottingham, UK. The study is about dynamic scheduling of steel production using a multi-agent system.

3.2.10.2 Project Description

The project was implemented on a private agent platform in C++ [54] with a simple XML [55] based language that agents use to communicate.

The system attempts to schedule or re-schedule production when raw materials are not available in a running production sequence, or when rush orders are submitted for production. The system is composed of autonomous agents, each one responsible for a certain resource. These agents react to events in the resources they are responsible for, communicate with other agents and so work together to achieve a global production schedule [38].

3.2.10.3 Results and Conclusions

The project presented useful information about dynamic re-scheduling in a production environment. These concepts or goals (robustness, utility and stability) will be incorporated into the current project alongside the priority values for production requests. Mathematical concepts such as Tabu Search [56] and Greedy Algorithms [57] as used in scheduling problems is noted and will be investigated further.

3.3 Conclusion

This section listed and described various implementations of multi-agent systems. Valuable knowledge about a wide range of multi-agent related subjects was gathered by studying these implementations. As can be seen in the reviewed implementations, multi-agent systems are very diverse and include a multitude of components and competing designs.

3.4 A Comparative Study of Agent Oriented Software Engineering (AOSE) Design Methodologies

An agent oriented software engineering design methodology provides a plan for designing and implementing a Multi Agent System (MAS). A good AOSE design methodology provides for all the phases in the software design lifecycle and optionally provides development tools and agent runtime platforms. Using an optimal design methodology will ensure a higher probability that the system being built will be completed successfully within the allocated time span.

Before a comparison of AOSE design methodologies can be made, evaluation criteria for methodologies have to be established. Also, evaluation techniques have to be selected in order to execute the comparison. After evaluation criteria and evaluation techniques have been identified, an evaluation framework can be constructed. The following section extracted evaluation criteria, techniques and frameworks from at least two studies. The information gained in this exercise was used to construct an appropriate evaluation framework.

3.4.1.1 Study 1: Dam and Winikoff [58]

Dam and Winikoff [58] proposed various criteria for evaluating methodologies. They used a questionnaire that was sent to the creators of each methodology, as well as a practical implementation of a multi-agent system by students to rank properties in each criteria group. The framework consists of criteria that address properties of classical software engineering as well as properties unique to AOSE.

3.4.1.2 Study 2: Shehory and Sturm [59]

Shehory and Sturm [59] developed a list of criteria to catalogue the features of AOSE methodologies. They differentiated between software engineering and agent based criteria. The analysis identified areas that need improvement to suit the needs of developers [60].

Shehory and Sturm later combined their work in [59] with Dam and Winikoff [58] in [61] to give an overview of what they think a mature AOSE methodology should include. The resulting framework Shehory and Sturm developed in [61] aims to evaluate general purpose methodologies (as opposed to domain specific methodologies).

“Having the development stages defined is not sufficient for using a methodology. A methodology should further elaborate the activities within the development lifecycle, in order to provide the user of the methodology with the means of using it properly and efficiently”[61]:

The agent runtime platform is a critical component in a multi-agent system. A methodology may be chosen as the most suitable in a comparative study, but be incompatible with a desirable runtime platform or linked to an unsatisfactory runtime platform. For this reason, a third study is evaluated that incorporates the runtime platform into the methodology evaluation.

3.4.1.3 Study 3: Sudeikat et al in [60]

Sudeikat et al in [60] propose an evaluation framework adapted from [58] based on the following observation: "Based on our experiences we argue that a complete evaluation of methodologies cannot be done without considering target platforms, because the differences between available implementations are too fundamental to be ignored. In order to conduct a suitable comparison we present a flexible evaluation framework that takes platform specific criteria into account" [60].

3.4.1.4 Methodology Selection Process

To arrive at an acceptable number of methodologies to evaluate, a pre-screening of methodologies was done. The screening criteria used is an adaptation of the "Pragmatics", "Process: Coverage of workflows", and the "Internal Architecture: Goals, Plans and Beliefs (BDI)" as presented in [60]. These criteria were chosen as they are considered to be the most important properties, and non-compliance in these properties would render any further evaluation of the methodology as a waste of time and effort.

To generate a list of methodologies for further review the following process was followed:

- Compilation of a list of available methodologies by scanning literature.
- Gathering of literature about evaluations and comparisons done of methodologies.
- Process acquired literature by selecting methodologies that comply with the pre-screening criteria.

A base list of methodologies and or platform combinations was determined by scanning literature and related websites. The result of the pre-screening process is presented in Table 3.1. Some entries that were thought to be methodologies turned out to be runtime platforms only. These entries are still included in the table:

Platform	Type	Covers most workflows?	BDI Compatible meta model?	Pragmatics	Evaluate further?	Notes
ADELFE	Methodology	YES	N\A	ADELFE Toolkit is free. OpenTool design IDE not free - trail only. Could not find non-trail version. No code generation capabilities	No	No support for Implementation or Deployment phases. Methodology consists of 18 activities to be followed and is described in detail with examples in the ADELFE Toolkit.
ADEM	Commercial Platform with own methodology	YES	YES	IDE, code generation, platform management, testing and debugging	YES	LS/TS – Living Systems® Technology Suite that includes the IDE, runtime platform and methodology
AGILE PASSI	Methodology	YES	?	AgentFactory and MetaEdit+ used in design and implementation. MetaEdit+ extremely expensive. AFAPL2 language used to program agents in AgentFactory	YES	Gather more information about methodology in review
AUML	Methodology	NO	N\A	N\A	NO	
BDI-ASDP	Methodology	YES	YES	Could not find any IDE or any other tools to support development	NO	
COMOMAS	Methodology	N\A	N\A	Could not find any IDE or other tools. Methodology seems old. Articles are scarce and not easily available.	NO	

Platform	Type	Covers most workflows?	BDI Compatible meta model?	Pragmatics	Evaluate further?	Notes
DESIRE	Methodology	N\A	YES	Could not find any IDE or other tools, seems abandoned. Articles are scarce not easily available.	NO	
GAIA	Methodology	NO	N\A	No IDE or code generation tools.	NO	Gaia2Jade process can be used to implement GAIA as JADE code - no code generation is available, only an extension process to be followed.
INGENIAS	Methodology	YES	YES	IDE, Code Generation for the JADE agent platform through the IDK tool.	YES	Evolved from the MESSAGE methodology
JACK	Commercial Platform with linked third party methodology	N\A	YES	IDE, Runtime platform, Code generation	YES	No included methodology could be found. The Prometheus methodology is compatible with the JACK platform.
JIAC	Platform with own methodology	NO	YES	IDE, Runtime platform, Code generation	YES	Appears to be mature. Platform has won many agent performance competitions.
MANUFAG	Platform only	N\A	N\A	N\A	NO	Microsoft .NET runtime based. Could not download platform. Can use incorporated principles, especially the PROSA architecture

Platform	Type	Covers most workflows?	BDI Compatible meta model?	Pragmatics	Evaluate further?	Notes
MAS-COMMONKADS	Methodology	YES	N\A	Could not find any IDE or other tools, seems old. Very few articles available.	NO	Requirements, Analysis and design phases - no implementation phase.
MASE	Methodology	N\A	N\A	N\A	YES	Obsolete - Current version is O-MASE. AgentTool1+2 used in MASE, AgentTool3 is used for O-MASE
MESSAGE	Methodology	NO	N\A	N\A	NO	
O-MASE	Methodology	YES	YES	AgentTool 3 IDE, Model verifier and Code generation for JADE.	YES	
OPM/MAS	Methodology	YES	YES	Not enough information about the methodology could be gathered.	NO	
PASSI	Methodology	YES	N\A	IDE, Code Generation for JADE. PASSI Toolkit (PTK) is a plug-in for Rational Rose from IBM which is very expensive and is used to design according to the methodology. AgentFactory used to generate JADE code	NO	

Platform	Type	Covers most workflows?	BDI Compatible meta model?	Pragmatics	Evaluate further?	Notes
POAD	A Methodology extension	N/A	N/A	N/A	NO	Can use agent pattern oriented concepts in this extension
PRACTIONIST	Platform with own methodology	?	YES	IDE, Runtime platform, Code generation	YES	
PROMETHEUS	Methodology	YES	YES	PDT generates code for JACK but JACK is not free. JACK also has the JDE IDE for code generation	YES	
ROADMAP	Methodology	YES	N/A	Very little information available. No code generation tools or IDE	NO	Extension to the GAIA methodology
SADAAM	Methodology	YES	N/A	Very little information available. No code generation tools or IDE. AgentFactory possibly used but could not confirm	NO	Uses agile software engineering process
SECURE TROPOS	Methodology	YES	YES	IDE, no code generation	NO	Can use secure concepts in Secure Tropos in chosen methodology

Platform	Type	Covers most workflows?	BDI Compatible meta model?	Pragmatics	Evaluate further?	Notes
SODA	Methodology	YES	N/A	Very little information available. No code generation tools or IDE	NO	No implementation phase, requirements phase questionable
SONIA	Methodology	YES	N/A	Very little information available. No code generation tools or IDE	NO	
STYX	Methodology	NO	N/A	Very little information available. No code generation tools or IDE	NO	
TROPOS	Methodology	YES	YES	IDE, Code Generation for JADEX	YES	
LOST WAX AGENT BUSINESS ANALYSIS	Commercial Platform with own methodology	N/A	N/A	N/A	NO	No response from company.
CYBELEPRO AGENT INFRASTRUCTURE	Commercial Platform	N/A	N/A	No IDE or code generation tools	NO	No methodology, only a runtime platform.

Platform	Type	Covers most workflows?	BDI Compatible meta model?	Pragmatics	Evaluate further?	Notes
ACTIVE EDGE	Commercial Platform with own methodology	?	NO	?	NO	No response from company.

Table 3.1: List of Methodologies

The following methodologies were chosen for further evaluation:

- ADEM Methodology + LS/TS – Living Systems® Technology Suite Platform.
- INGENIAS Methodology + JADE Platform.
- JIAC Platform + Built-in methodology.
- O-MASE Methodology + JADE Platform.
- PRACTIONIST Platform + Built-in methodology.
- Prometheus Methodology + JACK Platform.
- TROPOS Methodology + JADEX Platform.

The second phase of screening concentrates on pragmatic issues. In this phase the properties as defined in [60] are adapted and applied. These are:

- Tool support. An attempt is made to install the provided development tools included in the methodology and run an example multi-agent application.
- The usage of the methodology and its tools in existing projects are determined.
- Support for the methodology and its tools are determined.

ADEM Methodology + LS/TS – Living Systems Technology Suite Platform

This commercial platform is not available for anonymous download and evaluation. Upon further enquiry (after two attempts) concerning licensing options it was determined that licensing would not be feasible.

INGENIAS Methodology + JADE Platform

The IDK toolkit was successfully downloaded and installed. It was possible to open an example application in the visual development tool and to execute it on the JADE agent platform. It was determined that the visual design tool generates skeleton JAVA classes, and further use these classes a third party JAVA development environment is necessary such as Netbeans [62] or Eclipse [63]. No problems were encountered installing the plug-in, although the user interface is dated, not friendly and many steps are necessary to construct an application. The JADE platform and all design tools are free of charge.

JIAC Platform + Built-in methodology

The JIAC Platform provides a comprehensive visual development environment in the form of an Eclipse plug-in called Toolipse. Some difficulty was experienced in the attempts to successfully install the Toolipse plug-in. An example multi-agent application was successfully built and executed by following a step by step tutorial after installation succeeded. Questions regarding the errors encountered within the plug-in were sent via email to the development team of JIAC, but no response was received as the current version of JIAC is no longer supported. JIAC requires that a scripting language JIAC Agent description language (JADL) be learned by the programmer. The system ontology is also constructed using the JADL language. JIAC is a very compelling platform, but the included methodology seems lacking.

O-MASE Methodology + JADE Platform

The O-MASE methodology uses Netbeans plug-in called AgentTool3. No problems were encountered in installing and running the platform, although one test application did not function correctly. The AgentTool3 plug-in does not have a graphical editor, but the plug-in allows for the editing of the various agent platform files and the execution of the project. The JADE platform and all design tools are free of charge.

PRACTIONIST Platform + Built in methodology

Upon further investigation it was found that this platform/methodology combination is not very mature. The software components are at release candidate version 1. No concrete implementations of the platform/methodology combination could be found and the development team acknowledges that the software has not been extensively tested. As there are more mature options available, this combination is eliminated from further investigation. The Practionist platform and all design tools are free of charge.

Prometheus Methodology + JACK Platform

The JACK platform is distributed as a 60 day evaluation and can be downloaded after a short registration process. No major difficulties were encountered during the tools installation or execution. The JACK design tool is not very user-friendly. The Prometheus Design Tool which is an Eclipse plug-in can also be used to design a multi-agent system visually. The tool generates code that can be executed on the JACK platform. The JACK platform costs approximately R18 000.

TROPOS Methodology + JADEX Platform

The Tropos methodology is bundled with a visual development tool called TAOM4E [64], which is an Eclipse plug-in. The tool was easily downloaded and installed and worked without any problems on the first attempt. Although the tool has a few bugs, the functioning of the code generator for JADEX was not affected. The JADEX platform and all design tools are free of charge.

3.4.1.5 Findings and Conclusion

TROPOS + the JADEX platform seem to be the best choice after this review. JIAC is also a viable option if a suitable agent-oriented design methodology can be used alongside it.

3.5 Discussion of the TROPOS Methodology

3.5.1.1 Requirements Analysis

The purpose of this phase is to elicit a set of functional and non-functional requirements for the system to be designed [65].

Early Requirements Analysis

During this phase, the domain stakeholders are identified and modelled as social actors who depend on one another for the achievement of goals, plans to be performed and resources to be produced [65].

The main goal of this phase is to model the system as it is currently, not as is supposed to be [66].

Late Requirements Analysis

In this phase the conceptual model is extended by including an actor representing the system and adding dependencies between this new actor and the other actors in the environment [65]. These dependencies define the functional and non-functional requirements of the system [142]. Stakeholders delegate the goals they want to achieve to one or more system actors [142].

3.5.1.2 Architectural Design

The Architectural and Detailed Design phases focus on the system specification according to the requirements elicited in the Requirements analysis phases [65].

The Architectural design phase defines the system's global architecture in terms of sub-systems connected through data and control flows. These sub systems are represented as Actors, and the data and control flows as dependencies between the Actors. This phase maps Actors to software agents with certain capabilities [65].

3.5.1.3 Detailed Design

This phase aims to specify agent capabilities and interactions [65].

3.5.1.4 Implementation

This phase follows the detailed design specifications by following an established mapping between the implementation platform constructs and the detailed design notions [65]. By using the TAOM4E plug-in for Eclipse to model the system a skeleton collection of Java classes was generated from the model automatically. During the implementation phase these classes were given functionality by manual programming.

3.5.1.5 Key Concepts in the TROPOS Meta-Model

Actor

An Actor is an entity that has strategic goals and intentionality within the environment and represents a "physical, social or software agent as well as a role or position" [65]:

- **Role**

A role is defined as an "abstract characterisation of a social actor within some specialized context or domain..." [65].

- **Position**

A position "represents a set of roles, typically played by one agent. An Agent occupies a position" [65].

Goal

A goal represents an actor's interests. There are two types of goals, Soft Goals and Hard Goals:

- **Soft Goals**

Soft Goals are used to model non-functional requirements. These goals are "satisfied" while hard goals are satisfied. This means soft goals are evaluated after the hard goals are met and there is no clear cut way to determine if a Soft Goal has been met [65]. These goals represent concepts such as "...security, performance and maintainability" [67].

- **Hard goals**

The outcome of the execution of Hard Goals can be determined in the system. These are Goals that need to be executed for the particular process to be successful [65]. Hard Goals represent the functional requirements of the system [66].

Plan

A plan is a certain way of achieving a Goal [65] [66].

Resource

A Resource is a "physical or informational entity" that an Actor possesses and can deliver to another Actor [65].

Dependency

A Dependency is created between two Actors if the "depender" Actor depends on the "dependee" Actor to deliver a "dependum" object in order to achieve some Goal, Plan or provide some resource [65][66].

"Dependencies may be quaternary relationships that include the dependee and the depender actor, the dependum and a fourth argument, which specifies 'why' the actors depend one from another. Again, this 'why'-argument can be again a goal, a plan, or a resource and, in other words, it expresses what entity in the dependee actor needs the dependency" [66].

Capability

A Capability is the "ability of an actor of defining, choosing and executing a plan for the fulfilment of a goal, given the certain world conditions and in the presence of a specific event" [65].

Belief

A Belief represents the knowledge an actor has of the environment [65].

3.5.1.6 Modelling Activities

The following modelling activities contribute to the acquisition of a first early requirement model.

Actor Modelling

This phase involves identifying the Actors in the Environment and the Actors and Agents within the software system.

- **Early Requirements Phase**

Focus is placed on modelling Domain stakeholders and their intentions "...as social actors who want to achieve goals" [65].

- **Late Requirements Phase**

In this phase, the "system-to-be" actor is defined along with its interdependencies on domain stakeholders [65].

- **Architectural Design Phase**

In this phase the focus shifts to defining the structure of the "system-to-be" in terms of its sub-systems (actors) and interconnections through data and control flows [65].

- **Detailed Design Phase**

In this phase "the system's agents are defined specifying all notions required by the target implementation platform" [65].

- **Implementation Phase**

In the implementation phase "actor modelling corresponds to the agent coding" [65].

Dependency Modelling

Dependency Modelling "...consists of identifying actors which depend on one another for goals to be achieved, plans to be performed, and resources to be furnished" [65]:

- **Early Requirements Phase**

In the early requirements phase focus is put on modelling Goal dependencies between social actors in the organisation. "New dependencies are elicited and added to the model..." [65].

- **Late Requirements Phase**

In this phase, dependency modelling focuses on analysing the dependencies of the "system-to-be" actor [65].

- **Architectural Design Phase**

In this phase "data and control flows between sub-actors of the system-to-be actors are modelled in terms of dependencies, providing the basis for the capability modelling that will start later in architectural design together with the mapping of system actors to agents." [65].

After all these modelling activities Actor diagrams are produced. Actors are visualised as circles, hard goals as ovals and soft goals as cloud shapes and two-arrowed lines as dependency relationships between goals and actors [65].

Goal Modelling

Goal modelling involves the analysis of an Actors Goals, conducted from the point of view of the particular Actor. Three reasoning techniques are used [65]:

- **Means-end Analysis**

This method aims to identify Plans, Resources and Soft Goals that provide a means to achieve a particular Goal

- **Contribution Analysis**

This method identifies Hard Goals that can contribute either positively or negatively to the fulfilment of the particular Goal being analysed.

- **AND/OR Decomposition**

This method "...combines AND and OR decompositions of a root Goal into sub-Goals, modelling a finer Goal structure" [65].

Goal modelling is applied to Early and Late Requirements models and is used to refine these models as well as eliciting new Dependencies. During Architectural Design it is used to decompose the "...system-to-be Actor into a set of sub-Actors" [65].

Plan Modelling

Plan modelling is a complementary technique to Goal modelling. The same reasoning techniques used in Goal modelling is used, but they are applied to Plans instead of Goals. In particular, the AND/OR decomposition is used to decompose a root Plan into sub-Plans [65].

Capability Modelling

Capability modelling is started after the Architectural design phase when the system sub-Actors have been specified in terms of their own Goals and their dependencies on other Actors. Capabilities allow sub-Actors to define, choose and execute plans for achieving their own Goals. Social Capabilities are also provided to Actors to manage dependencies with other Actors. Previously modelled Goals and Plans become integral parts of Capabilities. In the Detailed design phase the Agent’s capabilities are specified further and coded during the Implementation phase [65]. Table 3.2 represents the steps of the Tropos methodology.

Modelling Activity	Requirement Analysis Early Requirements	Requirement Analysis Late Requirements	Architectural Design	Detailed Design
Actor Modelling	Identify "top-level" actors or stakeholders in domain	Introduce system as an actor called system-actor	Decompose system-actor into sub-actors. Identify all dependencies	Define agents to model capabilities of system-actor and its sub-actors
Goal Modelling	Refine goals using means-end analysis, AND-OR decomposition, and contribution analysis. Find new dependencies			
Plan Modelling	Refine plans using the three plan analysis methods analogous to goal analysis			
Dependency Modelling	Identify dependencies between stakeholders using goal modelling	Model dependencies between system-actor and other actors	Model dependencies between sub-actors of the system-actor to identify capabilities	
Capability Modelling			Identify capabilities of sub-actors required to handle dependencies with all others	

Table 3.2: Steps in the Tropos Methodology [68]

3.5.1.7 Visual Modelling Tools

The following tools were used to model the agent system following the Tropos methodology and generate agent code:

- The Eclipse Integrated Development Environment (IDE).
- The TAOM4E plug-in for Eclipse.

3.5.1.8 Design Patterns: Restrictions to the Methodology

Restrictions presented in [66] to the methodology was applied to the modelling activity in this project.

These restrictions reduce the Tropos modelling concepts. The restriction aims to enhance the following properties [66]:

- Definition of Semantics.
- Design simplicity and clarity.
- Model expressiveness.
- Implementation of the model.

To ease automatic implementation (code generation), the main modelling concept used was the Goal diagram. The diminished model contains the following concepts [66]:

- The basic Tropos constructs: Actor, Hard Goal, Soft Goal, Plan and Resource
- A restricted set of relationships:
 - AND/OR Decompositions using only constructs of the same type as the root construct to decompose the root construct into sub-constructs. The root construct may not reference itself in a decomposition (acyclic).
 - Means-End-relationships are between Plans (means) and Goals (end). Other means end relationships in the metamodel are not allowed.
 - Use- and Produce relations are means-end relationships between an entity and a resource. If the resource is the “end” the relations is labelled as a “produce-relation”, otherwise if the resource is the “means” the relationship is labelled as a “use-relation”.
 - A Delegation relation between two actors is limited to a Goal Delegation.
 - A Dependency relation between two actors containing a dependum can only contain a Goal as dependum and a Goal as the “why”-argument.
 - Contribution relationships can only exist between a Plan or Goal to a Soft Goal and have the following values: ++, +, -, --.

Combining Relationships

All relationships are acyclic or non-reflexive, so an entity cannot be related to itself. An entity can be composed of a number of sub-entities, but the sub-entities can belong to more than one decomposition. As illustrated in Figure 3.9, the decomposition of a Goal or Plan may not be composed of different types of relationships.

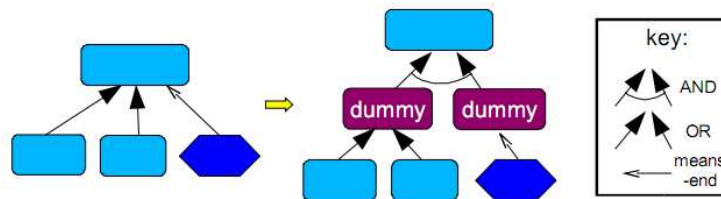


Figure 3.9: Modelling Relationship Combinations in the Diminished Tropos Model [66]

An example would be that if an AND/OR relationship is present in a Decomposition, a Means-end relationship may not be added. If an “illegal” combination in the decomposition is needed, dummy entities can be introduced into the model. These dummy entities are then used to adapt the model and overcome the modelling problem.

3.5.1.9 Design Patterns: General, Security and Interaction

Using established design patterns in software engineering can be of great advantage to the designer. The resulting design can be constructed quicker and be of a higher quality. The listed design patterns will have to be compatible with the restriction patterns to the methodology listed in the previous section; otherwise they cannot be applied.

Tropos Patterns

The following general Tropos design patterns or best practices when employing the Tropos methodology is given in [69]. Figure 3.10 indicates in what order (top to bottom) the patterns have to be applied. These patterns are very similar to the steps of the methodology already mentioned, but are much more clearly explained. The diagram shows the sequence of the application on the patterns in the design process.

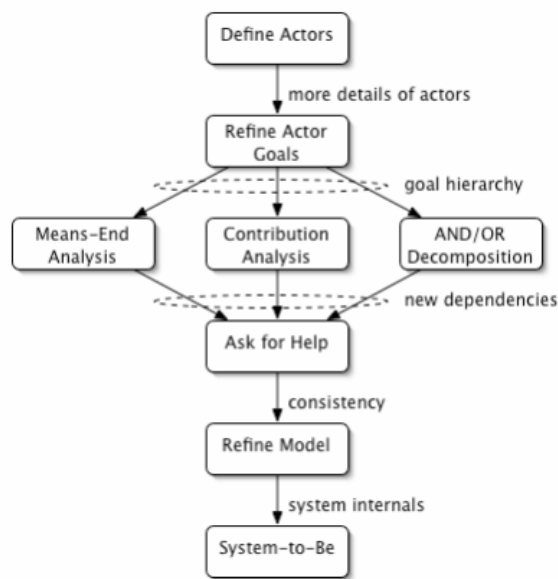


Figure 3.10: Design Pattern Sequence [69]

General Design Patterns

General design patterns are listed below and may be used during the design process (only patterns of interest are listed) [70]:

- **Booking Pattern**

The Booking pattern involves a client “booking” a resource from a provider. The provider can schedule access to the resource by the client [70].

- **Monitor Pattern**

The Monitor pattern is similar to Observer design pattern used extensively in conventional software engineering. It involves a client registering at a provider to receive updates about the status of certain resources [70].

- **Wrapper Pattern**

The Wrapper pattern wraps a legacy system into the multi-agent system. It translates communication between the legacy system and the MAS and ensures that the MAS and the legacy system remain decoupled [70].

Security Design Patterns

Security is a very important feature of any software system. Integrating security into the design from the outset will save time and money in the long term, as integrating security into a software system after implementation will be very time consuming, error prone and expensive. Security is considered a non-functional requirement and is thus specified apart from the functional requirements. These non-functional security requirements may in fact interact with the functional requirements [71]. The problems with security encountered during the design stages of a software system include [71]:

- Security requirements are not easy to analyse and model.
- Developers may not have the necessary skills required for secure software development.

The extension approach to the Tropos methodology as proposed in [71] can unfortunately not be implemented in this design, as the modelling environment does not support the new concepts introduced. This approach was later transferred into a new methodology extending Tropos called Secure Tropos [72].

As indicated in 3.8 and then further implemented in 4.3 a Role Based Access Control system was the main security mechanism implemented in this system. Because of this reason, the Access Controller security pattern was implemented. The Access Controller security pattern can be seen in Figure 3.11. It aims to limit access to system resources by using a security policy [73].

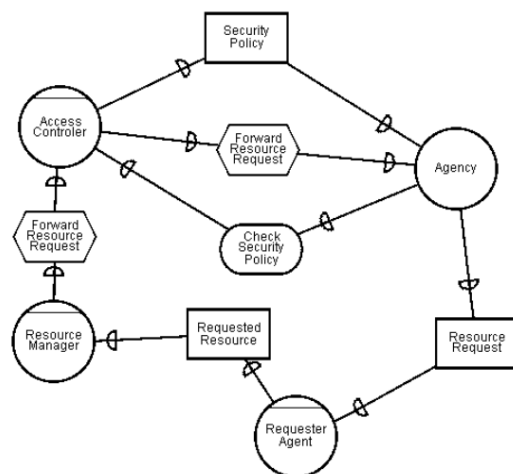


Figure 3.11: Access Controller Software Design Pattern [73]

3.6 A Study of Ontologies and Methods of Creating and Employing them within the System

3.6.1 Introduction

A study of ontologies and how to use them is necessary in the context of this project as all multi-agent software platforms include the concept of an ontology as one of their most important and required features to implement. The general layout of this chapter follows the layout as used in the excellent and very comprehensive ontology guide by Marek Obitko that can be accessed at [74].

3.6.2 Ontology Definition

The most widely used definition of an ontology is that it is an explicit specification of a conceptualisation [74]. An ontology does not store information about the particular state of a domain, it describes knowledge about the domain. [74] A knowledge base describes the particular state of a domain in the terms defined within the ontology in use in the domain. [74] Another definition of an ontology in literature is:

“Ontology engineering aims at making explicit the knowledge contained within software applications, and within enterprises and business procedures for a particular domain. An ontology expresses, for a particular domain, the set of terms, entities, objects, classes and the relationships between them, and provides formal definitions and axioms that constrain the interpretation of these terms.[75]”

Yet another definition states:

“In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application” [76].

In the context of an agent based system the following definition is more informational:

“Pragmatically, a common ontology defines the vocabulary with which queries and assertions are exchanged among agents. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner. The agents sharing a vocabulary need not share a knowledge base; each knows things the other does not, and an agent that commits to an ontology is not required to answer all queries that can be formulated in the shared vocabulary. In short, a commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology” [77].

Taking these definitions into account, it can be said that an ontology presents a framework in which domain knowledge can be stored, communicated and reasoned about by software components within and between software systems.

3.6.3 Ontology Languages

Ontologies are physically represented or encoded in a particular formal ontology language. Various ontology languages exist. Each language permits different levels of expressiveness (knowledge that can be represented in the language) vs. Inference capability (deductions that can be made from the knowledge represented in the ontology) [78]. Also, the availability of inference engines, parsers, software licence terms and tools for a particular ontology language plays an important role in the actual usability of an ontology language.

Examples of well known ontology languages that appear useable in this project include:

- **Cycl [79]**
Cycl is an ontology language based on first order logic linked exclusively to the Cyc and OpenCyc ontologies.
- **KIF (Knowledge Interchange Format) [80]**
KIF is an ontology language based on first order logic.
- **DAML+OIL [81]**
DARPA Agent Markup Language (DAML) and Ontology Inference Layer (OIL) is combined to form DAML+OIL. DAML+OIL is the predecessor of OWL.
- **Web Ontology Language - Description Logics (OWL-DL) [44]**
OWL-DL is a version of OWL that is designed specifically to enable automated logical inference by computers. It allows for good expressiveness by retaining computational completeness and decidability.

3.6.4 Ontologies

Using ontology languages, ontologies for various domains can be constructed. The available domain ontologies are too numerous to list. An ontology can be described as an upper or foundation ontology or as a domain ontology. The following sections explain this distinction.

3.6.4.1 Upper or Foundation Ontology

A foundation ontology or upper ontology is an ontology that contains very general concepts that are common to many different domains. These can be used as a base to construct more domain specific ontologies, enabling information transfer between these domains [82]. An example of a part of the MASON [83] upper level ontology is presented in Figure 3.12. Various upper ontology projects are presently in development.

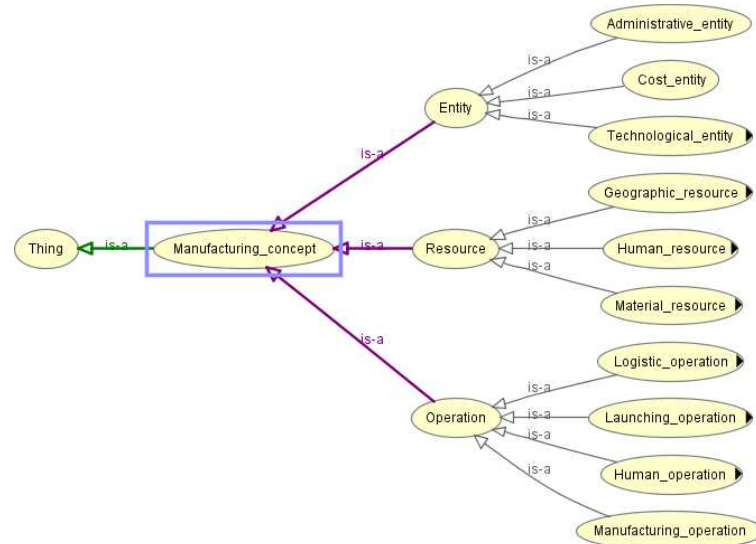


Figure 3.12: MASON Ontology

Examples of upper ontologies of interest are:

- **DOLCE [84]**
The DOLCE Ontology is an upper level ontology that can be used to construct domain specific ontologies. It is available in OWL.
- **GFO [85]**
The General Formal Ontology (GFO) is an upper level ontology available in OWL. It is mainly used to construct biological systems ontologies, but can be used to construct ontologies for other domains.
- **BFO [86]**
The Basic Formal Ontology (BFO) is an upper level ontology that can be used to construct domain specific ontologies. It is an offshoot of the DOLCE and SUMO ontologies, but does not contain any domain specific terms.
- **MASON [83]**
MANufacturing's Semantics ONtology (MASON) is an OWL-DL based upper ontology written specifically for manufacturing environments.
- **SUMO [87]**
The Suggested Upper Merged Ontology (SUMO) is an upper level ontology with domain specific divisions. It is free and owned by the IEEE. SUMO is written in the SUO-KIF ontology language, but is also available in a somewhat degraded form as an OWL ontology.
- **OpenCyc [88]**
OpenCyc is written in CycL, and is the open source free version of Cyc. It is a very large general upper level ontology. The OpenCyc ontology is available in OWL and contains a free inference engine and Java API.

3.6.4.2 Domain Ontology

A domain ontology models concepts specific to a particular domain. Domain ontologies are mostly incompatible with other domain ontologies, as the domains they model are not completely alike or the modellers model the domain differently. To make the concept of the domain ontology clearer, an example of a domain ontology used in a manufacturing control system is presented. ADACOR (ADaptive holonic COntrol aRchitecture for distributed manufacturing systems) [37] uses a proprietary frame based ontology as recommended by the FIPA Ontology Service Recommendations [82]. See Figure 3.13. JAVA classes are used to describe concepts and predicates in the domain the system operates, ensuring that the ontology is quickly generated and the resultant artefacts can be used directly.[82] It is possible to re-create the ADACOR ontology using the DOLCE ontology, as can be seen in [89].

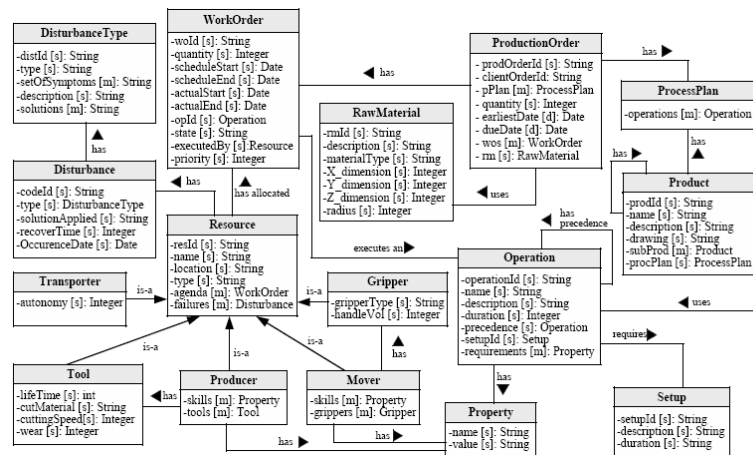


Figure 3.13: ADACOR Structure [89]

3.6.5 Ontology Tools

The tools used in ontology engineering consist mainly of ontology design software and inference engines used in conjunction with other programming languages.

3.6.5.1 Ontology Design Utilities

The most well-known visual ontology design tool is Protégé [90]. The tool was developed by the Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine and at the time of writing has more than 125 000 registered users.

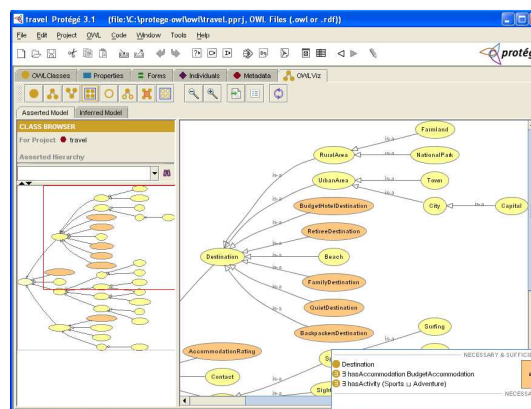


Figure 3.14: Protégé Ontology Editor [91]

It enables users to construct frame-based ontologies as well as ontologies for the semantic web. A screenshot of the editor in action with an OWL ontology being visualised can be seen in Figure 3.14. The tool is distributed as a stand-alone executable. Other well-known ontology design tools relevant to this project include:

- **Sigma [91]**
Sigma is an ontology browsing and creation tool for ontologies written in KIF. It is primarily used on the SUMO upper ontology.
- **DODDLE [92]**
DODDLE is a graphical ontology design tool capable of constructing OWL ontologies.
- **Owl Interactive[93]**
Owl Interactive is an ontology design tool capable of constructing OWL ontologies.
- **DL-Learner [94]**
DL-Learner is a supervised machine learning API. It uses an OWL-DL ontology combined with a user supplied example to infer OWL class expressions that are instances of the example provided. Using this method, unseen instances of the supplied example can be found in the ontology.

An example of such an activity would be to determine which chemical compounds cause cancer. In such an instance the ontology would model chemical compounds in general. A knowledge base would contain information about known cancer-causing compounds. Combining the API, ontology and knowledge base, class expressions can be derived for unstated cancer-causing compounds.
- **Jastor [95]**
Jastor converts OWL ontologies into JAVA interfaces, classes, factories and listeners to allow easy access to OWL ontologies in JAVA.

3.6.5.2 Ontology API's

Ontology API's allow software developers to interact with ontologies using a computer programming language. In this project, JAVA is the chosen programming language and only API's that can be used in JAVA was investigated:

- **AgentOWL [96]**
AgentOWL is an ontology tool that allows agents on the JADE multi-agent platform to communicate using OWL ontologies.
- **OntoBridge [97]**
OntoBridge is a tool used to load and work with ontologies in JAVA.
- **Jena [98]**
Jena is an API for manipulating ontologies in various ontology languages. Jena has grown out of work done by the HP Labs Semantic Web Programme, but has since evolved into an open source project. It has features such as ontology persistence and simplifies interaction with ontologies. It allows the integration of many popular inference engines such as Pellet [99], Racer [100] and Hermit [101].

- **OWL API[102]**

The OWL API package enables the reading and writing of OWL-DL ontologies from a JAVA program.

3.6.5.3 Ontologies and Multi-Agent Platforms

The usage of ontologies in some of the most prominent currently active free multi-agent platforms is described in this section.

- **JADE [49]**

The most popular and well-known agent platform is JADE. JADE has built-in support for a frame-based ontology and provides interfaces the programmer can implement to construct and use an ontology. Tools exist, such as the OntologyBeanGenerator 4.1 [103] plug-in for Protégé that enables Protégé to generate JADE compatible ontologies through a GUI. The tool can also convert OWL and RDF ontologies to JADE compatible ontologies.

- **JADEX [40]**

As JADEX can run on top of the JADE multi-agent framework, the same JADE ontologies is supported by it, provided that the JADE adapter package is used as the underlying agent platform. The Beanyner [104] plug-in for Protégé can also be used to generate JADEX or JADE compatible ontologies by using a GUI. The ontologies used in JADEX are composed of Java classes.

- **JIAC [105]**

JIAC uses an ontology language called JADL. JADL [106] is a LISP [107] like language that enables the construction of frame based ontologies. The language also allows the inclusion of functions and the integration of native Java code. JADL classes are eventually compiled to Java classes for use by the platform. JADL is based on first order logic and is also used to describe an agent's goals and plans.

3.6.5.4 Rule Engines and Semantic Reasoners

The terms rule engines and semantic reasoners are close in meaning to one another but also have some subtle differences. Both terms are types of inference engines, meaning that they infer new facts from data provided to them. The difference between the two is the type of databases used. Rule engines use rule bases while semantic reasoners use knowledge bases. Rule engines are mostly used in business applications to specify and use business rules. Business rules are removed from conventional programming structures and specified in a rule engine's rule base. In this way business rules such as "No sales to people under the age of eighteen" can be changed at runtime without recompiling code. Semantic reasoners use ontological knowledge bases to infer unstated information and reason about queries presented to them. Semantic reasoners can also function as rule engines depending on the way concepts in the ontology are related to one another and the query presented to the reasoned.

Possibly useful Java rule engines include:

- **Drools [108]**
DROOLS is a large business software package that includes a rule engine. Rules are stored in a local JBoss [109] server in a repository so that they are easily accessible. DROOLS is described in more detail later in the following section.
- **SWEET Rules[110]**
SweetRules is a rule engine focussed on the RuleML (Rule Markup Language) [111] and SWRL (Semantic Web Rule Language) [112] rule languages. SweetRules also supports OWL.
- **JRuleEngine [113]**
JRuleEngine is a JSR-94 [114] rule engine implementation. JSR-94 is a JAVA specification for rule engines.
- **JLisa [115]**
JLisa is a JSR-94 rule engine implementation using LISP as a rule language.
- **JESS [116]**
Jess uses its own rule language called JessML [116] and is a JSR-94 compatible rule engine. A commercial licence has to be acquired to run Jess outside an academic environment.
- **Jadex Rules [117]**
Jadex Rules is a small rule engine intended for use in other software projects. Rules are specified using JAVA and or a variant of the CLIPS language. Jadex Rules is built-in component of the Jadex Multi-Agent System.

Various interesting JAVA semantic reasoners are available:

- **HermiT [101]**
HermiT is a reasoner for OWL-DL ontologies. The API is released under the LGPL licence so no commercial licence is required to run it.
- **KAON2 [118]**
KAON2 an ontology manipulation tool with a built-in inference engine. An almost complete subset of OWL-DL and F-Logic [119] is supported ontology languages. A commercial licence has to be acquired to run KAON2 outside an open source environment.
- **Pellet [120]**
Pellet is an established, well-featured OWL-DL inference engine. A commercial licence has to be acquired to run Pellet outside an open source environment.
- **RacerPro [100]**
A commercial licence has to be acquired to run RacerPro outside an academic environment. RacerPro supports OWL-DL, although not completely.
- **QuOnto [121]**
- QuOnto is an OWL-Lite [122] ontology manipulation API with its own inference engine.

3.7 An Evaluation of Tools and Methodologies to Plan, Execute and Monitor Production

3.7.1 Introduction

The scheduling, re-scheduling and optimisation of production sequences within the system being developed is a planning problem. Several compatible planning methods and/or API's are available for evaluation. Many theoretical methods of planning are available with many articles describing possible ways to solve planning problems, but due to time-constraints not all of these can be investigated. A ready-to-use planning API written in JAVA is a requirement for the chosen planning method.

The chosen planning method should be able to evaluate the production plans in the production plan stack and generate a schedule that will make optimal use of all possible production hardware. Parallel execution of plans is a requirement of the chosen method. If failures occur in the production line, the planner should be able to re-plan, taking the failure into account (if possible).

3.7.2 Planning and Scheduling API's

Planners of interest include:

- **DROOLS Planner [123]**

The DROOLS Business Logic Integration Platform contains the DROOLS Planner as component. DROOLS combines various optimised search algorithms to solve planning problems. The planning constraints and scoring functions are represented by rules that the user has to create in the proprietary DROOLS Rule Language (DRL) language. The DRL language is used in the DROOLS rule engine. The DROOLS Planner uses the DROOLS rule engine during planning.

In DROOLS, planning problems have certain constraints attached to them. The first of these constraints are ones that should not be broken for the solution to be at least feasible (negative hard constraints). A negative soft constraint should not be broken if at all possible, although if it is broken the solution can still be feasible. A positive constraint or reward should be fulfilled if possible. This type of constraint is less important than negative soft constraints. An optimal solution is one that has the highest score of all the feasible solutions.

DROOLS is supplied with good documentation and several examples. Various presentations about DROOLS can be found on the Internet. Example problems such as the classic bin-packing and roster generation for exams and travelling sports teams are included.

DROOLS does not use a specific ontology language to represent the problem domain. Normal Java objects are used to describe the problem and solution. DROOLS provides interfaces that have to be implemented by the programmer in the problem-specific classes. DRL is used to express rules. The only information coded in DRL are the problem constraints expressed as score rules. This makes it easy to add, remove or change the problem constraints without changing any JAVA code or rebuilding the program.

- **DLPlan [124]**

DLPlan is a semantic planner. It uses OWL-DL as a language to represent the domain and problems it has to create plans for.

To achieve this it uses two other open source API's (OWLAPI to read OWL-DL) and PELLET to reason. DLPlan is an intriguing combination of technologies and it is particularly interesting because it uses the OWL-DL language. This language is also used to construct domain ontologies – another region of interest in this project. A possibility exists that this package could integrate nicely into the project. Of course this would only be possible if OWL-DL is chosen as the domain ontology language in this project.

DLPlan comes with the classic pizzeria example. All example problems revolve around assembling various types of pizzas. An OWL-DL ontology describes an pizzeria domain. DLPlan uses text files to specify the initial state of the domain and the problem to solve. DLPlan does not come with any more examples and community support is non-existent.

- **PL-Plan [125]**

PL-Plan may not be used for commercial projects. This licence limitation makes it unsuitable for use in this project. PL-Plan also uses the GraphPlan[126] algorithm.

- **JPlan [127]**

JPlan uses the GraphPlan algorithm to solve problems. Not much literature exists on JPlan and no documentation is provided with the API. Three planning examples are included. However, the GraphPlan algorithm is a very interesting problem solving algorithm that requires further investigation.

- **PDDL4J [128]**

PDDL4J is a platform containing a GraphPlan-based planning algorithm. The API does not have good documentation, but a few examples of planning problems are included.

3.7.3 Production Execution

3.7.3.1 Petri Net Based Execution

A Petri net (a type of directed graph) is a graphical and mathematical modelling tool that models in great detail and precision information processing systems that are: concurrent, asynchronous, distributed, non-deterministic and/or stochastic [129]. The Petri net was developed by Carl Adam Petri in 1962. An example of a Petri net diagram can be seen in Figure 3.15 where a parallel operation in a manufacturing system is modelled. Discussions of the usage of Petri nets in production scheduling, control and error recovery can be found in [130], [131], [132] and [133].

As can be seen in the articles mentioned in the Introduction section, Petri nets are a point of interest for research in industrial automation. Petri nets model complex processes with a solid mathematical base and standard notation. There are also JAVA Petri net manipulation tools and API's available to enable the programmatic use of Petri nets.

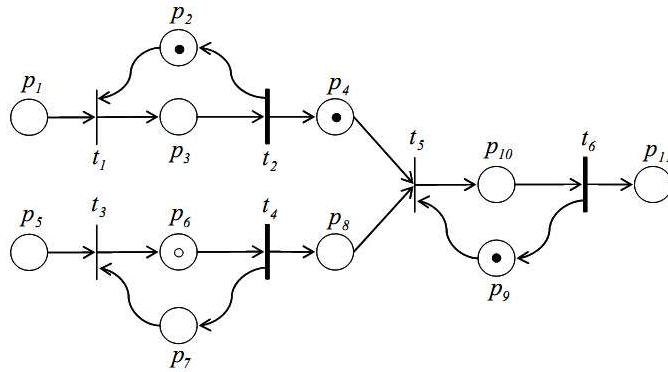


Figure 3.15: Parallel Manufacturing Tasks [131]

Petri nets rely on a large amount of very complex mathematical concepts which is not in my current skill set. Mastering the math required to effectively use Petri nets might take up too much development time. Also, the practical real time control of physical production processes might not be possible with the programming tools and API's that are currently available as Petri net implementations focus almost exclusively on simulation of processes rather than execution. Petri net tools of interest include:

- **JFern [134]**
JFern is a JAVA coloured Petri net implementation. The package includes a GUI visualisation tool to visualise and debug Petri nets that are constructed with the API. The package includes fairly good documentation and two examples.
- **JPetriNet [135]**
JPetriNet is a JAVA Petri net implementation. It allows for the modelling and analysis of conventional and timed Petri nets. No documentation or examples are included with the package.
- **Renew [136]**
Renew is a JAVA Petri net implementation. It allows for the modelling and representation of reference nets, an object-oriented version of Petri nets. Documentation as well as some examples is provided. This API is interesting because of the JAVA code that can be executed in transitions during the execution of the Petri net.
- **CO-OPN/2 [137]**
CO-OPN/2 (Concurrent Object-Oriented Petri Nets) is a formal specification language that was devised to support the specification of distributed software systems. Petri nets and the object-oriented design paradigm are combined with sophisticated synchronisation mechanisms to enable large distributed software systems. CO-OPN code can be converted into JAVA classes for execution.
- **Petri net Kernel [138]**
PNK is a JAVA Petri net implementation with a graphical Petri net designer. It can load, save and execute Petri nets in the PNML (Petri Net Markup Language) [139] format. Documentation and examples are provided.

In [133] an approach to error recovery in a Petri net based manufacturing control system is presented. When an error occurs at a certain place, a Petri net is dynamically constructed to recover from the error by moving execution to before the error occurred; however, choosing the recovery point to transfer execution to is not discussed in the article.

3.7.3.2 Adapted Workflow Engine Based Execution

A workflow engine is a software framework that allows a programmer to design a software system according to the workflow metaphor. The workflow metaphor is based on a series of steps that are performed in a certain order that describe actions that need to be performed on some object by specified people, groups or mechanical devices.

An investigation of workflow engines is included because of the similarities observed between workflows and the flow of products through a production or assembly line. Workflows also allow for the execution of concurrent processes. Concurrent processes are an integral part of many production or assembly lines.

The workflow metaphor might not be a perfect fit to what is needed in executing a production schedule. The available API's might not have the required functionality or be too focussed on business process workflows. Another problem might be that the API's might not support the dynamic generation and consequent execution of those generated workflows. Workflow engines of interest include:

- **Open business engine (OBE)** [140]

OBE is a business workflow engine that adheres to many industry standards. It comes with documentation and several examples. It seems that this framework is geared towards business workflows and might not be appropriate for the type of execution needed.

- **WADE** [141]

WADE is a workflow engine based on the JADE multi-agent platform. According to the documentation, WADE is focussed on lower level processes than conventional workflow engines that normally focus on higher level business workflows. Also, this platform is of great interest since the overall system is to be implemented on the multi-agent platform JADEX, which also runs on top of the JADE platform just like WADE. The API comes with documentation and an example. WADE is supplied with a graphical design tool called WOLF (an Eclipse plug-in) that facilitates the graphical construction of workflows. Unfortunately WADE does not support the continuation the workflow based on listening for external events.

- **DROOLS Flow** [142]

The DROOLS Business Logic Integration Platform contains the DROOLS Flow component. DROOLS Flow is a workflow engine. The engine contains many interesting features such as splits in flow that result in parallel processes and the specific inclusion of human tasks in the workflow. DROOLS Flow is supplied with extensive documentation and a visual flow design tool that is implemented as an Eclipse plug-in.

DROOLS also supports external events to trigger the continuation of execution of the workflow from a certain node. Timers can be added to the workflow either as an integral part of the workflow or they can be associated with a node – activating when the node is reached and performing some other action on each firing.

Upon contacting the DROOLS developer responsible for DROOLS Flow and explaining what I intend to use the API for, I was assured that using the API for the intended purpose would indeed be possible. Also, I was assured that all files required in execution of the workflows can be generated without using the visual design tools.

- **JADEX Processes [143]**

JADEX Processes is integrated into the latest release of JADEX (Version 2.0 RC2). It features a graphical design utility for designing workflows. The utility was successfully installed. The platform appears to contain all the required features. Documentation is available, but is not yet complete and no existing examples are available. The API is not very mature yet, but is interesting due to the fact that the multi-agent platform chosen for the project is JADEX, and this API is integrated into the latest version of JADEX.

3.7.3.3 Proposed Model for Production Execution

The following model for production execution is proposed. The model is influenced by the various completed literature reviews. A conceptual model of the proposed solution was developed and is presented in this section. The solution is closely based on interacting with the database and ontology design. The solution relies on some type of planner that can generate optimal schedules for it to execute.

Views of the system

The production execution system can be modelled in various ways; each model representing a different dimension. The models can be described as follows:

- **The Physical Model**

This model represents the physical location of devices on the production floor. Their positions relative to one another, physical coordinates relative to their parent device and location relative to abstract containment concepts such as “assembly cell” or “building” are stored as a tree in the database using parent-child relationships. These abstract containment concepts are not hard coded, allowing for a high level of flexibility. Different physical device layouts are possible since the overall system is reconfigurable.

- **The Assembly Flow Model**

This model hooks into the physical model of the system, but models the different ways assemblies can physically flow between devices. Due to the design of the database, bi-directional flows and multiple paths are possible to represent. Limitations on how the devices are connected to one another are represented by “ports” as introduced in [144].

- **The Production Sequence Model**

This model uses the previous two models to represent a legal production sequence. This model is represented as a tree structure containing various production objects and is explained in the following section.

Description of Production Sequence Model

Once the production stack (list of orders) with one or multiple orders is processed by the planning engine and a schedule is generated, the schedule will be decomposed and loaded into a production step tree. The tree will contain nodes that are composed of “production step” objects. Each production step object has other objects inside it that are necessary for the functioning of the object. These objects include:

- **Device IO objects**

A device IO object has the ability to communicate with a physical device in the production line on a specified IO. Communication with the actual device will be through the appropriate device agent.

- **Performative Objects**

Performative objects contain commands that can be sent to a physical device on the production line and represent actions the particular device should perform.

The production step tree will be executed alongside a condition tree. The condition tree sets certain pre- and post conditions for the execution of steps in the production tree. The functioning of this model can be seen in Figure 3.16.

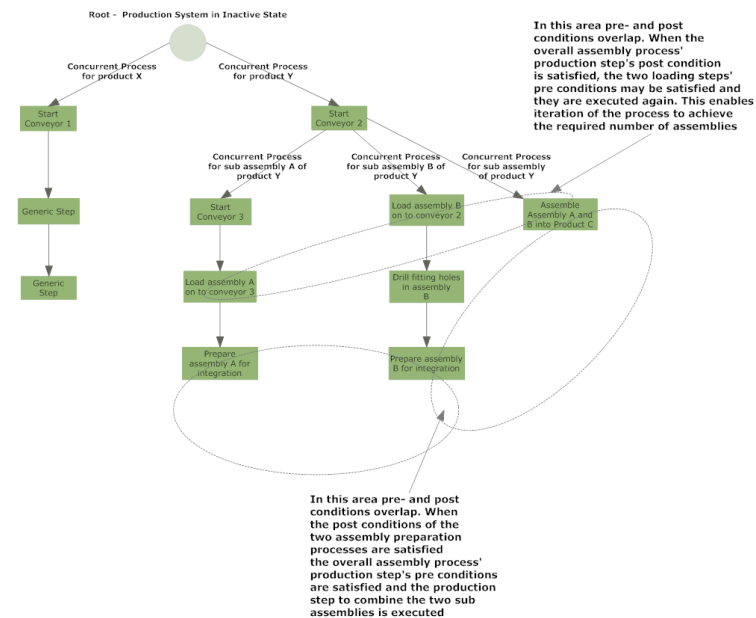


Figure 3.16: Overall Production Tree Example

As can be seen from Figure 3.16, the production steps are connected to each other in a hierarchical manner. Using a hierarchy, the production steps are chained together and an order of execution is established. When a node has more than one child node, each child node or production step represents the start of a new process that can execute concurrently with all other processes in the whole production line. If concurrent processes share a common parent production step, that step was necessary for them to be able to start successfully. Events (data changes on devices) that update conditions drive the production execution forward. Production steps can have one or more conditions that must be satisfied before it can execute. These are called pre-conditions.

After a production step has executed, one or more post conditions should be satisfied. If all the assigned post conditions are satisfied the production step executed successfully. The post-condition steps of a particular production step can also feature as pre-conditions for other production steps. Some pre-conditions (type 2 pre-conditions) are only taken into account the first time the production step is executed and ignored in further executions for the particular production process or run.

In Figure 3.16, a node or production step in the production tree is presented. The name of this node is “Assemble Assembly A and B into Product C”. The node has action objects (“Assemble Assemblies A and B” and the reverse action “Abort Assembly”). It has two pre condition objects (“Assembly buffer A contains an Item” and “Assembly buffer B contains an Item”). When all the pre conditions are satisfied, the production step can execute its action. When an action is executed, one or more post-conditions are updated. If these post-conditions are not satisfied, the execution of the production step has failed. In such an instance, all parent nodes or production steps’ reverse action will be called recursively. This is done to temporarily stop all upstream devices from the failed device from sending assemblies to the non-functioning device. Downstream devices in the process from the failed device can still continue to function.

To enable the production of a specified amount of products in a particular sequence, the pre- and post-conditions of production steps are linked in such a manner as to form a indefinitely repeating process. The number of times the process is repeated is monitored by the production execution agent.

3.7.4 Production Monitoring

A full production monitoring GUI is not within the scope of this project, although an OPC monitoring client was written to test out how various OPC access API’s and can be used to monitor all OPC devices in the component handling system. A screenshot of this client can be seen in Figure 4.10. To demonstrate and execute production tests, a simple operator GUI will be designed.

3.8 An Overview of Methods to Implement Security within the System

A security framework has to be built into the system from the ground up as it is very difficult trying to integrate security into a system as an afterthought. Security in this system can be defined as controlling and logging access to system objects or resources.

A system object or resource is any object within the system such as the database or a physical machine or an action. Access to system objects are pre-defined actions that users can perform on the objects. Users in the system that access objects can be human and/or computer.

The problem with integrating security into a system is complexity, which leads to the system taking longer to implement. Because of the limited time available to implement this project, security cannot be the main focus. Nevertheless, security is very important and a suitable security model has to be identified and integrated into the system. The security model can then be extended in the future, making the system more secure when time is available to do this.

Security in this system can be seen from the following perspectives:

- Protecting the system from external threats such as agents that try to gain access to the MAS in order to access resources.
- Protecting the system from unauthorized users or simple user error.
- Protecting the system from its own internal agents.
- Securely sending and receiving messages between agents in the MAS.

It is clear that a Role-based access control (RBAC) [145] architecture is one of the necessary security components of the system. Such an architecture requires a database to store access permissions. Therefore, the design and implementation of such a database will be necessary. It is not anticipated that the MAS will be open for direct connection from agents in other MAS's.

After evaluating security options the following methods to implement security within the system have been identified:

- Integrate a Role-based access control model into the system at design time coupled to a suitable RBAC database. This integration will have to be compatible with the chosen agent design methodology and code generation tools.
- Use an agent development methodology and or runtime environment with adequate built-in security features.
- The use of a plug-in into an existing platform to achieve security.

The following studies will be used to gather more information about the methods listed above:

- The NIST Model for Role-Based Access Control: Towards A Unified Standard [145].
- A Role based Access Control Model for Agent based Control Systems [146].
- Security Patterns for Agent Systems [73].
- Security in Multi-Agent Systems [147].

3.8.1 Role Based Access Control (RBAC)

Role based access control is described as: "RBAC provides a valuable level of abstraction to promote security administration at a business enterprise level rather than at the user identity level. The basic role concept is simple: establish permissions based on the functional roles in the enterprise, and then appropriately assign users to a role or set of roles.

With RBAC, access decisions are based on the roles individual users have as part of the enterprise. Roles could represent the tasks, responsibilities, and qualifications associated with an enterprise.”[145]

Various levels of RBAC exist. The higher levels include all the functionality of the lower levels, but add features, thus increasing complexity. The levels of RBAC are [145]:

- **Flat RBAC**

Flat RBAC is the simplest form of RBAC. Users are assigned to roles; roles are assigned permissions. Users can exercise all the permissions they have through role assignments at the same time.

- **Hierarchical RBAC**

Hierarchical RBAC add the requirement of role hierarchies. Senior roles inherit the permissions of their junior roles. Two types of hierarchical RBAC exist: General Hierarchical RBAC specifies an arbitrary partial order role hierarchy. Restricted Hierarchical RBAC imposes restrictions on how the hierarchy of roles is formed.

- **Constrained RBAC**

Constrained RBAC introduces the concept of Separation of Duties (SOD). In this model, the execution of a task is delegated over various users. Variations include static SOD based on role assignment and dynamic SOD based on role activation.

- **Symmetric RBAC**

Symmetric RBAC adds the requirement of permission roles review. This means that it should be possible to determine to which roles a permission was assigned and which permissions are assigned to which roles.

3.8.2 The RBAC Database Schema

The RBAC database schema was adopted from [145] and [148]. The level of RBAC chosen is the Restricted Hierarchical RBAC model. The restriction on the model is implemented by building a tree-like hierarchy of roles.

The RBAC schema is not a separate database from the system database. A system audit schema has been designed and attached to the RBAC schema to enable the tracking of permission usage. User actions were incorporated into the schema. These actions are generic verbs such as “Execute”, “Impersonate”, “Update” and “Delete” that can be assigned to different permissions to further refine them.

The concept of a User Group was introduced into the design. This means that roles are assigned to User Groups instead of Users. Users are assigned to User Groups. In this design it has been decided that human users, external software users and agents are all the same type of object. Using this approach a unified way of enforcing a RBAC based security model is established.

3.8.3 Integration of RBAC into the MAS Design

RBAC will be integrated into the MAS design during the design of the multi-agent component of the system. This will be achieved through following RBAC design patterns. This will ensure that security is integrated into the system on a relatively low level from the beginning of the design process.

3.9 Conclusion

The only way to completely realise the features and advantages of ontological technologies (ontologies, reasoning capabilities, etc.) in the product manufacturing or assembly domain is to use a common ontology for all the concepts in the domain, or use an ontology for each unique section and map the various ontologies to each other. This means that every concept of the production domain must be modelled in a common ontology and that relationships in the ontology between these concepts be established. The reason for this is that if all the concepts in the process are represented in a common ontology and relationships are defined between them, all aspects of the system can be taken into consideration on the computational level. This enhances the possibilities of handling concepts such as automatic planning re-configuration, error recovery and scheduling through the use of computational means.

Most multi-agent platforms contain no built-in inference engine that can infer new or unstated information based on input from sensors in its domain or reason about facts in a knowledge base. The multi-agent platform JADEX and JIAC have a Belief-Desire-Intention (BDI) rule engine that can logically reason about facts in its belief base and how these facts relate to the goals an agent has. Based on this logical reasoning it can execute plans to achieve its goals. The agent's belief base is updated to reflect the state of the domain in which it is running by providing the agent with information from sensors in the domain. The platform cannot, however, infer new information that is not available from sensors and integrate this information into its belief base. This is where a domain ontology coupled to a separate semantic reasoner can enhance the functionality of a multi-agent system.

The following technologies regarding ontologies, production planning and production execution have been chosen. Choosing and committing to a certain component is not an easy decision as there are so many components available. Also, other constraints such as project scope and time limits need to be taken into account. It became apparent that employing the preferred technologies discovered during this literature review would be too time consuming and outside the scope of the project. The effort is, however, not futile as it provides a base for future research that would improve upon this system.

3.9.1 Chosen Ontology Language

The preferred ontology language is OWL-DL as it permits sound and complete reasoning. Numerous utilities and API's are available in order to work with OWL-DL in Java. However, to make full use of the inference capabilities of a description logic ontology language such as OWL-DL a complete equipment, process and product ontology for manufacturing has to be designed first.

These ontologies in OWL-DL could not be found and designing these ontologies is a whole project in itself. The chosen ontology language for this project will therefore be plain Java classes based upon metadata gathered from the system database and domain.

3.9.2 Chosen Ontology Design Tool

The Protégé ontology design tool will be used to visually design the ontology. This tool is the de facto standard tool for visual ontology design. It was discovered that the OntologyBeanGenerator plug-in for Protégé can be used to construct JADE and JADEX compatible Java ontology classes.

3.9.3 Chosen Upper Ontology

Currently, the only upper ontology aiming to provide a base for manufacturing ontologies is MASON. This upper level ontology is specifically designed for manufacturing and is available in OWL-DL, but is a very high level ontology and contains only very abstract manufacturing concepts. Due to the fact that OWL-DL will not be used as the project ontology language, no upper ontology will be used in this project, but the currently available ontologies will be studied.

3.9.4 Chosen Ontology API

The preferred ontology API that can be used to programmatically manipulate (load, query, persist. etc) an OWL-DL ontology is Jena. The API is mature and specifically supports the Hermit inference engine by using a plug-in infrastructure. Due to the fact that a Java ontology will be used in the project, an ontology API is not required. The JADEX multi-agent platform is capable of using plain Java ontologies.

3.9.5 Chosen Inference Engine

The preferred inference engine is Hermit. The API is currently the only viable choice if an OWL-DL ontology was to be used in the project because it is the only OWL-DL inference engine released under a license compatible with this project (LGPL) and appears to be a solid piece of software. The developer of Hermit also developed the KAON2 inference engine. Due to the usage of a Java ontology in this project, an inference engine is not required.

3.9.6 Chosen Production Planner

Production planning is to be done using DROOLS Planner.

3.9.7 Chosen Production Executor

Initially DROOLS flow was tested within a small test scenario and found to be suitable as an execution engine due to its support of event-based flow control and apparent support for concurrency. Further testing using a real physical process, however, showed that it is not suitable when real-time concurrency is required and events occur much more frequently than in the simpler initial test scenario. A small truly multi-threaded, event-based production execution engine will therefore have to be developed that can be imbedded in each device agent.

Chapter 4: Implementation

4.1 Introduction

The implementation of the project will follow the following stages:

- Determining a method of communicating from Java programs to OPC devices and configuring OPC access within the project.
- The development of the multi-agent software system skeleton according to the chosen methodology.
- The development of a system database and system ontology.
- The implementation of the various agents using the various methods and technologies identified in the literature review.
- The combination of the various components into a functioning MAS.
- Testing the system on the test platform being constructed in the lab. This might not be completely possible since the construction of the test platform is an ongoing project.

4.2 Java to OPC Communication Method

4.2.1 Introduction

As the system communicates with production devices in the factory using OPC, and the system is to be implemented in the Java programming language, a method has to be found that will enable programs written in Java to reliably communicate with OPC. This has turned out to be a major hurdle, taking up large amounts of project time. The problems encountered during testing current Java to OPC communication API's at the time of writing were the following:

- The Java to OPC communication API's tested did not possess the required functionality.
- The API's were unreliable or too slow.
- The API's tested did not support all OPC data types.
- The API's tested had a combination of the three above mentioned problems and was prohibitively expensive.

Because of the problems encountered it was decided to design and implement a framework that could be placed between the system and any third party Java to OPC API - shielding the system from changes if another Java to OPC API is used.

To implement such a pluggable framework, many software design patterns were employed. Software design patterns are "best practices" in the software development community relating to the design and implementation of software [149]. The framework allows for the swapping out of different third party Java to OPC API's without requiring major code changes to the system.

4.2.2 Java to OPC Framework Design

4.2.2.1 Framework Core: Modified Composite Design Pattern

The core of the Java to OPC Framework is composed of a modified Composite design pattern (see Figure 4.1). The design pattern represents a tree-like data structure, with branches (AOpcComposite) and leaves (AOpcItem) inheriting the same super type (AOpcDataComponent). Branches and leaves thus have the same super type. This allows branches and leaves to be accessed as their super type for shared operations. Branch objects are containers and contain a reference to an object of the super type (AOpcDataComponent), thus it can contain a reference to either another AOpcComposite or just an AOpcItem object. In this way a tree of AOpcDataComponent objects is constructed. An AOpcItem object represents an actual OPC Tag or Item. All these classes are abstract, meaning they cannot be instantiated on their own, but have to be extended in concrete classes. This design pattern was chosen due to its conceptual similarity to the way OPC objects are represented.

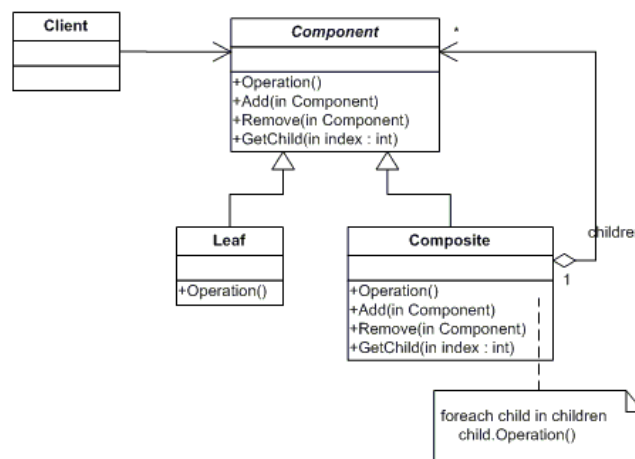


Figure 4.1: Composite Design Pattern [151]

An AOpcComposite object or branch can be any OPC concept that contains other OPC concepts. A branch can represent the following:

- An OPC server IP address container
- An OPC server container
- An OPC server group container

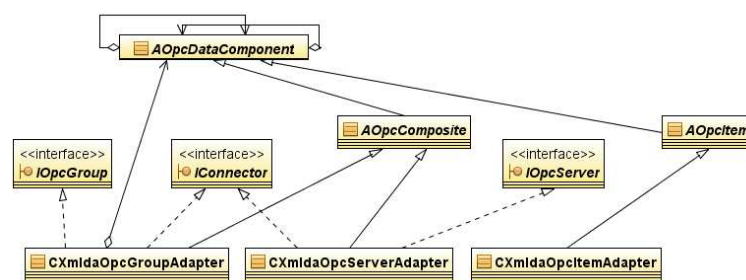


Figure 4.2: Modified Composite Design Pattern

As can be seen in Figure 4.2, a XML-DA concrete implementation of the Composite design pattern is presented. The interfaces (IOPCGroup, IConnector and IOPCServer) are modifications to the Composite pattern and are implemented by the concrete classes. They enforce functionality specific to an OPC Group and OPC server without introducing rigidity into the design.

Another software design pattern employed in this design is the Adapter software design pattern (see Figure 4.3).

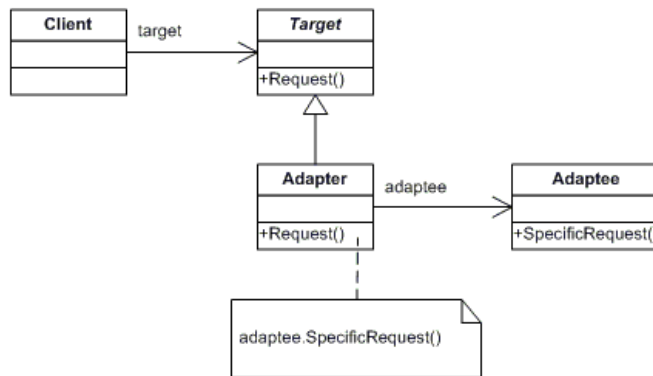


Figure 4.3: Adapter Design Pattern [152]

The Adapter design pattern provides an object of a certain type to the calling client code, while using an underlying object of a different type's services [149]. The concrete communication classes encapsulate required third party Java to OPC API classes, routing function calls between the API's public functions and the concrete communication class functions. Using this design pattern, Java to OPC communication API's from different companies conform to common interfaces and are used interchangeably by the system.

4.2.2.2 Other Framework Classes

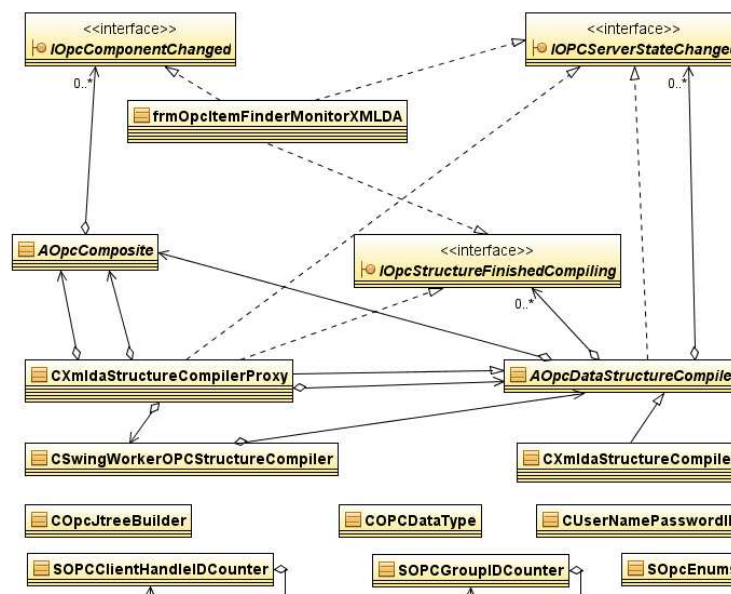


Figure 4.4: Other Framework Classes

Utility Classes

The framework classes are utility classes (see Figure 4.4) necessary for the framework to function, but are not part of the core Composite and Adapter design pattern model. The following utility classes of note are described:

CSwingWorkerOPCStructureCompiler

This class contains an AOPCDataStructureCompiler object and allows the AOPCDataStructureCompiler object within it to execute in a worker thread and safely update the GUI from the worker thread.

COPCDataType

This class represents an OPC data type and its equivalent Java data type and value.

SOPCEnums

This static class contains enumerations used throughout the system.

SOPCClientHandleIDCounter and SOPCGroupIDCounter

These static classes provides a unique integer to calling code on each call and is used where unique identifiers are required.

Event-Based Notification

In this design, three Interfaces are present (IOPCComponentChanged, IOPCServerStateChanged and IOPCStructureFinishedCompiling). These interfaces are used in objects that need event-based notification of changes to OPC components, OPC Server states and to notify a listener when a scan of OPC servers for components has completed. This is an implementation of the Observer software design pattern (see Figure 4.5). The observer design pattern allows an object to inform many other registered objects that a particular change in its state has occurred [149].

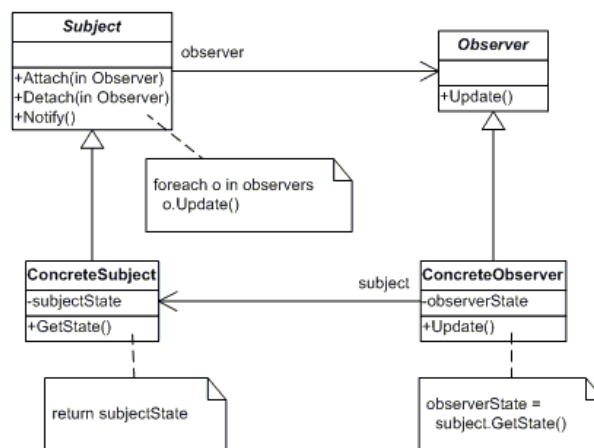


Figure 4.5: Observer Software Design Pattern [153]

OPC Data Structure Compilation

The AOPCDataStructureCompiler abstract class provides a base class for concrete objects that scan and compile a tree of OPC components. As such a search operation can take a few seconds; a graceful mechanism is required to update the GUI and keep it responsive while the process is active. The solution for this problem was found by implementing the Proxy software design pattern coupled with multi-threading.

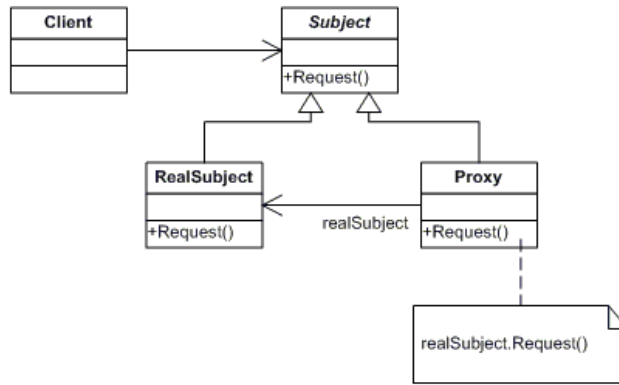


Figure 4.6: Proxy Software Design Pattern

The Proxy design pattern (see Figure 4.6) regulates access to a particular type of object and can be used by client code as a placeholder or surrogate for an expected object type [149].

Using the Proxy design pattern combined with a multi-threaded search operation, the user of the test GUI is immediately informed (see Figure 4.7) by a proxy object of the correct type that a scan has started after the Search button is clicked. When the search returns, the GUI is updated safely as can be seen in Figure 4.8.

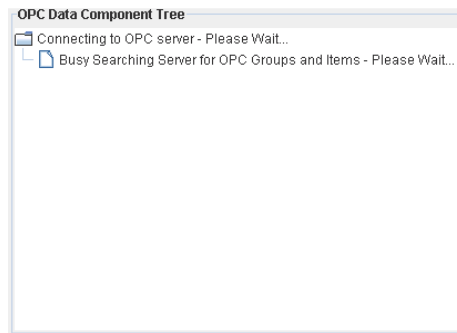


Figure 4.7: Proxy Design Pattern in Action Step 1

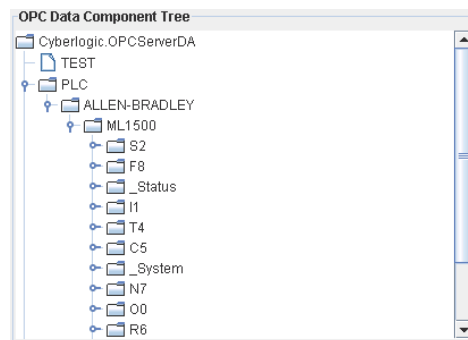


Figure 4.8: Proxy Design Pattern in action Step 2

In the meantime the OPC search proxy object has instructed a CSwingWorkerOPCStructureCompiler object to initiate an OPC search in its underlying OPC search object. At the end of the scan the GUI is updated with the results contained in the real scanning object. All the while the GUI stays responsive to other user commands. The advantages of the multi-threaded search process are not only limited to GUI interaction. It will improve performance in other system software components (that still have to be designed and implemented) that function without a GUI and rely on the search object for OPC search data.

4.2.2.3 Recursive Techniques

Various recursive code operations are present in the Java to OPC framework. Recursion is a powerful programming concept that involves a function calling itself repeatedly. This technique is very useful in constructing and traversing tree-like data structures. Recursion is used in the Java to OPC framework to accomplish the following:

- Search a given computer IP address for OPC servers, in turn searching each OPC server found for OPC groups and then the OPC items within them. During this process a tree data structure is constructed.
- Compile a visual representation of the tree data structure of OPC objects for use in the testing GUI.
- Flatten the tree data structure of OPC objects in order to enable client code to search the tree structure quickly for a specific OPC object. This is done with the help of the Iterator software design pattern.

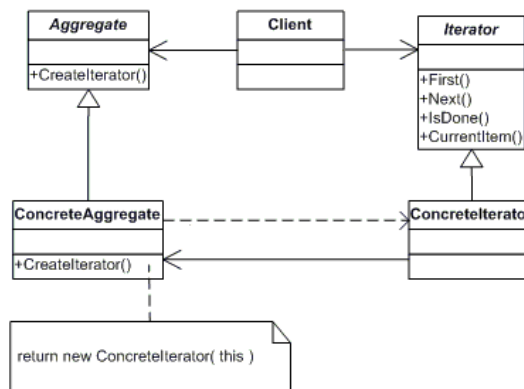


Figure 4.9: Iterator Software Design Pattern [154]

The Iterator software design pattern (see Figure 4.9) allows client code to access the elements of any collection sequentially [149]. The Java to OPC Framework OPC objects have been enhanced by implementing the Iterator software design pattern within them. This allows a recursive function to construct a flat view of the tree data structure.

4.2.2.4 Testing GUI

A testing GUI was developed to test the OPC communication API's.

The GUI allows the user to scan OPC servers and generate a tree-like structure of OPC components. The structure represents OPC concepts hierarchically and allows the user to easily navigate the OPC structure. OPC items can also be monitored from this GUI (see Figure 4.10).

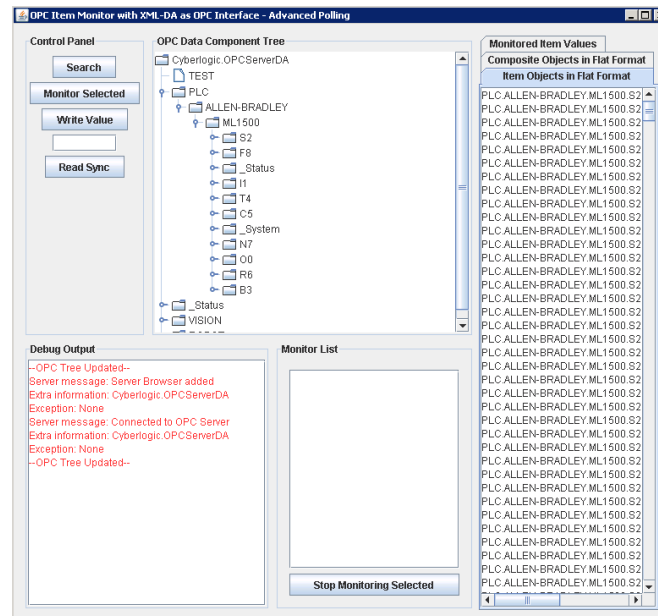


Figure 4.10: OPC testing GUI

4.2.3 Chosen OPC Access API

After much testing of other Java to OPC API's, XML-DA [4] was tested, written into the Java to OPC communication framework and adopted as the OPC access method of choice for this project. Although this method of access is not as fast as OPC-DA, it offers features such as:

- Adequate performance for the particular project implementation.
- Simplified security.
- Easier setup than OPC-DA.
- Reduced cost in the context of this particular project.
- Platform independence.

XML-DA uses web service calls as underlying communication method between the server and clients. To access the XML-DA server in this project, the Java API for XML Web Services (JAX-WS) [150] framework is used. XML-DA supports basic and advanced polling for changes in data items. Manual polling involves creating a subscription for certain items and calling the web service periodically to return the data for items that have changed since the last poll.

Advanced (delayed) polling is essentially the same as basic polling, but does not use periodic polling. A subscription is also created on the server and the server is polled initially. Data for all the subscribed items are returned in the initial poll.

The client then polls the server again immediately. If any of the item's data changed between the initial poll and the current poll, the values of changed items are returned. The client polls the server again immediately. If no data changes occurred in the subscribed items, the server keeps the poll request connection open for a specified amount of time (Hold Time + Wait Time).

If any data for items change within the period the connection is kept open, the server completes the connection and returns the changed items to the client. If no changes to the data of items occur in the time period that the connection is kept open, the server returns an empty response to the client. Either way, the client immediately polls the server again. The process is repeated until the client unsubscribes or stops polling the server. This method reduces to a minimum the amount of polling a client has to do. The timing parameters are explained below and illustrated in Figure 4.11:

- **Hold Time**

This parameter instructs the server to delay for a certain amount of time in returning a response when data for subscribed items has changed. It can be seen as the update rate parameter in traditional OPC-DA connections and is usually set to 500 milliseconds in this implementation. This is the minimum amount of time a request to the server will take.

- **Wait Time**

Wait time is the maximum amount of time the server will keep the connection from the client open after the Hold Time parameter has elapsed. The current implementation uses 25 seconds as the value for this parameter.

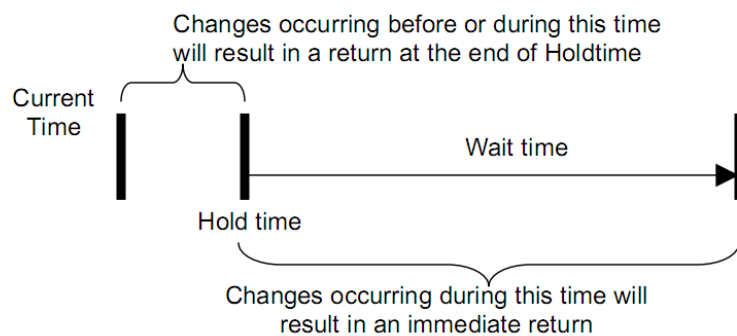


Figure 4.11: XML-DA Response Timing [156]

4.3 Design of the Multi-Agent System According to the TROPOS Methodology

4.3.1 Introduction

The chosen multi-agent design methodology is TROPOS. In this section a quick overview of the methodology as well as certain design patterns that will be used during design is given, followed by the actual design.

The TROPOS methodology is visually modelled in the I* notation [65] and supports all analysis and design activities in the software development process from application domain analysis to the system implementation using an incremental approach of refinement [65]. The methodology is composed of the following phases:

4.3.1.1 Identification of Domain Stakeholders and Their Goals

The domain is defined as a currently operational small manufacturing plant; while the system is defined as the software system being developed that will be introduced into the domain. Keep in mind that domain stakeholders need not only be human.

A stakeholder is defined within the context of the Tropos methodology as: “Stakeholders are modelled as social actors who depend on one another for goals to be achieved, plans to be performed, and resources to be furnished” [65]. Taking one of the classic definitions of a “stakeholder” into account as well (a list of definitions for stakeholder can be found in [151]), the definition of a stakeholder in this domain is any entity that depends on any other entity within the domain and that is directly affected by the system or directly affects the system. As the early requirements modelling progressed, many of the goals in the initial list changed. A list of the domain stakeholders with their goals excluding the system-to-be actor is introduced in the list below:

Factory

This stakeholder represents the current devices being used in the facility to produce products:

- Requires the highest possible utilisation of its devices.
- Requires proper maintenance of its devices.

Production Engineering

This stakeholder is responsible for planning and executing the following production processes:

- Generate optimised product design plans in the shortest period of time.
- Generate optimised production plans for products in the shortest period of time.
- Generate optimised product and process quality control plans in the shortest period of time.
- Generate optimised device layouts.
- Generate optimise product and part flows.
- Generate optimised device configurations (programs for PLC’s, Robots, etc.).
- Generate optimised device maintenance schedules.
- Schedule and delegate production execution activities to Operators and Factory Floor workers.
- Implement device layouts and configurations.
- Schedule and delegate device maintenance activities to Maintenance staff.
- Monitor operations and worker morale.
- Take corrective measures.

Shipping and Receiving

This shareholder handles the shipping and management of completed products and the procurement of parts used during production and maintenance:

- Requires precise information on what parts are needed for production so that orders can be placed at suppliers.
- Requires precise information on what parts are needed for maintenance so that orders can be placed at suppliers.
- Requires precise information on which products to ship to which customer and the due delivery dates.
- Maintains a store of parts used in production.
- Maintains a store of products before they are shipped to customers.
- Maintains a store of maintenance parts before they are shipped to customers.
- Delivers the correct quantity of parts to the correct place in the production process at the correct time to keep production going.
- Keeps records of incoming parts and outgoing products.

Staff (Goals common to all staff)

All staff have the following common goals:

- Require labour legislation compliant work schedules.
- Require labour legislation compliant work environment.

Maintenance

The Maintenance stakeholder maintains factory components:

- Requires a well-designed and scheduled maintenance plan to follow.
- Requires the correct equipment and replacement parts to be available when executing a maintenance plan.

Production Staff

Production Staff work in the production line, executing actions on products by hand or by using tools alongside factory devices:

- Requires training in order to correctly execute actions on parts and products.

Operational Staff

Operational Staff control and monitor the production processes. They do not directly perform actions on parts or products in the way Assembly Staff do:

- Control and monitor the production operations.
- Requires to be informed immediately of error conditions in production operations with enough information about the error to be able to quickly remedy the situation.

Customers

Customers are clients or customers receiving some type of output from the factory in exchange for payment:

- Expects a quick and easy way to determine which products to order.
- Requires an easy way to submit product orders.
- Requires an easy way to track product orders.
- Expects products of the correct quality and quantity at the agreed-upon price.
- Expect on-time completion of orders.
- Expects notification in the case of a status change involving their order.

Suppliers

Suppliers provide the factory with raw materials and components used in the production process for payment:

- Requires on-time payment of bills.
- Receive the correct orders at the correct time.
- Continuous communication with operation regarding the placement of orders.

Management

Factory Management workers manage all the aspects of the factory (General Management, Finances, Human Resources, and Communications):

- High-level control over all factory functions.
- Generates strategic plans.
- Generates budgets and manages finances.
- Handles staffing issues (hiring, firing, promotions, disciplinary issues, unions etc).
- Handles customer orders and procurements.
- Interacts and negotiates with all stakeholders.
- Requires high-level reports on real time and historical production operations.

Owner

These shareholders own the factory operations:

- Expects maximum return on investment.
- Expects good management of the factory.
- Requires an overview of operation activities for oversight and planning purposes.

Government

Government includes all regulatory bodies that enforce laws that directly impact on the operations of the factory. Examples of these laws would be labour and environmental laws:

- Requires adherence to labour regulations.
- Requires adherence to environmental regulations.
- Requires adherence to applicable property regulations.

Utility

This stakeholder supplies services to the factory such as electricity, water, sanitation and telecommunications:

- Requires on-time payment of utility bills.
- Requires customers to minimise the usage of resources during peak times.

4.3.1.2 Modelling using the TAOM4E tool: Lessons Learnt

The TAOM4E tool operates within the restrictions described in [66]. It is thus imperative that the restrictions be fully understood before modelling is attempted. It is highly recommended that the thesis “Knowledge Level Engineering of BDI Agents” (see [66]) be studied before modelling in the TAOM4E tool is attempted so that the tool and its limitations can be fully understood. The following observations were made during the use of the tool:

- A goal has to be seen from the perspective of what that particular stakeholder wants to achieve or do. Simple and straightforward goals such as “Acquire Parts from Supplier” will result in a better model and less remodelling than vague goals such as “Requires an easy way to submit orders”.
- The restriction to the methodology forcing decompositions to one type of object is very important. The use of dummy goals and plans to achieve this is used constantly and should always be kept in mind when decomposing goals or plans. The TAOM4E tool does not enforce the restrictions at design time, so care should be taken to continually enforce the restrictions during design.
- When creating dependencies between stakeholders, the restricted model only allows goals to depend on other goals (the normal Tropos model allows dependencies on other objects such as plans or resources). To simplify this process, position the goals the stakeholder cannot achieve on its own to one side of the stakeholder diagram.
- A “why” link used in the design means that instead of completely delegating a goal to another agent (the goal is removed from one agent and transferred to another) an AND decomposition is created between two goals; each goal being situated in a different agent. The “why” link works in the same way as a normal AND decomposition of a goal within an agent, the only difference being that the decomposition spans over two agents instead of one.
- To permanently remove an object from the model, right-click the object and click “Dispose from model”. Deleting an object removes it from the diagram, but not from the underlying model. If you delete an object and you want it back on a diagram, it can be found in the project outline and dragged onto the diagram.
- The current TAOM4E modelling tool does not allow for the “produce” relationship between a plan and a resource. It does allow for the “use” relationship between a plan and resource. This one-sided implementation can make the model harder to understand and it may be prudent to not use any resources in the model at all.

- Save images of diagrams during the design process. As all the diagrams share the same underlying model, changes to one diagram might produce unintended visual changes to existing diagrams and the diagram is essentially lost.
- The early and late requirements diagrams are of the same type (“Mixed Diagram”). The designer can create as many of these diagrams as he wants. It is possible to create a separate diagram for each actor and for actors to appear within the same model on different diagrams, but it was found that by doing this it becomes difficult to get an overview of the complete design easily and spot relationships. Also, the addition of objects to diagrams is not automatically reflected in the other diagrams within the model. This makes it hard to keep the diagrams synchronised.
- The disadvantage of using only one diagram is that the design will become cluttered if not managed correctly. A large design will also mean that some scrolling is required to navigate the diagram since it is too big to be displayed as a whole. Ultimately, the single diagram design was the better overall option in the larger model in this project.
- After the early requirements phase is complete it is time to start with the late requirements. To do this, add a normal actor representing the MAS to the early requirements diagram and delegate goals to the MAS. After the late requirements modelling is finished, right click the MAS actor and click “New Architectural Design Diagram”. The MAS actor will turn yellow and a new architectural design diagram will be created. Be sure of all the goals that have to be delegated to the MAS before continuing work on the architectural design diagram. In the architectural design diagram the goals assigned to the MAS will be visible. Add actors to this diagram that represent the software agents. Delegate the MAS goals to the various agents that have been identified. After the goals have been assigned to the agents, the same design process as in the early and late requirements diagram is followed within and between the agents.
- Later in architectural design it became impossible to create an image of the diagram as the tool would produce an error message (“Out of Java heap space memory”). The assumption is that the model became too large for the image creation portion of the TAOM4E tool to handle. After increasing the Java runtime’s memory allocation, the error changed from the out of memory error to a non-descript message. In the end the error could not be resolved.
- To generate the agent JADEX Agent Definition Files (ADF) files and Java classes, right click the MAS actor on the architectural design diagram. Select the “Agent Generator”. The files will be generated in a folder named “KLAagents” in the root of the Eclipse workspace. It is advisable to standardise on a naming convention for objects within the model so that the generated code can be more easily understood by the programmer. It is easy to confuse plans, goals and metagoals while customising the generated code.

The naming convention adopted in the design of this model is as follows:

- A goal’s name starts with “GOAL”.

- A plan's name starts with "PLAN".
- A dummy goal's name starts with "DUMGOAL" meaning "Dummy Goal".
- A goal delegated completely from another agent's name starts with "DGOAL" meaning "Delegated Goal".
- A goal delegated from the MAS agent's name starts with "MDGOAL" meaning "MAS Delegated Goal".
- A goal created in a cross-agent "why" decomposition situated in the "dependee" agent starts with "EGOAL" meaning "External Goal".
- A goal that is part of an OR decomposition's name starts with "OGOAL" meaning "OR Goal".

It is highly recommended that a versioning tool such as SVN be used to keep track of revisions of the project. A SVN plug-in is available for the Eclipse IDE. The reasons for keeping versions of the design are:

- It is good practice to use a versioning system in any design process.
- Due to the nature of the TAOM4E tool and TROPOS methodology, the designer might have to go back to an earlier version of the design after it becomes apparent that an incorrect design path has been followed.
- On rare occasions the TAOM4E tool exhibits some buggy behaviour. This may necessitate a return to a previous version of the design. Being able to easily go back to a previous version of the design ensures that the minimum amount of work is lost if such an unrecoverable error should occur.
- If really necessary, the designer can temporarily go back to a previous version of the design in order to retrieve images of the diagrams.

4.3.1.3 Early Requirements Modelling Using the TAOM4E Tool

During the early requirements phase, a goal diagram was produced. The diagram can be seen in Figure 4.12. This diagram was found to be too complex. It focussed too much on the goals and plans of stakeholders outside the MAS, adding no real value to the model. It did, however, provide valuable experience in the use of the TAOM4E tool and how to work within the restrictions. Another diagram was produced with the same number of stakeholders, but with greatly reduced complexity. On this attempt the multi-diagram approach was tested, but found to be unsatisfactory. The third and final attempt was simpler than the first, but consisted of only one diagram.

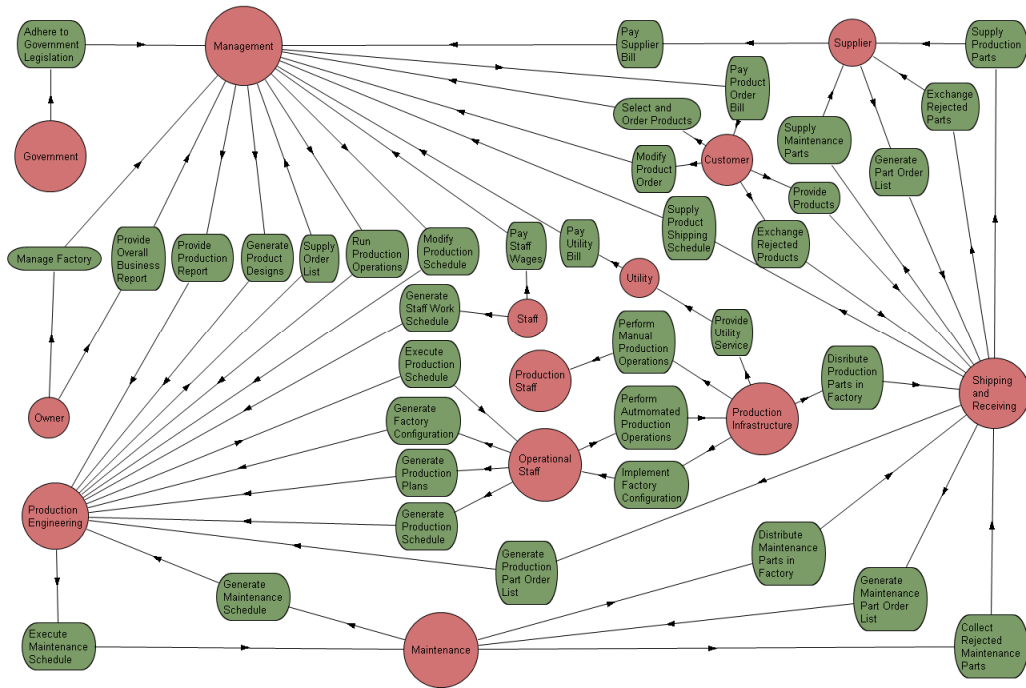


Figure 4.12: TROPOS ER Diagram

4.3.1.4 Late Requirements Modelling Using the TAOM4E Tool

The Late Requirements diagram can be seen in Figure 4.13.

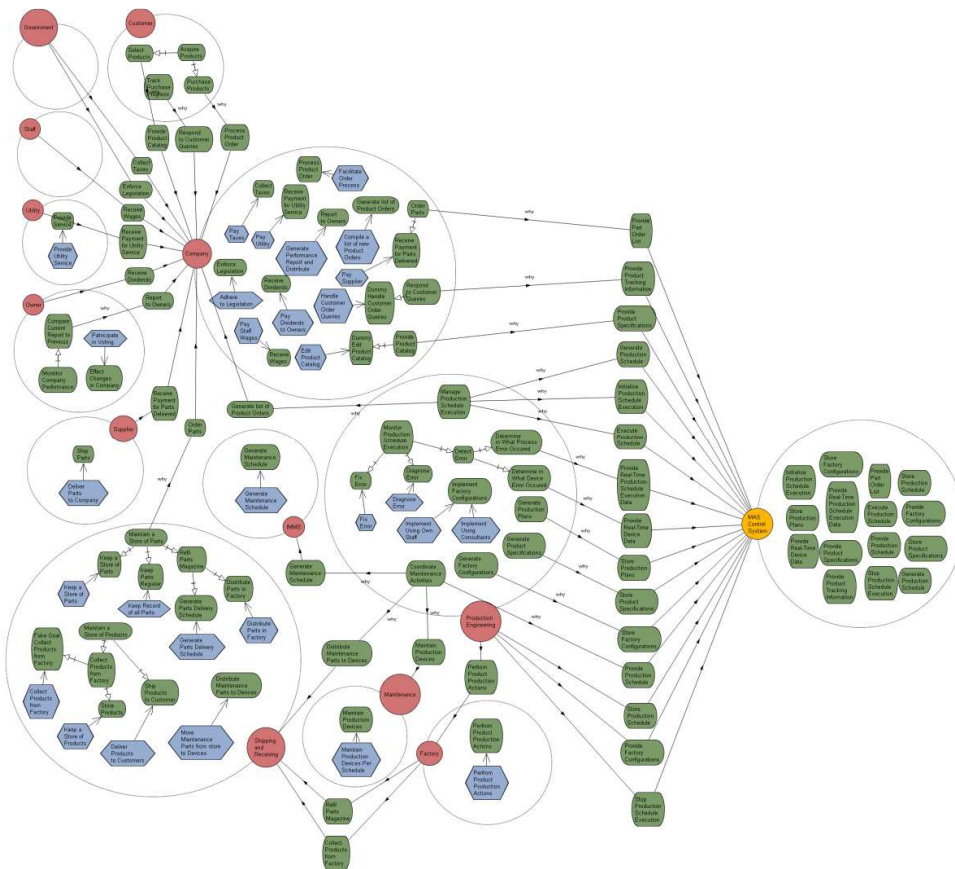


Figure 4.13: Late Requirements Diagram

At this point in the modelling process, the diagrams are essentially too large to display on one page. In this diagram, the System to be is introduced together with assigned goals from within stakeholders in the environment. These goals are to be assigned to agents in the next step of the design.

4.3.1.5 Architectural Modelling Using the TAOM4E [64] Tool

The Architectural Design diagram can be seen in Figure 4.14. In this diagram, agents are introduced and the goals assigned to the System, to be are assigned to the agents. New goals and goal dependencies are discovered and modelled. Unfortunately, the diagram is too large to display on one page in any detail.

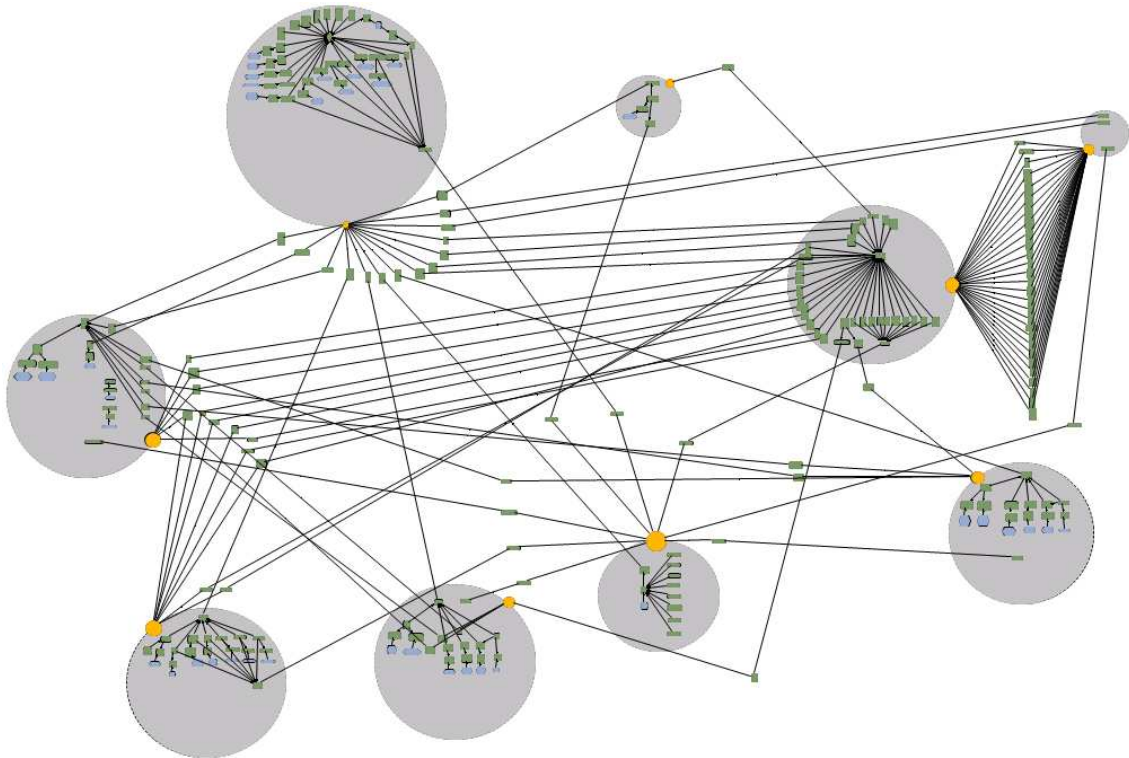


Figure 4.14: TROPOS Architectural Design Diagram

4.4 System Database and Ontology Design

4.4.1 Database Introduction

The system database is a very important component of the project. The database not only stores system data, it models the physical production environment and processes within it. Applicable concepts in a proposed equipment ontology found in [144] were built into the database.

The database went through many design and re-design cycles. Due to the re-configurable nature of the production system to be controlled, the database is designed to be very flexible. Achieving flexibility within the database results in more tables and more complex relationships between the tables. The database schema consists of 67 tables.

4.4.1.1 Security Model

Security is modelled according to the Restricted Hierarchical RBAC model. Users are assigned to User Groups. User groups are assigned Roles. Permissions are assigned to Roles. A Permission is linked to a Securable Object, a context Securable Object and an Operation combination.

An Operation is a generic verb such as “READ”, WRITE”, “EXECUTE” that gives a more fine-grained representation of what the Permission entails. The Securable object is the object in the Domain to which the Permission applies. The context Securable Object is a way to further refine the Permissions to eliminate ambiguity between similar Permissions.

A System audit Log is constructed by combining the User, Permission exercised and date. The system audit log allows the tracing of all User Permission consumption within the system.

The System table contains a reference to the current Administrator Super User of the system. This table also contains other system variables such as a system name and description.

4.4.1.2 Production Model

Various aspects of the Production Line are modelled in the database. The following list explains these aspects:

- **Device**

A Device is an abstract concept. It can represent a physical Device on the production floor, or any grouping concept such as a building, room or assembly cell. A Device Tree represents Devices in a hierarchical configuration. One Device may contain other Devices. The physical XYZ coordinates of each Device relative to its parent Device is stored in the database. The physical location of each device can therefore be computed. This also allows for an automatic computerised generation of a graphical representation of the Production Line.

A Layout represents a particular physical configuration of the production floor as stored in a Device Tree. This means different Device Trees can be stored in the database at the same time.

- **Port**

Although the Device Tree stores the physical Layout of the production floor, it does not represent how Devices are connected to one another. Devices are connected to each other by Configured Ports. A Port is an abstract concept that represents a physical input or output of material that is possible on a Device (as proposed in [144]). Ports have constraints that determine to which other Ports they can be connected. Devices can have more than one Port, meaning a Device can be connected to more than one other Device. Each Port has an XYZ coordinate that is relative to the XYZ position of the Device it is placed on. This also allows a more detailed auto generation of graphical representations of the production line.

- **Device Communication**

A Device Interface is a data access point on a Device. A Device Interface is an abstract concept with two current database implementations: User and OPC. This means that currently two types of Devices are supported in the system, devices that communicate through OPC and Devices that are other Agents. The OPC implementation directs communication for the Device Interface through the OPC protocol (which is also modelled in the database).

The User implementation directs communication to an Agent (the actual implementation of this is still to be determined). Production execution on the lowest level is composed of Configured Performatives. Performatives are descriptive commands that can be executed on a Device Interface of a Device. A Configured Performative is a combination of a Performative and actual data to send to the Device.

- **Production**

On the highest level is the Production Plan. A Production Plan describes how a Product is to be produced. A Production Step is a combination of a Configured Performative and a Device Interface. The Production Tree is a hierarchy of Production Steps. All Child steps of a particular Parent step (siblings) are to be executed concurrently by the system. Each Production Step can have many Conditions. A Configured Condition is the state (data value and quality of data) of a Particular Device Interface before or after the execution of the Production Step. If the Condition is specified as a “before” Condition, the Condition must be true before the Step can execute. If it is specified as an “after” Condition, the Condition must become true after the Step has executed in order to verify if the step was successful. Any number of Conditions can be linked together using Logical Operators such as “OR/NOT/ELSE” enable the fine-grained specification of Configured Conditions. Configured Conditions are stored in a hierarchical manner in the database so that Configured Condition sequences of any depth can be constructed.

- **Product**

A Product is modelled as a hierarchical collection of Assemblies. An Assembly can either be a single Assembly, or a collection of other Assemblies. Some Assemblies consist only of a single Part. These Assemblies are used as the first building blocks of more complex Assemblies. An Assembly can be connected to another Assembly. When a Super Assembly is created, the Assembly (Part) in both connecting Assemblies that are physically connected are specified. In this way it is possible to represent the makeup of a product, but also how the different sub-Assemblies of a Product is connected.

- **Order**

An order is a Production Plan combined with the quantity to produce, the Priority of the Order, its Status, its date of submission and date of completion.

4.4.1.3 Product Tracking Model

Parts in the system are affixed with one or more Tracking Identifiers with their accompanying data (RFID string, barcode string). Many different Tracking Identifiers can be registered in the database. Certain factory components in the Production line are specified as Tracking Components. These components can generate tracking data for one or more Tracking Identifier types. Tracking data is stored in a tracking table.

4.4.2 System Ontology

The ontology for the system mirrors the system database in many ways since database and ontology modelling both represents the domain it is to be used in. Just as the database design process, the ontology was also designed and implemented over a number of iterations. The database was a major source of knowledge and influenced the ontology design the most.

4.4.2.1 Ontology Design

The ontology is based on the Jade basic ontology. All other classes in the ontology inherit from these base classes. The base classes are made up of the following interfaces (shown in their inheritance hierarchy):

Concept

- AgentAction.

Predicate

From these base interfaces, the following base interfaces for the project ontology are derived:

Concept

- AgentAction.
 - ProductionDomainAction.
 - SecurityAction.
 - ProductionExecutionAction.
 - DeviceAction.
 - UserInterfaceAction.
 - DatabaseAction.
 - PlannerAction.
 - ProductAction.
- ProductionDomainConcept.
 - SecurityConcept.
 - ProductionExecutionConcept.
 - DeviceConcept.
 - PlannerConcept.
 - DataConcept.
 - ProductConcept.

Predicate

- ResultOfProductionAction.

As can be seen in the presented hierarchy, the actions and concepts mostly mirror one another. Actions represent actions that Agents can request other Agents to perform (the concept of a public function of a class in object oriented programming is a good analogy). Concepts are carriers of information that agents use to send data within the MAS. Predicates represent statements or propositions that agents can make and send to one another. The whole expanded ontology would be too large to represent logically in one diagram. A diagram of a subsection of the ontology is represented in Figure 4.15.

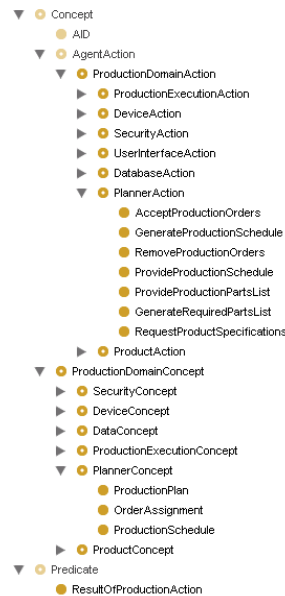


Figure 4.15: Production Ontology

4.4.2.2 ProductionDomainAction: Base Design

The ProductionDomainAction class serves as the base for all actions that can be performed in the MAS. It contains certain members or slots that are then common to all its children classes. The slots in this base class include a name and description. The other slots represent a security tuple. The security slots are:

- **ProductionDomainConceptOperationAppliesTo**
This slot represents the main concept in the domain that the operation in the action applies to.
- **ProductionDomainContextConcept**
This slot also represents a concept from the domain and is used to refine the security tuple further.
- **OperationToBePerformedByUser**
This slot contains a reference to an Operation. Operations are verbs such as “Create”, “Modify”, “Remove”. An Operation combined with the two domain concepts represents a unique action within the system and is used as the base of permissions in the RBAC database.

4.4.2.3 ProductionDomainConcept: Base Design

The ProductionDomainConcept class serves as the base for all concepts that can be represented in the MAS. It contains certain members or slots that are then common to all its children classes. The slots in this base class include a Name and Description. The other slot represents a Secured ID. The secured ID is a numeric ID that is matched to the Secured Object table in the database and forms part of a RBAC permission object.

4.4.2.4 ResultOfProductionAction: Base Design

The ResultOfProductionAction class is used to return a status (Failed, Succeeded), a result message, a Boolean value indicating if the result should be treated as an exception and the actual results of the action, represented as concepts in the domain. This object is sent back to the requesting agent that requested an action to be performed by the target agent.

4.5 Combination of Components into a Functioning MAS

4.5.1 Introduction

The implementation of the system from this stage followed the following steps:

- The generation, evaluation and modification of the JADEX agent definition files and Java classes.
- The generation, evaluation and modification of the Java ontology classes.
- Integration of all generated Java classes into the RGEMS Java class library.
- The development of the web service endpoint that will service operator requests.
- Development of a web service request management system that will run within the Web Service GUI Proxy Agent to serve as the public interface to the system.
- The development of the system configuration file and system boot-up procedure
- The development of the database access classes.
- The development of Java code (system logic) within the various agents.
- The integration and testing of all the various components.

4.5.2 Pre-Implementation Issues

Various unexpected issues with the generated JADEX ADF files, TROPOS Java classes and Java ontology classes were experienced during system implementation. These issues necessitated the creation of a small test MAS to explore possible solution on a small scale. The test MAS was used to resolve the problems encountered with the aim of transferring the knowledge gained during this test phase into the final MAS. The following problems were encountered:

- The generated Java ontology classes were not properly sorted into a logical collection of Java packages. This made it very difficult to find a particular concept in the ontology.
- The generated JADEX ADF files and TROPOS helper Java classes did not offer any support for the use of a formal ontology.

- During the development of the web service within the Web Service GUI Proxy Agent, it was discovered that the structure of the generated ontology classes were incompatible with the web service mechanism (JAX-WS) being used. The problem was that the generated ontology made use of some Java interface implementations instead of just class inheritance. This resulted in the web service mechanism being unable to publish some concepts in the ontology.

4.5.3 Test MAS

The test MAS consisted of 5 agents. The relationships between the agents were chosen to mimic those that can be found in the main MAS design. It has no real purpose other than to elicit knowledge and experience that can be transferred into the main MAS design as well as to solve the pre-implementation issues. The Test MAS was used to accomplish the following tasks:

- Successfully refactor the generated Java ontology classes into logical packages.
- Successfully alter the inheritance hierarchy in the generated Java ontology classes to enable the use of the classes in the web service.
- Integrate the Production Ontology into the generated JADEX ADF files and helper Java classes.
- Successfully host a dynamically generated web service from within the Web Service Agent given the internal execution model of JADE agents.
- Develop a method within the Web Service Agent to mediate between incoming and outgoing web service requests and ACL messages within the MAS.
- Develop database access classes.
- Develop the appropriate JADEX ACL message filters to apply to agents so that ACL messages in agents are routed to the correct plan.
- Find the correct way to insert and extract populated ontology objects into and from the ACL messages sent between agents.

4.5.3.1 Changes to the auto-generated Java Ontology classes

The Java ontology classes were refactored into appropriate Java packages ready for transfer into the main RGEMS Java repository. The web service incompatibility with the ontology classes issue was resolved by manually editing the Java ontology classes to introduce class inheritance where applicable.

4.5.3.2 Changes to the auto-generated TROPOS model

Unfortunately, the generated TROPOS helper Java classes could not be modified to support the use of the Production Ontology. The generated TROPOS ADF files are still useable, albeit with some changes. The generated TROPOS model is supplied with various classes that attempt to integrate the TROPOS methodology into JADEX in an automated fashion. These classes work when no ontology is used in the system and the level of control needed over the system is not that complex. Although it is not shown on the TROPOS diagram, each goal inside an agent has a plan associated with it.

These plans facilitate execution of the goal hierarchy within each agent, as well as sending and receiving messages between agents. This architectural design is kept in the modified model, but the automated way of invoking the correct plan for each goal is discarded. Although it is probably possible to re-write the TROPOS Java helper classes to provide support for ontologies and finer-grained control over messages, the decision was made to rather implement the system in a more straightforward way in order to save time and simplify the design. The implementation of the MAS was done according to the following method:

- The generated TROPOS ADF files were manually edited to incorporate all the knowledge gained in the test MAS ADF files.
- For each goal in the model, a Java helper class extending the JADEX Plan class was created. These classes are responsible for calling the correct sub-goals (in the correct order), passing along the correct result from AND decompositions and sending and receiving messages between agents (WHY decompositions).
- Some goals in agents (like the main web service message relay goal) are of a “service” type. This type of goal is not supported in the TAOM4E designer and is subsequently not included in the generated code and will have to be completely designed from the ground up.

4.5.3.3 Test MAS: Agent Society

As can be seen in Figure 4.16, AgentF serves as the Web Service Agent. This Agent receives web service requests from web service clients and then return data from the MAS back to the correct client in web service form.

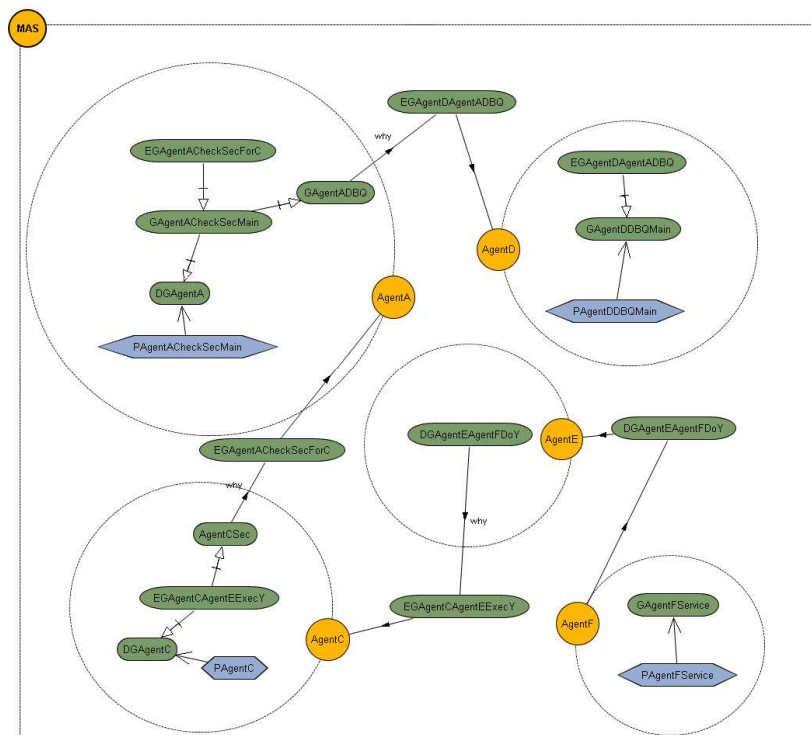


Figure 4.16: Test MAS Architectural Design Diagram

A basic web service client was written in C# using Visual Studio 2008 to test the web service agent's functionality. The public web methods available to clients are sourced from the GUI actions available in the Production Ontology. A request to login is initiated in the web service client application and the appropriate public web method in the web service agent is invoked. The web service running within AgentF (on a dedicated thread, not the default agent message thread) handles the each request in a new thread from a threadpool and generates the appropriate action object in the ontology, populating it with values received from the web service invocation. The web service within AgentF then sends itself (AgentF) the populated action request ACL message and puts the newly spawned web service request thread to sleep.

The sent ACL message enters AgentF's main message loop where the default agent message thread has been busy waiting for messages to arrive. The ACL message is evaluated and passed on to AgentE. A record of the conversation is made in AgentF so that when a result is returned from the MAS the correct web service thread is woken and sent the data. After AgentF sends the message on, it returns to a listening state for other messages. AgentF's main message loop accepts populated AgentAction ontology objects from its own web service endpoint and Predicate objects from agents in the MAS. It thus serves as a message "pump". Agents all have a built-in ACL message buffer that buffers incoming messages until the particular Agent can handle them.

In the screenshot seen in Figure 4.17, it can be seen that the first message Agent F sends itself has the status of "NOT_UNDERSTOOD". This error has been corrected in the actual MAS through the refinement of the message filters.

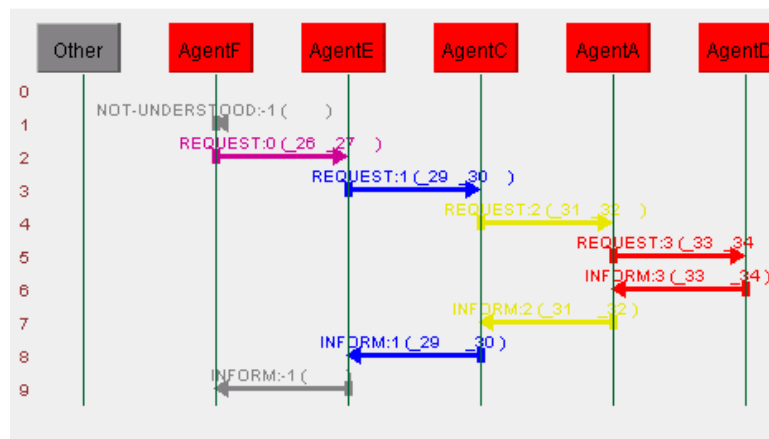


Figure 4.17: Test MAS ACL Message Passing

In figure Figure 4.18 an ACL message's content is examined (using the JADE sniffer agent). As can be seen in the content window, the object being passed is a JADE AgentAction object with content "RequestLoginUser", an AgentAction concept from the Production Ontology.

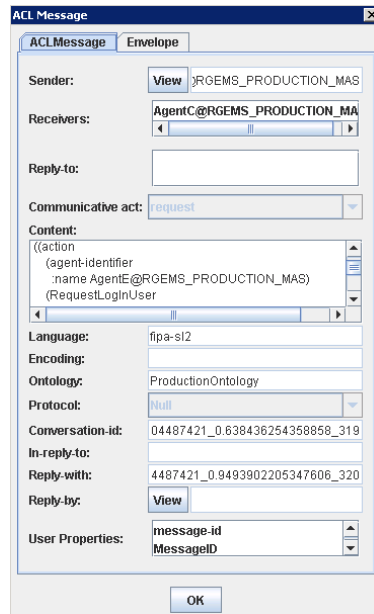


Figure 4.18: Test MAS ACL Message Example

4.5.4 System Implementation

The test MAS elicited a lot of knowledge about how to go about using the various components needed to further implement the system. It also provided a safe environment to test out various technologies. Much testing, debugging and configuration were needed to get the various components working together.

4.5.4.1 Database Access Classes

A reference implementation of the Java Persistence API (JPA) by Oracle Toplink Essentials is used as the ORM (Object Relational Mapping) API in the project. An ORM API allows object oriented access of database tables. Oracle Toplink Essentials is a free, community edition of the full Toplink product.

The Netbeans IDE allows for the graphical setup of the JPA API as well as the initial generation of entity classes (classes that represent the tables in the database) and controller classes (classes that contain methods to access the database for each entity class).

Database access is supposed to be managed through the Database Agent. A full implementation of the Database Agent was not done as it would take too much time to complete and not offer any real advantage at this time. The agents use the JPA generated classes to access the database directly.

4.5.4.2 MAS Start-up Procedure

The MAS start-up procedure consists of the following action performed in order and is specified in the "CProductionMAS" class:

- The MAS GUI main Java class is executed.
- The start-up command is given by the user by selecting the option from a menu. The bootstrap thread is started and it takes over the start-up process.
- The XML MAS configuration file is located and loaded.

- The bootstrap thread uses the parameters loaded from the configuration file to connect to various tables in the database (using the JPA API).
- The JADE runtime is retrieved and the main RGEMS container created within it.
- The Agents specified in the Boot table in the database are created within the main container by creating an agent controller for each agent. The Agent class for each agent is set to “jadex.adapter.jade.JadeAgentAdapter”. This specifies to JADE that the agents are JADEX agents.
- The agent controllers (agents) are started.

4.5.4.3 JADEX Agent Definition Files (ADF)

The ADF file for each agent was generated by the TAOM4E tool. Unfortunately, the files had to be altered significantly to allow for the functionality required. An ADF for each agent type is created – thus if two or more Device agents are started, they all use the same ADF file. The ADF file is an XML and contains the following sections:

- **Imports**

The Imports section imports all the source files that will be referenced in the ADF file. It works in the same way that one would normally import source files into a Java class. The code block below shows a piece of the Imports section in the Device Agent. The classes and packages being imported are Java classes and packages:

```
<imports>
  <import>jade.content.onto.basic.*</import>
  <import>jade.content.ContentElement</import>
  <import>jadex.adapter.fipa.*</import>
  <import>jadex.adapter.jade.*</import>
  <import>java.util.logging.*</import>
  <import>java.util.*</import>
  <import>jadex.util.*</import>
  <import>jadex.runtime.*</import>
  <import>jade.content.lang.sl.*</import>
  <import>jade.content.lang.xml.*</import>
  <import>jade.domain.JADEAgentManagement.JADEManagementOntology</import>
  <import>jade.domain.FIPAAgentManagement.FIPAMangementOntology</import>
</imports>
```

- **Capabilities**

Capabilities are predefined Goal and Plan packages that are imported into one or more agents. They can be thought of as adding a reference to a DLL or Library to a programming project. The code block below adds the Directory Facilitator capability collection to the Device agent. The Capability may contain many Goals and Plans:

```
<capabilities>
  <capability name="dfcap" file="jadex.planlib.DF"/>
</capabilities>
```

- **Beliefs**

Beliefs can be thought of as private variables (of any type) declared within the agent. It is what the Agent “knows”. Some beliefs of the Device agent is shown in the code block below:

```
<beliefs>
  <belief name="data_connections" class="CDeviceAgentListOfDataObjects">
    <fact>new CDeviceAgentListOfDataObjects()</fact>
  </belief>
  <belief name="opc_subscription_listener" class="COPCIOMediator">
    <fact>null</fact>
  </belief>
  <belief name="agent_state" class="Integer">
    <fact>new Integer(0)</fact>
  </belief>
  <belief name="production_executor" class="AProductionExecutor">
    <fact>null</fact>
  </belief>
</beliefs>
```

- **Goals**

Goals are containers for Plans. There are different types of Goals. Goals are actualised through Plans. Plans do not have to be linked to Goals and can stand alone. The code block below lists the Goals of the Device Agent. It imports two Goals from the referenced “dfcap” capability. The “Initialise_Agent” Goal is used to run the Initialise agent Plan at agent start-up. In the Plan section, the Plan named “realPlan_Intialise_Agent” is set to be triggered when this Goal is adopted by the agent:

```
<goals>
  <maintaingoalref name="df_keep_registered">
    <concrete ref="dfcap.df_keep_registered"/>
  </maintaingoalref>
  <achievegoalref name="df_search">
    <concrete ref="dfcap.df_search"/>
  </achievegoalref>
  <!--First goal triggered - initialises agent-->
  <achievegoal name="Intialise_Agent" exclude="when_failed">
    <unique/>
  </achievegoal>
</goals>
```

- **Plans**

Plans are defined in normal Java classes that extend the JADEX “Plan” class. This is where the Agent executes actual Java code to achieve some desired outcome. A part of the declared plans for the Device agent is shown in the code block below:

```
<plans>
  <!--Handles agent initialise with added IO Item service-->
  <plan name="realPlan_Initialise_Agent_Add_Services">
    <body>new realPlan_Initialise_Agent_Add_Services()</body>
    <trigger>
      <internalevent ref="realPlan_Initialise_Agent_Add_Services" />
    </trigger>
  </plan>
```

```

<!--Handles agent initialise-->
<plan name="realPlan_Initialise_Agent">
  <body>new realPlan_Initialise_Agent()</body>
  <trigger>
    <goal ref="Initialise_Agent"/>
  </trigger>
</plan>
<plan name="realPlan_StartProduction">
  <body>new realPlan_StartProduction()</body>
  <trigger>
    <messageevent ref="request_StartProduction_MSG"/>
  </trigger>
</plan>
</plans>

```

▪ **Events**

There are different types of events. The most common event defined within the ADF is a Message Event. A Message Event is raised when a specific message is sent to the Agent. The Message Event is declared with various parameters that together form a filter. If the incoming message matches the filter, one or more Goals or Plans can be executed if so configured and each linked Goal or Plan is sent the incoming message.

A Message Event has to be declared for each message the agent wants to send out of the agent. Each Agent has a few default Message Events. These are the standard FIPA messages such as “request”, “inform”, “agree”, “not understood”, “refuse” and “failure”.

The Message Event in the code block below is taken from the Device agent. It is responsible for matching the command to start the production execution process. The type of message it will allow is a “request”; it is encoded in the “FIPA-SL2” language; the message uses the “ProductionOntology” ontology; the content of the message will be of the “Action” type and the specific type of the message will be an instance of “DefaultExecuteProcessTreeAndConditions”.

The Internal Event shown in the code block below is an example of an Event that an agent can raise inside itself. Plans and Goals can be triggered if they are configured to listen for this event.

```

<events>
  <messageevent name="request_StartProduction_MSG" direction="receive" type="fipa">
    <parameter name="performative" class="String" direction="fixed">
      <value>SFipa.REQUEST</value>
    </parameter>
    <parameter name="language" class="String" direction="fixed">
      <value>SFipa.FIPA_SL2</value>
    </parameter>
    <parameter name="ontology" class="String" direction="fixed">
      <value>ProductionOntologyOntology. ONTOLOGY_NAME</value>
    </parameter>
    <parameter name="content-class" class="Class" direction="fixed">
      <value>jade.content.onto.basic.Action.class</value>
    </parameter>
    <match>((Action)$content).getAction() instanceof DefaultExecuteProcessTreeAndConditions
  </match>

```

```

</messageevent>
<internalevent name="realPlan_Initialise_Agent_Add_Services">
</internalevent>
</events>

```

▪ Properties

Properties are almost like Beliefs in that they are variables that have been declared within the Agent. Properties are like public variables defined within a class in object oriented software engineering. The following Properties are defined in the Device agent. The Properties defined in the agents mainly have to do with adding Content Codecs to the JADEX agent so that it can process the built-in JADE ontologies and the ProductionOntology:

```

<properties>
<!-- Only log outputs >= level are printed. -->
<property name="logging.level">Level.SEVERE</property>
<!-- The default parent handler prints out log messages on the console. -->
<property name="logging.useParentHandlers">true</property>
<property name="contentcodec.fipa-management-s12">
  new JadeContentCodec(new SLCodec(2), FIPAMangementOntology.getInstance())
</property>
<property name="contentcodec.jade-management-s12">
  new JadeContentCodec(new SLCodec(2), JADEManagementOntology.getInstance())
</property>
<property name="contentcodec.ProductionOntology-s12">
  new JadeContentCodec(new SLCodec(2), ProductionOntologyOntology.getInstance())
</property>
</properties>

```

▪ Configurations

An agent can have various Configurations. When the agent is started, a configuration name is passed as a parameter. The default configuration is called "default". A Configuration is a way to specify a different start-up procedure for the Agent. The Configuration section of the Device agent is shown in the code block below. As can be seen, the first Goal to be run when the agent starts is the "Initialise_Agent" Goal. The Goal that will run just before the agent is destroyed is called "df_deregister". This goal will remove the agent registration from the Yellow Pages (discussed in the next section):

```

<configurations>
  <configuration name="default">
    <goals>
      <initialgoal ref="Intialise_Agent"/>
      <endgoal ref="df_deregister"/>
    </goals>
  </configuration>
</configurations>

```

4.5.4.4 Agent Auto-Registration in Yellow Pages

Agents are not automatically registered in the Yellow Pages when they join the MAS. The Yellow Pages are maintained by the Directory Facilitator (DF) agent and can be queried by other agents. The Yellow Pages is a directory of agents, their information and the services they offer. To enable the agents to register themselves, each agent's first initial Goal is to register itself at the DF agent. The Plan used to actualise the registration is common for each agent type in the MAS: `realPlan_Initialise_Agent`. In the Plan, a pre-defined Goal "df_keep_registered" is executed with agent specific parameters. The "df_keep_registered" Goal is of a "maintain" type. This means it will keep executing to maintain the registered state of the agent.

Device Agents use a derived version of the `realPlan_Initialise_Agent` named `realPlan_Initialise_Agent_Add_Services`. After it has connected to an underlying Device data source, it drops the initial "df_keep_registered" goal, thus deregistering itself from the Yellow Pages. It then executes the derived Plan to register itself again, only this time adding the "IO_ITEM_SERVICE" to its description, along with the names of all the Device IO Items it is connected to. The derived plan uses the same "df_keep_registered" goal to maintain the agent registration. This derived Plan overrides the empty "addAdditionalServices" method in the original Plan class (Template Method Design Pattern). This method adds the "IO_ITEM_SERVICE" to a Device agent. Other agents can now use the DF agent to search for and subscribe to Device IO's in Device Agents.

4.5.4.5 Web Service Request Management

The Web Service Agent is comprised of three main classes. The JADDEX Plan class, the web service "face" class and the "conversation" class. The Plan object starts the web service in a dedicated thread so that the main Plan thread is free to handle incoming and outgoing messages. The face object contains all the public web service functions that are to be published. These functions are published when the web service endpoint is started. The web service request management method can be better understood by describing its start-up procedure:

- After MAS boot up, the various agents run their default Goals. The default Goal for the Web Service Agent is to start running the web service endpoint.
- The Plan to actualise the Goal is started. In this plan, JAX-WS is used to publish the face object containing the public web methods in a dedicated thread. The main Plan thread is now put to sleep by waiting for incoming messages. The main Plan thread now serves as a message pump, running an "infinite" loop, until the web service is shut down.

Each time a web service request is received through JAX-WS, it is handled on a dedicated thread for that particular connection. JAX-WS has a pool of threads ready to handle new connections. An example of a web service Login request coming into the system is described below. An operator has to Login before being able to call any other web methods:

- The web service client calls the public Login web method over the network.

- The Login method in the face object starts executing. A conversation object is created. This object contains a unique Transaction ID as well as a thread synchronisation object. The conversation is added to the active conversation collection in the Plan object.
- A Login Request object from the ontology is created and populated with the data the Operator supplied. The Agent then sends this populated action object to itself and puts the thread handling the web service request to sleep by waiting on the synchronisation object in the conversation object.
- The message the Agent sent itself arrives at the Agent's message queue. The main Plan thread in the Plan object is woken and collects the message from the message queue. The Plan object checks the type of the message it has received. If the message is an Action object, it registers the message in an ongoing conversation collection and routes the message into the MAS to the correct Agent. If the message is of a Predicate type, it routes the message out of the MAS through the web service endpoint by matching the message it has received to an ongoing conversation in its collection. If it finds a match, the synchronisation object in the conversation object is used to wake the sleeping web service request thread. The message returned from the MAS is then available for the web service request thread to retrieve. The ongoing conversation is then deregistered.
- Before the Login result is returned to the Operator, a new session is registered in the Plan object if the Login attempt was successful. The Operator has to use the string session ID it receives upon successful login for all future requests to the system. If a valid session ID is not supplied during a call to a public web method, the system rejects the request.
- The Operator is now logged into the system. The initial conversation object is removed from the conversation collection in the Plan object and the web method terminates. The thread used to handle the request is returned to the thread pool, ready to service future requests. The only object still remaining referencing the operator is its Session object in the Plan object Session collection.

4.5.4.6 Pluggable Production Executor

Device agents contain a production executor object as one of its Beliefs. The production execution object has been written so that it can easily be switched out with another production execution implementation. The Device agent contains a reference to the abstract super-class of all the production executors: `AProductionExecutor`. This class has concrete methods and variables common to all production executors, and several abstract methods that need to be implemented in a child class that inherits from it. The abstract methods are explained below:

- **Execute:** This method is called by the Device agent to start the production execution process.
- **Halt:** This method is called by the Device agent to stop the production execution process.

- **Load Production Schedule:** This method is called by the Device agent when the production execution process is initialised.
- **IO Data Changed, Debug Data Changed and State Data Changed:** These methods deliver updated IO items to the production executor object that the Device agent is subscribed to.
- **Write IO Data, Write Debug Data and Write State Data:** These methods implemented in the child class rather than the abstract super-class to allow for more flexibility when writing to IO Interfaces.

4.5.4.7 Device IO Objects

As mentioned in section 4.4.1.2, the system currently supports two types of Devices: OPC and User. This means that the underlying data source Device agents connect to can be of more than one type. To allow for all the different data sources to be used interchangeably, several polymorphic object-oriented designs have to be implemented. Several classes were designed to achieve the desired behaviour. The classes are explained below:

- **ADeviceData**

This is the abstract base class for Device IO wrapper classes. The child wrapper classes wrap the underlying connection objects that physically connect to Devices (Adapter Design Pattern). As there are two types of underlying data sources (OPC and User) there are two wrapper classes: cDeviceDataOPC and cDeviceDataUser. The abstract ADeviceData class has several abstract methods that the child wrapper classes have to implement. In these methods, the child wrapper classes mediate between the underlying data connection object and the public methods made available in the ADeviceData class.

- **IDeviceDataObserver**

An ADeviceData object can have observers. Observers are objects that get notified when certain changes occur in the ADeviceData object. An observer of an ADeviceData object has to implement the IDeviceDataObserver interface and add itself to the list of observers if it wants to be informed of changes in the ADeviceData object (Observer Design Pattern). Two objects implement the IDeviceDataObserver interface: AProductionExecutor and CExternalDeviceDataObserver. The AProductionExecutor is an observer because it needs data from the Production Floor devices as it changes to steer its process. The CExternalDeviceDataObserver directs updates in ADeviceData objects to another subscribing agent within the MAS.

- **AIOInterfaceMediator**

Connections to the underlying device are not made independently by each ADeviceData object. This would be very inefficient. In both the OPC and User data type cases, connections to the underlying data sources are pooled by a mediator object.

The mediator object manages the connections to the underlying data source and updates its ADeviceData objects with data it receives. It also manages the writing of data from an ADeviceData object to the underlying data source. As with the other objects described so far, two implementations of AIOInterfaceMediator exists in the system: COPCIOMediator and CUserIOMediator.

Chapter 5: Results

5.1 Test Scenario

5.1.1 System Start-up

When the RGEMS MAS Java application is started, a simple GUI is loaded. The user selects the “Start Platform” option from the “File Menu”. In Figure 5.1, the output stream of the various MAS components is rerouted to a text window in the GUI. In the screenshot, a partial view of the MAS starting up can be seen.

```
USER >> INITIATE SYSTEM START
SYSTEM >> STARTING BOOTSTRAP_THREAD
BOOTSTRAP >> NEW_THREAD: [Thread-5] RUNNING
BOOTSTRAP >> RETRIEVING SYSTEM BOOTSTRAP VARIABLES FROM CONFIG FILE: [production_mas_config.xml]
BOOTSTRAP >> USING DATABASE: [rgems_core @ 192.168.120.12]
BOOTSTRAP >> RETRIEVING ACTIVE SYSTEM PROFILE
10 Nov 2011 4:01:12 PM org.hibernate.annotations.common.Version <clinit>
INFO: Hibernate Commons Annotations 3.1.0.GA
[TopLink Info] 2011.11.10 04:01:17.419-ServerSession(2329159)-TopLink, version: Oracle TopLink Essentials - 2.0.1 (Build b09d-fcs (12/06/2007))
[TopLink Info] 2011.11.10 04:01:18.424-ServerSession(2329159)-file/C:/dev/main/rgems/build/olasses/rgems_core_login_successful
BOOTSTRAP >> RETRIEVING LIST OF ORPHANED SESSIONS
BOOTSTRAP >> RETRIEVING LIST OF AGENTS
BOOTSTRAP >> RETRIEVING LIST OF ORPHANED ORDERS
BOOTSTRAP >> STARTING RUNTIME DATA RESET
BOOTSTRAP >> DELETING ORPHANED SESSIONS
BOOTSTRAP >> ORPHANED SESSION: [16f391-c855-425f-8d90-b0621x179b6] DELETED
BOOTSTRAP >> ORPHANED SESSION: [d7f5e80-0728-4ee3-ac49-20c2ebd89850] DELETED
BOOTSTRAP >> ORPHANED SESSION: [5581825-423c-4266-bff6-c084e050780f] DELETED
BOOTSTRAP >> ORPHANED SESSION: [c239239-9a25-473e-b8f6-3a078d7a2e50] DELETED
BOOTSTRAP >> ORPHANED SESSION: [22c4891f-518b-4d2c-9a0f-c8b80017b7c7] DELETED
BOOTSTRAP >> ORPHANED SESSION: [7471c05d-3ed6-486e-42b8-dbe1a0e29e09] DELETED
BOOTSTRAP >> DELETING ORPHANED AGENTS
BOOTSTRAP >> DELETING ORPHANED ORDERS
BOOTSTRAP >> LOADING CONFIGURATION FOR SYSTEM: [NAME: RGEMS Test Platform] [DESCRIPTION: RGEMS MAS Test Platform] [VERSION: 0.1] [SITE: BHP20]
BOOTSTRAP >> RETRIEVING LIST OF BOOT-UP AGENTS AND STARTING SESSIONS
BOOTSTRAP >> AGENT: [Security] OF TYPE: [SECURITY.agent.xml] FOUND
BOOTSTRAP >> USER SESSION FOR AGENT: [Security] WITH USER: [system] CREATED
BOOTSTRAP >> AGENT: [Production Planner] OF TYPE: [PRODUCTION_PLANNER.agent.xml] FOUND
BOOTSTRAP >> USER SESSION FOR AGENT: [Production Planner] WITH USER: [system] CREATED
BOOTSTRAP >> AGENT: [Production Execution] OF TYPE: [PRODUCTION_EXECUTION.agent.xml] FOUND
BOOTSTRAP >> USER SESSION FOR AGENT: [Production Execution] WITH USER: [system] CREATED
BOOTSTRAP >> AGENT: [MBSERVICE] OF TYPE: [MBSERVICE_UI.agent.xml] FOUND
BOOTSTRAP >> USER SESSION FOR AGENT: [MBSERVICE] WITH USER: [system] CREATED
BOOTSTRAP >> AGENT: [Test Device] OF TYPE: [DEVICE.agent.xml] FOUND
BOOTSTRAP >> USER SESSION FOR AGENT: [Test Device] WITH USER: [system] CREATED
BOOTSTRAP >> AGENT: [Test Device Proxy Agent] OF TYPE: [AGENT_DEVICE_PROXY.agent.xml] FOUND
BOOTSTRAP >> USER SESSION FOR AGENT: [Test Device Proxy Agent] WITH USER: [system] CREATED
BOOTSTRAP >> RETRIEVING JADE RUNTIME
BOOTSTRAP >> STARTING MAIN JADE AGENT CONTAINER
10 Nov 2011 4:01:20 PM jade.core.Runtime beginContainer
INFO:
-----
This is JADE snapshot - revision 6357 of 2010/07/06 16:27:34
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
-----
Retrieving CommandDispatcher for platform RGEMS_PRODUCTION_MAS
10 Nov 2011 4:01:20 PM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jcp://192.168.120.63:1099
-----
10 Nov 2011 4:01:20 PM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
10 Nov 2011 4:01:20 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
10 Nov 2011 4:01:20 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
10 Nov 2011 4:01:20 PM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
10 Nov 2011 4:01:21 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
10 Nov 2011 4:01:21 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
10 Nov 2011 4:01:21 PM jade.core.messaging.MessagingService clearCachedSlice
INFO: Clearing cache
10 Nov 2011 4:01:21 PM jade.core.messaging.MessagingService clearCachedSlice
INFO: Clearing cache
10 Nov 2011 4:01:21 PM jade.core.AgentContainerImpl joinPlatform
INFO:
-----
Agent container RGEMS_MAIN_CONTAINER@192.168.120.63 is ready.
-----
BOOTSTRAP >> CREATING AGENT CONTROLLERS
BOOTSTRAP >> AGENT_CONTROLLER: [Security] CREATED
```

Figure 5.1: MAS Debug Output Window

After the MAS has booted up, the JADE Agent Management GUI can be seen (Figure 5.2) if it was configured to be shown in the MAS configuration file. In it, the agents running on the “RGEMS_PRODUCTION_MAS” are listed. Using this GUI, messages sent between agents can be intercepted and viewed. Agents can also be removed or added.

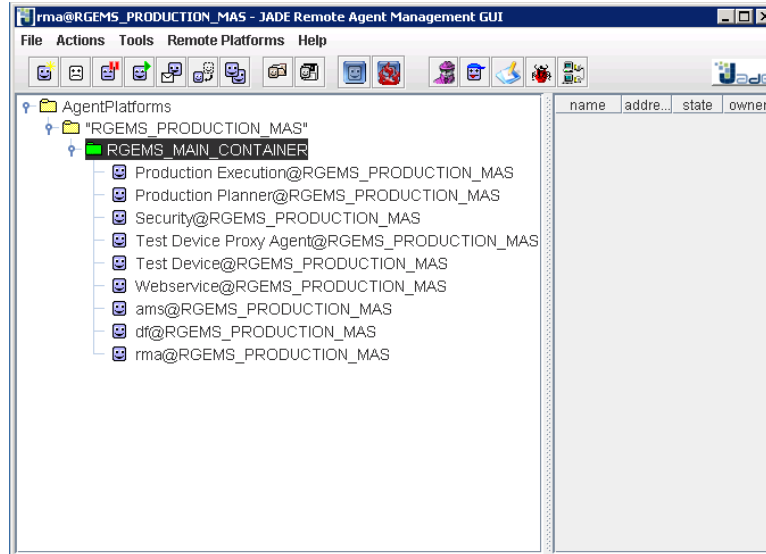


Figure 5.2: JADE Agent Management GUI

The JADEX Control Centre (Figure 5.3) can also be seen if it was so configured in the configuration file. Using this GUI, the JADEX specific BDI features of the agents in the MAS can be viewed.

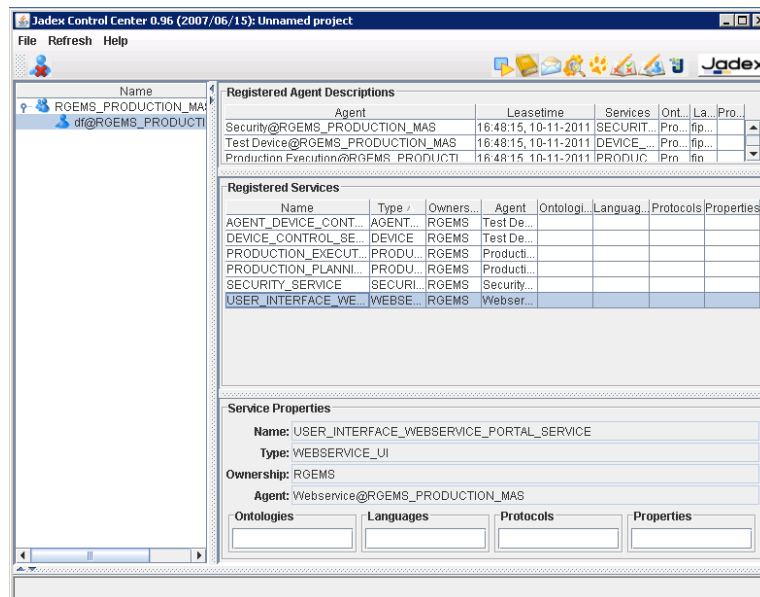


Figure 5.3: JADEX Control Centre

5.1.2 Operator Interaction

At this point, operators can connect to the MAS using the web service client application (or any other web service client written against the web service). Many IDE's can automatically interrogate the WSDL file of the web service endpoint when a client is being developed. During this interrogation, the IDE automatically creates the required data and connection classes to connect and use the web service. This makes creating a web service client relatively easy.

A screenshot of the client is shown in Figure 5.4. The operator types in a user name and password and clicks Login. The operator will then be authenticated by the Security Agent. If authentication succeeds, a unique session is created. A GUI User Interface Agent is created within the MAS that will handle all further interaction between the web service session and the MAS.

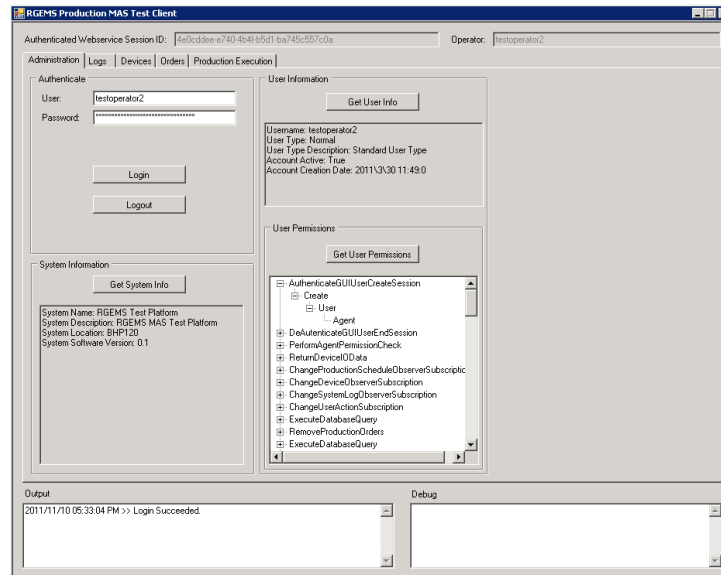


Figure 5.4: Web Service Demo Client Application Admin Screen

On this page, the operator can query the MAS (if the operators' User has the correct Permissions) on information such as "System Info", "User Info" and "User Permissions". In the current test scenario, only one Device is actually configured and used. As the database design is very flexible, the one device is a logical Device incorporating all the physical hardware in on the Production Floor. In Figure 5.5, the Device Interfaces for the "Simulation Test Device 1" can be seen. All Device Interfaces in this test scenario use OPC to communicate with their physical devices.

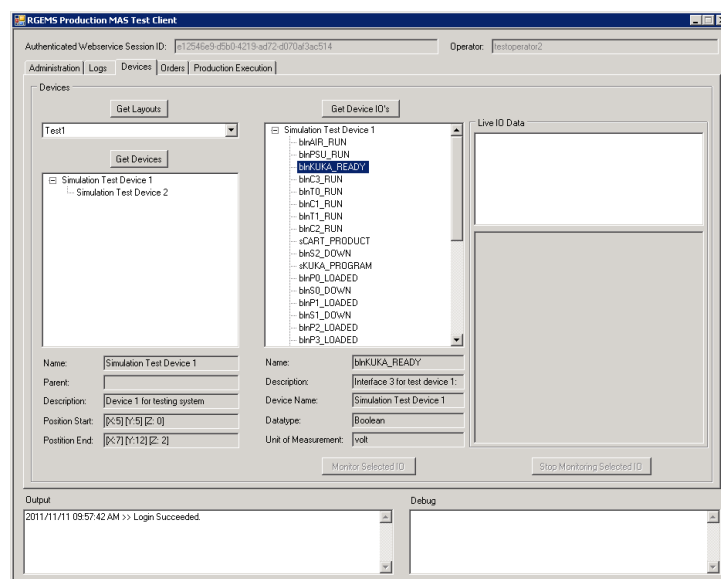


Figure 5.5: Web Service Demo Client Application Device Screen

In the Production Execution window, production in the system can be controlled. In the test scenario, an empty Production Schedule is generated by the Production Planner Agent when the operator clicks on “Generate New Production Schedule”. The schedule is returned to the operator so that it might be customised. When the operator clicks “Initialise Production”, the schedule is sent from the operator client interface to the Production Execution agent for processing. This Agent is to break the schedule up into smaller schedules that are to be sent to various device agents.

In the test scenario, one Device Agent is initialised. It connects to its Device Interfaces and starts listening for updates from the physical devices on the Production Floor. When the operator clicks “Start Production” the Production Execution Agent instructs the Device Agent to start its Production Execution engine. The test Production Execution engine is an event-based multi-threaded executor that executes a set of “*if..then*” rules specified in Java. As the rules constraints become satisfied, different commands are sent to the appropriate Device Interfaces. These commands are then sent to the actual shop floor devices.

The test process was split up into concurrent blocks. Each block contains actions that must be executed in order in that particular block. Blocks can be executed in parallel. Each block has its own dedicated thread within the executor. In the next section, the Test Process is demonstrated.

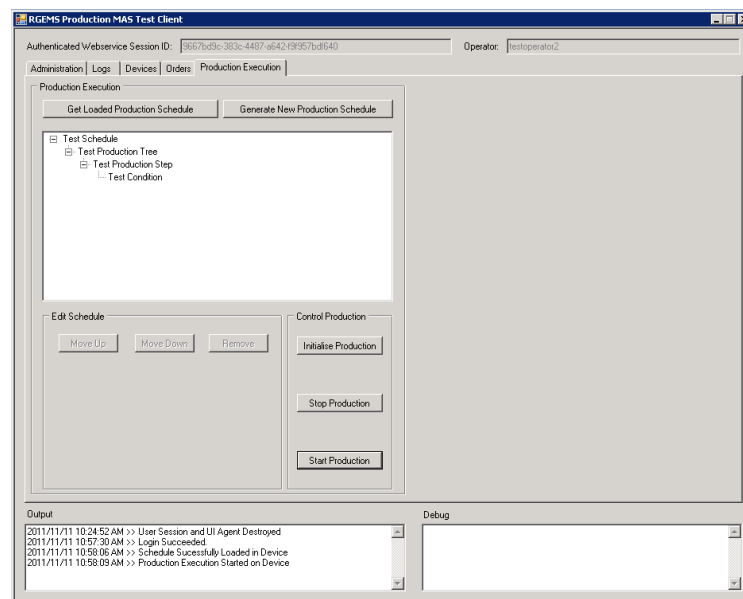


Figure 5.6: The Production Execution Window

5.1.3 Production Execution of the Test Process

The physical conveyor system and process is not yet complete, but a simple Test Process on the various conveyors and related devices can be demonstrated. An operator instructs the system to start production. At this point, the conveyor and transverse unit motors are started by the production executor in a Device agent. Transverse units allow pallets to be transferred between connected parallel conveyors.

Once all the motors are running, the production executor listens for events occurring on the physical test process. Pallets are loaded onto the running conveyor system (conveyor 1) by hand and are stopped at position “A” by a pneumatic stop actuator. The default position for a pneumatic stop actuator is “up” or “stop”, meaning in its default state it stops pallets from continuing. See Figure 5.7.

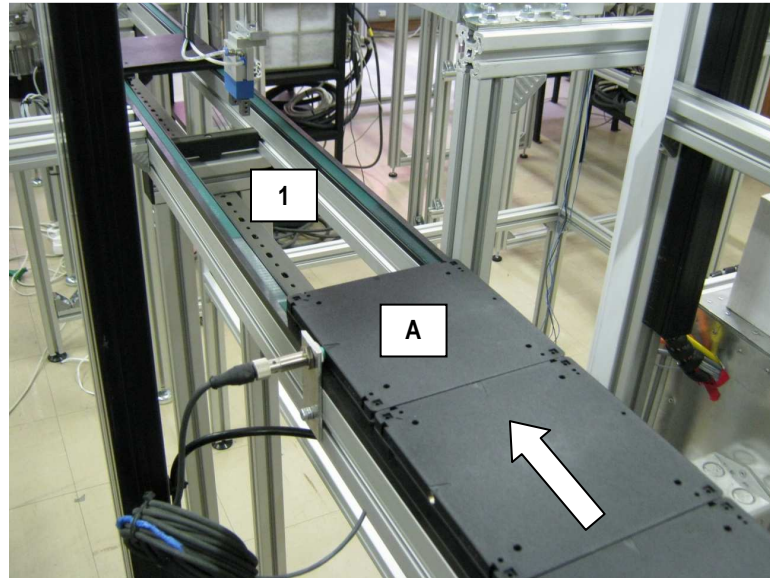


Figure 5.7: Test Process Start

The pallets are not connected to the conveyor; they are placed upon the two green conveyor belts and are moved along purely by friction. When a pallet is stopped by an actuator, the two belts beneath it continue moving, rubbing gently against the pallet. The contact points on the pallets are smooth and anti-static.

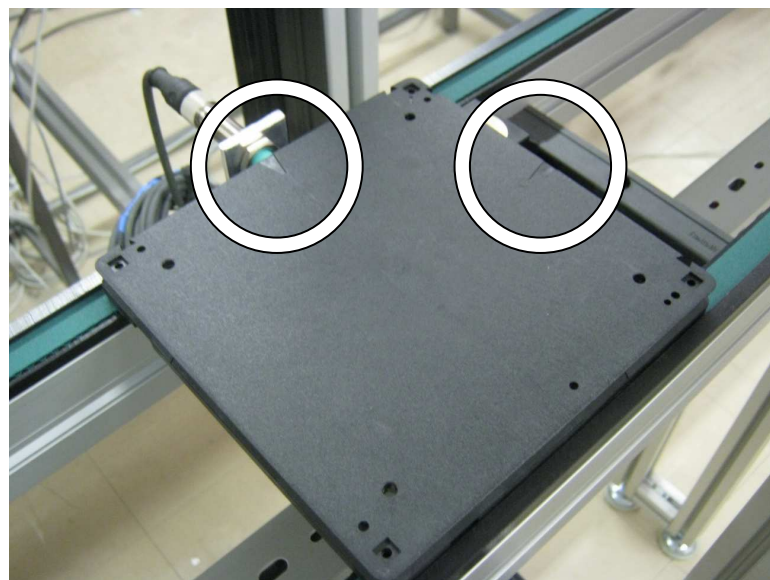


Figure 5.8: Pallet

Each pallet has metallic inserts in the middle of two of its sides (see Figure 5.8). This allows induction proximity sensors to detect if a pallet is present. Using induction proximity sensors in conjunction with the metallic inserts in a pallet allows the system to detect if an individual pallet is completely present at a precisely defined location. These sensors ignore the sides of a pallet until it reaches the metal insert. This is especially useful when pallets are moving and stacked next to one another on the conveyor. In this situation the possibility exists that a normal proximity sensor would always erroneously detect that a pallet is present. Please note that this is not RFID.

When the production executor within a Device agent detects that a pallet is available at position “A”, a pallet is allowed to move through into the system to position “B”, where it is again stopped (see Figure 5.9). At this point, a robotic arm is instructed to partially produce the desired product by picking and placing sub-components of the product onto the product holder. The production executor sends a string representing the product to be produced to the main PLC. The PLC takes over the actual commands to be sent to the robotic arm and reports back when it is done.

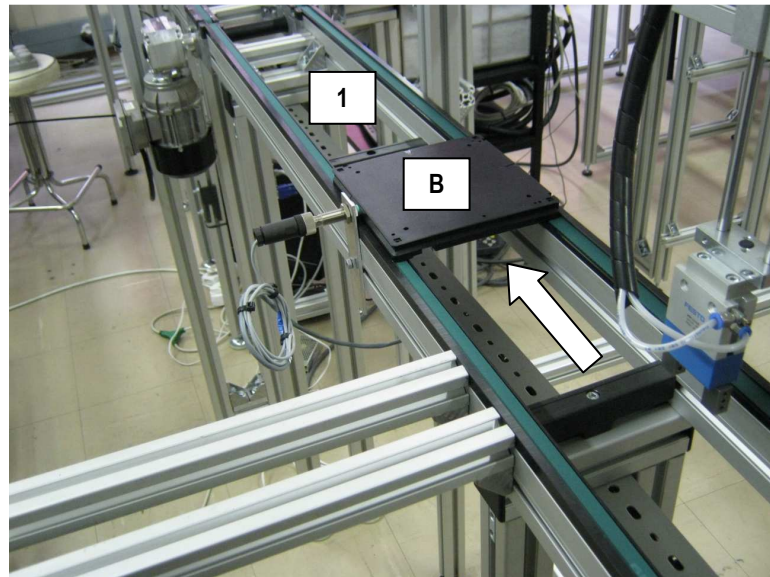


Figure 5.9: Test Process Step 2

When this process is completed, the pallet is released and continues on to conveyor 2 (see Figure 5.10). On conveyor 2 at position “C”, the pallet is either transferred to conveyor 3 where it is stopped at position “D”, or moved straight onto a stop at position “E”. The pallet is only transferred onto the third conveyor if the production executor detects that the third conveyor is running.

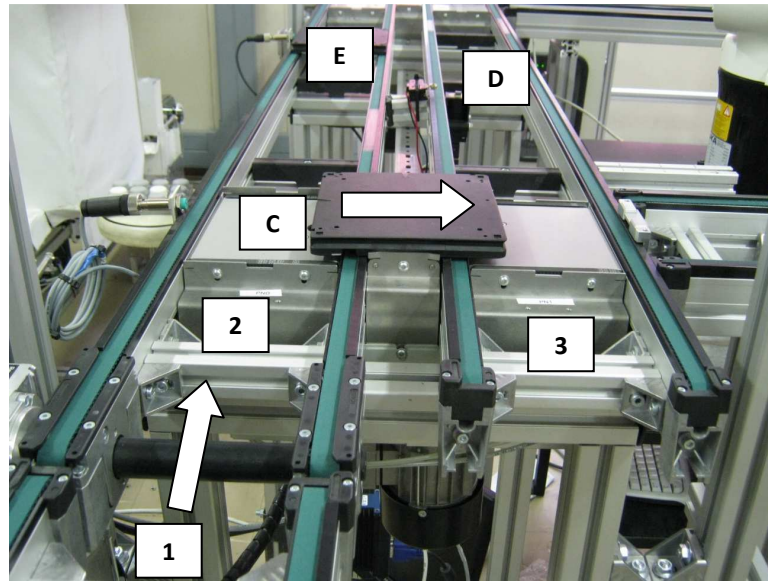


Figure 5.10: Test Process Step 3

When both positions “D” and “E” are occupied by pallets, the KUKA robot is instructed to complete both products, by picking and placing sub-components onto both product containers. The actual low level control of the KUKA robot is done by KRL code in the KUKA controller. Only a string representing the product to be produced is sent to the KUKA controller by the production executor. If the production executor did not transfer a product onto conveyor 3, the KUKA robot is only instructed to produce the product at position “E” on conveyor 2.

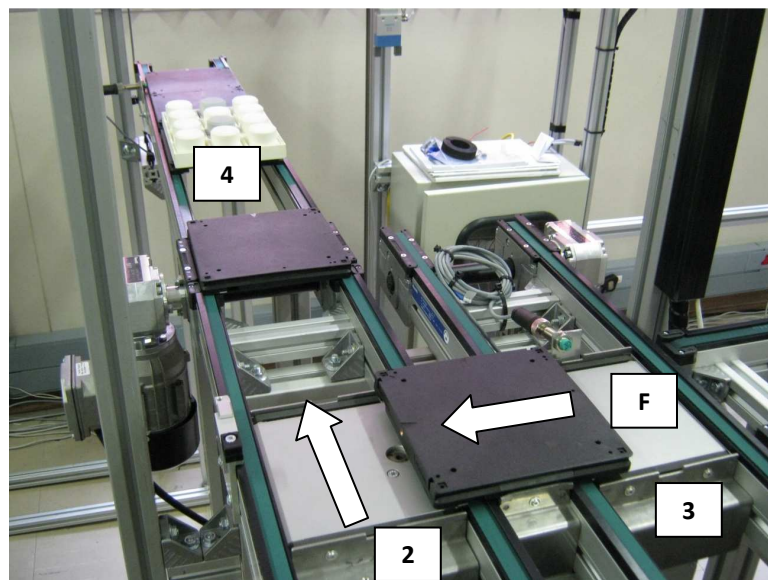


Figure 5.11: Test Process Step 4

When this process is completed, the KUKA robot signals that it is done. The production executor releases both pallets and they continue to move on towards conveyor 4. The pallet in position “D” on conveyor 3 is transferred back to the end of conveyor 2 at position “F” (see Figure 5.11).

The pallets collect at position “G” on conveyor 4, where a robotic arm is instructed to collect the whole product container from the pallet (this mechanism still to be implemented on the product container) and move it over to another conveyor to the left.

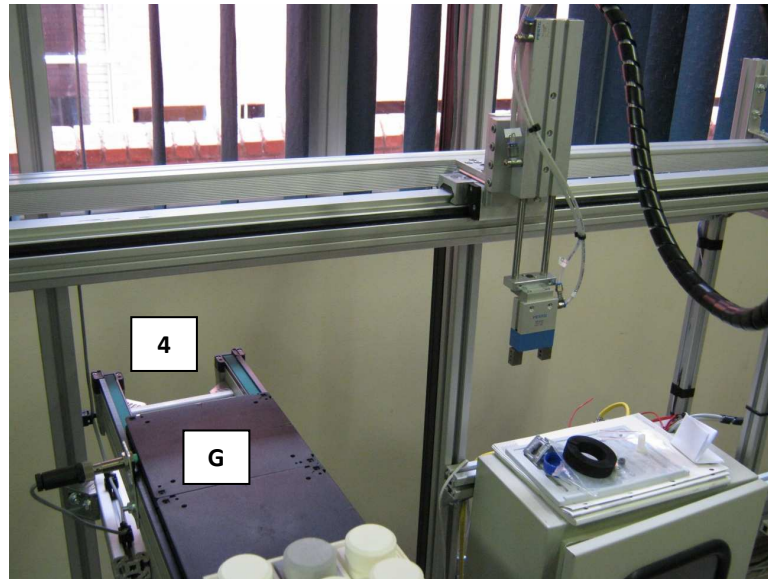


Figure 5.12: Test Process Step 5

5.1.4 Conclusion

The following observations were made concerning the method of control in the Test Process:

- OPC is fast enough for event-based control and monitoring (where events that occur in the physical process triggers code to execute in the Device agent production executor). The test process was successfully controlled using OPC, but OPC cannot be relied upon to always write or read data from a device in a pre-set amount of time, causing timing problems. One example of this is the stop actuators on the conveyors that need to be retracted an exact amount of time to allow only one pallet through at a time. The PLC has to take over the timing aspect of the retract command as it can do so in a pre-set amount of time, every time the command is received from the Device agent.
- True concurrent processing is needed to control even a simple process such as this. There are many sub-processes running concurrently within the overall process and this translates into many events being triggered concurrently in the Device agent’s production executor.

Chapter 6: Conclusion

6.1 Introduction

This chapter summarises the project, examines the research goals and objectives, lists the contributions made, provides a critique of work done, and envisions future work.

6.2 Summary

In Chapter 1 the project was introduced and placed within context. Chapter 2 introduced the test environment as well as introducing OPC. In Chapter 3 various literature studies were conducted to gather more information about the various aspects of multi-agent systems. Chapter 4 documented the implementation of the system and Chapter 5 presented the results obtained from testing.

The literature studies in Chapter 3 revealed many different approaches to multi-agent systems. These approaches were studied and information regarding production scheduling and control, communication, agent types, ontology design, database design and MAS design methodologies was extracted.

The implementation in Chapter 4 revealed what necessitated the development of various components to overcome design challenges such as OPC communication, methodology design tool limitations, operator interaction and ontology implementation.

Chapter 5 demonstrated the system in a limited test environment. It was shown that the MAS design was able to control a production process using OPC as the underlying communication protocol.

6.3 Research Goals and Objectives

The first goal of the evaluation and testing of the OPC communication architecture for real time device control and monitoring was met through the setup of the OPC infrastructure in section 2.4.2 and the implementation and subsequent testing of the Java to OPC framework in section 4.2.

The second goal of the evaluation and testing of a MAS as the core application architecture for the production control application was met by implementing a MAS in an actual test environment in Chapters 4 and 5. Using the Java to OPC framework as communication mechanism, the agents in the MAS were able to control the assembly process.

6.4 Contributions

The project delivered the following contributions:

6.4.1 Production Ontology

A detailed production ontology was developed that was influenced by various reference architectures as well as the domain. The ontology was developed in a well-known ontology development tool and is stored in a well known format.

6.4.2 System Database

A very detailed system database schema was developed. The schema is highly flexible in its representation of various components of the production process. A full implementation of Hierarchical RBAC was also done in the database schema and integrated with the production concepts as well as the production ontology.

6.4.3 Web Service to ACL Translation Procedure

A novel interaction management system between web-service and ACL communication was implemented within an agent (see. Appendix A and Appendix B). The very same Java ontology classes that are used in inter-agent communication are also used to transfer data between agents and the web-service clients outside the MAS. The system is also capable of handling concurrent communication between the MAS and more than one web-service clients.

6.4.4 Integration of RBAC into the Production Ontology

The core requirements of RBAC functioning were integrated into the ontology classes at design time, allowing the ontology classes to be linked to the RBAC implementation in the system database.

6.4.5 IT Infrastructure

A solid IT infrastructure has been put in place at the RGEMS lab. Current and future RGEMS students will be able to access a full range of normal IT services and complete their projects faster and easier. The establishment and setup of the secured OPC architecture will allow future students to focus more on their projects instead of struggling to connect to their devices. The establishment of static DHCP IP assignments has brought order and consistency to the IP network. The establishment of an internal DNS server and NAT services out to the CUT student network allow students to seamlessly use both the RGEMS and CUT student network in a secured manner. The configuration of software licensing server software allows students to use several industrial design software packages that were previously left unused. The establishment of a SVN software versioning server allows software developers within RGEMS to collaborate on projects as well as backing up source code in a central location. The establishment of the Hyper-V virtualisation environment has brought much needed flexibility to the hosting of various services.

6.5 Critical Analysis of Work

At the end of the project, several observations can be made about how things could have been done better and what mistakes were made.

6.5.1 MAS, Ontology and Database

The core of the project application is MAS-based. The majority of MAS architectures and supporting API's currently available are written in Java. This fact led to the adoption of Java as the programming language of the project application.

In Java, OPC is not very well supported by API's. The Java / OPC disconnect took a relatively long time to overcome. If a language such as C++ or C# was chosen as the programming language for the project application, the OPC issues would have been greatly reduced as the OPC API's for these languages are much more readily available. The problem would then have been that an appropriate MAS would have been very difficult or impossible to find. The issue of free software would also have been a problem, as Java has much more free software available for use on the Internet.

Attempts to adhere to standards also complicated the design and implementation. The ability of the project application to communicate with other MAS's and operator clients using standardised protocols insures that the application is flexible to the outside at the cost of introducing complexities within the system. An example of this is the ontology and database. These two components mirror one another in several aspects. If a change is made to one, it is very often necessary to modify the other. This entails not only changing the ontology or database schema on its own, but the Java classes representing these components in the application have to be modified as well.

The other problem with the separation of the database and ontology is transferring data between the two as this is a manual process and the code for each translation has to be written both ways. When the database or ontology is changed, the translation code has to be changed as well. Many database and ontology concepts have recursive relationships defined within them. These translations are especially hard to implement.

A possible solution to these problems would be to sacrifice some flexibility on the outside interaction side of the application and use more focussed platforms that have better development tools and support.

6.5.2 OPC Access

Because of the limitations of OPC access in Java at the start of this project (explained in more detail in 4.2), a significant amount of time was spent on establishing OPC access in Java. This time could have been spent on other parts of the project. The solution was found in using the web-service based XML-DA, but even this solution necessitated the writing of a complete XML-DA client which is accompanied by its own set of complexities. It is recommended that if OPC access is to be attempted through Java, a study be made of new OPC access API's that might have been developed recently. The successor of XML-DA, OPC UA is worth looking into.

6.6 Future Work

6.6.1 Production Planner Implementation

One of the sub-systems that are still to be implemented fully in this project is the production planner. Once this component is designed within the Production Planner agent, an accompanying production execution manager for the Production Execution agent has to be developed. An execution engine for production plans distributed by the manager also has to be implemented in the Device agents.

6.6.2 Product Tracking

Product tracking functionality was designed into the system but never implemented. After RFID infrastructure is introduced into the test system this component can be fully implemented.

6.6.3 Operator Functionality

Currently the operator web service endpoint and client GUI only support the functionality required to start and stop the test implementation of the system. As the system grows more complete, the web service endpoint will be augmented with public functions that can be accessed by operator clients.

References

- [1] *Central University of Technology (CUT)*, Available Online: <http://www.cut.ac.za>, last accessed in January 2010.
- [2] *Research Group in Evolveable Manufacturing Systems (RGEMS)*, Available Online: <http://www.rgems.co.za>, last accessed in January 2010.
- [3] *Stellenbosch University*, Available Online: <http://www.sun.ac.za/index.asp>, last accessed in November 2011.
- [4] *About OPC - What is OPC?*, Available Online: http://www.opcfoundation.org/Default.aspx/01_about/01_what_is_opc.aspx?MID=AboutOPC, last accessed in October 2008.
- [5] *Allen-Bradley*, Available Online: ab.rockwellautomation.com, last accessed in November 2011.
- [6] *Automation Systems - DeviceNet Network Overview*, Available Online: <http://www.ab.com/en/epub/catalogs/12762/2181376/214372/1768364/3404052/>, last accessed in November 2011.
- [7] *PI- PROFIBUS & PROFINET International: PROFIBUS*, Available Online: <http://www.profibus.com/technology/profibusb/>, last accessed in November 2011.
- [8] *Subversion*, Available Online: <http://subversion.apache.org/packages.html>, last accessed in May 2009.
- [9] *SharePoint | Collaboration Software for the Enterprise*, Available Online: <http://sharepoint.microsoft.com/en-us/Pages/default.aspx>, last accessed in November 2011.
- [10] *squid: Optimising Web Delivery*, Available Online: <http://www.squid-cache.org/>, last accessed in November 2011.
- [11] *Cut the cost of printing*, Available Online: <http://www.papercut.com/>, last accessed in November 2011.
- [12] *Kaspersky Lab: Antivirus software*, Available Online: <http://www.kaspersky.com/>, last accessed in November 2011.
- [13] *Hyper-V*, Available Online: [http://technet.microsoft.com/en-us/library/cc753637\(ws.1\).aspx](http://technet.microsoft.com/en-us/library/cc753637(ws.1).aspx), last accessed in November 2011.
- [14] *Routing and Remote Access Service*, Available Online: <http://technet.microsoft.com/en-us/library/cc754634%28WS.10%29.aspx>, last accessed in November 2011.
- [15] *Adaptec*, Available Online: <http://www.adaptec.com/en-us/>, last accessed in November 2011.
- [16] *MySQL : The world's most popular open source database*, Available Online: http://www.mysql.com/?bydis_dis_index=1, last accessed in December 2008.
- [17] *Microsoft SQL Server*, Available Online: <http://www.microsoft.com/hk/sql/default.msp>, last accessed in May 2010.
- [18] *D-Link*, Available Online: <http://www.dlink.com>, last accessed in November 2011.
- [19] *DD-WRT*, Available Online: <http://www.dd-wrt.com/site/index>, last accessed in March 2009.
- [20] *OpenVPN*, Available Online: <http://openvpn.net/>, last accessed in March 2009.
- [21] *OPC Crosslink Suite*, Available Online: http://www.cyberlogic.com/OPC_Crosslink_Suite.html, last accessed in April 2009.
- [22] *Kepware Technologies - OPC Servers / Communications for Automation*, Available Online: <http://www.kepware.com>, last accessed in November 2011.
- [23] *DCOM*, Available Online: <http://msdn.microsoft.com/library/cc201989.aspx>, last accessed in April 2010.
- [24] *The Official Microsoft IIS Site*, Available Online: <http://www.iis.net/>, last accessed in April 2009.

- [25] *IBM*, Available Online: <http://www.ibm.com/us/en/>, last accessed in January 2010.
- [26] *WebSphere Application Server Version 6.1*, Available Online: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/rprf_plugin.html, last accessed in January 2010.
- [27] *The GNU General Public License - GNU Project - Free Software Foundation (FSF)*, Available Online: <http://www.gnu.org/licenses/gpl.html>, last accessed in November 2009.
- [28] *GNU Lesser General Public License - GNU Project - Free Software Foundation (FSF)*, Available Online: <http://www.gnu.org/licenses/lgpl.html>, last accessed in January 2010.
- [29] Ilkka Seilonen, "An Extended Process Automation System: An Approach based on a Multi-Agent System," Ph.D Dissertation, Department of Automation and Systems Technology, Helsinki University of Technology, Helsinki, Espoo, Finland, 2006.
- [30] Janne Jussila, "Agent-Based Approach to Supervisory Information Services in Process Automation," M.S. Thesis, Department of Automation and Systems Technology, Helsinki University of Technology, Helsinki, Espoo, Finland, 2006.
- [31] Milan Fajt, "Information Agents in Process Automation," M.S. Thesis, Department of Automation and Systems Technology, Helsinki University of Technology, Helsinki, Espoo, Finland, 2003.
- [32] "Contributions to Conception, Design and Development of Collaborative Multi-Agent Systems," Ph.D. Dissertation, Faculty of Economics, Business Information Systems Department, Babeş-Bolyai University, Cluj-Napoca, Cluj County, Romania, 2005.
- [33] D Roy, F Vernadat, and D Anciaux, "SYROCO: A novel multi-agent shop-floor control system," *Journal of Intelligent Manufacturing*, vol. 12, no. 3, pp. 295-307, June 2001.
- [34] M Pěchouček and V Mařík, "Industrial deployment of multi-agent technologies: review and selected case studies," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 3, pp. 397-431, December 2008.
- [35] A G Higuera and A C Montalvo, "RFID-enhanced multi-agent based control for a machining system," *International Journal of Flexible Manufacturing Systems*, vol. 19, no. 1, pp. 41-61, March 2007.
- [36] L Mönch and M Stehli, "ManufAg: a multi-agent-system framework for production control of complex manufacturing systems," *Information Systems and E-Business Management*, vol. 4, no. 2, pp. 159-185, April 2006.
- [37] P Leitão, F Restivo, and G Putnik. (2001, Jan., 1). "A multi-agent based cell controller," Available Online: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.4358>, last accessed in December 2008.
- [38] P I Cowling, D Ouelhadj, and S Petrovic, "A Multi-Agent Architecture for Dynamic Scheduling of Steel Hot Rolling," *Journal of Intelligent Manufacturing*, vol. 14, no. 5, p. 457, 470 2003.
- [39] Shanku Chakraborty, "Agent Based Approach to Fault Recovery in a Process Automation System," M.S Thesis, Automation Technology Laboratory, Helsinki University Of Technology, Helsinki, Espoo, Finland, 2003.
- [40] W Lamersdorf, L Braubach, and A Pokahr, "Jadex: Implementing a BDI-Infrastructure for JADE Agents," *EXP - In Search of Innovation*, vol. 3, no. 3, pp. 76-85, March 2003.
- [41] J J Carroll et al., "Jena: Implementing the Semantic Web Recommendations," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, New York, United States of America, May 2004, pp. 74-83.
- [42] *Apache Tomcat*, Available Online: <http://tomcat.apache.org/>, last accessed in May 2010.
- [43] *SQL Introduction*, Available Online: http://www.w3schools.com/sql/sql_intro.asp, last accessed in December 2012.
- [44] *OWL Web Ontology Language Overview*, Available Online: <http://www.w3.org/TR/owl-features/>,

last accessed in May 2010.

- [45] *Foundation for Intelligent Physical Agents*, Available Online: <http://www.fipa.org/>, last accessed in May 2010.
- [46] *Rockwell Automation*, Available Online: www.rockwellautomation.com, last accessed in May 2010.
- [47] K Finkenzeller, *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*, New York: John Wiley & Sons, 2003.
- [48] *Microsoft.NET Framework*, Available Online: <http://www.microsoft.com/net/>, last accessed in May 2010.
- [49] *Jade - Java Agent DEvelopment Framework - Technical Description*, Available Online: <http://jade.cselt.it/description-technical.htm>, last accessed in May 2010.
- [50] J C Collis, D T Ndumu, H S Nwana, and L C Lee, "The ZEUS Agent Building Tool-kit ," *BT Technology Journal*, vol. 16, no. 3, pp. 60-68, November 2004.
- [51] "Reference Architecture for Holonic Manufacturing Systems: PROSA," *Computers In Industry, Special Issue on Intelligent Manufacturing Systems*, vol. 37, no. 3, pp. 255-276, November 1998.
- [52] *Java SE Documentation at a Glance*, Available Online: <http://java.sun.com/javase/reference/index.jsp>, last accessed in May 2010.
- [53] *Microsoft Windows*, Available Online: <http://www.microsoft.com/windows/default.aspx>, last accessed in May 2010.
- [54] *ISO/IEC JTC1/SC22/WG21 - The C++ Standards Committee*, Available Online: <http://www.open-std.org/jtc1/sc22/wg21/>, last accessed in May 2010.
- [55] *Extensible Markup Language (XML)*, Available Online: <http://www.w3.org/XML/>, last accessed in 12 2011.
- [56] E Alba and R Martí, "Tabu Search," in *Metaheuristic Procedures for Training Neural Networks*, vol. 36 New York, United States of America: Springer US, 2006, pp. 53-69.
- [57] *Greedy Algorithms*, Available Online: <http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch17.html>, last accessed in May 2010.
- [58] K H Dam and M Winikoff, "Comparing Agent-Oriented Methodologies," *Lecture Notes in Computer Science*, no. 3030, pp. 78-93, 2004.
- [59] O Shehory and A Sturm, "Evaluation of Modeling Techniques for Agent-Based Systems," in *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, May 2001, pp. 624-631.
- [60] J Sudeikat, L Braubach, A Pokahr, and W Lamersdorf, "Evaluation of Agent-Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform," in *Agent-Oriented Software Engineering V*, vol. 3382/2005 Berlin: Springer Berlin / Heidelberg, 2005, pp. 126-141.
- [61] A Sturm and O Shehory, "A Framework for Evaluating Agent-Oriented Methodologie," in *Agent-Oriented Information Systems*, vol. 3030/2004 Berlin, Germany: Springer Berlin / Heidelberg, 2004, pp. 94-109.
- [62] *Netbeans*, Available Online: <http://netbeans.org/>, last accessed in December 2008.
- [63] *Eclipse*, Available Online: <http://www.eclipse.org/>, last accessed in December 2008.
- [64] *TAOM4E*, Available Online: <http://sra.itc.it/tools/taom4e/>, last accessed in December 2009.
- [65] J Mylopoulos, F Giunchiglia, P Giorgini, A Perini, and P Bresciani, "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203-236, October 2004.
- [66] M Morandini, "Knowledge Level Engineering of BDI Agents," M.S. Thesis, Faculty of

- Mathematical, Physical and Natural Science, University of Trento, Trento, Province of Trento, Italy, 2005.
- [67] A Perini, J Mylopoulos, and F Giunchiglia, "The Tropos Software Development Methodology: Processes, Models and Diagrams," in *Agent-Oriented Software Engineering III*, vol. 2585/2003 Berlin, Germany: Springer Berlin / Heidelberg, 2003, pp. 162-173.
- [68] M P Singh and A U Mallya, "Incorporating Commitment Protocols into Tropos," in *Agent-Oriented Software Engineering VI*, vol. 3950/2006 Berlin, Germany: Springer Berlin / Heidelberg, 2006, pp. 69-80.
- [69] M Weiss and H Mouratidis, "Patterns for Modelling Agent Systems with Tropos," in *Software Engineering for Multi-Agent Systems IV*, vol. 3914/2006 Berlin, Germany: Springer Berlin / Heidelberg, 2006, pp. 207-223.
- [70] A Pirotte, M Kolp, and T T Do, "Social Patterns for Designing Multiagent Systems," in *Proceedings of the Fifteenth International Conference on Software Engineering & Knowledge Engineering*, San Francisco Bay, United States of America, July 2003, pp. 103-110.
- [71] G Manson, P Giorgini, and H Mouratidis, "Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems," in *Advanced Information Systems Engineering*, vol. 2681/2010 Berlin, Germany: Springer Berlin / Heidelberg, 2003, p. 1031.
- [72] P Giorgini and H Mouratidis, "Secure Tropos: a Security-Oriented Extension of the Tropos Methodology," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 2, pp. 285-309, April 2007.
- [73] H Mouratidis, P Giorgini, and M Schumacher, "Security Patterns for Agent Systems," presented at the *Proceedings of the Eight European Conference on Pattern Languages of Programs*, Irsee, Germany, June 2003.
- [74] *Introduction to Ontologies and Semantic Web*, Available Online: <http://www.obitko.com/tutorials/ontologies-semantic-web/introduction.html>, last accessed in January 2010.
- [75] L et. al Pouchard, "Ontology Engineering for Distributed Collaboration in Manufacturing," , Tuscon, Unisted States of America, March 2000.
- [76] T Gruber, *Encyclopedia of Database Systems*, 2008, Provides a definition of ontology as a technical term for computer science, tracing its historical context from philosophy and AI.
- [77] T Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," *International Journal Human-Computer Studies* , vol. 43, no. 5-6, pp. 907-928, August 1993.
- [78] A Gómez-Pérez and O Corcho, "A RoadMap to Ontology Specification Languages," in *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, Juan-les-Pins, France, October 2000, pp. 80-96.
- [79] *The Syntax of CycL*, Available Online: <http://www.cyc.com/cycdoc/ref/cycl-syntax.html>, last accessed in January 2010.
- [80] *Knowledge Interchange Format (KIF)*, Available Online: <http://www.ksl.stanford.edu/knowledge-sharing/kif/>, last accessed in February 2010.
- [81] *DAML+OIL Reference Description*, Available Online: <http://www.w3.org/TR/daml+oil-reference>, last accessed in February 2010.
- [82] R Sharman, R Kishore, and R Ramesh, "Foundations for a Core Ontology of Manufacturing," in *Ontologies*, vol. 14 New York, United States of America: Springer US, 2007, pp. 751-775.
- [83] S Lemaignan, A Siadat, J Y Dantan, and A Semenenko, "MASON: A Proposal For An Ontology Of Manufacturing Domain," presented at the *Distributed Intelligent Systems: Collective Intelligence and Its Applications*, Prague, Czech Republic, June 2006.
- [84] *Laboratory for Applied Ontology - DOLCE*, Available Online: <http://www.loa-cnr.it/DOLCE.html>, last accessed in February 2010.
- [85] *General Formal Ontology (GFO)*, Available Online:

- med.de/ontologies/gfo/index.jsp, last accessed in January 2010.
- [86] *Basic Formal Ontology (BFO)*, Available Online: <http://www.ifomis.org/bfo/>, last accessed in April 2010.
 - [87] *The Suggested Upper Merged Ontology (SUMO) - Ontology Portal*, Available Online: <http://www.ontologyportal.org/>, last accessed in February 2010.
 - [88] *OpenCyc*, Available Online: <http://www.opencyc.com>, last accessed in April 2010.
 - [89] S Borgo and P Leitão, "The Role of Foundational Ontologies in Manufacturing Domain Applications," in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, vol. 3290/2004 Berlin, Germany: Springer Berlin / Heidelberg, 2004, pp. 670-688.
 - [90] *The Protégé Ontology Editor and Knowledge Acquisition System*, Available Online: <http://protege.stanford.edu/>, last accessed in December 2008.
 - [91] *Sigma*, Available Online: <http://sourceforge.net/projects/sigmakee/>, last accessed in April 2010.
 - [92] *DODDLE*, Available Online: <http://sourceforge.net/projects/doddle-owl/>, last accessed in April 2010.
 - [93] *Owl Interactive*, Available Online: <http://sourceforge.net/projects/owl-interactive/>, last accessed in April 2010.
 - [94] *AKSW: Projects / DL Learner*, Available Online: <http://dl-learner.org/Projects/DLLearner>, last accessed in April 2010.
 - [95] *Jastor*, Available Online: <http://jastor.sourceforge.net/>, last accessed in April 2010.
 - [96] *AgentOWL*, Available Online: <http://sourceforge.net/projects/agentowl/>, last accessed in April 2010.
 - [97] *OntoBridge*, Available Online: <http://sourceforge.net/projects/ontobridge/>, last accessed in April 2010.
 - [98] *Jena Semantic Web Framework*, Available Online: <http://jena.sourceforge.net/>, last accessed in April 2010.
 - [99] *Pellet: The Open Source OWL 2 Reasoner*, Available Online: <http://clarkparsia.com/pellet>, last accessed in April 2010.
 - [100] *Racer Systems GmbH & Co. KG : Products*, Available Online: <http://www.racer-systems.com/products/racerpro/index.phtml>, last accessed in April 2010.
 - [101] *Hermit Reasoner: Home*, Available Online: <http://hermit-reasoner.com/>, last accessed in April 2010.
 - [102] *OWL API*, Available Online: <http://owlapi.sourceforge.net/>, last accessed in April 2010.
 - [103] *OntologyBeanGenerator 4.1*, Available Online: http://protegewiki.stanford.edu/index.php/OntologyBeanGenerator_4.1, last accessed in June 2009.
 - [104] *Jadex Tool Guide*, Available Online: <http://jadex.informatik.uni-hamburg.de/docs/jadex-0.96x/toolguide/index.single.html#tools.beanynizer>, last accessed in July 2009.
 - [105] *DAI-Labor / JIAC*, Available Online: <http://www.dai-labor.de/index.php?id=1805&L=1>, last accessed in December 2008.
 - [106] T Konnerth, B Hirsch, and S Albayrak, "JADL – An Agent Description Language for Smart Agents," in *Declarative Agent Languages and Technologies IV*, vol. 4327/2006 Berlin, Germany: Springer Berlin / Heidelberg, 2007, pp. 141-155.
 - [107] *The Implementation of LISP*, Available Online: <http://www-formal.stanford.edu/jmc/history/lisp/node3.html>, last accessed in May 2010.
 - [108] *Drools - JBoss Community*, Available Online: <http://www.jboss.org/drools/>, last accessed in September 2009.
 - [109] *JBoss - Global Leader in Open Source Middleware Software*, Available Online:

- <http://www.jboss.com/products/wfk/>, last accessed in September 2009.
- [110] *SweetRules Project Home Page*, Available Online: <http://sweetrules.projects.semwebcentral.org/>, last accessed in April 2010.
 - [111] *RuleML Homepage*, Available Online: <http://ruleml.org/>, last accessed in April 2010.
 - [112] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, Available Online: <http://www.w3.org/Submission/SWRL/>, last accessed in April 2010.
 - [113] *JRuleEngine - OpenSource Java Rule Engine*, Available Online: <http://jruleengine.sourceforge.net/>, last accessed in April 2010.
 - [114] Q H Mahmoud. (2005, July, 26). "Getting Started With the Java Rule Engine API (JSR 94): Toward Rule-Based Applications," Available Online: <http://java.sun.com/developer/technicalArticles/J2SE/JavaRule.html>, last accessed in April 2010.
 - [115] *JLisa - A Rule Engine for Java*, Available Online: <http://jlisa.sourceforge.net/>, last accessed in March 2010.
 - [116] *Jess, the Rule Engine for the Java Platform*, Available Online: <http://www.jessrules.com/>, last accessed in March 2010.
 - [117] *Jadex Rules*, Available Online: <http://jadex-rules.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview>, last accessed in March 2010.
 - [118] *KAON2 -- Ontology Management for the Semantic Web*, Available Online: <http://kaon2.semanticweb.org/>, last accessed in April 2010.
 - [119] M Balaban, "The F-Logic Approach For Description Languages ," *Annals of Mathematics and Artificial Intelligence*, vol. 15, no. 1, pp. 19-60, April 2005.
 - [120] *Pellet: The Open Source OWL 2 Reasoner*, Available Online: <http://clarkparsia.com/pellet>, last accessed in March 2010.
 - [121] *QuOnto*, Available Online: <http://www.dis.uniroma1.it/~quonto/>, last accessed in April 2010.
 - [122] *Owl Lite*, Available Online: <http://www.w3.org/TR/owl-ref/#OWLLite>, last accessed in March 2010.
 - [123] *Drools Planner - JBoss Community*, Available Online: <http://www.jboss.org/drools/drools-planner.html>, last accessed in February 2010.
 - [124] *DLPlan*, Available Online: <http://sourceforge.net/projects/dlplan/>, last accessed in April 2010.
 - [125] *PL-PLAN, an AI Planner*, Available Online: <http://plplan.philippe-fournier-viger.com/index.html>, last accessed in April 2010.
 - [126] *GraphPlan*, Available Online: <http://www.cs.cmu.edu/~avrim/graphplan.html>, last accessed in April 2010.
 - [127] *JPlan: Java GraphPlan Implementation*, Available Online: <http://sourceforge.net/projects/jplan/>, last accessed in April 2010.
 - [128] *PDDL4J*, Available Online: <http://sourceforge.net/projects/pdd4j/>, last accessed in April 2010.
 - [129] T Murata, "Petri nets: Properties, analysis and applications," in *Proceedings of the IEEE*, August 2002, pp. 541-580.
 - [130] W Zhang, T Freiheit, and H Yang, "Dynamic Scheduling in Flexible Assembly System Based on Timed Petri Nets Model ," *Robotics and Computer-Integrated Manufacturing*, vol. 21, no. 6, pp. 550-558, December 2005.
 - [131] L L Zhang and B Rodrigues, "A Petri Net-based Approach to Reconfigurable Manufacturing Systems Modeling," *Journal of Systemics, Cybernetics and Informatics*, vol. 7, no. 1, pp. 18-24, 2009.
 - [132] R H Ghouil, A Benjelloul, S Kechida, and H Tebbikh, "A Scheduling Algorithm Based on Petri Nets

- and Simulated Annealing," *American Journal of Applied Sciences*, vol. 4, no. 5, pp. 269-273, 2007.
- [133] N G Odrey and G Mejía, "An Augmented Petri Net Approach for Error Recovery in Manufacturing Systems Control," in *Robotics and Computer-Integrated Manufacturing*, Toronto, Canada, July 2004, pp. 346-354.
- [134] *JFern, Java-based Petri Net framework*, Available Online: <http://sourceforge.net/projects/jfern/>, last accessed in March 2010.
- [135] *JPetriNet*, Available Online: <http://sourceforge.net/projects/jpetrinet/>, last accessed in March 2010.
- [136] *Renew*, Available Online: <http://www.renew.de/>, last accessed in March 2010.
- [137] *CO-OPN and COOPNBuilder*, Available Online: <http://smv.unige.ch/research-projects/co-opn>, last accessed in April 2010.
- [138] *Petri Net Kernel*, Available Online: <http://www2.informatik.hu-berlin.de/top/pnk/>, last accessed in April 2010.
- [139] *Petri Net Markup Language*, Available Online: <http://www2.informatik.hu-berlin.de/top/pnml/about.html>, last accessed in April 2010.
- [140] *The Open Business Engine*, Available Online: <http://obe.sourceforge.net/>, last accessed in April 2010.
- [141] *Wade - Workflows and Agents Development Environment*, Available Online: <http://jade.tilab.com/wade/>, last accessed in April 2010.
- [142] *Drools Flow*, Available Online: <http://www.jboss.org/drools/drools-flow.html>, last accessed in April 2010.
- [143] *Jadex Processes*, Available Online: <http://jadex-processes.informatik.uni-hamburg.de/xwiki/bin/view/About/Overview>, last accessed in April 2010.
- [144] N Lohse, H Hirani, and S Ratchev, "Equipment Ontology for Modular Reconfigurable Assembly Systems," *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 301-314, October 2005.
- [145] R Sandhu, D Ferraiolo, and R Kuhn, "The NIST Model for Role-Based Access Control: Towards a Unified Standard," in *Proceedings of the fifth ACM workshop on Role-based access control*, Berlin, Germany, July 2000, pp. 47-63.
- [146] M Drouineaud, A Luder, and K Sohr. (2003). "A Role based Access Control Model for Agent based Control Systems," Available Online: www.informatik.uni-bremen.de/~sohr/papers/indin03.pdf, last accessed in December 2009.
- [147] N Borselius, "Security in Multi-Agent Systems," in *Proceedings of the 2002 International Conference on Security and Management*, Nevada, United States of America, 2002, pp. 31-36.
- [148] *Fine Grained Role Based Access Control (RBAC) system*, Available Online: http://www.sqlrecipes.com/database_design/fine_grained_role_based_access_control_rbac_system-3/, last accessed in January 2010.
- [149] C G Lasater, *Design Patterns*, Texas: Wordware Publishing, Inc., 2007.
- [150] *jax-ws: JAX-WS Reference Implementation*, Available Online: <https://jax-ws.dev.java.net/>, last accessed in December 2009.
- [151] R K Mitchell, B R Agle, and D J Wood, "Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts," *Academy of Management Review*, vol. 22, no. 4, pp. 853-886, 1997.
- [152] Moore Industries- International, Inc. (2006, January, 1). "Introduction to Fieldbus," Available Online: http://www.fieldbus.org/images/stories/newsroom/articles/wuhan_article_rev_01_lhbt20090528_ff_nxpowerlite_.pdf, last accessed in May 2010.
- [153] R M Metcalfe and D R Boggs, "Ethernet: Distributed Packet Switching for Local Computer

- Networks," in *Innovations in Internetworking*, C Partridge, Ed. Massachusetts, United States of America: Artech House, Inc, 1998, pp. 25-34.
- [154] *Proficy HMI/SCADA - iFIX 5.0*, Available Online: <http://www.ge-ip.com/products/3311#overview>, last accessed in May 2010.
- [155] *NetModule JOPC-Bridge 4.5*, Available Online: http://www.netmodule.com/en/products/netmodule_jopcbridge/, last accessed in December 2008.
- [156] S Poslad, P Buckle, and R Hadingham, "Open Source, Standards and Scaleable Agencies," in *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, vol. 1887/2001 Berlin, Germany: Springer Berlin / Heidelberg, 2001, pp. 296-303.
- [157] M Balachandran, "Developing Intelligent Agent Applications with JADE and JESS," in *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 5179/2010 Berlin, Germany: Springer Berlin / Heidelberg, 2008, pp. 236-244.
- [158] *Apache Struts*, Available Online: <http://struts.apache.org/>, last accessed in May 2010.
- [159] *iBatis*, Available Online: <http://ibatis.apache.org/>, last accessed in May 2010.
- [160] F Bellifemine, G Caire, A Poggi, and G Rimassa, "JADE: A software framework for developing multi-agent applications. Lessons learned," *Information and Software Technology*, vol. 50, no. 1-2, pp. 10-21, January 2008.
- [161] *CVS - Open Source Version Control*, Available Online: <http://www.nongnu.org/cvs/>, last accessed in May 2010.
- [162] *OWL Web Ontology Language Guide*, Available Online: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/#DescriptionLogics>, last accessed in April 2010.
- [163] P Leitão, A W Colombo, and F Restivo, "Formal Specification of ADACOR Holonic Control System: Coordination Models," in *Proceedings of the 44th IEEE Conference on Decision and Control*, Seville, Spain, December 2005, pp. 2137-2142.
- [164] *Protégé Editor Example Image*, Available Online: <http://protege.stanford.edu/plugins/owl/images/OWLViz-BackpackersDestination.png>, last accessed in January 2010.
- [165] *RacerPro*, Available Online: <http://www.racer-systems.com/products/racerpro/index.phtml>, last accessed in March 2010.
- [166] *Tapaal*, Available Online: <http://www.tapaal.net/index.php?id=22>, last accessed in April 2010.
- [167] *Visual Studio*, Available Online: <http://msdn.microsoft.com/en-gb/vstudio/default.aspx>, last accessed in December 2008.
- [168] *Composite Design Pattern in C# and VB.NET*, Available Online: <http://www.dofactory.com/patterns/patterncomposite.aspx>, last accessed in December 2009.
- [169] *Adapter Design Pattern in C# and VB.NET*, Available Online: <http://www.dofactory.com/patterns/PatternAdapter.aspx>, last accessed in December 2009.
- [170] *Observer Design Pattern in C# and VB.NET*, Available Online: <http://www.dofactory.com/Patterns/PatternObserver.aspx>, last accessed in December 2009.
- [171] *Iterator Design Pattern in C# and VB.NET*, Available Online: <http://www.dofactory.com/Patterns/PatternIterator.aspx>, last accessed in December 2009.
- [172] M J Huber, "JAM: a BDI-theoretic mobile agent architecture," in *Proceedings of the third annual conference on Autonomous Agents*, Seattle, Washington, May 1999, pp. 236-243.
- [173] *OPC XML-DA Specification, Version 1.0*, July 12, 2003.
- [174] *Object Management Group*, Available Online: http://www.omg.org/technology/documents/spec_catalog.htm#Middleware, last accessed in December 2011.

Appendices

Appendix A Example of Operator Web service Endpoint Public Function

```
/**
 * webMethod
 * @param sessionString Valid Session String to validate login
 * @return Defaultuser object containing the details of the current user
 * @throws Exception
 */
@WebMethod(operationName = "RequestReturnUserDetails_ADVANCED_POLL")
public DefaultUser RequestReturnUserDetails_ADVANCED_POLL(@WebParam(name =
    "sessionString") String sessionString)
    throws Exception
{
    //Object to return
    DefaultUser objectToReturn = null;
    //Create Conversation Registration Object
    cwebserviceConversation regconv = new cwebserviceConversation();
    try
    {
        if(_real_agentAgent.getActiveSessions().containsKey(sessionString))
        {
            //Create Message
            IMessageEvent event = _agent.createMessageEvent("request");
            event.getParameterSet(SFipa.RECEIVERS).addValue(_agent.
                getAgentIdentifier());

            //Populate action object
            DefaultExecutedDatabaseQuery request =
                new DefaultExecutedDatabaseQuery();
            request.setWebServiceTransactionID(java.util.UUID.randomUUID().
                toString());
            request.setUserSessionID(sessionString);
            request.setOperationToBePerformedByUser(new DefaultRetrieve());
            request.setProductionDomainConceptOperationAppliesTo(
                ProductionOntologyOntology.USER);
            request.setProductionDomainContextConcept(
                ProductionOntologyOntology.DATABASE);

            //Create Action wrapper Object
            Action act = new Action();
            //Embed Action in wrapper action
            act.setAction(request);
            AID agentid = new AID(_agent.getAgentName());
            act.setActor(agentid);

            //Put wrapper action into message
            event.setContent(act);
            regconv.setAuthenticatedSessionID(request.getUserSessionID());
            regconv.setWebServiceTransactionID(request.
                getWebServiceTransactionID());

            //Monitor object for this thread
```



```

        Object monitor = new Object();
        regconv.setThreadMonitorObject(monitor);
        //Register Conversation in conversation list in agent
        _real_agentAgent.RegisterWebserviceConversation(regconv);
        //The ws calling thread will wait here on the monitor object
        //The agent will wake this thread using this monitor object when it gets a reply
back from another agent
        //A lock is established on the monitor object before sending the message
        synchronized(monitor)
        {
            //Send message to agent
            _agent.sendMessage(event);
            monitor.wait(_MonitorWaitTime);

            //Reply is received from MAS
            DefaultResultOfProductionAction result =
                (regconv.getActionPredicate());

            if(result.getIsException())
            {
                throw new Exception(result.getResultMessage());
            }

            if(result.getRequestStatus() instanceof DefaultSucceeded)
            {
                objectToReturn = (DefaultUser) (result.
                    getListOfGeneratedConcepts().
                    getProductionConceptList().get(0));
            }

            //Remove Conversation
            this._real_agentAgent.DeRegisterWebserviceConversation(regconv.
                getWebServiceTransactionID());
        }
    }
    else
    {
        regconv.setExceptionMessage(
            "Please login before calling service function");
        throw new Exception(regconv.getExceptionMessage());
    }
}
catch(Exception ex)
{
    Logger.getLogger(cWebserviceFace.class.getName()).log(Level.SEVERE,
        null, ex);
    this._real_agentAgent.DeRegisterWebserviceConversation(regconv.
        getWebServiceTransactionID());
    throw new Exception(regconv.getExceptionMessage());
}
return objectToReturn;
}

```

Appendix B Web service Service Plan

```
public class servicePlan_Run_Webservice extends Plan
{
    //web service class containing public web methods
    cwebserviceFace _face;
    //webservice
    Endpoint _endpoint;
    //Que strategy for thread pool
    SynchronousQueue _que;
    //Threadpool for webservice requests
    ExecutorService _pool;
    //URL to publish webservice on
    String _webserviceURL = "";
    //Collection of conversations the agent is handling from the web service
    private ConcurrentHashMap<String, cwebserviceConversation> _webserviceConversations =
        new ConcurrentHashMap<String, cwebserviceConversation>();
    //Linking of web service conversation with ACL message ID
    private ConcurrentHashMap<String, String> _webserviceConversationsACL =
        new ConcurrentHashMap<String, String>();
    //List of active sessions
    private ConcurrentHashMap<String, DefaultActiveSession> _ActiveSessions =
        new ConcurrentHashMap<String, DefaultActiveSession>();

    /**
     * The plan executes here in one thread in a continuous loop paused by a blocking call that
     * waits for
     * messages from the webservice or other agents
     */
    @Override
    public void body()
    {
        IWaitqueue que = getWaitqueue();
        que.addMessageEvent("collector_event");
        try
        {
            //Start the web service
            startWebservice();

            //This is the main message pump for the agent. It loops until the indicated variable
            //becomes false
            while((Boolean) (this.getBeliefbase().getBelief(
                "global_agent_offers_service").getFact()))
            {
                try
                {
                    //This is a blocking call - thread will wait here and continue
                    //once a message is recieved. Agent will cache messages that arrive
                    //while this code is executing
                    IMessageEvent message = this.waitForMessageEvent(
                        "collector_event");

                    //Two types of messages can arrive at the agent:
                    //1. Action objects from the webservice
                    //2. Predicate objects from other agents in the MAS to WS clients

                    //Look for Action objects here
                }
            }
        }
    }
}
```

```

        if(message.getContent() instanceof Action)
        {
            RouteActionToMAS(message);
        }
        //Look for Predicate objects here - wake the waiting thread in the webservice
        else
        {
            NotifySleepingWSThread(message);
        }
    }
    catch(Exception e)
    {
        System.out.println("AGENT: " + this.getAgentName() +
            ">> EXCEPTION IN WEBSERVICE MESSAGE PUMP: [" + e.
            getMessage() + "]"");
    }
}
//Loop exited, stop the webservice
Stopwebservice();
}
catch(Exception e)
{
    System.out.println("AGENT: " + this.getAgentName() +
        ">> ERROR STARTING WEBSERVICE: [" + e.getMessage() + "]"");
    Stopwebservice();

    this.fail(e.getMessage(), e);
}
}

/**
 * Start the web Service
 * @throws Exception
 */
private void Startwebservice()
    throws Exception
{...}

/**
 * Reset all web Service related objects
 */
private void ResetwebserviceObjects()
{...}

/**
 * Stop the web Service
 */
private void Stopwebservice()
{...}

/**
 * Add a webservice conversation to the conversation list
 * @param conv Conversation object used to match stateless web Service calls to answers
recieved from
 * agents in the MAS
 */
public void RegisterwebserviceConversation(cwebserviceConversation conv)
{...}

```

```
/**
 * Remove a Conversation from the conversation list. Should occur after every time
 * data is returned to a web service call
 * @param webServiceTransactionID
 */
public void DeRegisterWebserviceConversation(String webServiceTransactionID)
{...}

public cwebserviceConversation GetWebServiceConversation(
    String webServiceTransactionID)
{...}

public ConcurrentHashMap<String, DefaultActiveSession> getActiveSessions()
{...}

public ConcurrentHashMap<String, cwebserviceConversation> getWebServiceConversations()
{...}

private void NotifySleepingWSThread(IMessageEvent message)
{...}

private void RouteActionToMAS(IMessageEvent message)
{...}
}
```