

UNIVERSIDADE NOVA DE LISBOA

Faculdade de Ciências e Tecnologia
Departamento de Engenharia Electrotécnica



**AN ONTOLOGY-BASED REPRESENTATION OF AN AGENT-
BASED CONTROLLED ROBOTIC CELL**

Ricardo N. S. Cruz Gomes

Dissertação apresentada na Faculdade de Ciência e Tecnologia da
Universidade Nova de Lisboa para a obtenção do grau de Mestre em
Engenharia Electrotécnica e de Computadores

Supervisor: José António Barata de Oliveira

Abstract

Customers demand for high product customization and differentiation, and short product life-cycle. As such, industries have to adapt their manufacturing systems more frequently in order to remain competitive.

Changing manufacturing systems within a short period of time requires a huge effort in terms of time and money, reducing this effort would make industries more competitive.

The proposed solution consists in developing an ontology-based multi-agent system to control manufacturing systems.

Defining the ontology for the manufacturing system allows the control to perform its operation, and when changes arise, it is required to change the ontology so that the control became aware of the changes to control the manufacturing system.

An ontology-based control allows for a smaller setup time since the control is not specific for one physical system and can be applied to different ones, therefore it reduces the effort in adapting manufacturing systems to required changes allowing industries to become more competitive.

Flexibility is given by the multi-agent system that controls the physical system with the ontology.

Stating this, the solution of an ontology-based control for manufacturing systems provides the required results.

Preface

This thesis marks the end of 5 years of work as a university student and this journey had the influence of several people to whom I would like to express my gratitude.

In respect to this thesis, I would like to express my gratitude to my project supervisor M. Sc. Vladimir V. Herrera for his advices and constant guidance, then to my thesis supervisor Prof. José L. Martinez Lastra for the opportunity of making this thesis at the Tampere University of Technology as well as for the professors at my home university from the Pedagogic Council of the Department of Electrotechnical Engineering of the New University of Lisbon and also to Prof. José Barata for accepting being my thesis supervisor at my home university, I would like also to express my gratitude to the people of FAST lab that dealt with me everyday during its development.

Since the present has no meaning without past, I would like to thank those from the “núcleo duro” (university group of friends) for their support and friendship, and also to some friends namely: Francisco Ganhão, César Soares, Tiago Fonseca, José Lúzio, Ruben Lino, Gonçalo Luís, Ricardo Morais and Luciano Batalha.

Finally, this was not possible without the complete support of my family to whom I would like to express the most kind and special gratitude.

To all of you a great thank you.

Tampere, November 2009

Ricardo Gomes

Table of Contents

1. Introduction.....	1
1.1 Background.....	1
1.2 Problem definition	2
1.2.1 JUSTIFICATION OF THE WORK	2
1.2.2 PROBLEM STATEMENT.....	3
1.2.3 WORK DESCRIPTION	3
1.3 Outline.....	5
2. Theoretical Background	6
2.1 Ontology	6
2.1.1 DEFINING ONTOLOGY	6
2.1.2 REPRESENTATION OF ONTOLOGY.....	6
2.1.3 ONTOLOGY LANGUAGES	7
2.1.4 ONTOLOGY BUILDING	11
2.1.5 ONTOLOGY LIFE CYCLE	12
2.1.6 ONTOLOGIES IN THE FACTORY AUTOMATION DOMAIN.....	12
2.2 Control Perspective.....	14
2.2.1 OVERVIEW	14
2.2.2 CONTROL NEEDS.....	15
2.3 Multi-Agent Systems.....	16
2.3.1 OVERVIEW	16
2.3.2 STATE-OF-THE-ART ON MULTI-AGENT SYSTEMS DEVELOPMENT METHODOLOGIES	16
2.3.3 APPLICATIONS IN THE FACTORY AUTOMATION DOMAIN	20
2.4 Agent Platforms	21
3. Manufacturing System Control Solution	24
3.1 Functional View of the Proposed Solution.....	24
3.2 Technical Architecture	24
3.3 The Physical Scenario.....	25
3.4 Manufacturing System Ontology	30
3.4.1 GENERIC DEVICE ONTOLOGY	31
3.4.2 CONVEYOR ONTOLOGY.....	35
3.4.3 DIVERTER ONTOLOGY.....	37
3.4.4 TOOL ONTOLOGY	38
3.4.5 ROBOT ONTOLOGY.....	39
3.4.6 SENSOR ONTOLOGY	44
3.4.7 THE STOPPER ONTOLOGY.....	48

3.4.8	PRODUCT ONTOLOGY.....	48
3.4.9	PALLET ONTOLOGY.....	49
3.4.10	PRODUCTION SYSTEM ONTOLOGY.....	51
3.5	Multi-Agent System.....	58
3.5.1	COMMUNICATION BETWEEN AGENTS.....	59
3.5.2	LAUNCHER AGENT.....	63
3.5.3	TRANSPORT AGENTS.....	65
3.5.4	ROBOT AGENT.....	73
3.5.5	PALLET AGENT.....	75
3.5.6	EXIT HANDLER AGENT.....	78
3.5.7	SENSORIAL AGENT.....	79
3.5.8	MONITOR AGENT.....	81
3.6	Interface to physical controller.....	83
3.7	The Solution Testbed.....	86
4.	The Testbed Results.....	87
5.	Conclusions and Future Work.....	95
6.	Bibliography.....	97

List of Figures

Figure 1: Conceptual overview of the solution	2
Figure 2: OWL layers of expressivity, modified from (Martínez Lastra, Delamer and Ubiz Lopez 2007)	8
Figure 3: Example of a SPARQL query	11
Figure 4: Example of a nRQL query, adapted from (G. &. RacerSystems 2007).....	11
Figure 5: Classical approach to production control - Hierarchical, adapted from (Bussman, Jennings and Wooldridge 2004).....	14
Figure 6: Goal-Driven approach to production control - Cooperative, adapted from (Bussman, Jennings and Wooldridge 2004).....	15
Figure 7: The Gaia model, adapted from (Bussman, Jennings and Wooldridge 2004)..	18
Figure 8: Use Case diagram	24
Figure 9: Solution Architecture	25
Figure 10: Manufacturing system's schematic	26
Figure 11: Manufacturing system's picture at the FAST Lab Manufacturing System ...	26
Figure 12: Service Conveyor.....	27
Figure 13: ByPass Conveyor	27
Figure 14: InterCell Conveyor.....	28
Figure 15: Diverter.....	28
Figure 16: SCARA Robot	29
Figure 17: Pallet.....	29
Figure 18: Class Taxonomy of the Generic Device Ontology	32
Figure 19: Properties from Generic Device Ontology	33
Figure 20: Restrictions of Types class belonging to the Generic Device Ontology.....	34
Figure 21: Class Taxonomy from Conveyor Ontology	35
Figure 22: Added properties of the Conveyor Ontology	36
Figure 23: Added restrictions of Types class belonging to the Conveyor Ontology.....	36
Figure 24: Restrictions of CarryDimensions class from Conveyor Ontology	36
Figure 25: Properties and Skills individuals of the Conveyor Ontology	37
Figure 26: Class Taxonomy of the Diverter Ontology	38
Figure 27: Properties and Skills individuals belonging to Diverter Ontology	38
Figure 28: Class Taxonomy of the Tool Ontology	39
Figure 29: Properties and Skills individuals of the Tool Ontology	39
Figure 30: Properties of the Robot Ontology	40
Figure 31: Restrictions of the Arm class from the Robot Ontology	40
Figure 32: Class Taxonomy from Robot Ontology	41
Figure 33: Restriction of the Scara class of the Robot Ontology	41
Figure 34: Restrictions of Link class of the Robot Ontology	42
Figure 35: Restrictions of Joint class of the Robot Ontology	42
Figure 36: Properties and Skills individuals of the Robot Ontology.....	43

Figure 37: Class Taxonomy of the Sensor Ontology	45
Figure 38: Inductive sensor class restrictions of the Sensor Ontology.....	46
Figure 39: Properties and Skills individuals of the Sensor Ontology.....	47
Figure 40: Properties and Skills individuals of the Stopper Ontology	48
Figure 41: Class Taxonomy from the Product Ontology.....	49
Figure 42: Class Taxonomy belonging to the Pallet Ontology	49
Figure 43: Added properties of the Pallet Ontology.....	50
Figure 44: Added restrictions of the Pallet Ontology.....	50
Figure 45: Properties and Skills individuals of the Pallet Ontology	51
Figure 46: Class Taxonomy of the Production System Ontology	51
Figure 47: Properties of the Production System Ontology	52
Figure 48: Restrictions of Production System class of the Production System Ontology	52
Figure 49: Restriction of Types class added to each device's ontology to represent its attached devices, from the Production System Ontology	52
Figure 50: Restriction to all Types classes of the devices belonging to the Production System Ontology	53
Figure 51: Restriction to conveyor Types class of the Production System Ontology	53
Figure 52: Restriction to diverter class of the Production System Ontology.....	53
Figure 53: Individuals of the Production System Ontology.....	54
Figure 54: Referential axis of the physical system.....	55
Figure 55: Class Taxonomy of the control from the Production System Ontology	56
Figure 56: Properties for the control class of the Production System Ontology	56
Figure 57: Restrictions for the control class that belongs to the Production System Ontology.....	56
Figure 58: Control individuals of the Production System Ontology	57
Figure 59 Schematic of the devices and respective controlling agents	57
Figure 60: UML Class Diagram of the Ontologies for Agent Communication.....	61
Figure 61: Activity Diagram of Launcher Agent	64
Figure 62: Query Specific Agent Information of the Launcher Agent.....	65
Figure 63: Behaviour sequence of a Transport Agent	66
Figure 64: Service Request from Service Responder behaviour, adapted from FIPA Request IP	66
Figure 65: Query free places from Query Responder Behaviour, adapted from FIPA Query-If IP	67
Figure 66: Query Shared Area from Query Responder, adapted from FIPA Query-If IP	68
Figure 67: Query pallets' information interaction from Query Responder, adapted from FIPA Query-If IP	69
Figure 68: Behaviour's Activity Diagram of Receive Informs.....	70
Figure 69: Activity Diagram of the Conveyor Agent depicting the delivery of a pallet to the following Transport agent	71

Figure 70: Behaviour's Action Diagram of the Receive Informs from agent Decision Point	72
Figure 71: Behaviour sequence of Robot Agent	73
Figure 72: Service Responder Interaction of the Robot Agent, adapted from FIPA Request IP	74
Figure 73: Receive Informs Activity Diagram of the Robot Agent	75
Figure 74: Behaviour of Pallet Agent	76
Figure 75: Get next Manufacturing Process Activity Diagram of the Pallet Agent	77
Figure 76: Receive Informs Activity Diagram of the Pallet Agent	78
Figure 77: Behaviours of Exit Handler Agent.....	78
Figure 78: Receive Informs Activity Diagram of the Exit Handler Agent	79
Figure 79: Behaviours of the Sensorial Agent	80
Figure 80: HeartBeat Signaling Activity Diagram of the Sensorial Agent.....	80
Figure 81: Get sensor info Activity Diagram of the Sensorial Agent	81
Figure 82: Activity Diagram of the Monitor Agent.....	81
Figure 83: Heart Beat Activity Diagram of the Monitor Agent	82
Figure 84: Monitoring Activity Diagram of the Monitor Agent	82
Figure 85: Interaction between agents and the physical system.....	83
Figure 86: Log Window of the Launcher Agent operation.....	87
Figure 87: Log Window of the Launcher Agent launching a Pallet Agent	87
Figure 88: Log Window of a Pallet Agent running	88
Figure 89: Content of a Service Request Message	88
Figure 90: Log Window of the ByPass Conveyor 2 Agent running.....	89
Figure 91: Log Window of the Agent Inter Cell Conveyor 2 running	90
Figure 92: Log Window of the Agent Service Conveyor 2 running 1	90
Figure 93: Log Window of the Agent Service Conveyor 2 running 2	90
Figure 94: Log Window of the Agent Decision Point running	91
Figure 95: Log Window of the Agent Decision Point 2 launching an Exit Handler Agent.....	91
Figure 96: Log Window of the Exit Handler Agent	92
Figure 97: Log Window of robot initialization of the Robot Agent.....	92
Figure 98: Log Window of a start operation of the Robot Agent.....	93
Figure 99: Log Window of an end operation of the Robot Agent.....	93
Figure 100: Log Window of Pallet agent when operations completed.....	93
Figure 101: Log Window of the Sensorial Agent running.....	94
Figure 102: Log Window of the Monitor Agent receiving messages.....	94
Figure 103: Log Window of the Monitor Agent substituting Sensorial Agent	94

List of Tables

Table 1: Comparison between reasoners Pellet and RacerPro.	9
Table 2: Comparison between query languages, nRQL and SPARQL.	10
Table 3: Comparison of ontologies.	13
Table 4: Comparison between Java-based agent platforms	21
Table 5 Device's status memory mapping area	84
Table 6 Actuating memory mapping area	84
Table 7 Sensing and Actuating memory mapping area of RFID	85

Acronyms

ATP	Agent Transfer Protocol
CORBA	Common Object Request Broker Architecture
DAML	DARPA Agent Mark-up Language
DARPA	Defence Advanced Research Project Agency
DL	Description Logics
DLL	Dynamic-Link Library
E2PROM	Electrical and Erasable Programmable Read Only Memory
FIPA	Foundation for Intelligent Physical Agents
IIOP	Internet Inter-ORB Protocol
IP	Interaction Protocol
JADE	Java Agent Development Framework
JMI	Java Metadata Interface
JNDI	Java Naming and Directory Interface
KQML	Knowledge Query and Manipulation Language
MAS	Multi-Agent System
MTP	Message Transfer Protocol
nRQL	New Racer Query Language
OIL	Ontology Interchange Language
OWL	Web Ontology Language
OWL-DL	Web Ontology Language Description Logics
PLC	Programmable Controller
QL	Query Language
RDF	Resource Description Framework
RFID	Radio Frequency Identification
RMI	Remote Method Invocation

SCARA	Selective Compliant Assembly Robot Arm
SHOE	Simple HTML Ontology Extensions
SPARQL	Simple Protocol and RDF Query Language
SODA	Societies in Open and Distributed Agent
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XML	eXtensible Mark-up Language
XOL	XML-Based Ontology Exchange Language

1. Introduction

This thesis presents a solution for modelling and controlling manufacturing systems. In respect to modelling this thesis presents an approach to create the knowledge that represents manufacturing systems, as well as its control; in terms of controlling manufacturing systems this thesis provides a solution based in a distributed, reactive, autonomous, social and proactive controller, independent of the physical manufacturing system. For the control to actuate and sense the physical system it requires an interface to use it.

In this first chapter, the description and justification of the developed work is presented and organized in three sub-chapters. First sub-chapter, background, where the motivation for the work is described, then, the second sub-chapter, problem statement, where the work is justified and stated. The last sub-chapter presents thesis outline.

1.1 Background

After 1950, the demand for products increased at a relatively high speed that led to a situation in which it was difficult to satisfy the product demand. This need for a faster product supply made the industry evolve towards a mass production paradigm, where automated production lines provided answer to the product demand at lower prices; however automation at that time was still very limited.

In the decade of 1980, the general demand for products decreased and thus it gave the possibility to raise the quality of the produced products. This led to some new concepts in the production field such as Quality Management and Process Control. In the 1990's, industries become more competitive in terms of product quality, flexibility, delivery and agility as predicted by a group of scholars from Iacocca Institute of the Lehigh University in USA in 1991. They predicted that agile manufacturing is needed to make industry competitive and that was already happening in the field (Yusuf, Sarhadi and Gunasekaran 1999).

At the beginning of the present century, industry's market changed from vendor's oriented perspective to customer's oriented. In this change, customers demanded product customization which aroused more competition between vendors. For companies to respond accordingly and do not loose competitiveness in their market some changes had to come up. Reduction of product life cycle, reduction of time to market and increase in product differentiation at reduced costs are the factors that can make companies more competitive in their market. To accomplish that, companies have

to introduce changes to current products or newer ones with a higher frequency, since these are of possible customization, they are more complex from the production point of view. Consequently, manufacturing systems need to evolve towards more adaptable manufacturing systems in order to give answer to the demand (Bussman, Jennings and Wooldridge 2004).

1.2 Problem definition

1.2.1 Justification of the work

For industries to become more competitive require high adaptability to changes in their environment and doing so makes them agile. In order to become agile, industries' products need to adapt according to customer needs or wishes and to make this possible manufacturing systems have to adapt its configuration to the changes that occur frequently. Until now, this reconfiguration is time consuming and cost ineffective due to the required amount of programming and reconfiguration of manufacturing systems (Kidd 1995). The limitations of these systems are the lack of modularity from high levels (software and control level) which make systems less flexible due to the difficulty in adapting to low level (device level) changes, and the lack of modularity from low levels make integration and upgrading of components more complex.

In order to cope with these problems, manufacturing systems should have a smaller setup time and should also permit easy integration and reusability of existing systems (Mehrabi, Ulsoy and Koren 2000). Smaller setup time, easier integration and reusability of existing systems requires the control domain to define boundaries to become modular and independent of the manufacturing system. With a specification of the manufacturing system, used to perform control, it allows the definition of a modular and independent control of the system. A system specification is the knowledge representation of the manufacturing system, therefore different systems have different specifications. The use of knowledge representation to control manufacturing systems makes integration and reusability of systems easier (Figure 1 presents it).

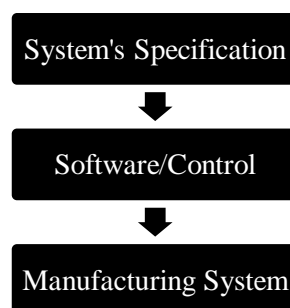


Figure 1: Conceptual overview of the solution

1.2.2 Problem statement

Currently manufacturing systems are limited in terms of providing control in a generic way, i.e., independent of the physical system. One way of providing generic control to a manufacturing system is through the use of a control that is specification-based, i.e. the control is not focused in one system, is focused in a system specification which allows the control to be applied to different systems.

This thesis provides a solution to control a manufacturing system, it ranges from the higher level, the knowledge representation of the manufacturing system to the control level where entities perform the control of the manufacturing system based on the available knowledge. However there is no such standard way for those entities to perform the control as well as a standard way for developing the knowledge representation of the system.

1.2.3 Work description

1.2.3.1 Objectives

The following objectives have been considered during the development of this thesis work:

1. Design and develop a knowledge-based representation of a manufacturing system.
2. Integrate the knowledge-base representation with the control entities of the manufacturing system.
3. Design and develop the control of the manufacturing system based on the knowledge provided by the representation of the system.
4. Integrate the controlling entities with the physical manufacturing system.
5. Assess the developed work with an automated system execution based on the knowledge representation.

1.2.3.2 Solution

The proposed solution consists of:

- A knowledge representation of each device where together represent the manufacturing system.
- Available knowledge to any decision making process of the control entity.
- Distributed and autonomous control entities applied to each device of the manufacturing system, which interact with each other to achieve the manufacturing objectives based in a provided knowledge representation.

For the task of knowledge representation, the choice of the modelling language has crucial effects for the implementation of the ontology. The language should have several characteristics such as: expressiveness, inference mechanisms, language support tool, permit knowledge exchange between applications, allow integration for

representing the knowledge through the internet, and also the existence of translators that have minimum losses. The Web Ontology Language - Description Logics (OWL-DL) was selected since it fulfils the aforementioned characteristics and therefore it is the most suitable solution for this task.

The control has to be robust, and for that a distributed option minimizes risks, since a failure affects a device and not the whole system. Consequently, the control should be assigned independently to each device, requiring interactions among them to control the manufacturing system. To achieve that, the control has to be autonomous, to sense and act on its environment, it also has to be reactive, and to trigger the system and achieve manufacturing goals, it also has to be proactive. The control that fulfils the previous mentioned characteristics is the Agent (López Orozco and Martínez Lastra 2007).

The platform that supports the development of Agents must be compliant with standards, since The Foundation of Physical Agents (FIPA) is the entity responsible for the promotion of Agent standards, the selected platform must be FIPA compliant. Java Agent Development Framework (JADE) was selected since it is FIPA compliant.

1.2.3.3 Assumptions and Limitations

In order to develop the knowledge-based representation and the controlling scheme of the manufacturing system, the concepts of scalability and re-usability should be considered. Scalability since the domain of manufacturing systems can range from small production system to a large-scale production system and re-usability to avoid repeating developed work. Regarding the controlling scheme, the concept of robustness should also be applied so that, in case of failure, the system is able to continue its normal operation by using alternative modules of control when one controlling module fails. In terms of processing capacity, the paradigm of ubiquitous computing should be considered where each controller consists of an independent and limited processing unit.

This thesis applies to manufacturing systems where each device can be controlled independently and are a separate physical module, this way device boundaries are well defined. It is worth emphasising that this thesis apply to modules of devices where device's boundaries are not well defined and the control of one device physically implies also the control of the other. Due to that, the control cannot be applied to each device therefore not allowing the control of a device to be independent of the others.

1.3 Outline

This thesis is organized into five chapters, namely Introduction, Theoretical Background, Solution, Results, Conclusions and Future Work.

Chapter 2 presents the theoretical background of the technologies used in terms of the state-of-the-art and also with the methodologies that gave support for the solution of this thesis. Chapter 3 presents the designed and developed solution, where first the modelled knowledge base representation is presented, followed by the developed control of the manufacturing system. Chapter 4 presents the obtained results regarding the case scenario in which the solution was applied. Chapter 5 presents the conclusions of this thesis and points out future directions regarding this thesis.

2. Theoretical Background

This chapter describes the paradigms, theories and methodologies used in this work and it is organized in four sub-chapters: Ontology, Control Perspective, Multi-Agent Systems and Agent Platforms.

Sub-chapter 1 presents ontology definition, formal language and several examples that apply in the manufacturing domain. Sub-chapter 2 presents an overview of manufacturing systems and manufacturing control needs. Sub-chapter 3, presents an overview and a review of the state-of-the-art of the methodologies for Multi-Agent Systems (MASs), several implementation examples of MAS in the domain, and finally existing agent platforms are presented.

2.1 Ontology

2.1.1 Defining ontology

An ontology is the formalization of knowledge. Gruber defines an ontology as “*a specification of a conceptualization*” where a conceptualization is an abstract simplified view of the world that is intended to represent (Gruber 2007). A database, a program or a conceptualization are not an ontology, reasoned by an internal sharing of some formats defined a priori and a conceptualization is only a vision or concept that is not specified thus alone is not an ontology.

An ontology is a formal representation of knowledge (Martínez Lastra, Delamer and Ubiz Lopez 2007), a body of knowledge describing some particular domain using a vocabulary representation.

2.1.2 Representation of ontology

An ontology is a representation described in an organized way by several components; these are classes, relations, functions or other objects. Humans represent an ontology through sets of declarative statements in natural language, however machines are not able to understand it and thus, it requires a formal language to allow machines to interpret it.

Specifying the vocabulary of an ontology makes the knowledge understood by both human and machines. Nonetheless, it is of relevance that an ontology is not “active” since it cannot be run as a program, it only represents the knowledge from a specific domain (Gasevic, Djuric and Devedzic 2006).

Ontology makes software more efficient, adaptive and intelligent, among some other reasons, the following are of relevance:

- Allow people or software to share knowledge from domains.
- Make domain assumptions explicit.
- Allow re-use and analyse of domain knowledge.

Ontology is not only a representational model, with the addition of reasoning and inference capabilities new knowledge emerge and is added to the domain (IBM 2006).

An Ontology can be classified into two main types, either a two-dimensional categorization based on their internal structure and the subject of conceptualization, or a categorization based on their level of dependence on a particular task or point of view. Classifications can also range in terms of the subject of conceptualization: knowledge representation ontology, general or common ontology, top-level or upper-level ontology, domain ontology, domain-task ontology, method ontology, application ontology.

General or common ontology specifies the knowledge that can be reused across domains; top-level or upper-level ontology gives a general description of concepts and provides a framework to be specialized for different domains; domain ontology specifies at a generic level tasks, activities and processes which can be reused across domains, usually as a specialization of a top-level ontology; domain-task ontology gives a specification for tasks regarding a particular application domain; method ontology specifies a problem-solving or reasoning methods to achieve given tasks and lastly application ontology gives a specification regarding application dependent concepts by extending existing upper-level and task ontologies by reusing concepts defined in general ontologies (Martínez Lastra, Delamer and Ubiz Lopez 2007). Other classifications can also be found in related literature

2.1.3 Ontology Languages

There are different types of languages and the ones used nowadays appeared with the boom of the internet and thus are called Web-based ontology languages or ontology mark-up languages. The syntax of these languages is based in eXtensible Mark-up Language (XML-W3C 1998) and the more known ones are Simple HTML Ontology Extensions (SHOE 1999), XML-Based Ontology Exchange Language (Karp, Chaudhri and Thomere 1999), Resource Description Framework (RDF) and RDF Schema (RDF-W3C 1999), Ontology Interchange Language (OIL) (Horrocks, et al. 2008), Defence DARPA Advanced Research Project Agency Agent Mark-up Language + OIL (DAML-OIL-W3C 2001) and Web Ontology Language (OWL-W3C 2004). The ones that are supported nowadays are RDF, RDF Schema and OWL.

The knowledge representation paradigms that support ontology languages were based in First Order Logics, frames combined with First Order Logics and Description Logics.

Description Logics (DL) emphasises by its logic-based semantics and computational properties for reasoning systems (Corcho, Fernández-Lopez and Gómez-Pérez 2006).

The selection of the appropriate language is of high importance to avoid problems during the development process of the ontology and during its application. Some characteristics that an ontology language must have are expressiveness, inference mechanisms, a language support tool, ontology exchange and integration for applications and web (HTML, XML), and also existence of translators with minimum losses.

2.1.3.1 OWL

The Web Ontology Language (OWL-W3C 2004) is a recommendation of the World Wide Web Consortium. OWL is based on XML, XML Schema, RDF and RDF Schema.



Figure 2: OWL layers of expressivity, modified from (Martínez Lastra, Delamer and Ubiz Lopez 2007)

XML defines the syntax structure of the document, XML Schema defines the structure and the datatype restrictions of the XML document, RDF represents the data model of objects or resources and the relationships among them, and RDF Schema gives the vocabulary to describe properties and classes of RDF resources. On the top of this hierarchy lays the OWL language which has three sublanguages that vary in terms of expressiveness, OWL-Lite, OWL - Description Logics (OWL-DL) and OWL-Full. Figure 2 presents the OWL layers and the relation with RDF and XML.

OWL-Lite is the less expressive sub-language which provides a classification hierarchy with simple constraints that permit an easier implementation of inference engines compared to other richer languages. OWL-DL adds maximum expressiveness allied to computational completeness, i.e., all the conclusions are computable guaranteed. OWL-Full like OWL-DL adds all the characteristics of the previous language subset and enhances expressiveness possibilities, however there is a price to pay for all this added expressiveness, computational complexity is highly increased (OWL-W3C 2004).

Stating this and based in the characteristics for choosing a language presented in the sub-chapter 2.1.3, the chosen language for modelling in the scope of factory automation and in this thesis is OWL-DL.

2.1.3.2 Inference Engines

An inference engine is a so called “black box” that is used to infer types and also to get new knowledge from a developed ontology, it provides consistency and taxonomy¹ checking, and it can also query the knowledge inside the ontology.

These are also known as reasoners and they complement the ontology editor with the offered operations. For the ontology to be queried it requires a query language and in the following sub-chapter two are presented.

There are several reasoners available; two examples are Pellet and RacerPro. Pellet is a free open-source Java-based reasoner, and RacerPro is a commercial lisp-based reasoner, both are OWL-DL aware engines which provide support for description logic-based query languages. RacerPro is mainly developed to provide support for its proprietary query language nRQL although RacerPro supports Simple Protocol and RDF Query Language (SPARQL) in a limited scope. Table 1 presents a comparison between the features of these two inference engines.

Table 1 was developed based on information present in (ZHANG 2005), (SPARQL-W3C 2008), (Sirin, Parsia and Grau, et al. 2007), (Clark&Parsia 2004), (G. & RacerSystems 2007) and (Haarslev, Möller and Wessel 2004).

Table 1: Comparison between reasoners Pellet and RacerPro.

Features	RacerPro	Pellet
Licence	Commercial	Open-Source
Implementation	Lisp	Java
Query Language (QL)	nRQL	SPARQL
QL Syntax	Complex	Simple
OWL-DL aware	Yes	
DL-based QL	Yes	Yes (SPARQL-DL)
Interfaces	nRQL parser	SPARQL parser
	RacerPorter	
	SPARQL (AllegroGraph)	Jena
	OWLAPI adaptor	OWL API
	OWL link (DIG interface)	
	JRacer	DIG
LRacer		

Regarding Pellet, it provides full support for the SPARQL query language which is a recommendation from the W3C; its support for a description logic-based query

¹ For the scope of this thesis Taxonomy is in respect to classification naming.

language is based on the full implementation of the subset SPARQL-DL of the SPARQL language.

Pellet is a free-open-source reasoner, it provides support for a non proprietary query language and is simple to use, these characteristics makes it the selection as the inference engine for the scope of this thesis.

2.1.3.3 Query Languages

A query language is a language used to query a database or an information system in order to get knowledge or data.

SPARQL

SPARQL (SPARQL-W3C 2008) is a query language for RDF and it is a recommendation of W3C. It has capabilities for querying, value testing and constraining queries to a RDF graph source (node graph where nodes are connected with relations), its queries consist in triple patterns, conjunctions, disjunctions and optional patterns. Besides that, SPARQL can be used across different data sources and its query results can be organised in result sets or in a RDF graph.

SPARQL-DL

The query language SPARQL-DL is an extension to SPARQL which adds the possibility to query the expressiveness of an OWL-DL source in a more satisfactory way (Sirin and Parsia 2007). This is a more expressive language when comparing to other existing description logics query languages and it is based in OWL-DL semantics. The added features associated with this query language makes it a valuable choice when deciding which query language to use since other query languages are mainly developed to query RDF graphs thus those do not satisfy when querying OWL-DL. Table 2 presents a comparison between two description logics query languages.

Table 2: Comparison between query languages, nRQL and SPARQL.

Features	SPARQL	nRQL
Proprietary	No	Yes
DL-based QL	Yes(SPARQL-DL)	Yes
QL Syntax	Simple	Less Simple
Queries	Tbox/Abox/Rbox	Abox/Tbox

As previously mentioned, nRQL is a proprietary query language for RacerPro and is a description logics-based query language (Kaplunova, Möller and Wessel 2007); however its syntax has a strong mathematical base with logical basis which makes it

less simple (Zhang, et al. 2007), on the other hand, SPARQL-DL is not proprietary and its syntax is simple.

Since the description logic-based query language SPARQL-DL is simple and not proprietary, it was selected for the scope of this thesis.

Figure 3 and Figure 4 present an example of a query to get all the siblings in a family ontology. Siblings have common parents and the parents are related with their children by means of the property *hasChild* in the ontology.

```
PREFIX fam:< http://www.owl-ontology.com/FamilyOntology.owl#>
SELECT ?x ?y
WHERE{?z fam:hasChild ?x.
        ?z fam:hasChild ?y}
```

Figure 3: Example of a SPARQL query

```
(retrieve (?x ?y)
  (and (has-child ?z ?x)
        (has-child ?z ?y)))
```

Figure 4: Example of a nRQL query, adapted from (G. & RacerSystems 2007)

?x and ?y are the resulting variables, the siblings, and ?z is an auxiliary variable used to relate the siblings by means of their parents and is not part of the result.

2.1.4 Ontology Building

A general methodology is presented to implement an ontology using OWL-DL and it consists of the following steps (Martínez Lastra, Delamer and Ubiz Lopez 2007):

1. Determination of the domain and scope of the ontology to acquire the necessary knowledge of the domain.
2. Reusing existing ontologies consist of reusing already developed ontologies on an ontology and for that several open-source communities have developed ontologies and published them online through libraries (ProtégéWiki 2006), wikis and also in a semantic web search engine (olp.dfki.de s.d.) (UMBC 2006).
3. Use reliable support for the taxonomy like standards, laws and regulations, glossaries from professional or industrial associations, technical publications, books, and also dictionaries or encyclopedia.
4. Define classes and a class hierarchy according to the third step. Class hierarchy is established through classes, sub-classes, sub-sub-classes and so on, where the OWL notation specifies that the top-level classes are the super-classes and any class that has a sub-class is also defined as a super-class; classes that are at the same level are considered siblings.
5. Define classes properties. Properties introduce the required expressiveness for the reasoning capabilities, since it creates relations between classes and

datatypes. Properties are inherited to subclasses and in OWL there are two types of properties: object properties and datatype properties. Object properties relate individuals from classes and datatype properties relate individuals from a class to a datatype value. Domain and range of a property define the classes in which individuals are linked. The individuals *from* a class are specified in the domain classes and the individuals *to*, are indicated by the range classes. Object properties have also other characteristics like functional, functional inverse, inverse and transitive. Datatype properties can be functional or not, the range of datatype properties specify the datatype value and in terms of restrictions the allowed values that apply to this type of properties is the definition of the allowed values for the property. Properties can also be organized in a hierarchy and in this way it enhances asserting and querying. With the addition of more sub-properties it is possible to get more detailed information from the ontology thus sub-properties permit a narrower classification.

6. Create instances and individuals which represent the objects of interest in the domain.

2.1.5 Ontology Life Cycle

Developing an ontology is a step by step process which starts by getting the knowledge from the domain of interest, then check existing ontologies from the domain taking advantage of ontology reusability. In order to formalize this knowledge the appropriate ontology language must be chosen and after development, comes the maintenance phase which consists in updating the ontology in order to be continuously used, also merging more ontologies contributes to the expansion of knowledge in the modelled domain.

2.1.6 Ontologies in the Factory Automation Domain

Currently there are some ontologies that can be used within the domain of factory automation, such as:

- DOLCE Ontology (Descriptive Ontology for Linguistic and Cognitive Engineering) is an ontology for generic engineering purpose and it can be used for the factory automation domain, although this ontology was not developed for the factory automation domain (Masolo, et al. 2003).
- The functional knowledge of manufacturing processes (Mizoguchi and Kitamura 2000).
- MASON ontology which models three concepts: entities, operations and resources (Lemaignan, et al. 2006).
- OntoMAS is intended to design modular assembly systems and is presented under the concepts of product, assembly process, assembly equipment and structure definition (Lohse, Ratchev and Barata 2006).

- MKS (Manufacturing Knowledge System) uses the concepts of process, equipment, facilities, and operational procedures to represent the domain (Pan, Tenenbaum and Glicksman 1989).
- Ontology for information exchange among controlling entities (Mönch and Stehli 2004) (Pouchard, Ivezic and Schlenoff 2000)

Table 3 presents a comparison between the previous ontologies in terms of the modelling concepts and it is possible to verify that for most of the ontologies that can model more concepts are not domain specific, i.e. the ontology was not developed focused only in the factory automation domain. Based on this, developing the knowledge makes it a very hard task. On the other side, ontologies that are domain specific, i.e. were developed for the factory automation domain have limitations, whether it lacks representing functionality, process, the system as a whole or the control associated to the physical system.

Stating this, it is required an ontology which models functionality, processes, devices, control, and through the combination of modules (devices) represent the complete system, and at the same time are domain specific, i.e. focused in factory automation.

Table 3: Comparison of ontologies.

Represent	Functionality	Process	Device	Modular	Domain specific	Domain general	Modules together represent system as a whole	Control
DOLCE	yes	yes	yes	Yes	no	Numerous	yes	yes
OntoMAS	no	yes	yes	Yes	yes	No	no	no
Mizoguchi and Kitamura 2000	yes	no	yes	Yes	yes	No	no	no
MASON	yes	yes	yes	Yes	no	Yes	yes	yes
MKS	no	yes	yes	yes	yes	No	no	no
Mönch and Stehli 2004	no	yes	limited	yes	yes	No	limited	yes
Pouchard, Ivezic and Schlenoff 2000	no	yes	limited	yes	yes	No	no	yes

2.2 Control Perspective

2.2.1 Overview

In order to cope with the adaptability of industry shop floor, the control scheme should have a plan that performs accordingly to classical methods, use a hierarchical and schedule-driven approach where the higher hierarchies provide instructions or scheduled operations to the lower and those act upon accordingly. Thus there is only minimal feedback and exchange of information at the same level in all processes, this means that when a problem occurs the entire system has to be rescheduled being detected at the next scheduling cycle. Figure 5 presents the model of the explained production control. This model only works if there are no problems during operation time, if any problem arises the controllers cannot respond, since the schedule or plan is made in an optimal way, i.e. all resource utilization is maximized to optimally to reduce costs. Thus there is no way for controllers to act upon a problem without interfering with neighbouring controllers operation, resulting in a cascade interfering effect in the hierarchy.

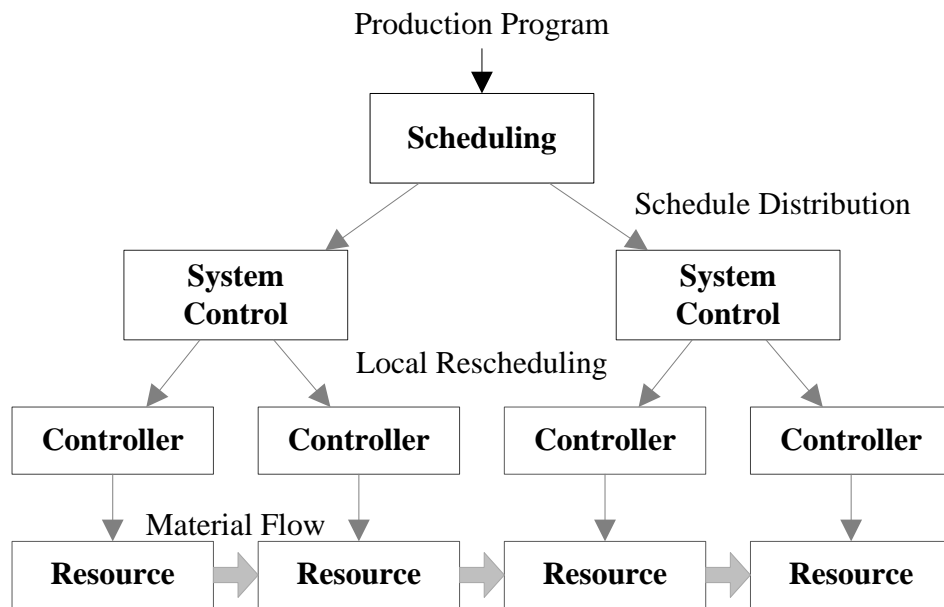


Figure 5: Classical approach to production control - Hierarchical, adapted from (Bussman, Jennings and Wooldridge 2004)

An alternative production control model consists in giving more autonomy to the device controllers, hence controllers are able to make better decisions according to the present state, in case of failure they can also act upon due to their autonomy. This approach is based on goal-driven production control in which each controller acts upon accomplishing its assigned goal. To accomplish goals, controllers have to co-operate with others in order to achieve the production goal. Since controllers are autonomous, there is a need for distributing the control since it is not centralised anymore. The main

schedule can be divided by into sub-schedules and deliver them to each controller unit. Figure 6 presents the model of the production program explained before.

This model not only provides a scalable control but also makes the system robust to failures and easily adapt to changes. As simply as changing the goal of the production plan the system changes accordingly and make it evolve in the right direction. Thus, applying this scheme in the industrial level provides companies the capability of having a flexible production system that supports smaller product life cycles and highly customized products.

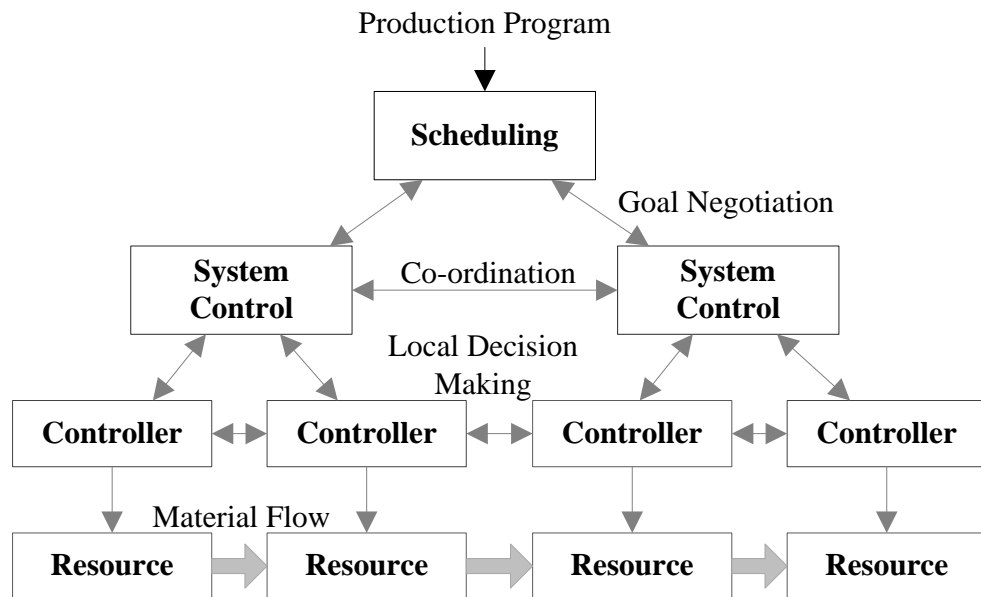


Figure 6: Goal-Driven approach to production control - Cooperative, adapted from (Bussman, Jennings and Wooldridge 2004)

2.2.2 Control Needs

The previous production control model, requires being distributed and not centralised, cope with environment changes and act upon, interact proactively with other controllers to achieve goals. Listing these characteristics results:

1. Distributed
2. Reactive
3. Autonomous
4. Proactive
5. Social

A controller with the aforementioned characteristics is an Agent (López Orozco and Martínez Lastra 2007). Proactive stands for its own initiative in performing operations, reactive for its reactions to environment events, social because it has the capability of interacting with other agents and autonomous since it manages to achieve its own goals.

2.3 Multi-Agent Systems

2.3.1 Overview

Computer hardware developed significantly in the last decade at reduced price came along. Due to these facts and specially the price factor, computing has gone into places and devices that were unthinkable just a few years ago, thus we are living towards a new computing paradigm, the ubiquitous computing which is characterized by processing capacity everywhere. With processing capacity everywhere, the possible abstraction level is higher on control systems, making them become more *intelligent*. Of relevance is that due to the ubiquitous paradigm, each device has its own controller with processing capacity which leads to a distributed way of computing (Becta 2007).

One thing that needs to be changed towards the future is the machine-oriented view of programming to a more human abstract view, like concepts and metaphors since having a good evolution in terms of hardware is not enough, it is necessary to use software paradigms that go in the same direction. As computer systems evolve, there is a need for them to be controlled towards our interests while interacting with other systems or users. However, computer hardware has evolved towards distributed systems but implementing the previous paradigms is still not a simple task. Joining these two concepts means that systems and users with different goals need to interact in order to achieve goals. One way of doing it is by means of cooperation and agreements, the same way we, humans, do it in everyday life.

Tools for developing distributed computing are already developed but tools that model our interest and allow for interaction with other systems or users in a distributed environment are a relatively recent research topic and are known by Multi-Agent Systems.

A Multi-Agent System is an environment in which agents interact with each other in order to achieve goals. These interactions are typically message exchange supported by a computer network infrastructure. To achieve goals, agents in this environment have to cooperate, coordinate and negotiate with other agents similarly to the way humans do it (Wooldridge 2002).

As explained before, agents are reactive, autonomous, proactive and social, the issue is how to develop a control with agent characteristics (Bussman, Jennings and Wooldridge 2004).

2.3.2 State-of-the-art on Multi-Agent Systems Development Methodologies

There are several proposed methodologies within the scope of Multi-Agent Systems, however only some of them are relevant for the development of this thesis.

Some of the identified methodologies, relevant for the scope of this thesis, are the methodology of Kinny and Georgeff, the GAIA methodology and its extensions, and an Agent-Oriented methodology by Elammari and Lalonde due to the representation of the concepts of roles, responsibilities, services and interactions.

2.3.2.1 Kinny's and Georgeff methodology

Analysing responsibilities leads to the identification of services which agents can provide and the authors proposed a methodology divided in internal and external perspectives.

The external perspective models purpose, responsibilities, services and interactions of an agent; the internal perspective is based on specific agent architecture and models beliefs, goals and plans.

The external outlook is composed by two models that are independent of the architecture used for the internal outlook.

- An *agent model* - describes the hierarchical relationship among different abstract and concrete agent classes.
- An *interaction model* - describes the responsibilities of an agent class, the services it provides, the interactions it engages in, and the control relationships between the agent classes.

The steps towards the analysis of external models are:

1. Identify roles in the application domain and elaborate an agent class hierarchy.
2. For each role, identify the associated responsibilities and the services provided to fulfil those responsibilities and decompose agent classes to the service level.
3. For each service identify the interactions associated with the provision of the service, the speech acts required for those interactions and their information content.
4. Identify events and conditions to be considered, actions to be performed and other information requirements.
5. Determine the control relationships between agents.
6. Refine the agent hierarchy and the control relationships.

Agent identification is guided by the identified roles, however these are not completely defined before the roles have been decomposed to the service level. Hence system structure can be optimized due to the reorganization of the agents at the service level. During or after this phase, each agent is modelled in terms of the goals it is intended to achieve, the beliefs it may adopt and the plans to achieve the goals (Kinny and Georgeff 1997).

2.3.2.2 The GAIA methodology

The GAIA methodology proposes an analysis and design of agent systems based on the abstraction of roles and responsibilities, it also considers social and agent levels without assuming any specific agent architecture.

The agent based system is modelled in terms of agent roles where each role is characterised by three attributes: responsibilities, permissions, and protocols. The responsibilities of a role define its functionality or what it is supposed to do; the permissions of a role define the allowed resources to carry out a role, and the protocols of a role define the interaction of agents in order to accomplish their goals.

To create the role models, first identify the roles in the system, then for each role identify and document the associated protocols and finally develop the role models based on the protocol model. Afterwards, the created models are transformed into three types of models: an agent, a service and an acquaintance model.

The agent model defines agents by associating roles and creating an agent-type hierarchy. The service model specifies the services of each role and also its properties such as preconditions and results. The acquaintance model defines which agents communicate directly to each other.

To create an agent model, it is necessary to associate roles into agent types and refine them to form an agent-type hierarchy, then document the instances of each agent-type. To create a services model, it is necessary to examine protocols and, safety and liveness properties of roles. Finally, create an acquaintance model based on the interaction and the agent model (Wooldridge, Jennings and Kinny 2005). Figure 7 presents the models of the Gaia methodology.

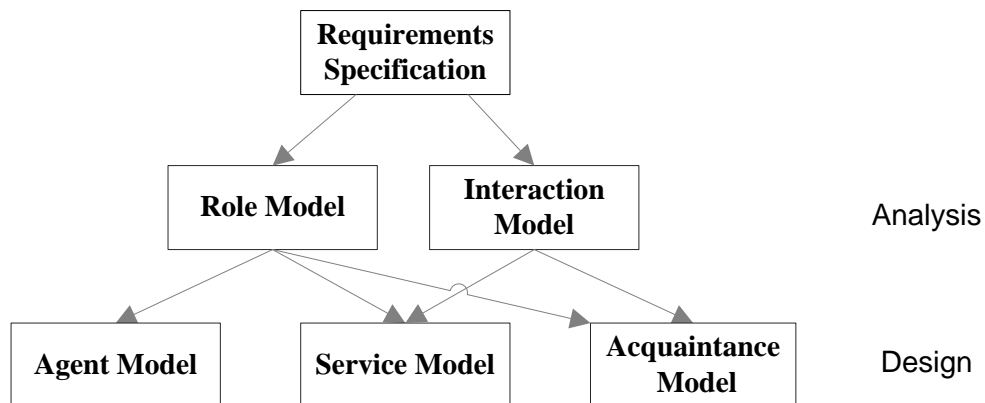


Figure 7: The Gaia model, adapted from (Bussman, Jennings and Wooldridge 2004)

2.3.2.3 Extension of the Gaia Methodology

The Gaia methodology does not specify how to model social roles, other methodologies do it, due to that the present methodology complements the Gaia methodology. Societies in Open and Distributed Agent (SODA) spaces methodology introduces social

roles and assigns tasks to individual roles and therefore to single agents, for social tasks a group of agents is assigned to it. Agents in the group play a social role and interact in order to achieve the social task. The complete methodology can be found at (Juan, Pearce and Sterling 2002).

2.3.2.4 Elammari and Lalonde, an Agent Oriented methodology

Elammari and Lalonde propose a methodology that ranges from high-level to implementable models. This methodology is composed by two phases, discovery and definition, and it generates five models:

1. High-level model – identifies agents and their high-level behaviour.
2. Internal agent model – describes an agent internal behaviour and structure.
3. Relationship model – captures dependencies and jurisdictional relationships.
4. Conversational model – describes the co-ordination between agents.
5. Contract model – defines a structure for commitments between agents.

The discovery phase provides a high-level model that identifies roles and responsibilities, it provides as well a view with the causal sequences of the system which allow the identification of active roles in the problem description.

The remaining four models are created in the definition phase. The internal agent model defines the internal structure of a model in terms of goals, beliefs, plans and tasks. The relationship model describes the relationships between agents and consists in two sub-models:

- Dependency diagram – an agent provides a service requested by other.
 - Goal dependency – an agent depends on other to achieve a certain goal
 - Task dependency – an agent requires other agents to perform a task
 - Resource dependency – an agent depends on another that provides a resource
 - Negotiated dependency – an inter-agent negotiation is required
- Jurisdictional diagram – describes the authority status of agents with respect to other agents.

The conversational model defines the necessary messages to be exchanged between agents for the relationship model, i.e., the exchanged messages for the dependency and jurisdictional diagrams.

The contract model defines the obligations and authorizations among agents. A contract specifies participants, authorisations, obligations, beliefs and policies, thus it helps agents defining their expectations about their relationships with other agents (Elammari and Lalonde 1999).

2.3.2.5 Conclusions

The concept of roles or responsibilities is present in all methodologies, as well as the concept of communication or interaction among agents which is justified since agents

exist inside a society and depend on the activity of other agents to achieve goals whether as a server, provider or both.

The solution that this thesis provides, in respect to the Multi-Agent System includes the concept of agents as service providers from the methodology of Kinny and Georgeff where agents have specific roles; the concept of roles and responsibilities from the Gaia methodology which add the concept of responsibility to an agent; the concept of dependency and interaction from the methodology of Elammari and Lalonde since agents depend on other agents to achieve goals (ex: to perform a task, to acquire a resource), and require a way of interaction to reach other agents, by negotiation.

2.3.3 Applications in the Factory Automation Domain

Several Multi-Agent System methodologies have already been developed this section presents some application cases in which MAS were applied in order to perform control within the factory automation domain.

The methodology named designing agent-based control systems, consists in a MAS applied to control the blackboard welding shop; it assembles through the welding of sheets the blackboard of the truck driver's cab. This methodology was implemented at the DaimlerChrysler truck plant at Worth, Germany (Bussman, Jennings and Wooldridge 2004).

The methodology Coalition Based Approach for Shop floor Agility consists in a MAS architecture based on the concepts of collaborative organizations that give support for evolvable assembly systems. Its agent architecture is composed by resource agents, coordinator agents, cluster manager agents and broker agent, and this approach has been implemented in the NovaFlex manufacturing system (Barata, Camarinha-Matos and Onori 2005).

The Actor-Based Assembly System (ABAS) is a methodology characterised by the unitary actor, the assembly actor. From the point of view of a MAS each actor is an Agent. This methodology was implemented in a highly dynamic reconfigurable testbed system present at the Tampere University of Technology (Lastra and Colombo 2006).

An application of an agent system to control a real scale based prototype of chilled-water system of the US Navy can be found in (Maturana, Staron and Hall 2005).

Factory BrokerTM is a solution to Holonic Control Systems where a holon is a production equipment capable of performing manufacturing operations, and its controller has agent's characteristics. The solution is a holarchy for industrial automation where holons are functionally decomposed. The holarchy is composed by workpiece, machine, transport, loader, shift table and unloader holons, and is implemented at DaimlerChrysler, Germany, in a flexible manufacturing transfer line (Colombo, Neuberg e Schoop s.d.)

FABMAS is an Agent-Based System for Production Control of Semiconductor Manufacturing Processes, where its agent architecture is composed by resource, lot, work area, work center, preventive maintenance, monitoring and scheduler agents, and is implemented in a discrete event simulation scenario (Mönch, Stehli e Zimmermann 2004).

Another application of MAS is a simulation-based benchmarking platform based on a real test case scenario, developed at the university of Karlsruhe, where the application of agents prove that planning quality increases for a well-defined shop floor scenario. Its architecture is composed by Product, Resource, Order and Staff agents (Frey, et al. June 2003).

2.4 Agent Platforms

Multi-Agent Systems require a platform that gives support to agent life-cycle, message transportation, and also to register and look up actions. This platform should be a reference in the domain according to standards and since the promotion of standards in the agent domain is provided by the Foundation for Intelligent Physical Agents (FIPA), the chosen platform should be FIPA compliant.

Table 4: Comparison between Java-based agent platforms

Platform	Available Documentation	Open Source	Complexity of use	Support	Mobility
JADE	Good	Yes	Simple	FIPA, MTP, RMI IIOP, XML	Weak
AgentBuilder	Good	No	Complex	KQML, TCP/IP	Strong
Jack	Limited	No	Simple	FIPA	No
MadKit	Limited	Yes	Simple	CORBA	No
Zeus	Good	Yes	Complex	FIPA, KQML	No
Aglets	Good	Yes	Simple	ATP	Weak
Ajanta	Good	No	Complex	JMI, RMI ATP, XML	Weak
Tryllian	Good	No	Simple	FIPA, SOAP, XML, JXTA JNDI	Strong
Grasshopper	Good	No	Simple	FIPA, RMI, IIOP	Weak
FIPA-OS	Good	Yes	Simple	FIPA, IIOP RMI, XML	Limited

Among the platforms found, the Java-based ones provide support for standards and due to that other platforms were not considered, exception for MadKit, it supports agents developed in other languages such as Python, Scheme (Kawa), BeanShell and JESS. Table 4 presents a comparison between those platforms (Gutknecht, Ferber and Michel, et al. 2000) , (FIPA-OS 1999), (Tryllian Agent Development Kit 1998), (Nguyen, et al. 2002), (Inc 2004), (Gungui, Martelli and Mascardi 2008), (Ricordel and Demazeau 2000), (Gutknecht and Ferber 2000), (Glanzer, Hammerle and Geurts 2001), (Dale, Knottenbelt and Labo n.d.).

For the scope of this thesis, the agent platform Java Agent Development Framework (JADE) was chosen since it is open-source, supports FIPA and several other standards and protocols, it has been used in several projects, has good documentation and it is also well accepted within the community of users.

3. Manufacturing System Control Solution

This chapter presents the developed work organized in six sub-chapters: Use Case, Architecture, Physical Scenario, Ontology, Multi-Agent System, Interface to physical controller and Testbed. The Use Case sub-chapter presents the system's use case diagram; the architecture sub-chapter presents an overview of the solution's architecture; the Physical Scenario sub-chapter presents the physical scenario to the applied solution; the Ontology sub-chapter presents the developed ontology which represents the system; the Multi-Agent System sub-chapter presents the control of the system using reactive, autonomous, proactive and sociable concepts; the sub-chapter interface to physical controller presents how the MAS interacts with the physical devices and finally a testbed sub-chapter where a test to the solution is presented.

3.1 Functional View of the Proposed Solution

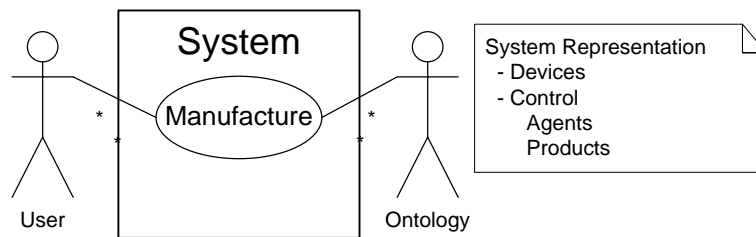


Figure 8: Use Case diagram

The proposed solution consists of a MAS control system applied to a manufacturing system based on an ontology. Figure 8 presents its use case diagram.

The solution's use case is composed by two actors and the system; the actors are a user and the ontology; the system is the MAS, the physical controllers and the manufacturing system; the user wants to manufacture products and the ontology reflects the desired products as well as a system representation in terms of devices and control which also includes a representation of the MAS.

3.2 Technical Architecture

The proposed architecture is composed of a MAS which controls the physical system based on an ontology which represents the physical system as well as its control. The physical system is composed by devices which are controlled by a centralized programmable logic controller (PLC) where each device is controlled independently.

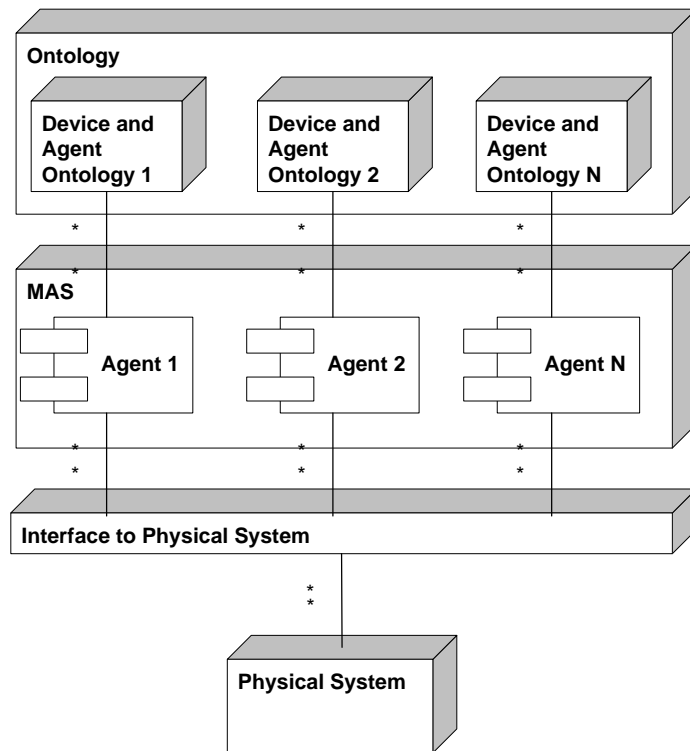


Figure 9: Solution Architecture

For the MAS to control the physical system it requires sensing and actuating, and to accomplish that an interface handles those interactions which are supported by a dedicated Ethernet network. Figure 9 presents solution's architecture.

3.3 The Physical Scenario

The physical system used for testing the proposed solution consists of two symmetric robotic cells that are connected through a conveyor system. Each cell has a conveyor that goes in its working area and another that avoids it. To connect to the other cell there is another conveyor that connects the end of the previous conveyors to the entrance of the other cell. The conveyor that goes through the cell and the one that avoids it are parallel and are connected on their entrance and exit points. Figure 10 presents the schematic of the previous mentioned descriptions of the manufacturing system and Figure 11 presents a picture of the testbed located at the FAST laboratory in the facilities of the Tampere University of Technology. These cells are mainly used for the manufacture and assembly of the automotive industry electronics and consumer products. For the specific case of this testbed cells were previously dedicated to the manufacture and assembly of covers for mobile phones.

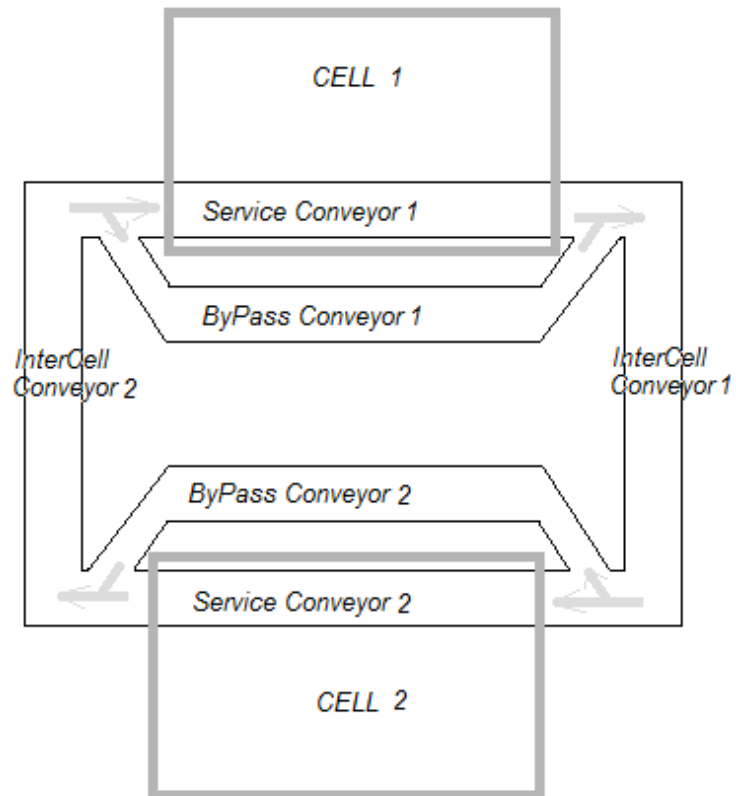


Figure 10: Manufacturing system's schematic



Figure 11: Manufacturing system's picture at the FAST Lab Manufacturing System

The Service Conveyor is presented in Figure 12 has two stoppers where each one has an RFID reader and a sensor of presence attached to.



Figure 12: Service Conveyor

A ByPass Conveyor has one stopper and one sensor of presence attached to and both are positioned close to the exit of the conveyor as can be seen in Figure 13.



Figure 13: ByPass Conveyor

An InterCell Conveyor has the same configuration as the ByPass Conveyor and is presented in Figure 14.

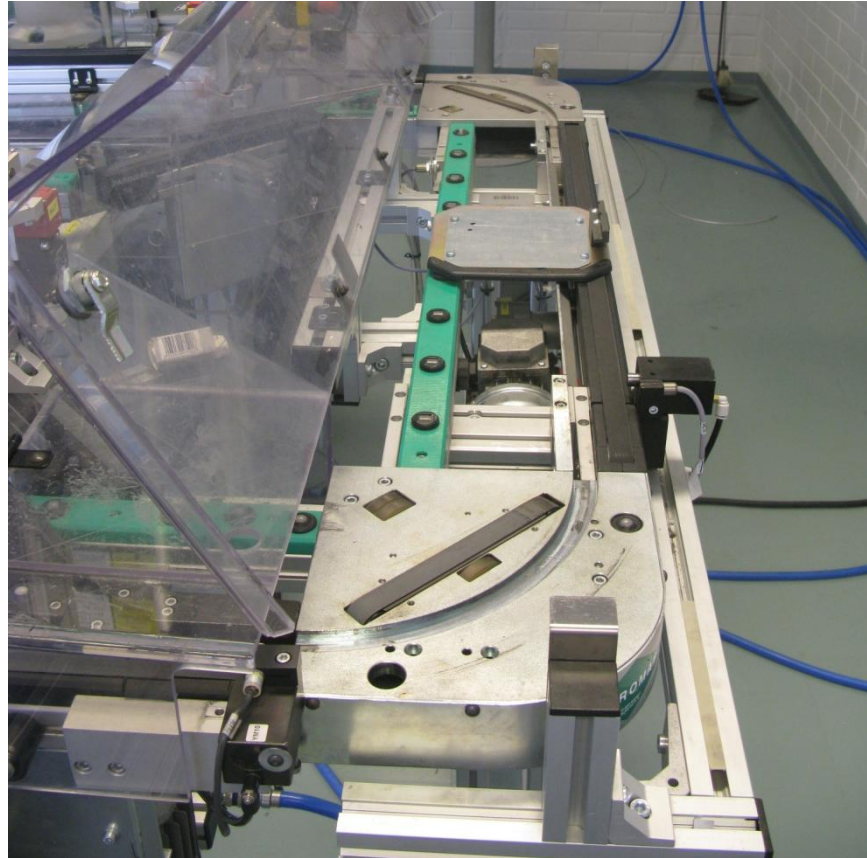


Figure 14: InterCell Conveyor

In order to route pallets allowing them to go through the Service Conveyor or through the ByPass Conveyor there is a diverter in charge of that operation. It is located at the entrance of the cell which coincides with the end of the InterCell Conveyor of the previous cell, as presented in Figure 15.



Figure 15: Diverter

Each cell has one high-speed assembly robot SONY SRX 611 as shown in Figure 16.



Figure 16: SCARA Robot

Each cell has several devices and to actuate on them directly, a centralized unit named programmable controller (PLC) is assigned for that operation.

Lastly, pallet is the device that carries the materials or products and it circulates in the system through the use of the conveyors. Figure 17 presents an example of a pallet from the testbed scenario.

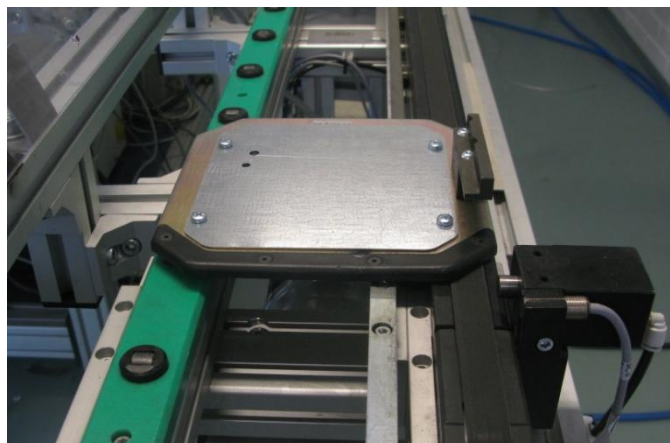


Figure 17: Pallet

3.4 Manufacturing System Ontology

A requirement to control a manufacturing system independently of the physical manufacturing system is to have a system specification and a specification-based control, i.e. a control which performs based on a system specification therefore independent of the physical manufacturing system.

In order to develop a system specification, a formal representation of the manufacturing system is required, it should be understood by both machines and humans so that automated operations can be performed, so an ontology is the best option to create the system specification.

As previously mentioned the ontology language that better suits modelling for the factory automation domain is OWL-DL due to the added expressiveness of the description logics and the supporting tools available. Therefore an ontology representing the physical system is presented in the following sub-chapters, using the tool Protégé².

Based on the requirements for developing an ontology presented in the sub-chapter 2.1.4, the ontology represents each device present in the manufacturing system and then, by joining the several device ontologies create the representation of the manufacturing system. The representation of each device contemplates also the processes and functionalities associated with it and lastly, a specification of the control entities is also appended to the ontology. Representing each device alone means that each device ontology can stand outside the scope of the manufacturing system since it represents only the device.

Before presenting the ontology it is important to review some ontology's concepts such as *property*, *restriction*, *namespace prefix*, *literal* and *individual*. Properties introduce the required expressiveness for the reasoning capabilities since they create relationships among classes and also with datatypes; restrictions specify what and the amount that can be filled in a property assigned to a class. Restriction types are:

- *allValuesFrom*, specify that all the values of the property are from a specific class however the amount is not specified.
- *someValuesFrom*, specify that individuals from the class to which the property applies has at least one individual from the class to which this property refers to.
- *hasValue* mentions directly the value of the property either an individual or a literal.
- *cardinality*, *minCardinality* and *maxCardinality* specify the amount of individuals or values, depending on property's type that the property has.

² The Protégé Ontology Editor and Knowledge Acquisition System available at <http://protege.stanford.edu>, last visit on August 2009.

Individuals are instances of classes and represent the objects in the domain of interest; literals are the datatype values that properties have; namespace prefix is used as an abbreviation of the ontology's namespace. Namespace is an URI and is used to identify an ontology, besides that it is also useful to refer to a specific ontology when is added to another and for querying purposes.

The knowledge from the domain that gave support to modelling the ontology is presented in Appendix A, and is the reference that gave support for the decision making process of choosing the right taxonomy and important knowledge to develop the ontology.

3.4.1 Generic Device Ontology

Based on domain knowledge and during devices ontology development metadata was perceived due to found equivalences, it means that same concepts apply to different devices. Therefore, the Generic Device Ontology is the base ontology to develop any device's ontology and the class taxonomy is presented in Figure 18.

Classes

This ontology is represented by the classes *Types*, *Skills* and *Properties*, as sub-classes of these, the classes *State*, *ControlOperations* and *CostOfOperation* are present under the class *Properties*. Regarding the reuse of ontologies **Dimensions** ontology, **Effector** ontology, **Location** ontology and **Unit** ontology were reused for each device's ontology. The *Unit* ontology was found at OntoSelect's website³ and the respective owl file at⁴.

The class *Types* represent all types of the device; for example, a robot can be a SCARA robot, a humanoid robot or other type, thus it is possible to differentiate them using this class where each type may have a sub-class or a sub-sub-class associated with. The class *Skills* represent the processes or operations the device is able to perform. The class *Properties* relates to all properties that describe a device. In terms of sub-classes, the class *State* relates to possible states that a device can have, the class of *ControlOperations* relates to the control operations that are available to control the device, and the class of *CostOfOperation* represents the cost associated in performing a device's operation. This is a generic cost and it is relevant since an operation has an associated cost, whether it is a monetary cost, time cost or other cost type. Other classes or sub-classes can be added in order to complete the ontology.

³ OntoSelect website - <http://olp.dfki.de/ontoselect?wicket:bookmarkablePage=wicket-0:de.dfki.ontoselect.SearchOntologies>

⁴ OWL Unit ontology - <http://www.loria.fr/~coulet/ontology/unit/version1.9/unit.owl>



Figure 18: Class Taxonomy of the Generic Device Ontology

The *Dimensions* ontology represents a device's dimension, the *Effector* ontology represents the concept of the effector of a device or in other words the device's actuator. The *Location* ontology represents the concept of location of a device and finally the *Unit* ontology represents the concept of unit to classify the required datatype values associated with an individual. The *Dimensions* ontology represents the concept with 3 sub-classes that are directly related with a three dimensional representation and those classes are height, width and length. The *Effector* ontology represents the concept through four effectors: electric, mechanic, pneumatic and hydraulic, where each is defined as a sub-class. The decision support to define these sub-classes is based on the devices present in the manufacturing system namely, electric effector devices, pneumatic effector devices, mechanic effector devices and hydraulic effector devices. In the present manufacturing system there is no hydraulic effector device, however it is highly used in heavy manufacturing systems. The *Location* ontology represents the concept of location in terms of the distance of the device to a referential. Due to that, the *Location* ontology is composed by the classes x, y and z. The remaining classes are intended to give information regarding the space orientation of the thing that is being represented.

Properties

To relate the mentioned classes or datatypes with the main instance of a device ontology properties are used. The developed properties are *hasSkill*, *hasProperty*, *hasControlOperation*, *hasCostOfOperation* and *hasState*. Each property is defined with a “has” part in its name, meaning that the individuals in which the restriction to the property applies have a relationship with other individual(s) from other class.

In order to add dimension, effector, location and unit information to a device’s instance, the defined properties are *hasUnit*, *hasLocation* and *hasLocationUnit*, *hasDimensions* and *hasDimensionsUnit*, and *hasEffectorType* where these also follow the previous approach regarding the naming. The properties *hasLocationUnit* and *hasDimensionsUnit* are related with adding unit information to respectively location and dimension’s information. The properties *hasDimensions* and *hasLocation* have sub-properties that relate each of their sub-classes so that direct properties can be associated and more meaningful knowledge can be represented. Figure 19 presents the previous mentioned properties of the generic device ontology.

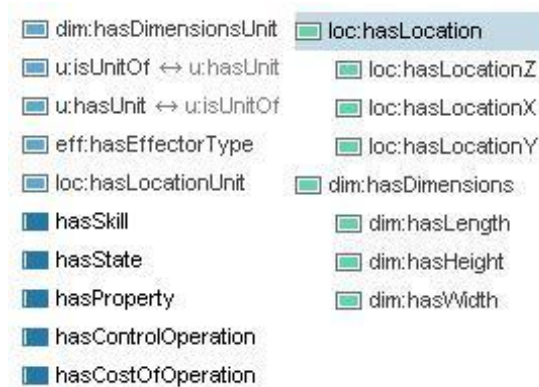


Figure 19: Properties from Generic Device Ontology

To represent the knowledge of a device, its main individual is created under the class or a sub-class of the class *Types*, then by defining restrictions to the class, properties are associated to it so that information is added to the instance (to the device).

Restrictions

The restrictions that apply to this ontology specify the properties that a generic device ontology has. Mainly these are defined under the class *Types* since that is the class in which the main instances of the device are created and thus, is where all the knowledge is associated with the main individual and where other restrictions apply in order to add knowledge to the main individual.

In order to define the skills of a device, a restriction to the property *hasSkill* with the type *someValuesFrom* is defined, i.e. the individual from that class have at least one individual from that relationship. Since skills instances are created under the class *Skills*, the target individuals of this restriction belong to its class. A device has at least

one state and to specify that, the property *hasState* is assigned to a restriction type *someValuesFrom* where the target individuals belong to the class *State*. In order to represent the operations of a device the property *hasControlOperation* is associated with a restriction type *someValuesFrom*, since a device has at least one control operation. Control operation's individuals are created under the class *ControlOperations* which is restriction's target class. In order to add other properties to the device a restriction type *someValuesFrom* is assigned to the property *hasProperty* where the target class is the class *Properties*.

The *Generic Device* ontology reuses other ontologies such as *Dimensions*, *Location*, *Effector* and *Unit* and since this knowledge is of importance to the main instance restrictions apply to add that knowledge. A device has a physical dimension and it can be represented in terms of its height, length and width, and to add that specification the properties *hasHeight*, *hasLength* and *hasWidth* are assigned to a restriction. Since a physical device is defined by each of those components and one of each component, the restriction type is cardinality one which means that for that property only one individual or literal can be assigned to it. Since these properties relate to values, unit information can be added through the property *hasDimensionsUnit* assigned also to a restriction type cardinality one where unit is the target class. Other characteristic of a device is its actuator, and to represent it the property *hasEffectorType* is assigned to a restriction type cardinality one since a device has only one actuator type. In respect to the devices present at the testbed scenario devices have actuator however devices out of the range of this scenario may not have an actuator and therefore this restriction does not apply for those cases. Finally, a device has a location and it can be represented in terms of its distance to a coordinates referential and thus the properties *hasLocationX*, *hasLocationY* and *hasLocationZ* are assigned to a restriction which type is cardinality one since only one of each specify the distance in terms of its coordinates to the referential. These properties relate to literal values, and thus unit information is added through the property *hasLocationUnit* with a restriction type cardinality one where the target class is the *Unit* class. Figure 20 presents the restrictions mentioned previously.

- ⊖ dim:hasDimensionsUnit **exactly** 1
- ⊖ dim:hasHeight **exactly** 1
- ⊖ dim:hasLength **exactly** 1
- ⊖ dim:hasWidth **exactly** 1
- ⊖ eff:hasEffectorType **exactly** 1
- ⊖ hasControlOperation **some** ControlOperations
- ⊖ hasProperty **some** Properties
- ⊖ hasSkill **some** Skills
- ⊖ hasState **some** States
- ⊖ loc:hasLocationUnit **exactly** 1
- ⊖ loc:hasLocationX **exactly** 1
- ⊖ loc:hasLocationY **exactly** 1
- ⊖ loc:hasLocationZ **exactly** 1

Figure 20: Restrictions of Types class belonging to the Generic Device Ontology

In the manufacturing domain, a cost is associated with the act of performing a control operation on a device and to represent this cost, the property *hasCostOfOperation* is assigned to a restriction type cardinality one since one operation has exactly one cost. *CostOfOperation* class is the target class of this property, its restriction is presented in **Error! Reference source not found..**

3.4.2 Conveyor Ontology

The conveyor system present in the physical system is composed by conveyor belts, due to that a sub-class *ConveyorBelt* is defined under the class *Types* which belongs to the *Generic Device* ontology. Regarding properties that characterize a conveyor it has capacity, a specification of the dimensions that it is able to carry, the directions in which it can work, the range in speed that it is able to perform transportation and also the weight that it can transport. Using the *Generic Device* Ontology as the base for developing conveyor's ontology and adding the mentioned characteristics under the class *Properties* it results in the class taxonomy of Figure 21.



Figure 21: Class Taxonomy from Conveyor Ontology

Properties

To add the knowledge of conveyor domain, it requires adding properties to its ontology; the properties are *hasCapacity* and *hasDirectionType* as presented by Figure 22. Adding these properties avoids using the more general property *hasProperty* and thus it increases the detail level of the available knowledge.

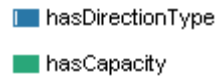


Figure 22: Added properties of the Conveyor Ontology

Restrictions

The restrictions that apply to this ontology relate directly to the previous mentioned properties and the added classes to the *Properties* class. In order to specify the capacity and the working direction of a conveyor, *Types* class has a restriction type cardinality one that applies to the property *hasCapacity* as well as for the property *hasDirectionType*. Figure 23 presents the mentioned restrictions.



Figure 23: Added restrictions of Types class belonging to the Conveyor Ontology

The sub-class *CarryDimensions* of the class *Properties* specify the dimensions which the conveyor is able to transport and to represent it *hasDimensions* properties are required, namely *hasHeight*, *hasWidth*, *hasLength* and *hasDimensionsUnit*, with a restriction type cardinality one. Figure 24 presents the mentioned restrictions of the class *CarryDimensions*.



Figure 24: Restrictions of CarryDimensions class from Conveyor Ontology

Individuals

To add more detailed knowledge to the ontology, classes can be instantiated and in Figure 25 the classes *ControlOperations*, *CostOfOperation*, *Direction*, *State* and *Skills* are instantiated to specify the knowledge of the conveyor.

The skill of a conveyor is the provision of a transport operation; the possible working directions of the conveyors present in the conveyor system is one; the cost associated to

the provision of a transport operation is not valuably specified; the available working speeds of the conveyor are two; the allowed control operations are activate or deactivate conveyor, and define one of two possible speeds; the possible states are on, off, with an error and working with a fast or a slow speed. Figure 25 presents the mentioned characteristics in terms of instance creation.

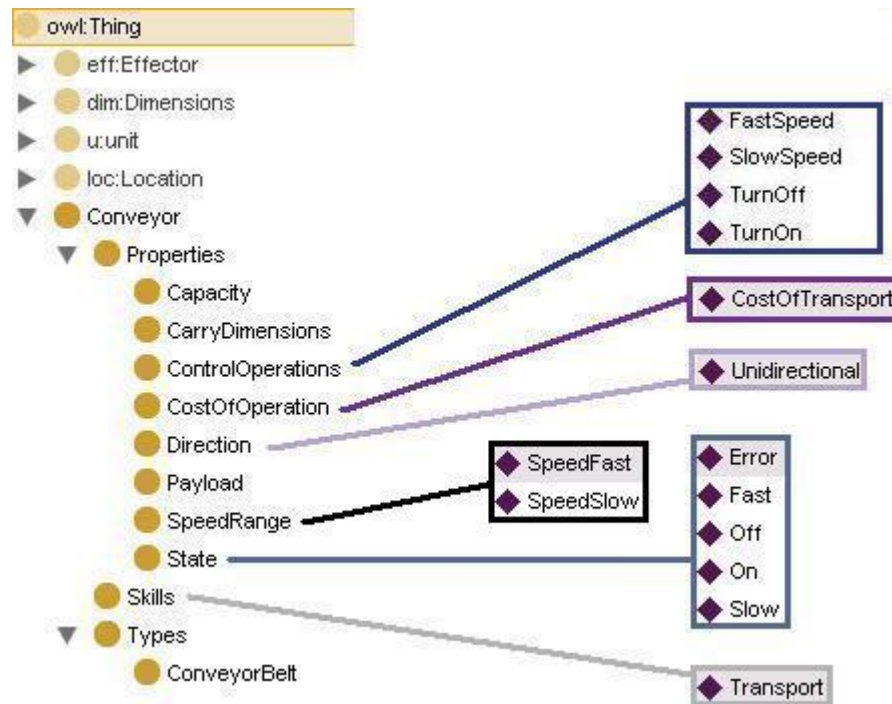


Figure 25: Properties and Skills individuals of the Conveyor Ontology

3.4.3 Diverter Ontology

A diverter is a device that can perform routing operation in a conveyor system. Since it is a device, the *Generic Device* Ontology applies to represent it as the base ontology for development. The diverter present in the physical system has two possible options of delivering pallets, one allows pallets to go in the cell and the other avoids it. Reasoned by that, its type is binary and Figure 26 presents its class taxonomy. In terms of properties, no properties are added to this ontology.

Individuals

The diverter has two different working positions, thus it has two control operations, one for each position and since the operations are symmetric there is a cost associated to both operations. In terms of device's state, it can be working or not working, in respect to its working position it can be sending pallets in the cell or by passing them, or it can also have an error. Figure 27 presents the individuals created with the mentioned characteristics.

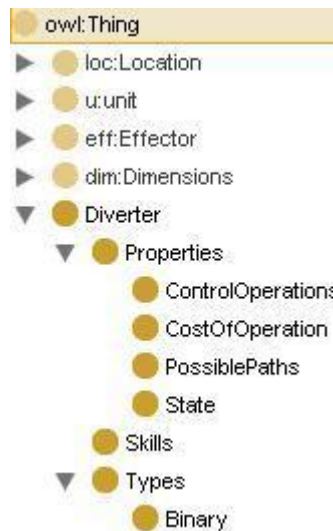


Figure 26: Class Taxonomy of the Diverter Ontology

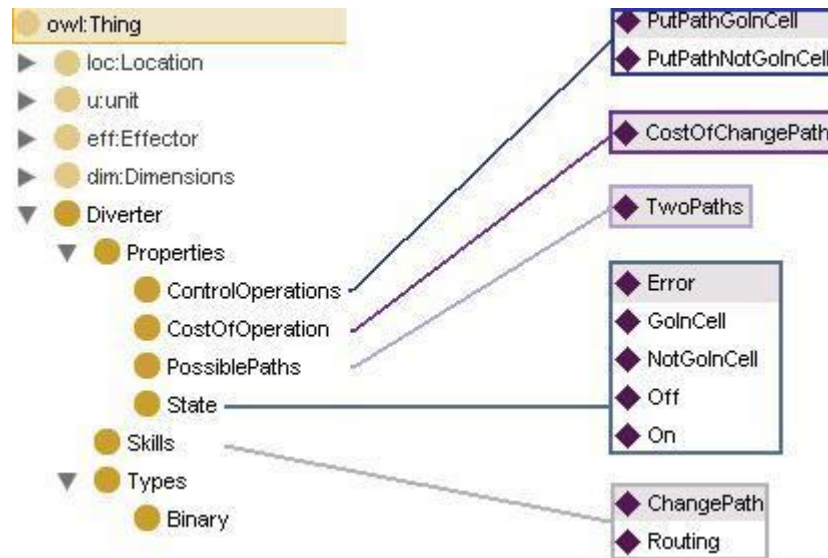


Figure 27: Properties and Skills individuals belonging to Diverter Ontology

3.4.4 Tool Ontology

A tool is a device used to perform a specific task and like the previous devices, it is developed based on the *Generic Device* Ontology. The tool present in the physical system is a gripper for a robotic SCARA arm and what it does is opens and closes its fingers. Characteristics of the tool are its opening range and the maximum pressure applied in its fingers. Figure 28 presents the mentioned characteristics.

Properties

To represent the case of a tool being attached to a device, the property *hasTool* is defined.

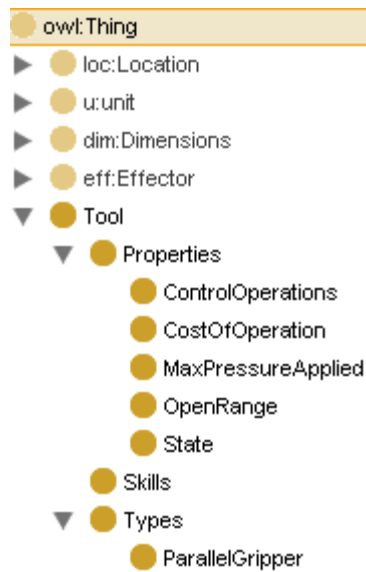


Figure 28: Class Taxonomy of the Tool Ontology

Individuals

The tool present in the physical system is a gripper and to control it, is possible to perform the operations of open and close its fingers; in terms of the cost of performing these operations a specific cost is associated to each operation. The states that characterize a gripper are activated, deactivated, closed, opened, and lastly the error state. Figure 29 the instances of the classes *Properties* and *Skills* are presented.

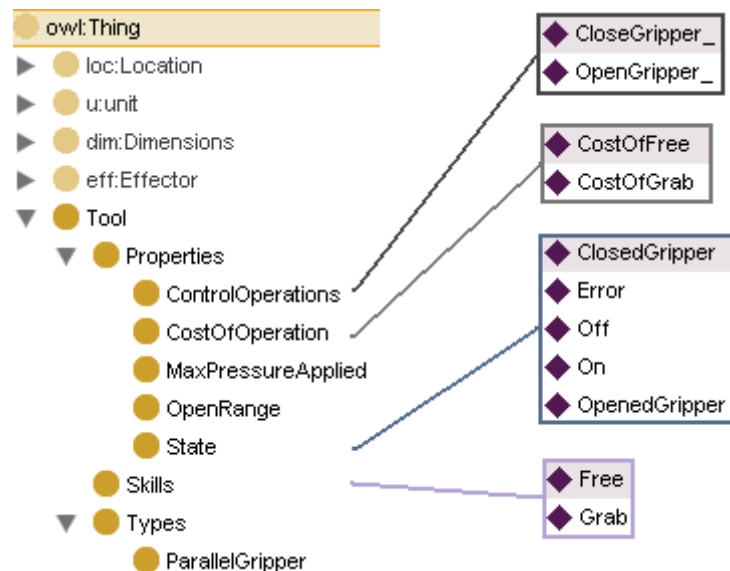


Figure 29: Properties and Skills individuals of the Tool Ontology

3.4.5 Robot Ontology

A robot is a machine designed to execute one or more tasks repeatedly, with speed and precision, therefore it can be modelled using the *Generic Device* Ontology as a developing base. The available robot in the physical system is a SCARA robot.

Robots have tools to perform specific tasks, require arms to make tools reach certain positions, have a working space, a working environment, an accuracy in terms of its moving parts as well as speed and payload, and there are also several different types of robot. Figure 32 presents the class taxonomy of the *Robot* ontology.

Properties

To relate the new sub-classes of *Robot* ontology, the properties *hasArm*, *hasJoint* and *hasLink* are defined, allowing to relate an arm to a robot or several joints and links to an arm. Figure 30 presents the previous properties.



Figure 30: Properties of the Robot Ontology

Restrictions

A manipulator robot has an arm, an arm is composed of links and joints, and it can also have a tool, its position varies compared to robot's location, therefore restrictions to properties are required to specify the previous conditions. One restriction was added to specify that a manipulator has at least one arm.

To represent that an arm is composed of at least one link, has at least one tool and has a location, restrictions apply respectively to the properties *hasLink*, *hasTool* as well as for location properties. Figure 31 presents the mentioned restrictions to properties.

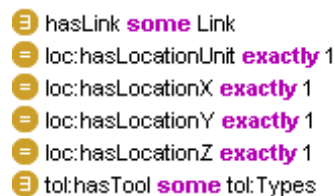


Figure 31: Restrictions of the Arm class from the Robot Ontology

As mentioned previously, the manipulator present in the physical system is a SCARA robot, is composed by four links and thus a restriction of type cardinality four is applied to the property *hasLink*.

With respect to the type SCARA robot, it is characterised by having an arm which type is SCARA, therefore restrictions mentioning these conditions are necessary under the sub-class *Scara*. The restriction type *AllValuesFrom* apply to the *hasArm* property with a target class *ScaraArm* specifying that the property only relates with individuals from the class *ScaraArm*. To specify that a SCARA robot only has one arm, the restriction type cardinality one is applied to the same property, Figure 33 presents these restrictions.

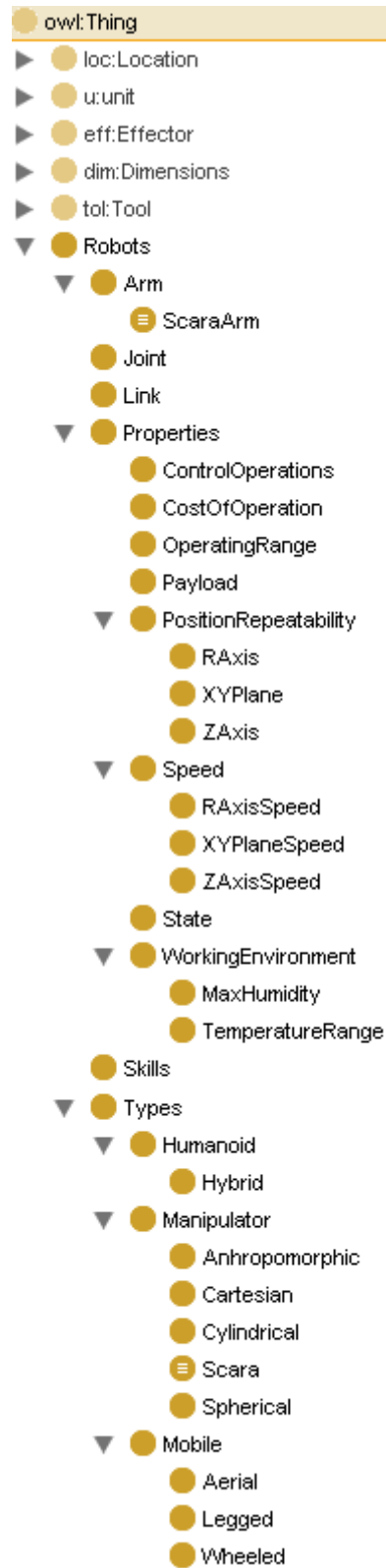


Figure 32: Class Taxonomy from Robot Ontology

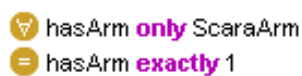


Figure 33: Restriction of the Scara class of the Robot Ontology

A link is a rigid body which maintains a fixed relationship between joints of a robot arm, thus connects at least one joint and a maximum of two. Its dimension is of relevant importance since it influences arm's size and to represent it restrictions apply to the property *hasJoint* and dimensions properties. Figure 34 presents the previous described restrictions that apply to class *Link*.

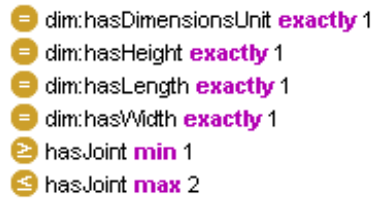


Figure 34: Restrictions of Link class of the Robot Ontology

A joint specifies the connection between two links, or between one link and a fixed base. In order to specify these conditions two restrictions of type cardinality one and two are applied to the property *hasLink* and presented in Figure 35.

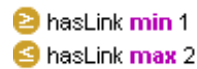


Figure 35: Restrictions of Joint class of the Robot Ontology

Individuals

Individuals of this ontology relate with the specification of robots and are two since are the available robots at the physical system. The operations of a robot are, move one determined link, move its end effector to a specified position, pick and place, and assemble parts 1 or 2. In respect to their arms, these are composed by three links and three joints and in terms of the available operations are move a determined link, move to a specified position, pick and place. To each of these operations there is a cost associated. Finally, in terms of states, a robot can be activated or deactivated, busy, performing an operation and it can also be in an error state. Figure 36 presents the individuals previously described instantiated in the respective classes.

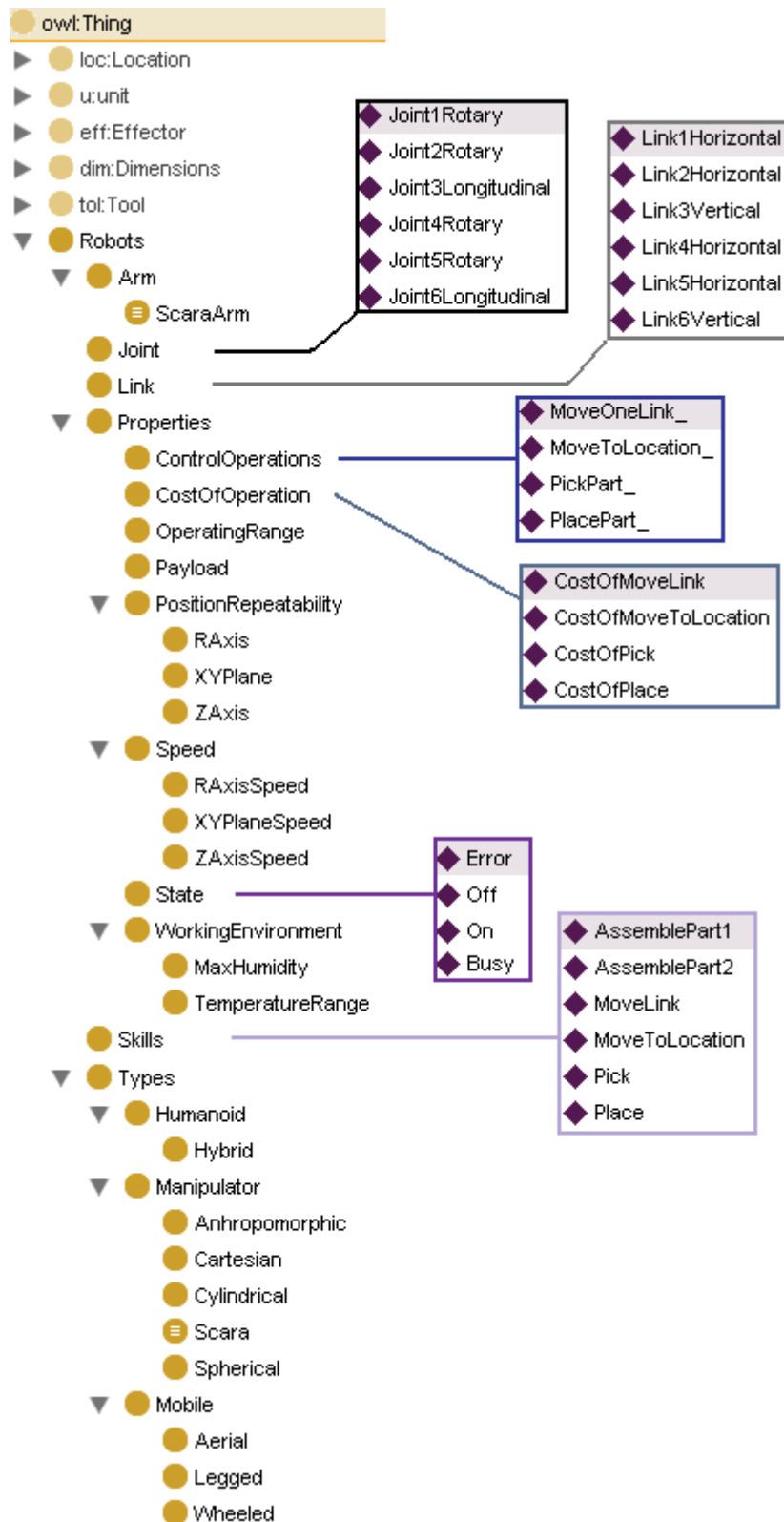


Figure 36: Properties and Skills individuals of the Robot Ontology

3.4.6 Sensor Ontology

A sensor is an electronic device that is used to measure a physical quantity. In the physical system there are presence sensors and RFID sensors, since these are devices, to create its ontology the *Generic Device* Ontology is the developing base.

As mentioned previously, the sensors in the physical system are of two types, presence or RFID sensors; the presence sensors belong to the electromagnetic type and are inductive sensors; RFID sensors are composed by a reader and a tag where the reader reads tag's information and are wireless sensors. In order to model that in the ontology, sub-classes of the class *Types* specify these different types of sensor. In terms of the skills of a sensor, in this scenario sensors can give presence information for the case of the inductive sensors, indicating if there is or not a metal piece. For the RFID sensors it can read or write a value in the tag when it is within reader's working area.

Regarding their characteristics, a sensor can be characterised in terms of different aspects such as accuracy, excitation, housing material, memory information, physical resistance, repeatability, saturation, hysteresis, transfer function, working distance, working conditions, and others. Figure 37 presents the complete developed class taxonomy of the *Sensor* ontology with the mentioned descriptions.

Properties

The properties of this ontology can be added in terms of the property *hasProperty* and due to that no other properties regarding the creation of relationships with the class *Properties* are defined. There is one added property that represents the notion of complementary part of a sensor. An example is that a sensor alone does not give any information, it requires a complementary part, for the RFID sensor is the tag; however this does not apply to all type of sensors.

Restrictions

Since an RFID reader has a complement part, a tag, it is necessary to have that represented in the RFID individuals. The restriction type *allValuesFrom* apply to the property *hasComplement* specifying that a RFID reader has complement type only Tag. In terms of skills, what it can do is read and write on a tag.

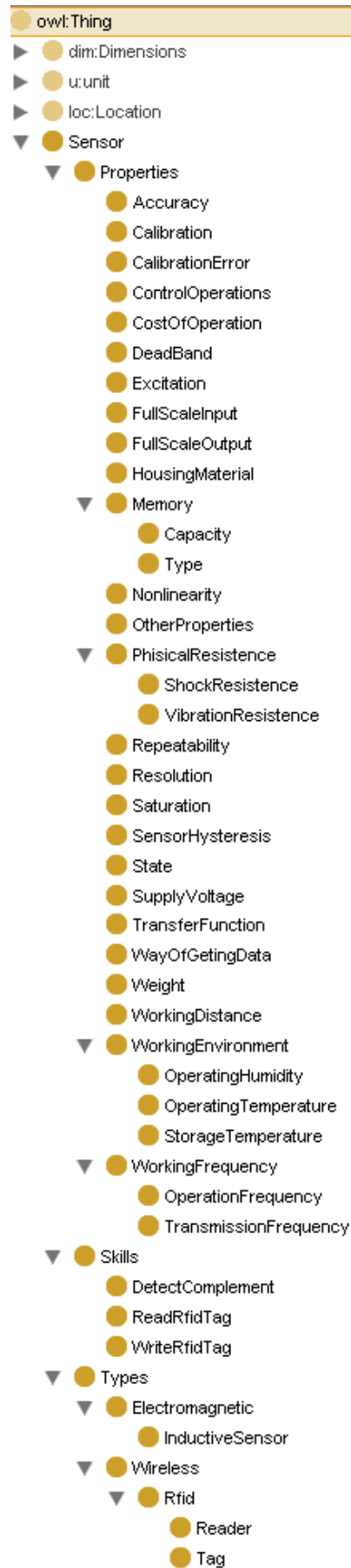


Figure 37: Class Taxonomy of the Sensor Ontology

In respect to the inductive sensors, what they do is detect a component and for that, the restriction in Figure 38 represents it. The first restriction specifies that for the *hasSkill* property it does not relate with individuals from other classes than the *DetectComplement* class, however it does not specify anything with respect to the amount of individuals that it can relate. To specify the number of individuals in which that property can relate, the second restriction specifies that the property relates with at least one individual from the *DetectComponent* class, i.e. is only related with detecting components.

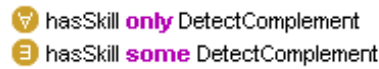


Figure 38: Inductive sensor class restrictions of the Sensor Ontology

Individuals

RFID sensors present at the physical system can perform read or write operations, presence sensors are able to detect the presence of a metal piece. The operations that control a sensor are read and write for a RFID sensor and presence checking for a presence sensor and each of these operations have a cost associated. RFID readers have E2PROM memories with 256 bytes of capacity. The states of a sensor are activated, deactivated and for the cases of error, an error state. Figure 39 presents the individuals of the classes *Properties* and *Skills* based on the previous descriptions.

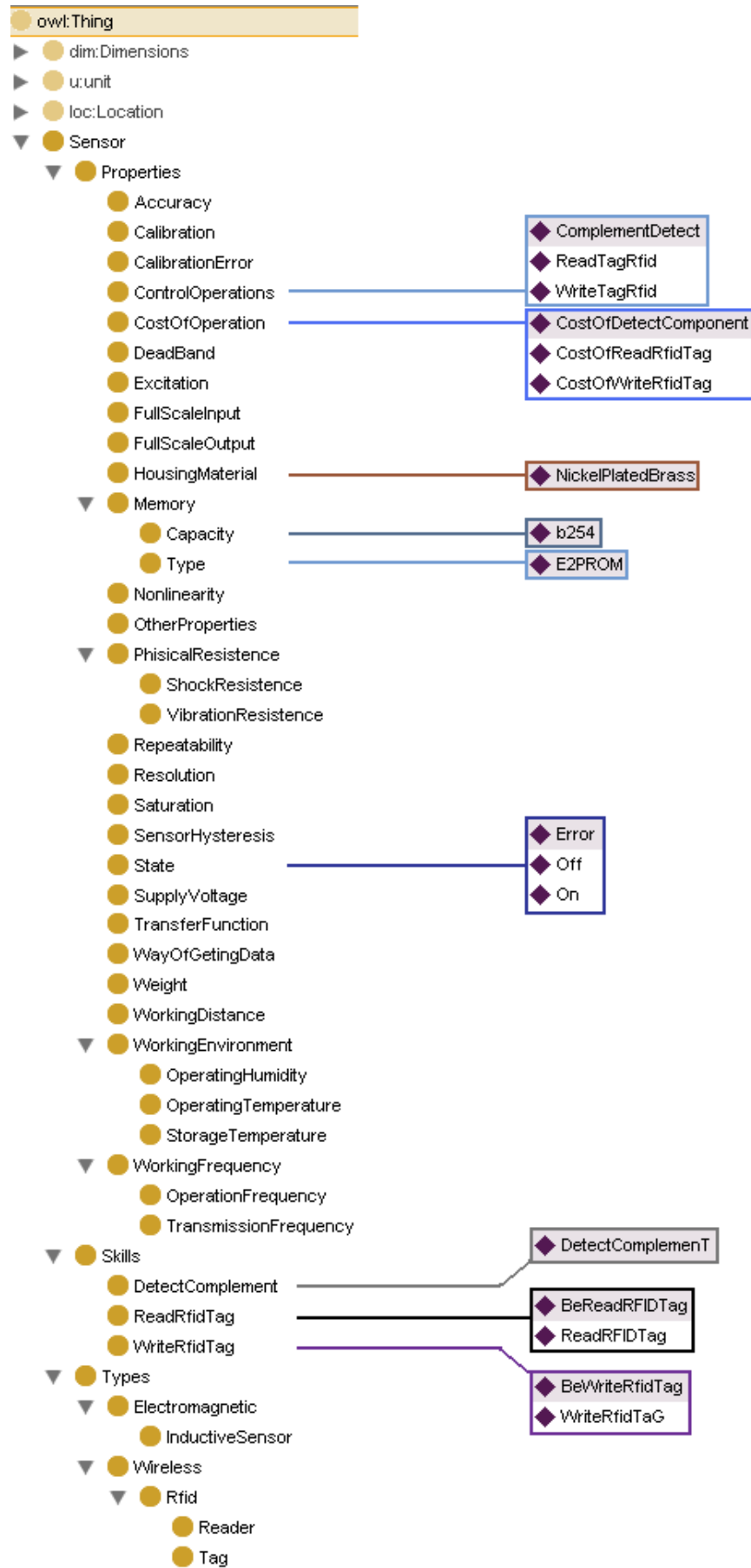


Figure 39: Properties and Skills individuals of the Sensor Ontology

3.4.7 The Stopper Ontology

A stopper is a device that is used to block pallets in a transport system. In the present physical system there is one type of stopper and when active it stops pallets, to let a pallet continue its path the stopper has to release it. Therefore, the operations that are applied to a stopper are block or unblock the stopper, where each operation is considered to have the same cost since those are symmetric. A stopper can be blocking or not blocking and thus it is a binary stopper. In terms of states, a stopper can be activated, deactivated, blocking or not blocking and also with an error. Activated or deactivated is in respect to the device being turned on or off for the cases where before using a stopper it is necessary to turn it on, otherwise blocking or none blocking is enough.

A Stopper is a device and reasoned by that it is modelled based on the *Generic Device* Ontology, class's instances are added based on the previous descriptions and are presented in Figure 40.

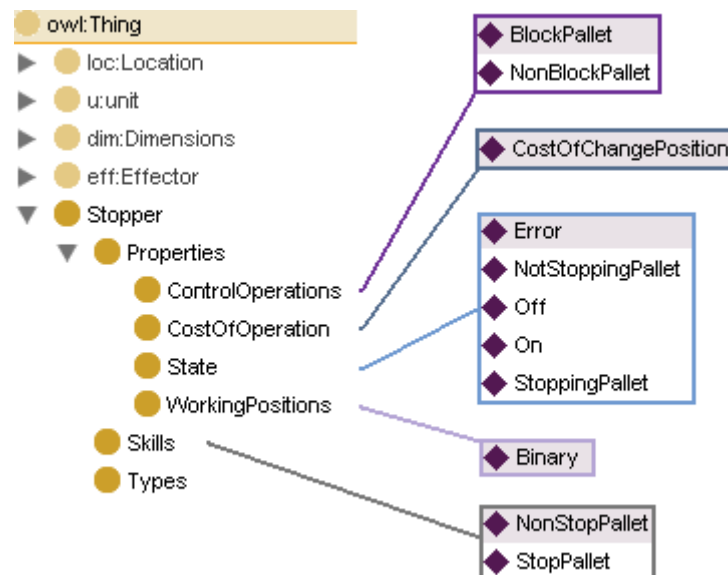


Figure 40: Properties and Skills individuals of the Stopper Ontology

3.4.8 Product Ontology

A product is what is created, assembled or developed in a production system. A product is not a device and to create its representation the complete *Generic Device* ontology is not useful for modelling this ontology, therefore only part of it is used. Figure 41 presents the class taxonomy of the Product ontology where the *Effector* ontology and the classes *Skills*, *ControlOfOperations* and *CostOfOperation* were removed. The reason for that is that a product does not provide any operations to the manufacturing system therefore its information is useless. Other classes such as *Properties*, *State* and *Types* are kept so that any product's property, state or type can be represented in the ontology. However it is of relevance to mention that the product modelled for this testbed is quite simple and therefore its representation is also quite limited.

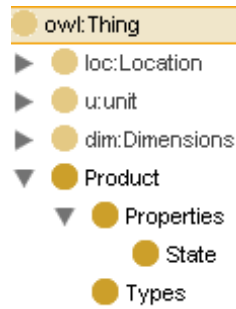


Figure 41: Class Taxonomy from the Product Ontology

Properties

In order to represent the manufacturing processes that a product requires the property *hasManufacturingProcess* allows the relationship between a product individual and manufacturing processes, since products have different priorities in a manufacturing system, priority specification is also required and for that the datatype property *hasPriority* specifies the relationship between a product's individual and a priority value.

In this ontology there are no individuals since products only exist during execution time.

3.4.9 Pallet Ontology

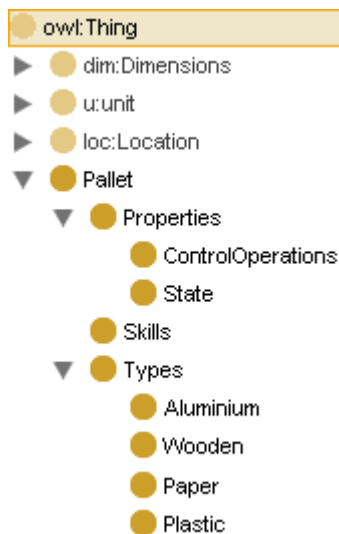


Figure 42: Class Taxonomy belonging to the Pallet Ontology

A pallet is a portable platform on which goods are placed for storage or transporting purposes, its size can vary depending on the number of goods and in terms of types, there are wooden, plastic, metal or paper pallets. A pallet is a device and thus the *Generic Device* ontology is used as the modelling base, Figure 42 presents its class taxonomy based on previous descriptions.

Properties

Pallets carry goods, its representation relate the *Pallet* class with the goods class. In this case scenario, goods are the products therefore, the property *hasProduct* is defined and used for this relationship. In the physical system, pallets have RFID tags that can be used for identification and to represent it, the property *hasRfidTagValue* is defined, it is a datatype property so that it represents RFID tag value. Figure 43 presents the added properties.

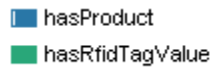


Figure 43: Added properties of the Pallet Ontology

Restrictions

In order to specify that pallets have products a restriction applies to the property *hasProduct* and, for this case scenario a pallet carries one product therefore, its restriction type is cardinality one. Pallets have a RFID tag value and to represent that the restriction cardinality one applies to the property *hasRfidTagValue*. The mentioned restrictions apply to the *Types* class and these are the added restrictions to the *Generic Device* Ontology as presented in Figure 44.

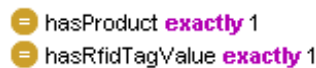


Figure 44: Added restrictions of the Pallet Ontology

Individuals

Pallets carry products or are used for storage purposes. In the physical system pallets are used to carry products to which manufacturing operations are performed, therefore pallet's skill is carry products. In terms of its states, a pallet can have or not a product and it can be in the system when is carrying a product or not for the case when it is out of the system. Figure 45 presents the instances based on previous descriptions.

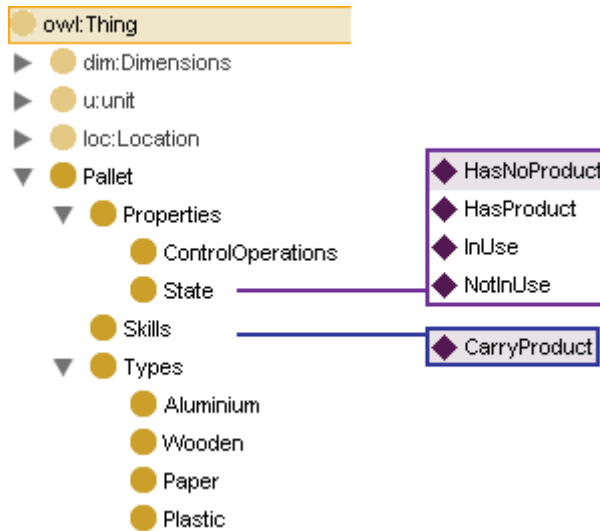


Figure 45: Properties and Skills individuals of the Pallet Ontology

3.4.10 Production System Ontology

The previous presented ontologies represent each of the devices present in a manufacturing system for the specific case of the physical scenario. Using all those ontologies in an ontology creates the representation of a production system and thus Figure 46 presents the class taxonomy of the production system ontology.

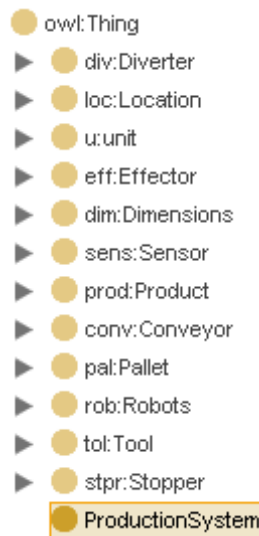


Figure 46: Class Taxonomy of the Production System Ontology

Properties

A production system has devices and for the case of the conveyors those are connected with each other forming a transport system due to that a relationship between them is required to represent transport system connections. Since a production system has several devices a unique identifier is also required so that all different devices are uniquely identified.

In order to represent that a production system has devices, the property *hasDevice* allows the relationship between *Production System* class and the respective device's class. Besides that, it is also used to represent the attached devices of a device.

Specifying the transport system for the production system consists in representing the physical connections of the transport devices and for that the property *hasConnection* represents the transport device to which it is connected to, or in other words, the transport device that is physically after it.

Finally, in order to represent the unique identifier of a device it is required to create a relationship from a device to a string so that the string uniquely identifies the device. For that, the datatype property *hasUniqueID* is defined and Figure 47 presents the previous mentioned properties of the production system ontology.



Figure 47: Properties of the Production System Ontology

Restrictions

To specify that a production system is composed of several types of devices, the property *hasDevice* is used, restricted by a restriction type *someValuesFrom* which mean that it has at least one relationship with an individual from the target class. The target class is not one but several since production systems are composed of different types of devices. Figure 48 presents the restrictions that apply to the *Production System* class.

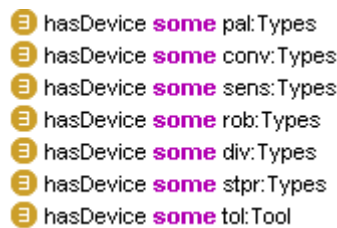


Figure 48: Restrictions of Production System class of the Production System Ontology

Several devices have other devices attached to, an example are conveyors which have stoppers and sensors attached to, and in order to represent that relationship the property *hasDevice* applies under the restriction type *someValuesFrom* where the target class is the default since any device applies specifically to it. Figure 49 presents the mentioned restriction that applies to the class Types of each device that has devices attached.

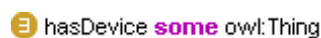


Figure 49: Restriction of Types class added to each device's ontology to represent its attached devices, from the Production System Ontology

In order to represent that devices' individuals have a unique identifier the restriction type cardinality one applies to the property *hasUniqueID* and this restriction applies to all devices ontologies. This restriction is created in *Types* class since device's main instance is created there, Figure 50 present this restriction.


 hasUniqueID exactly 1

Figure 50: Restriction to all Types classes of the devices belonging to the Production System Ontology

Conveyor devices have a physical organization representing their interconnection and to represent that in *Production System* ontology the restriction type cardinality one applies to the property *hasConnection*. Diverters are also part of the transport system since they perform routing operation, therefore it requires the specification of the following transport devices. A diverter can have several following transport devices and due to that restriction is not cardinality one but *someValuesFrom* which is also applied to the property *hasConnection*. This restriction applies to *Types* class of the respective transport device ontology. Figure 51 presents the previous mentioned restriction for the conveyor ontology and Figure 52 presents the restriction for diverter ontology.

 hasConnection exactly 1

Figure 51: Restriction to conveyor Types class of the Production System Ontology

 hasConnection some owl:Thing

Figure 52: Restriction to diverter class of the Production System Ontology

Individuals

In this ontology, device's individuals represent the devices present in the physical system, these are associated with the production system individual through the addition of device's individual to the property *hasDevice* of production system individual.

As mentioned in chapter 3.3 - The Physical Scenario, the system is symmetric where each cell has the same devices. Each cell has three conveyors, one service conveyor that goes in the working area of the cell, one that avoids it called bypass conveyor and another that connects to the other cell named inter cell conveyor; it has also a diverter, a robot, and several sensors and stoppers. A robot has an arm and an arm has a tool so that it performs operations. For the system to have a flow of goods it needs pallets which can carry products in the system. Figure 53 presents the individuals created within the production system ontology according to the physical devices.

When creating the individuals of the main devices it is necessary to fill in the properties that characterized the device. With respect to conveyors, a service conveyor has two physical modules attached to it, composed by stopper, presence sensor and RFID reader; a bypass conveyor has one physical module composed by stopper and presence sensor, as well as for an inter cell conveyor, the diverter has one module composed by

stopper and RFID reader. For robots to perform operations on pallet's products it requires a way to know when and which pallet is there since pallets have RFID tags robots have RFID readers to get pallets information.

The physical modules in the different configurations present in the physical system are modelled as physically in the same location. Due to that, location's properties are filled with the same values. Conveyors' location is considered to be its beginning point to perform the transport, for others devices its centre is considered.

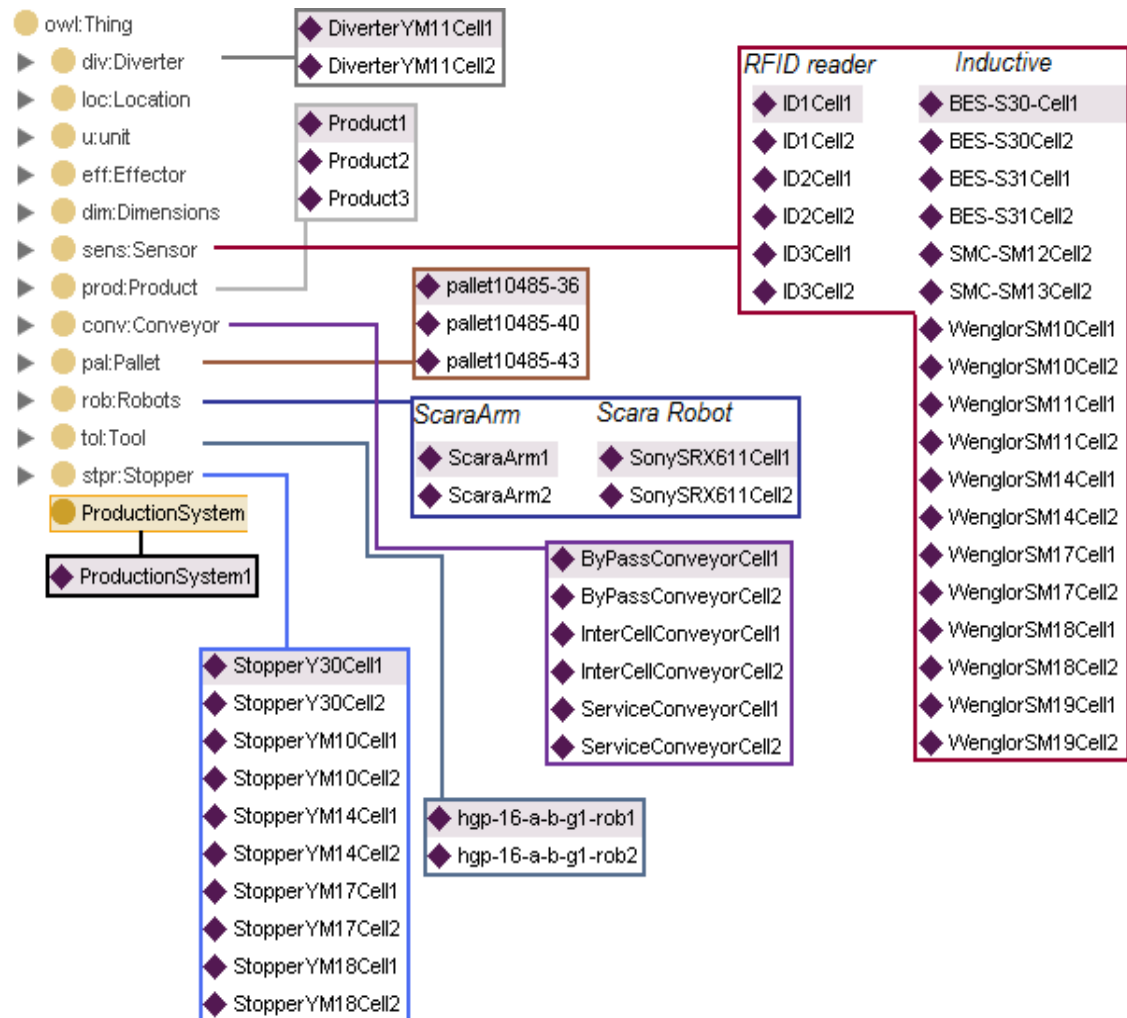


Figure 53: Individuals of the Production System Ontology

In order to create the representation of the location it is necessary to specify its referential axis in the physical system, Figure 54 presents its position.

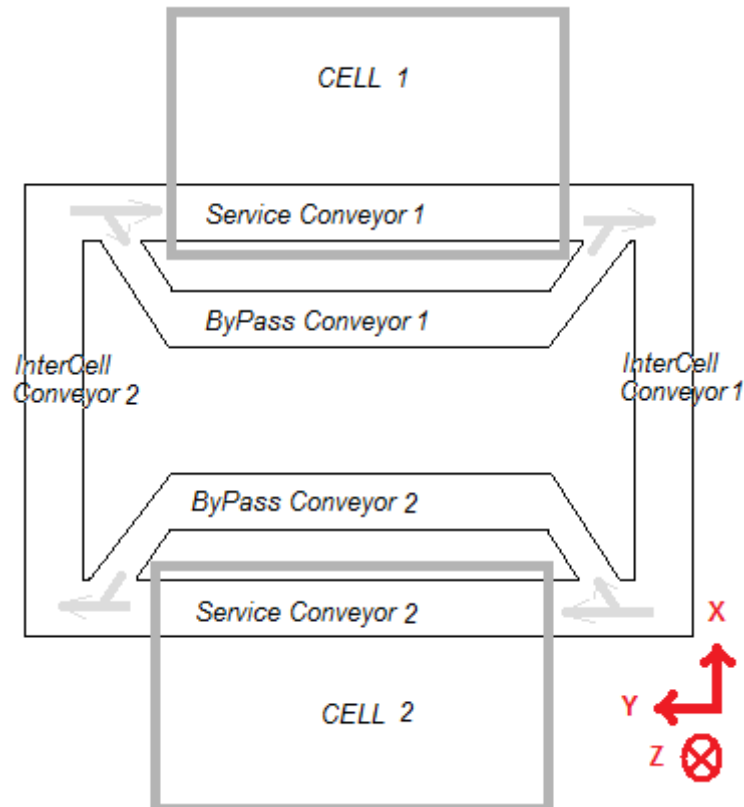


Figure 54: Referential axis of the physical system

3.4.10.1 Control Representation

The physical system requires control in order to perform its operation. The control paradigm applied to this system is integrated through distributed, autonomous, proactive, social and reactive entities known as agents where these are assigned individually to control each device. Since devices have other devices attached to, the agents are not assigned strictly to all devices but rather to main devices and these are the devices where the control is applied in order to provide its complete functionality. One example of that is the agent applied to control a conveyor, it requires sensor's information to sense its environment and actuate on the stoppers to control the pallets on it, therefore the controller entity assigned to control the conveyor it also controls its attached devices, in this case sensors and stoppers.

In order to represent this knowledge in the ontology, a class is added. Figure 55 presents the control class of the production system ontology.

Properties

In order to specify the device to which the controller is assigned, the property *controlsDevice* is defined.

In the transport system there are cases where conveyors have a common area with another, to represent that the property *hasSharedResource* allows its representation and the property *hasSharedResourceAgent* specifies the agent that controls or competes for the shared area. Figure 56 presents the previous mentioned properties.

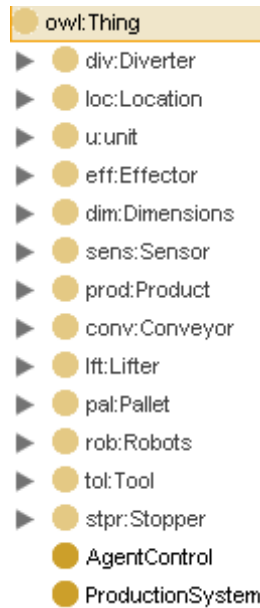


Figure 55: Class Taxonomy of the control from the Production System Ontology



Figure 56: Properties for the control class of the Production System Ontology

Restrictions

In order to specify the controlled devices by the controlling entities a restriction type *someValuesFrom* applies to the property *controlsDevice*. In order to represent the shared area of a conveyor, a restriction type cardinality one applies for both properties *hasSharedResource* and *hasSharedResourceAgent*. In terms of scalability if a shared area has more than two conveyors, then another controller should be assigned to handle the decision making process of that location. As devices have a unique identifier, controllers also have one, therefore a restriction type cardinality one applies to the property *hasUniqueID*. Figure 57 presents the mentioned restriction that applies to the **Control** class.

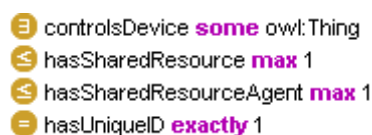


Figure 57: Restrictions for the control class that belongs to the Production System Ontology

Individuals

The individuals of the *Control* class represent the control entities present in the system. The control is represented in terms of the main devices present in the physical system, these are each of the conveyors and diverters that belong to the transport system where the controller of a diverter is called decision point since a routing decision is made at

that location. A controller is also assigned to each robot and pallet, where pallets require to be served by the system.

As mentioned previously in terms of scalability for the case of conveyor's shared areas, if there are more than two conveyors the controller that is assigned to those locations is a decision point agent. Figure 58 presents the individuals of the *Control* class.

- ◆ AgentByPassConv1
- ◆ AgentByPassConv2
- ◆ AgentDecisionPoint1
- ◆ AgentDecisionPoint2
- ◆ AgentInterCellConv1
- ◆ AgentInterCellConv2
- ◆ AgentPallet10485-36
- ◆ AgentPallet10485-40
- ◆ AgentPallet10485-43
- ◆ AgentRobot1
- ◆ AgentRobot2
- ◆ AgentServConv1
- ◆ AgentServConv2

Figure 58: Control individuals of the Production System Ontology

The association of the agents belonging to the *Control* class to the respective devices to which they apply the control is presented in Figure 59.

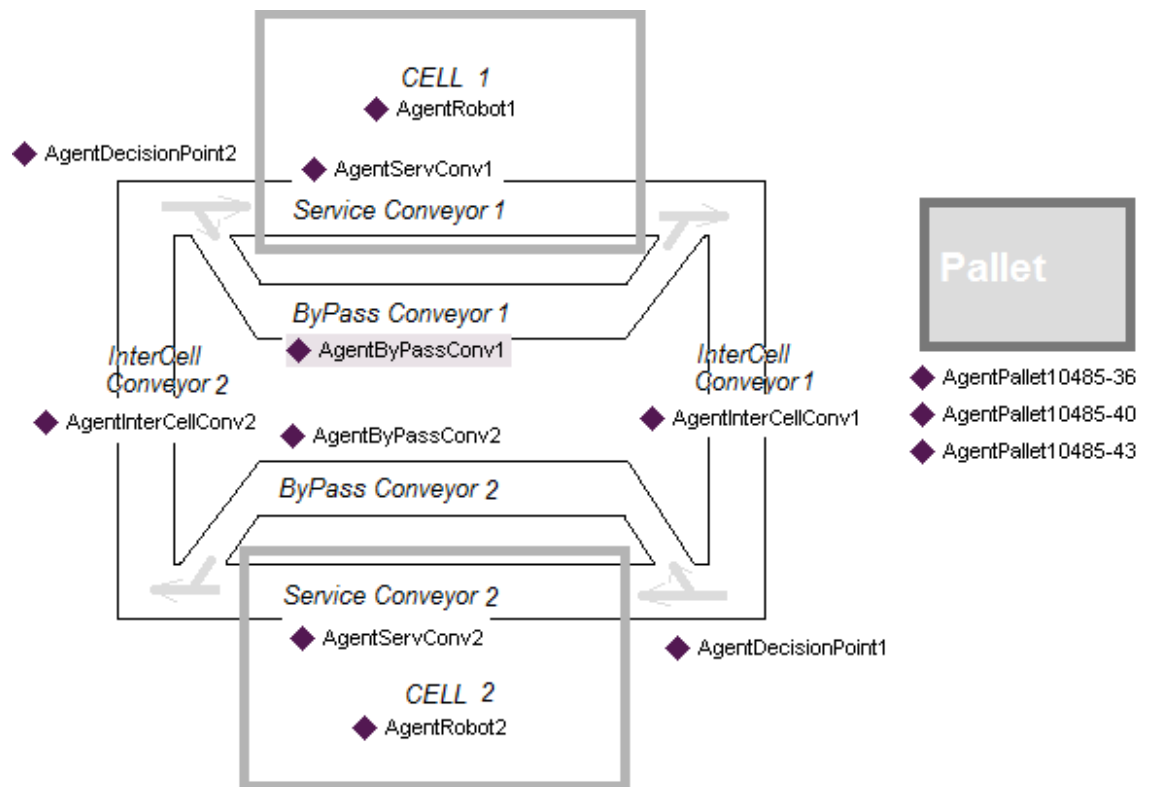


Figure 59 Schematic of the devices and respective controlling agents

3.5 Multi-Agent System

The desired control to apply to the system is in terms of a goal-oriented approach, the paradigm that permits this has to fulfil also the concepts of autonomy, reactivity, sociability, proactivity and distributional. As mentioned in chapter 2.2.2 - Control Needs, the paradigm that suits these needs is the Agent paradigm; however hardware still needs to evolve so that controllers are physically distributed.

Agent's Solution

Taking the concepts of roles and interactions from the methodologies presented in the sub-chapter 2.3.2 - State-of-the-art, the methodology for the multi-agent system consists in each agent plays a role in the system and through interaction it provides services to other agents.

Being this the case, agents are service providers, and in this case scenario agents are in charge of controlling conveyors, diverters, robots, pallets and their attached devices such as stoppers and sensors as presented in Figure 59. Assigning an agent per device as a service provider, makes a conveyor agent as a transport service provider, a robot agent as an assembly, disassembly or other related service provider, and a diverter agent as a routing service provider. At this point, there is no trigger in the system, there is no entity requesting any of the provided services, therefore agents provide services and perform the control of their devices. The trigger is made by the pallet agent which interacts with other agents in order to fulfil the necessary operations of the product that is carried on the pallet that it controls.

Connection to the Ontology

As mentioned previously, one of the things that is necessary to have so that the control of a manufacturing system is independent of its physical system is to have a system's specification. Giving this specification to the control permits it to perform the control based on it, being a specification-based control instead of a control specific for a physical system. For each agent to have the knowledge representation of the physical system it requires making that knowledge available, at launch time, the agent is loaded with its associated knowledge so that when it starts its operation it has the knowledge of the device that is in charge of controlling. A specific agent is assigned to this task, launching all other agents with the associated knowledge, its name is the launcher agent. This approach solves the problem of having multiple accesses to a shared resource that in this case is the ontology.

In order to access the knowledge present in the ontology it is required to use an OWL API, which for this thesis Jena API is the option since is developed in Java and it can be obtained under an open source licence. In order to make the knowledge useful, it is necessary to use an inference engine so that it allows the execution of queries and the

inference of knowledge. In respect to that, Pellet's API was used for this thesis and is also Java-based and is open source.

3.5.1 Communication between Agents

Agents need to interact in order to achieve own goals and thus have to communicate and understand each other. In order for them to communicate they have to speak the same language; however speaking the same language does not make them understand each other, for that the content of the language has also to be defined. With a language defined, agents can exchange messages; however those will not be understood since agents do not share a common syntax and semantics. Having a common syntax and semantics, agents enables agents to understand messages and its content and thus, are enables them to communicate and understand each other. For the scope of this thesis the agent communication language used is the Agent Communication Language (FIPA 2002) developed by FIPA, and in terms of the syntax and semantics an ontology for the communication between agents is presented below in the sub-chapter 3.5.1.2.

Through message exchange, agents can *Request*, *Query*, *Inform*, *Refuse* or send other type of messages. FIPA also specifies standards for interaction protocols between agents and for the scope of this thesis the protocols of interest are the *Request* (FIPA 2002) and the *Query* (FIPA 2002) protocol standards.

3.5.1.1 Agent Ontology - Concepts, Predicates and AgentActions

Agents are able to communicate among themselves, however to understand each other they require knowing communication's syntax and semantics. To overcome this problem, agents can share an ontology. An ontology in JADE is represented in terms of concepts, predicates and agentactions.

Concepts are expressions that represent entities with a complex structure which are defined in slots like (Person :name John :age 31), concepts are meaningful when referenced in other concepts or predicates otherwise concepts are meaningless. Example: (Car :model Astra :brand Opel :owner (Person :name John :age 31)) (Caire and Cabanillas 2006).

Predicates are expressions that refer to the status of the world either true or false. Example: specifying that Peter works for the company FastLab results in the predicate (Works-for (Person :name Peter :age 23) (Company :name FastLab)), therefore predicates can be meaningfully used as message's content (Caire and Cabanillas 2006).

AgentActions are a special concept that specifies actions which can be performed by agents. Example: (MakeHole (Robot :name ABB_1) (Place :name Platform_1)). This specification is useful when requesting other agents (Caire and Cabanillas 2006).

3.5.1.2 Ontology for Agent's communication

The ontology developed for agent's communication is based in the OWL ontology developed for the physical system and is adapted to agent communication requirements.

Concepts

The UML class diagram representing the concepts of the ontology for the communication between agents is presented in Figure 60.

Thing, is the generic concept and is composed by a name and a unique identifier. This concept is the base for all the other concepts.

AgentOnto, represents the concept of an agent, it is extended from the concept **Thing** and it adds an agent identifier. **PalletAgentOnto**, is the concept of pallet agent and it extends the concept of **AgentOnto** since it is also an agent. It adds pallet's information such as the RFID tag value and priority, which are extracted from the product that the pallet device carries. **Working Area** class, represents the concept of a working area in the physical system, it extends the concept of **Thing** and it adds location information. **Location** concept is described below. **Service** class specifies a description of a service, is an extension to the concept **Thing** and it adds location and type information. **Product** and **Manufacturing Process** classes represent respectively the concepts of product and manufacturing processes and are an extension of the concept **Thing**.

In order to represent devices there is another concept named **Device**, which is also an extension of the concept **Thing**. Device's concept adds type's information, and to represent each of the devices present in the physical system consists in extending the concept **Device** where class and type are named with device's name and type respectively. Therefore the concept **Conveyor** is an extension of the concept **Device** and its slot type is also fulfilled with its device's type, conveyor. The concept of other devices present in the physical system follows the same procedure besides the device pallet. Pallet extends the concept **Device** and it adds RFID value, priority and location which are extracted from the product that it carries. Location corresponds to the desired destination so that the manufacturing process is performed on the product that is being carried.

In order to represent properties the concept **Property** is defined. **Skill**, **State**, **ControlOperation** and **Cost** represent their concepts respectively and are described in terms of the name that its concept represents. **Location** class represents a physical location in terms of the distance to a referential axis and in terms of the coordinates x, y and z.

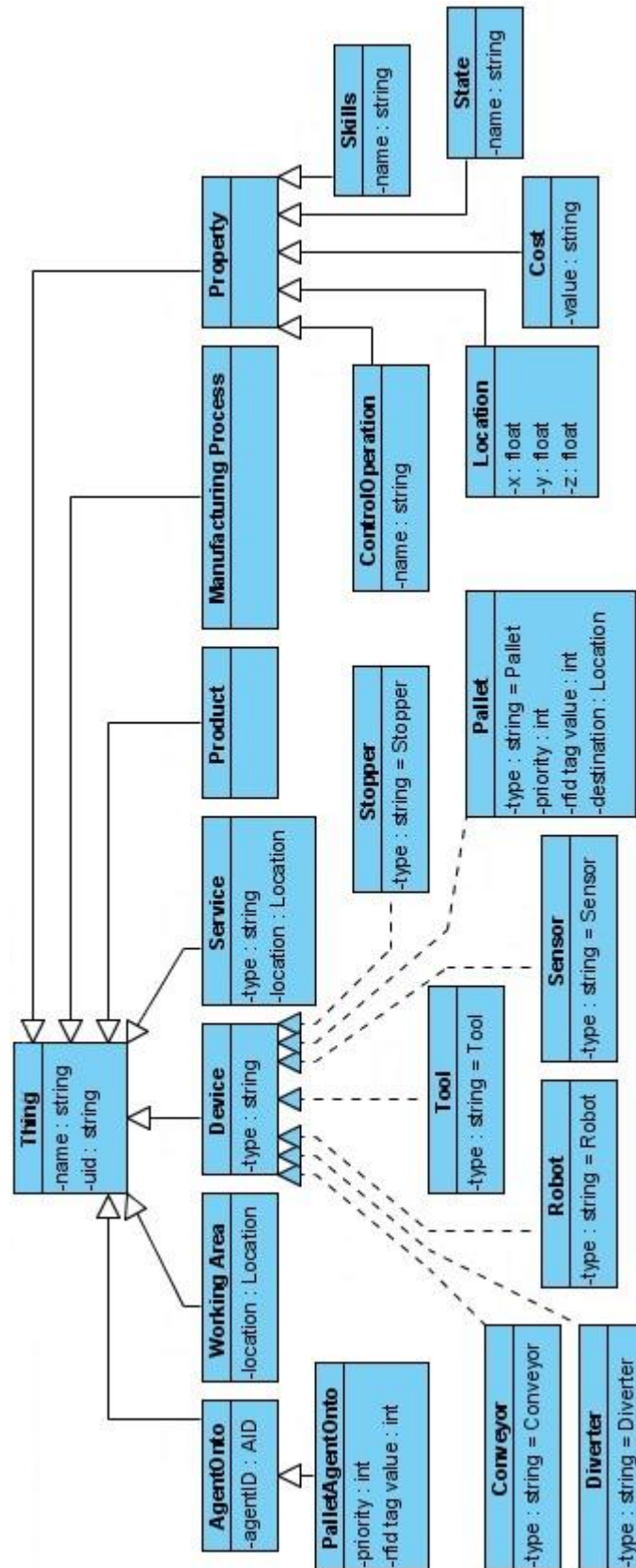


Figure 60: UML Class Diagram of the Ontologies for Agent Communication

Predicates

hasService specifies the relationship between the agents and its provided services and is represented by the concepts *AgentOnto* and a list of *Service*.

hasControlOperation represents the control operations that can be applied to a device, it is described by the concepts of *Device* and a list of *ControlOperations*.

hasDevice represents the attached devices of a device, it is described by the concept *Device* where a device is related to a list of devices representing the attached ones.

hasFollowingTransportAgent this predicate represents the two agents of the transport devices that are connected; its slots are the concept of *AgentOnto* for the previous and following agent description.

hasLocation specifies the location of a thing, its representation consists in a slot for the concept *Thing* and other for the concept of *Location*.

hasManufacturingProcesses it describes the relationship between a product and the manufacturing processes that it needs. Its representation is specified by the concepts of *Product* and a list of the concept *ManufacturingProcess*.

hasPallets represents the pallets that are under the domain of one device's agent, usually for conveyor agent. Its description consists in the description of the agent, using the concept *AgentOnto* and a list of the pallets that it has, fulfilled with the concept *Pallet*.

hasProduct describes the relationship between a pallet and the products that it carries physically. It is represented by a pallet's description and a list of products where the concepts *Pallet*, *Device* and *Product* are used respectively.

hasProperty represents a generic relation between a thing and a property that classifies it. The concepts *Thing* and *Property* describe them respectively.

hasSkills is intended to specify the skills that a device has, its slots are a device description and a list with the associated skills. The concepts of *Device* and *Skill* apply respectively to each of the intended slots.

hasState describes the state of a thing, its slots are thing and state where each one represents its concept, *Thing* and *State*.

hasTool represents the tool which is attached to a device, its slots are a description of the device that has the tool and the respective tool.

AgentAction

ServiceAction, specifies the service that is requested by an agent, it is described by the respective service and the agent requester. The slots are fulfilled with the respective concepts description.

Vocabulary

The vocabulary used to fulfil the string slots is defined in a vocabulary section so that when an agent is aware of the ontology is also aware of its vocabulary.

3.5.2 Launcher agent

The Launcher agent is the agent in charge of launching the complete Multi-Agent System that controls the physical system, its operation starts by getting the control representation of the system from the ontology in terms of the agents that will control it. Then it gets the respective device information of each agent as well as the attached devices, and with this information launches each of the agents.

Agent's behaviour is presented in Figure 61 and it starts by loading the ontology, then it queries to get controller agents that are in charge of controlling the physical system; for each controller agent, it performs queries to get its knowledge in terms of device information and attached devices. Since each agent can have different information other queries are performed to get specific agent's information and Figure 62 presents its activity diagram. Once required information is loaded the agent is launched, and after launching all the controlling agents the sensorial and monitor agent are launched.

The operation of querying the ontology to get specific agent's information consists in getting information that is only represented for that agent which is the case of the conveyor, decision point and pallet agents, the transport agents.

Transport agents have the information in respect to the physical connection to other transport devices, therefore when launching a transport agent its neighbouring agents are obtained, respective to the neighbouring devices. For a conveyor agent, information regarding the existence of a shared area is also obtained; for the decision point agent, information in terms of the position locations available at the destination transport agents is obtained to be aware of the possible working areas; for a pallet agent, its RFID tag value, name, priority and the manufacturing processes required from the carried product are also queried.

In terms of reconfigurability adding or removing devices to the system is allowed and it does not interfere with the normal operation of its control, as long as these devices are not transport devices in which the case this thesis does not provide a solution but some ideas that could be implemented in the future.

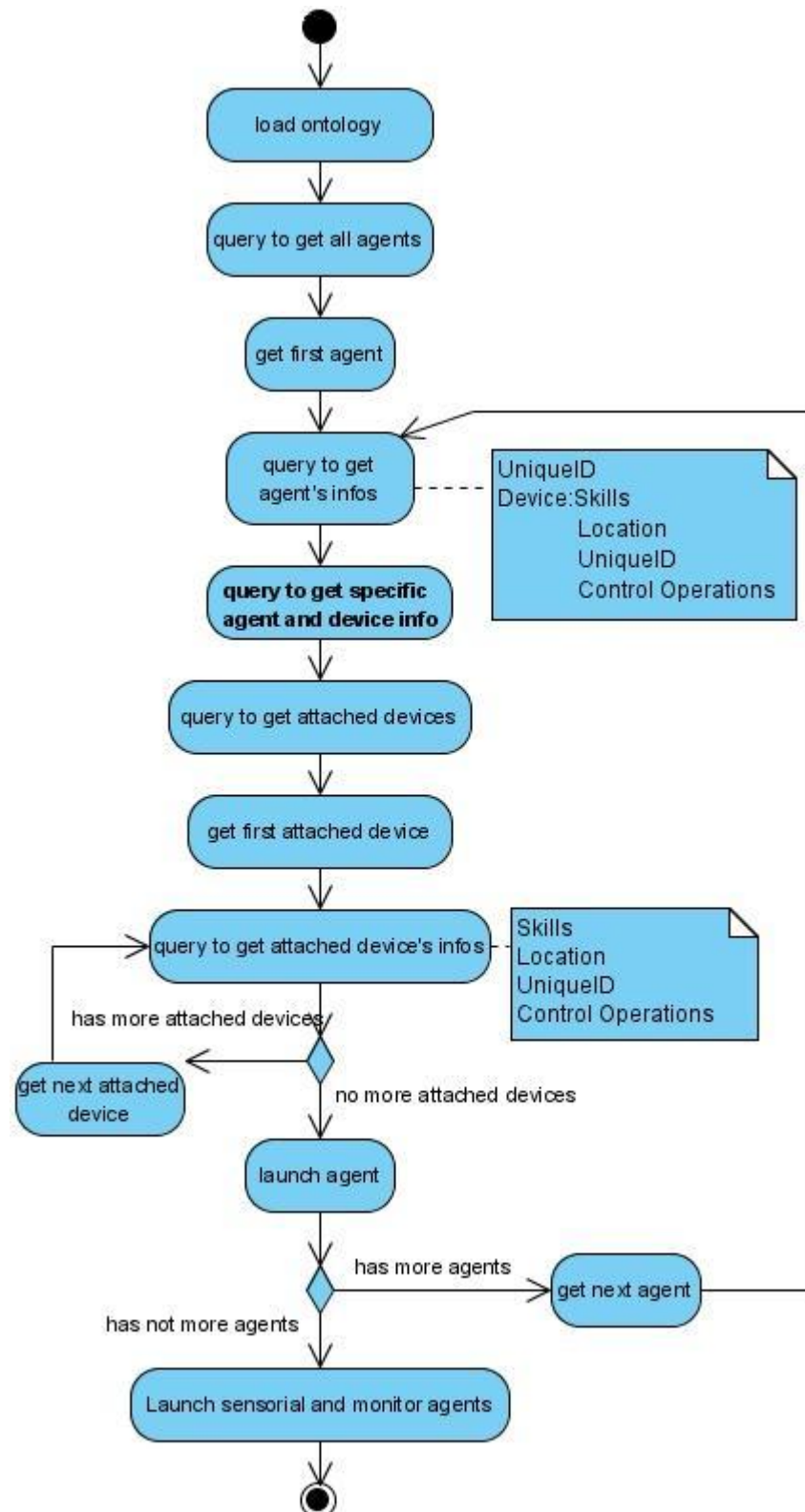


Figure 61: Activity Diagram of Launcher Agent

Adding a new device to the running system consists in updating system's ontology with the new device and then launch its agent using the Launcher agent. Removing an agent, is simple as removing the device and agent, and also updating the ontology, the device

must not be under operation otherwise it will ruin the operation. Remember, adding or removing devices on the fly is not possible for transport agents.

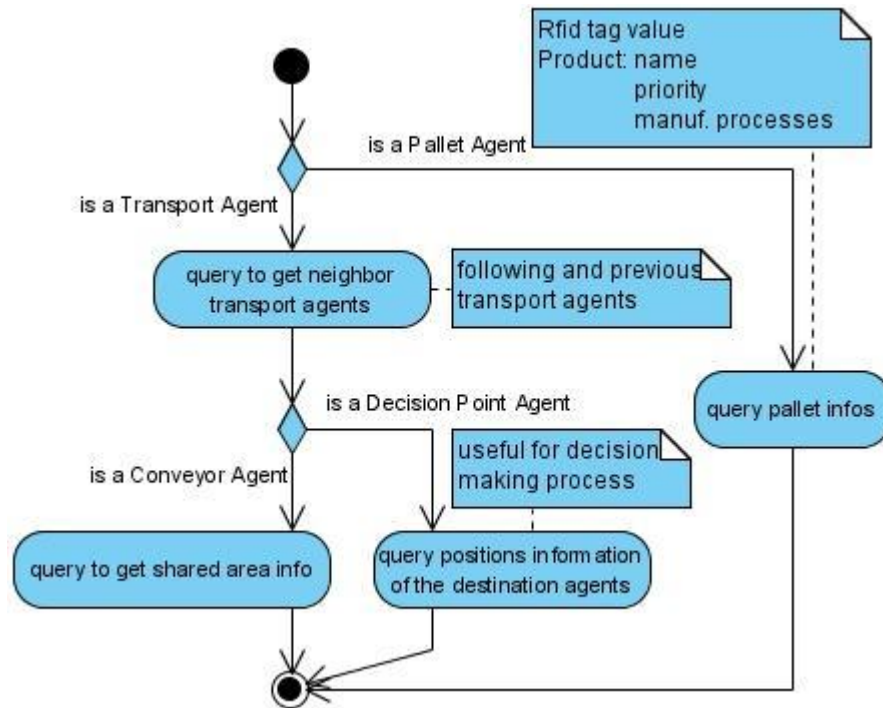


Figure 62: Query Specific Agent Information of the Launcher Agent

3.5.3 Transport Agents

Transport agents provide a service related to the movement of pallets in the manufacturing system, these are the conveyor agents and the decision point agents. Transport agents are aware of to which transport devices have a physical connection, so that a pallet can be routed through the complete transport system.

Conveyor agents control a conveyor device and provide a transport service, decision point agents control a diverter device and provide a routing service, however diverter agent can also be assigned to places or devices where a routing decision is required. Through interaction with other agents it is possible to request transport agent services and query information about its state. Behaviours of a transport agent are present in Figure 63. Transport agent operation starts by initializing the agent, where its knowledge is analysed, then the agent registers itself, which corresponds to the act of registering agent name and services which are device skills. Then agent launches its three main operating behaviours which operate during agent lifecycle and these are service responder, receive informs and query responder; service responder behaviour handles service requests from other agents; the query responder handles queries regarding agent state; finally receive informs behaviour handles reception of information such as the end of operation in a pallet at a specific conveyor position, information of a pallet that has been delivered to the following transport agent and also sensor information.

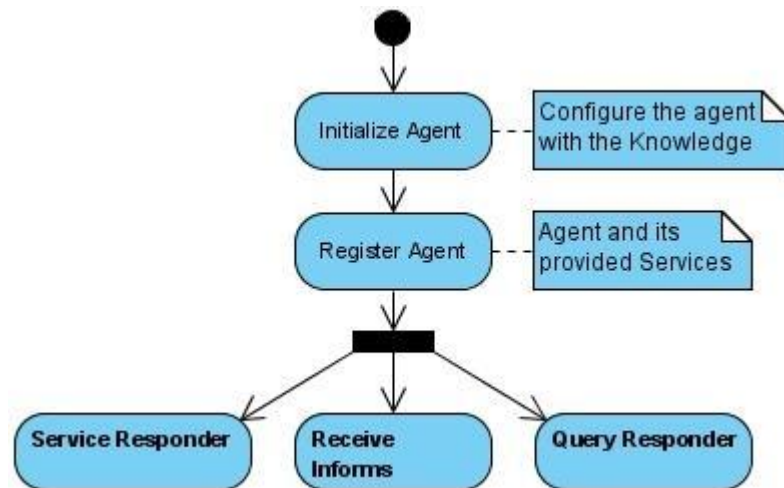


Figure 63: Behaviour sequence of a Transport Agent

Service requests require an interaction protocol so that agents are able to *Request* a services and *Query* information, Figure 64 presents service request interaction protocol and Figure 65 presents *Query* free places interaction protocol.

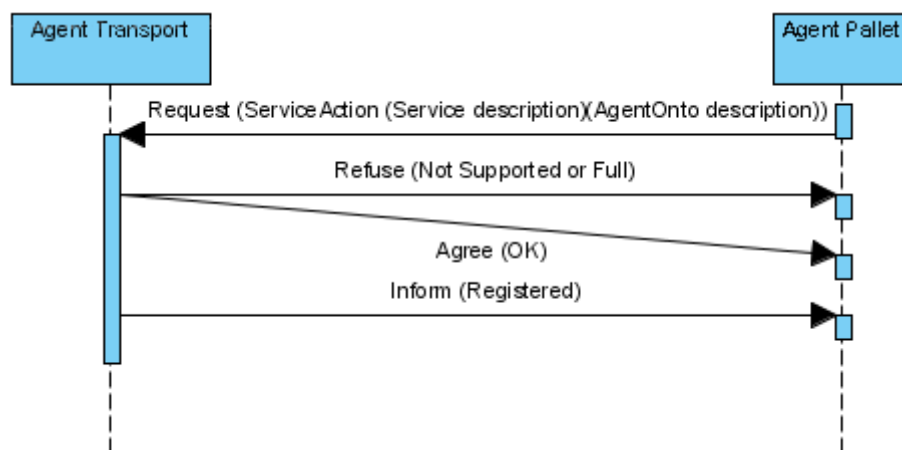


Figure 64: Service Request from Service Responder behaviour, adapted from FIPA Request IP

The service request interaction protocol is generic to all service requests within the MAS where the answer can be adapted to the service that is being requested. *Request* has an *AgentAction* called *ServiceAction* and is composed by a *Service* description and its *Requester*; *Service* description consists in name, type, unique identifier and desired destination location, *Requester* description consists in name, agent identifier and unique identifier. The *Answer* depends on the *Request*, if message's content is incorrect, the service is not provided or the agent is busy, the *Answer* is of type *Refuse*, otherwise is *Agree* followed by a message type *Inform* for a successful request.

The *Query* free places interaction protocol is a protocol which message content has a description of the queried transport agent and *State* description has value "empty" specifying that the query regards empty places. In terms of answers to this interaction protocol, if there are no free places answer message type is *Refuse* and its content

specifies full, otherwise is *Agree* followed by an *Inform* type message with the number of available places in the state description. When *Request* message content is not the expected one, answer message type is also *Refuse* and state description mentions unknown.

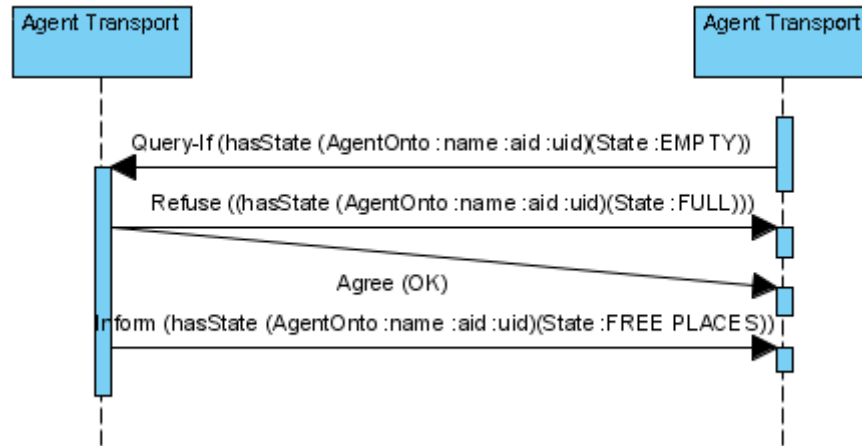


Figure 65: Query free places from Query Responder Behaviour, adapted from FIPA Query-If IP

The following sub-chapters describe in more detail the conveyor and the decision point agents which are transport agents.

3.5.3.1 Conveyor agent

A conveyor agent has the objective of controlling a conveyor device, manage pallets and provide a transport service to the multi-agent system. Its behaviour diagram is presented in Figure 63 and its description is presented below.

When a conveyor agent is launched its operation starts by analyzing the knowledge that was loaded with it. The information regards conveyor's device and its attached devices, and it ranges from device's skills, control operations, location and unique identifier. In terms of conveyor capacity, it is obtained through the organization of attached devices by location, ordered according to the flow of products. Capacity corresponds to the different working positions due to the attached devices. Taking the service conveyor from the physical system as an example, it has two presence sensors, two RFID readers and two stoppers, physically distributed in two groups with have two different locations, thus it has capacity two. With the location of each position it is also possible to represent the order and position of each working area where pallets can be stopped.

As explained before, the behaviours which describe a transport agent are service responder, receive informs and query responder. Each one has a specific function and for the conveyor agent query responder behaviour gives also other answers such as query shared area and query pallets information.

The query shared area is used to query if conveyor's shared area is free, it is a common area which a conveyor can have in its end and related to other conveyor. In order to

avoid collisions in the shared area, the conveyor agent queries the other agent about the availability of the shared area before releasing a pallet to the following transport agent. Figure 66 presents the interaction protocol of the query shared area of the query responder behaviour. The first message is of type *Query* and its content has the predicate *hasState* since the question relates with the state of the shared area. The *hasState* predicate is composed of a device description together with a state description, the device in this case represents the exclusive resource, its name and unique identifier are specific to the area in case. The desired state is empty, therefore content's state is "empty". In terms of the answers to this protocol, when the shared area is occupied the answer is negative, message type is *Refuse* and the content is the same as the query message changed the state field mentioning full. On the other hand, if the shared area is not occupied there is a positive answer and a message type *Agree* is sent, then to confirm it a message type *Inform* which content is the same type as the initial *Query* is sent.

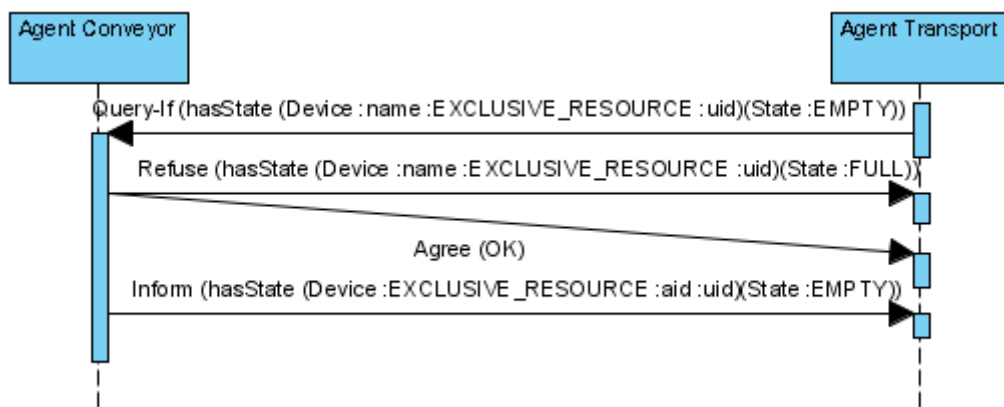


Figure 66: Query Shared Area from Query Responder, adapted from FIPA Query-If IP

The shared area is also the area between transport agents where both control agents do not have sensorial information in respect to the pallet since it is transiting from one transport device to the other.

In order to know the information of the pallet present in a conveyor, the interaction protocol from Figure 67 describes it. This interaction protocol belongs to the query responder behaviour of the agent. The message that starts the interaction protocol is of type *Query-if* and its content is the predicate *hasPallets* that is composed by an agent description respective to the conveyor agent and a list of pallets with its information. The first message corresponds to a question to know pallets information present at the conveyor and after a positive feedback, with a message type *Agree*, an *Inform* message is sent where its content is a *hasPallets* with a list describing the pallets. Pallet's description is characterised by name, type pallet, unique identifier, priority, RFID tag value and the location destination.

Other behaviour that a conveyor agent has is the receive informs which is characterised by handling the reception of inform type messages which inform the agent with respect

to finished operations at specific working areas, delivered pallets to the following transport agent and other information. Figure 68 presents behaviour's activity diagram where all the cases are described.

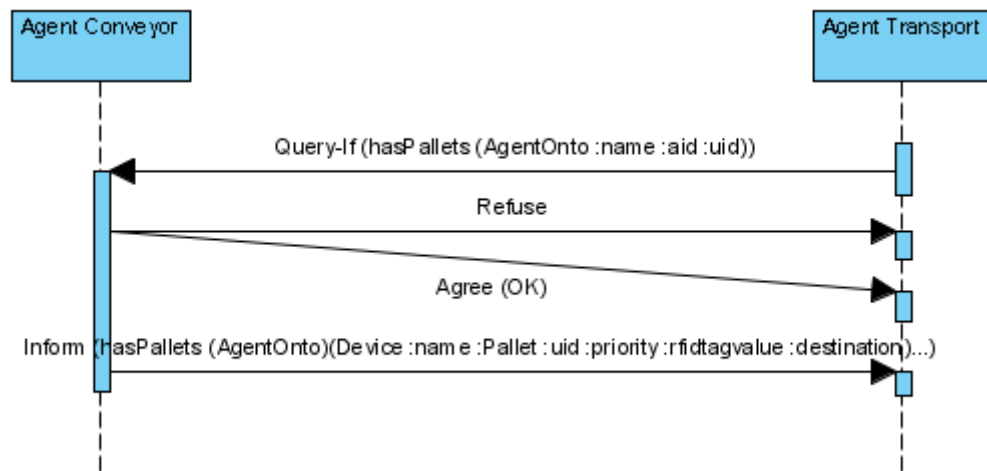


Figure 67: Query pallets' information interaction from Query Responder, adapted from FIPA Query-If IP

Its operation starts when it receives a message that is according to the specified template of the receiving operation, its template defines that the message is of type *inform* and not *Query-if* or *Request*.

The expected messages are sensor's information, working area information and also information about a pallet that has been delivered to the following transport agent.

Sensor's information message corresponds to a message sent by the agent that manages sensor's information. The arrival of a message of this type means that a physical event has occurred; therefore, it updates sensorial information and a decision making process is made to actuate on the physical device.

The delivered *inform* message has the information regarding the delivery of a pallet previously sent to the following transport device and agent. After analyzed conveyor's occupation is checked, if the conveyor is empty then it is turned off in order to save energy.

The working area *inform* message, gives information in terms of the working area and it can inform that a pallet is going to have an operation at that location and thus it has to be stopped or that an operation at that location has ended. The working area information is verified in terms of the validity of its location, if it corresponds to a location that does not exist the message is discarded. After the arrival of a message of this type location state information is updated.

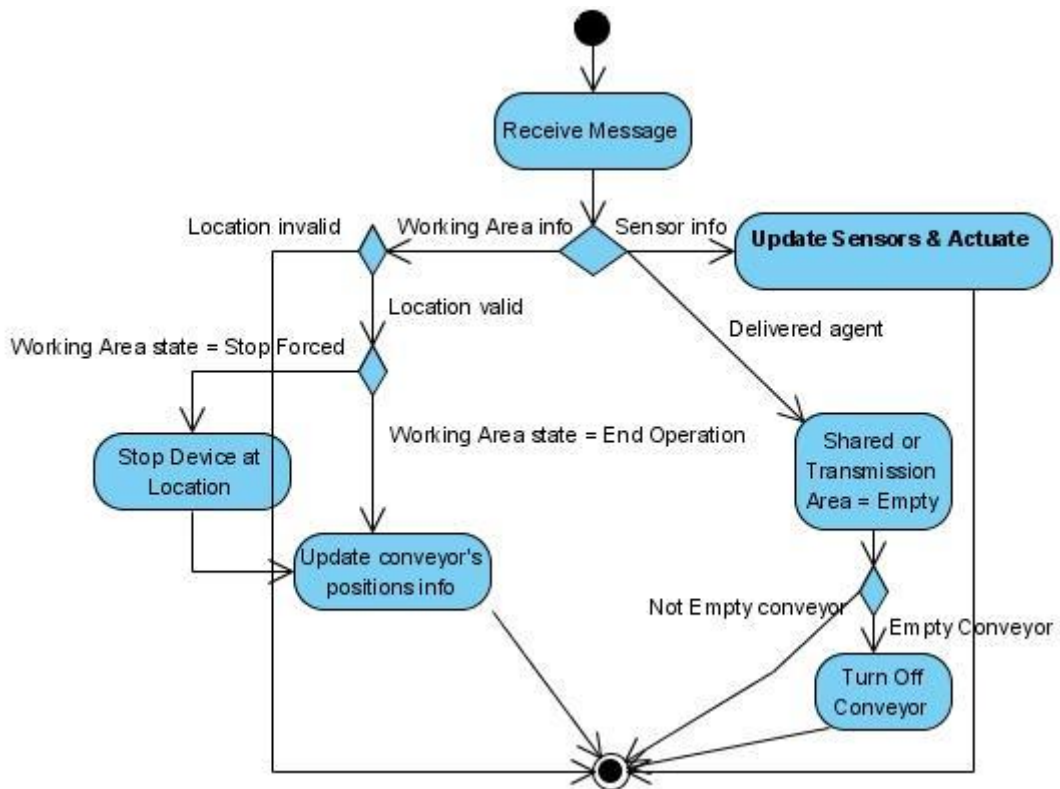


Figure 68: Behaviour's Activity Diagram of Receive Informs

When a pallet is in the last position of a conveyor, it has to be delivered to the following transport device and agent. For that it is required to know if the transmission or shared area is free and that the following transport device has space for the pallet. In order to achieve that, it is necessary to query the agent that is competing for the shared area and then query the following transport device agent. For the cases where there is no shared area only the following transport agent is queried, for both cases if a negative answer is received the process is repeated until the agent is served. Being the agent served, the pallet is sent to the destination transport device. Figure 69 presents the sequence of operations that represent the mentioned process.

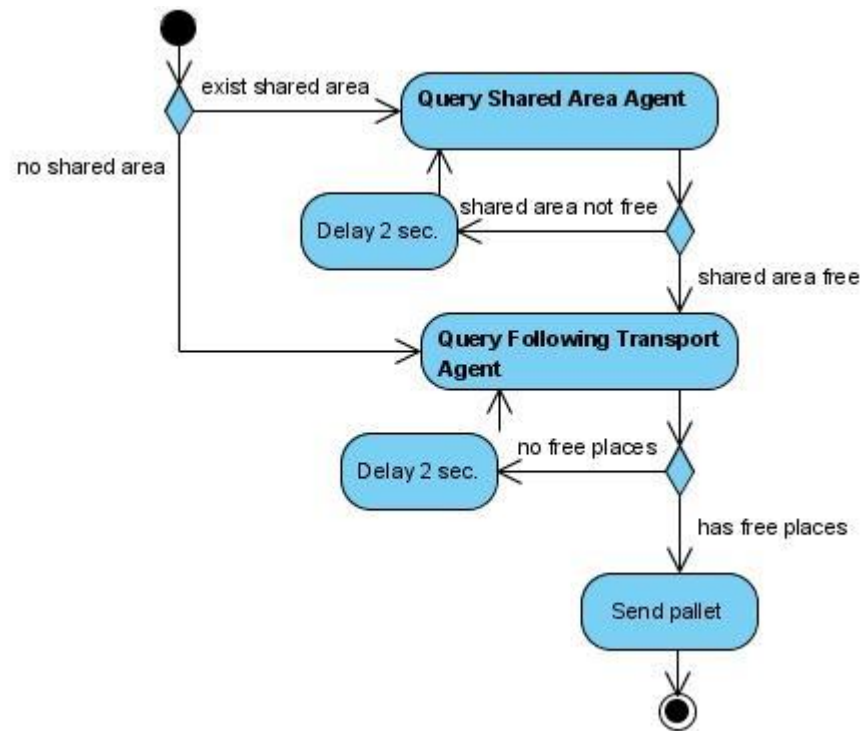


Figure 69: Activity Diagram of the Conveyor Agent depicting the delivery of a pallet to the following Transport agent

3.5.3.2 Decision Point Agent

A decision point agent is a transport agent since it belongs to the agents that provide services related to the transportation of pallets, more specifically a decision point agent provides a routing service given a set of entrance and exit points; a decision point agent is assigned a location and manages the delivery of pallets to the correct exit point according to pallet's desired destination.

Figure 63 presents the behaviour sequence which characterize this agent. In the initialization behaviour, knowledge is obtained and it consists in device's information, attached devices, and information regarding the previous and following transport agents; the following behaviour is where the agent registers itself and the provided services which correspond to device's skills; then, the behaviours which characterise agent's operation are launched, namely service responder, receive informs and query responder. Service responder behaviour answers service requests through the interaction protocol present in Figure 64; query responder behaviour provides answer to queries regarding the free places of the diverter as presented in Figure 65; finally the receive informs behaviour handles information messages which can be sensor's or pallet delivery information and its behaviour is presented in Figure 70.

Decision point agent starts operation upon receiving a message matching a specified template from the receiving behaviour, the template defines a message type *Inform* and not *Query-if* or *Request*, the expected messages are sensor and pallet delivery

information where sensor information represent a physical event and pallet delivery information the delivery of a pallet to another transport agent.

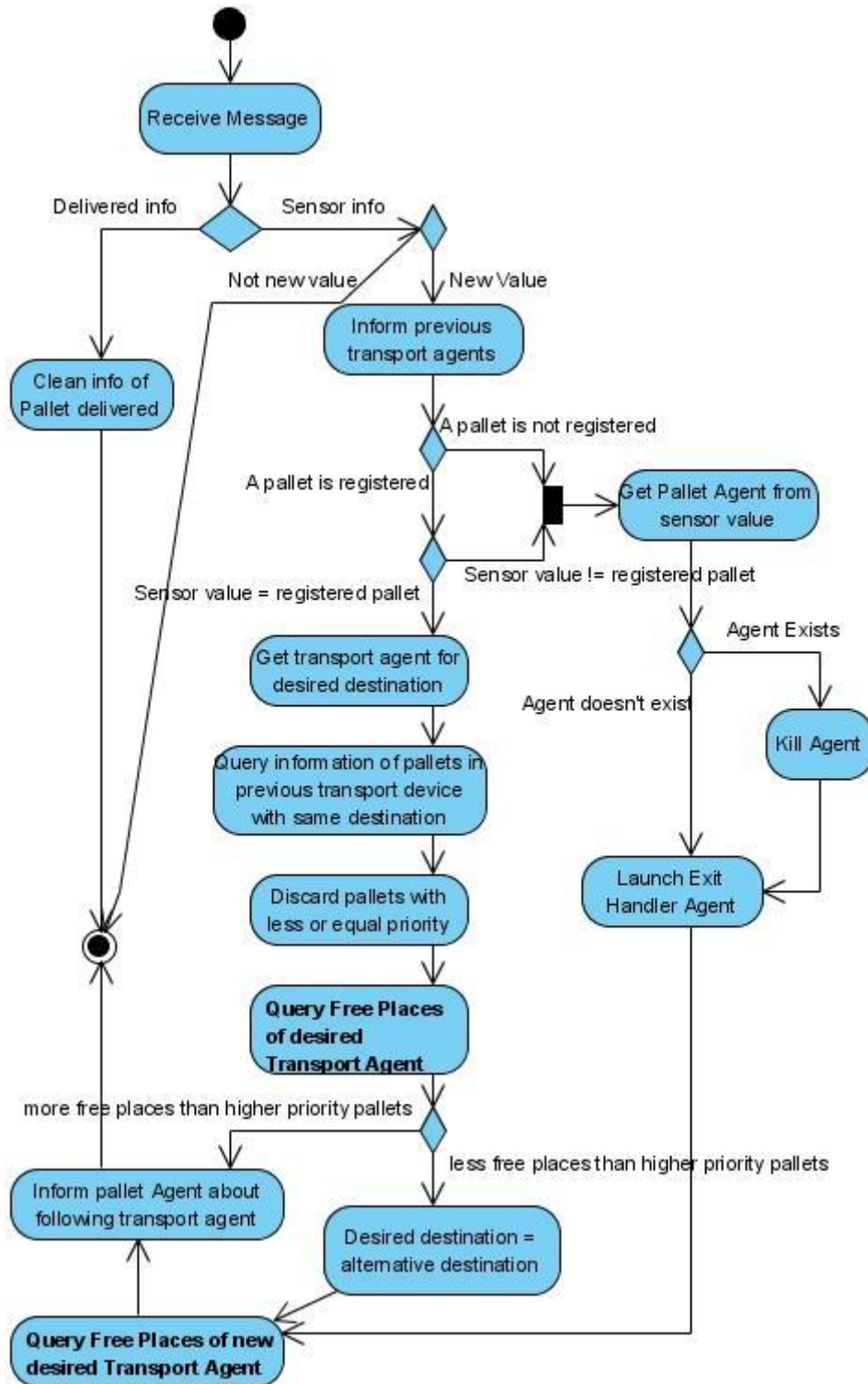


Figure 70: Behaviour's Action Diagram of the Receive Informs from agent Decision Point

In this physical system the decision point agent is assigned to diverters which have RFID sensor and stopper as attached devices. When receiving sensor information, it means that a new pallet is present at agent's working area and thus it requires

acknowledging the previous transport agent which confirms the delivered pallet. In order to know pallet's desired destination it requires having requested the service first, if that is the case, the decision making process starts, otherwise the pallet is routed out of the system since it is not synchronized. This operation is done by the exit handler and if the pallet has a controlling agent this one is killed, pallet checking is made through its RFID sensor value, and once validated the desired destination is known. The decision making process considers traffic in previous and following transport devices, and for that queries are made in order to obtain information regarding pallets with the same destination and higher priority, followed by querying the availability of desired destination transport agent. If the free places are more than the higher priority pallets, the pallet is routed to the transport agent which takes it to the desired destination; otherwise is routed to the alternative transport device since higher priority pallets are coming after. In the present system alternative path is the bypass conveyor. Before delivering the pallet to the alternative transport it queries it in order to get information in respect to free places. In both cases, when the transport device and agent are chosen the pallet agent is informed about it.

3.5.4 Robot Agent

Robot agents provide manufacturing operations to pallets, it not only controls a robot device but it also provides the controlled operations as a service to the multi-agent system. Its operation consists in performing a manufacturing process to a pallet which has previously requested the service, the operation is performed once the pallet is within robot's working area.

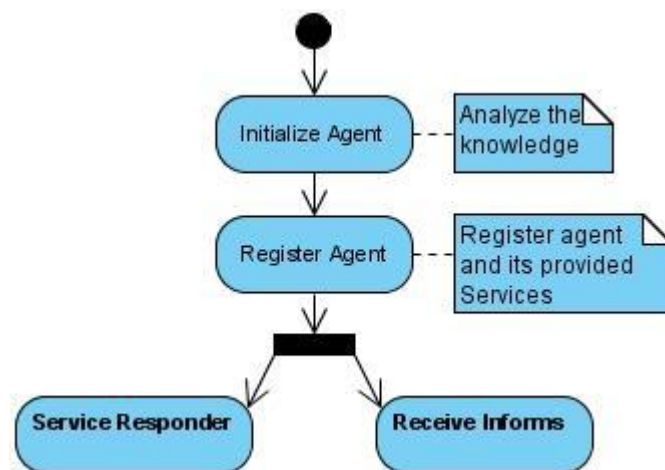


Figure 71: Behaviour sequence of Robot Agent

Agent behaviour starts by analysing the knowledge that has been loaded, followed by the registration of the agent and the provided services. Being these behaviours completed the behaviours service responder and receive informs are launched and these characterize agent's operation as presented in Figure 71.

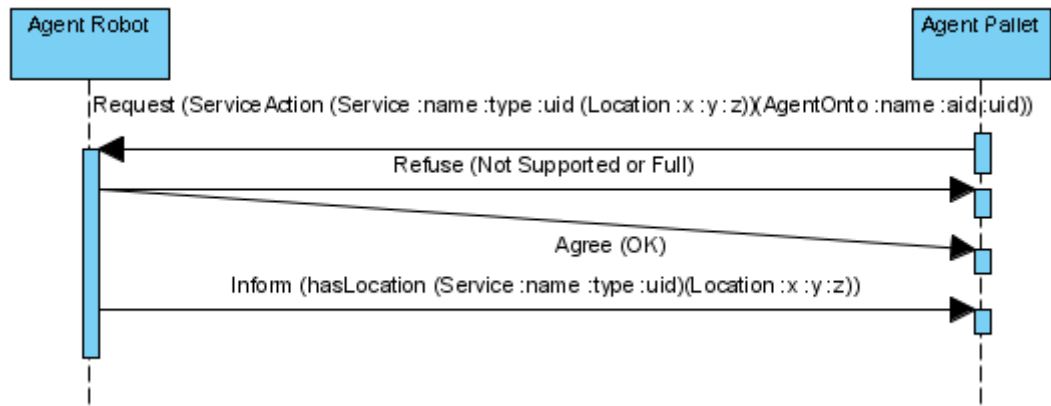


Figure 72: Service Responder Interaction of the Robot Agent, adapted from FIPA Request IP

Service responder behaviour is used to receive service requests and its interaction protocol is presented in Figure 72, the interaction protocol starts by sending a message type *Request* which content are a service and requester description, where service description is an *AgentAction* named *ServiceAction* used to request any service within the multi-agent system. A negative answer, *Refuse* type message, is provided for cases where the robot is busy or the requested service is not available or provided; otherwise a positive answer is sent to the requester; first in terms of an *Agree* type message, and then confirming the request with an *Inform* type message which content specifies the location to which the requester agent has to go in order to be served. Providing location information in terms of the place where to perform the operation gives freedom to the agent providing the service to choose the best location, especially for the cases where it has more than one working area since it can distribute the work load.

Receive informs behaviour, is used to receive information from other agents that in this case is sensorial information, its activity diagram is presented in Figure 73. It starts when a message according to the specified template arrives, the template specifies that the message is of type *Inform* and not *Query-if* or *Request*, the expected content is sensor's information and at the arrival of a sensor message, sensor value and location are extracted. Its information can range from an end operation to the pallet's RFID tag value. Upon receiving an end operation message, it requires informing pallet agent as well as respective conveyor so that both are aware of the ended operation. Then, if pallets are waiting for an operation to be performed, the first from the list is served. For the case of the sensor value being an RFID tag value it corresponds to a pallet tag number, if this number corresponds to a pallet that has previously registered within the robot agent, the requested operation is performed if the robot is free. Once the operation starts, robot agent informs pallet agent about the beginning of operation. If the robot is not free then pallet's information is queued.

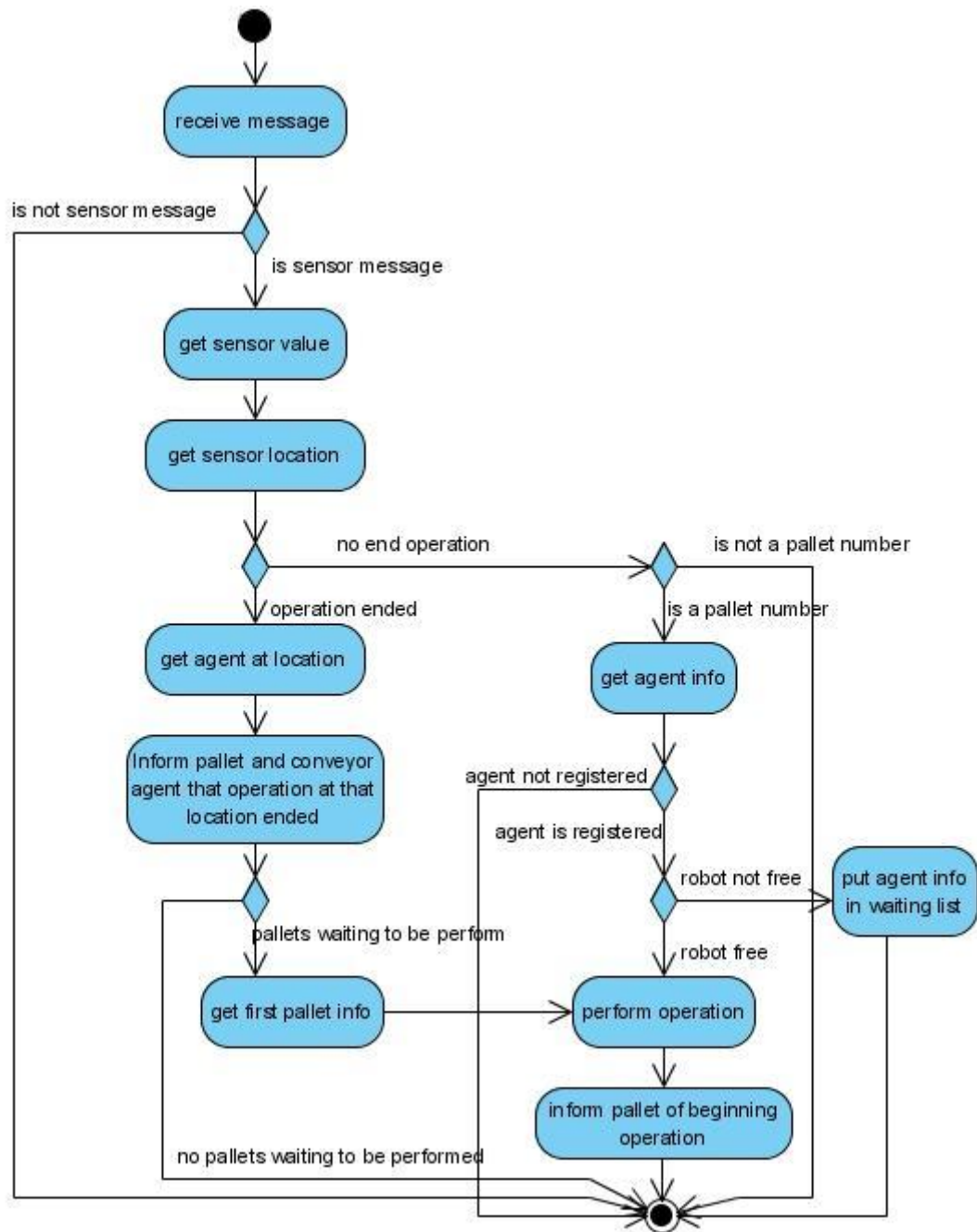


Figure 73: Receive Informs Activity Diagram of the Robot Agent

3.5.5 Pallet Agent

Pallet agent is in charge of the pallet device, where a pallet carries a product which has a number of manufacturing processes that need to be performed in order to manufacture the product. To achieve that, requires the agent to manage in order to get the manufacturing processes done by other devices and thus it needs a way to request the processes and also to reach them, to request operations require knowing where those operations are listed and how to request them.

Pallet agents provide a carrying service and during operation serves manufacturing processes following product required operations list. To get the manufacturing

processes served requires searching it and then request it, based on the provided location information route the pallet to the specified location. In order to get to the specified location it also has to request transport agents, upon completed operation, agent repeats the process for the remaining operations.

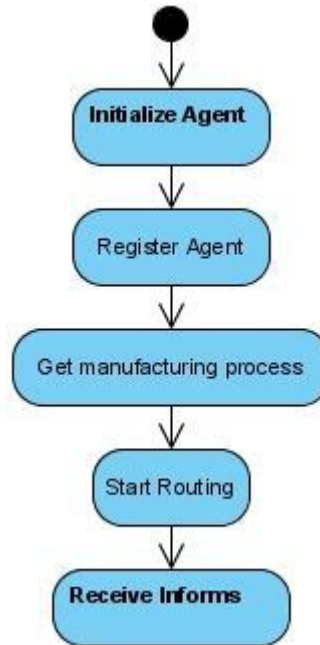


Figure 74: Behaviour of Pallet Agent

The behaviour of a pallet agent is characterised in Figure 74, it starts by analyzing the loaded knowledge, then registers itself as well as the provided services which are device's skills. In order to start its operation, it searches for the manufacturing process that the product requires and with the search result it requests one of the providers then, with the location information provided in the answer it starts the routing in order to physically reach the service provider and for better behaviour understanding Figure 75 presents the activity diagram searching and routing in order to get the manufacturing process.

The process of searching for a manufacturing process consists in looking up in the service repository for a specific service, if exists the service is requested and if accepted routing location destination is updated where its interaction protocol is presented in Figure 72. In the case of a request failure, other providers, if any, are requested and for the case of no providers the search is delayed for a small amount of time, for a possible availability in the system. During delay time routing destination is updated with no destination so that the pallet is hanging around the system waiting for the availability of the service, if the service does not become available, it is discarded and the following one is performed.

Agent's main behaviour is named receive informs and its activity diagram is presented in Figure 76, it starts at the arrival of a message that is according to a template which

defines message type *Inform* and not *Query-if* or *Request*, its information content can range from working area to following transport agent information; upon reception of a working area information message, its content represents working area's status, its location is validated according to desired operation location; working area's status can range from begin to end of operation; when receiving a begin of operation, operation status is updated; for the case of an end of operation received, the following manufacturing process has to be requested and thus the get next manufacturing process behaviour is launched.

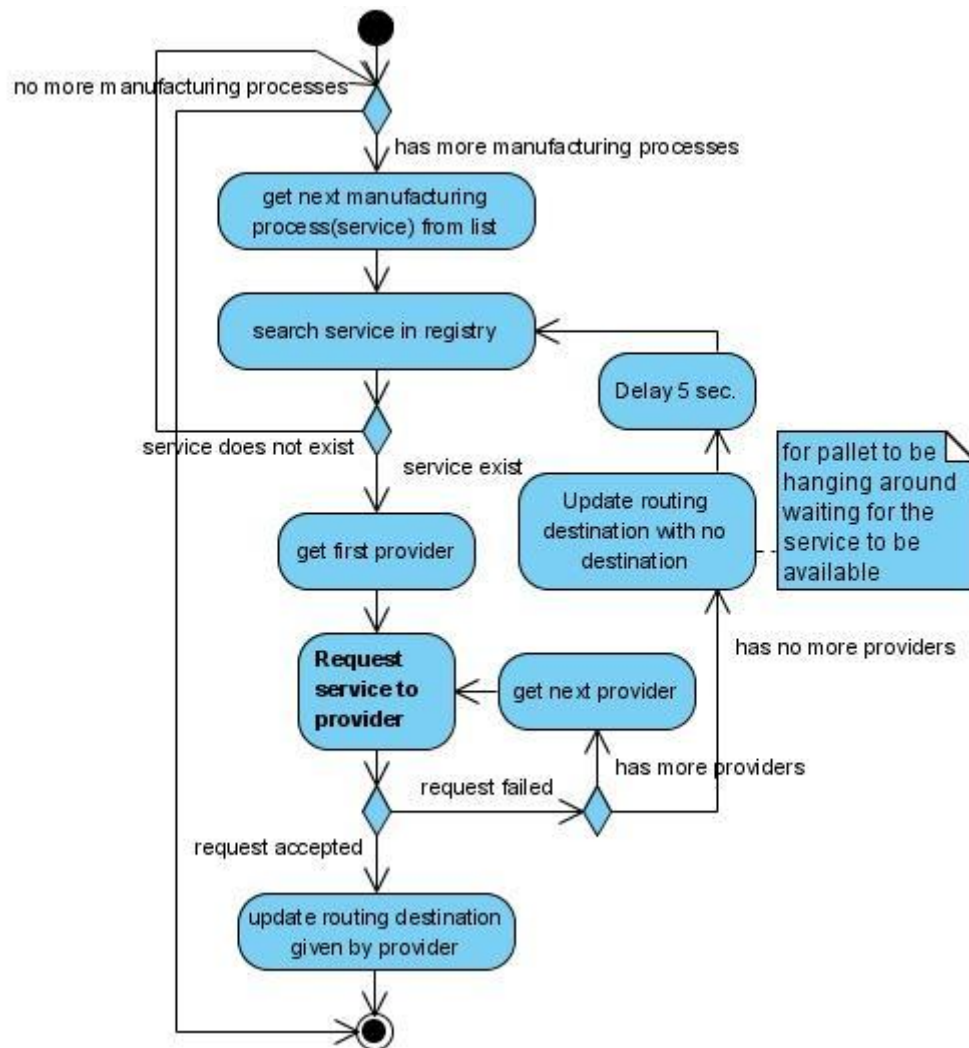


Figure 75: Get next Manufacturing Process Activity Diagram of the Pallet Agent

Upon receiving following transport agent information message, means that the pallet is physically going to the following transport device; therefore, uses it to request the following transport device, interaction protocol is presented in Figure 64 and its operation continues repeatedly until the last operation is performed on the pallet.

For a pallet to reach a desired destination, it requests transport agents one after the other until it reaches destination, this way pallet agent do not have any routing decision since

that is done by the transport agents which carry the pallet, therefore pallet agent is less complex since all the routing complexity is at the transport agent side.

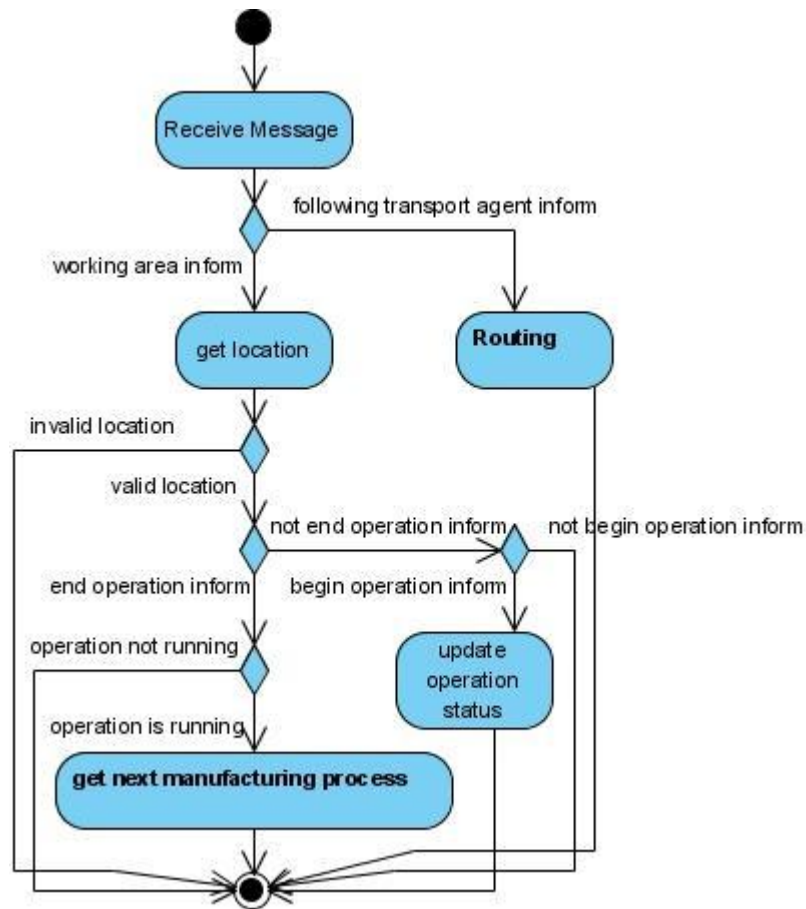


Figure 76: Receive Informs Activity Diagram of the Pallet Agent

3.5.6 Exit Handler Agent



Figure 77: Behaviours of Exit Handler Agent

Exit handler is the agent in charge of routing out a pallet that is not synchronized in the system and is launched by a decision point agent. The behaviours that characterize the agent are present in Figure 77, it starts by analysing the knowledge loaded with the

agent which corresponds to pallet information; on the following behaviour, agent registers itself and the provided services, and finally the last behaviour describes agent's main operation which consists in receiving informative messages and act upon.

Main behaviour is triggered by the arrival of messages, namely receiving information which corresponds to the following transport agent provided by the current transport agent while exit handler agent is routing out the pallet. With the information of the following transport agent, exit handler agent performs request operations to the following transport agent using the interaction protocol present in Figure 64, and once it reaches the exit point its operation ends. Figure 78 presents main behaviour activity diagram, the receive informs behaviour activity diagram.

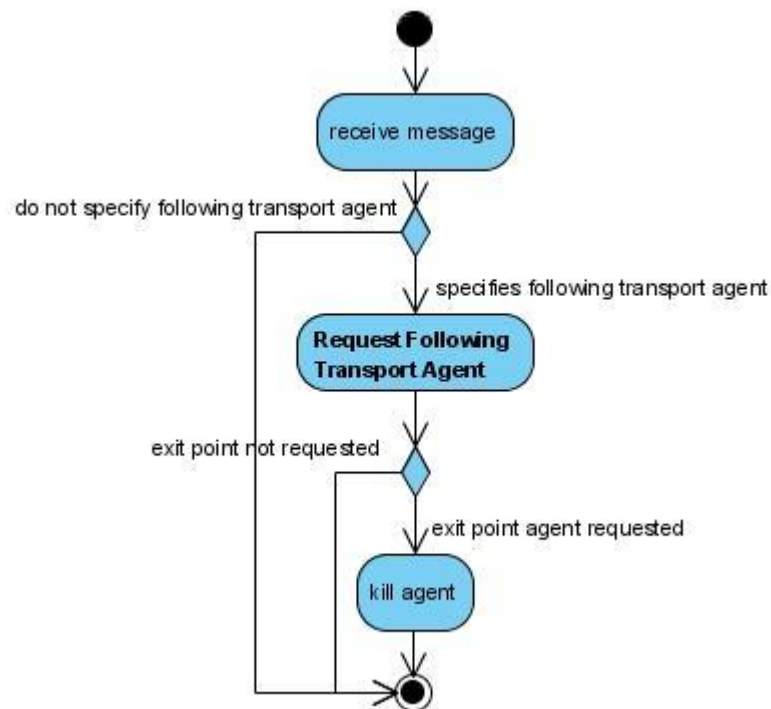


Figure 78: Receive Informs Activity Diagram of the Exit Handler Agent

3.5.7 Sensorial Agent

The sensorial agent is in charge of getting sensor's information from the physical system. Due to that it has to distribute sensor's information to the other agents. Its behaviours are presented in Figure 79.

The agent starts by initializing agent's information, then registers itself and the provided services that for this case it consists in providing sensor's information. The following behaviours characterize agent's operation and are get sensors info and heartbeat signalling; heart beat signalling is used to send a "liveness signal" and its behaviour is described in Figure 80; get sensors info behaviour is in charge of managing all sensor's data and is present in Figure 81.

Heartbeat behaviour sends messages periodically and is based in the protocol IPC 2541 where periodic messages are sent to specify that this agent is not blocked, this means that the standard is encapsulated over an ACL message so that is FIPA compliant. This behaviour is implemented in this agent since the complete MAS relies on the information it provides and thus requires being always working. With the behaviour of sending periodic messages, it is possible to check when the agent is blocked and act upon since for those cases no message is sent. Considering an agent for the task of monitoring increases robustness, its name is Sensorial agent and is presented below.

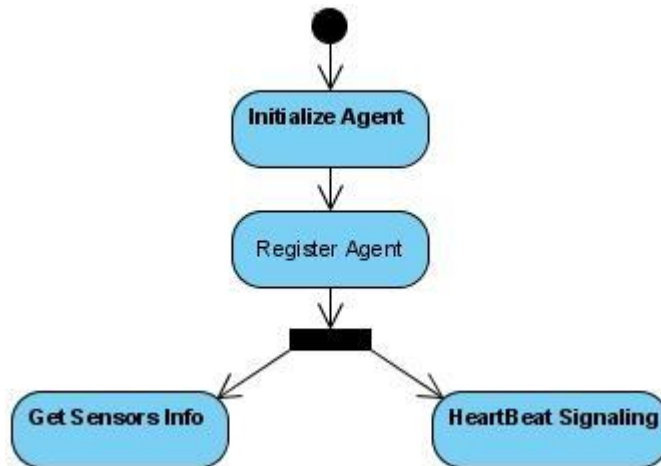


Figure 79: Behaviours of the Sensorial Agent

The behaviour for managing sensor's information consists in getting all sensor data, analyse it by comparing to the previous sample, if has changes send the information to the respective agents, once all sensor data e checked the process is repeated.

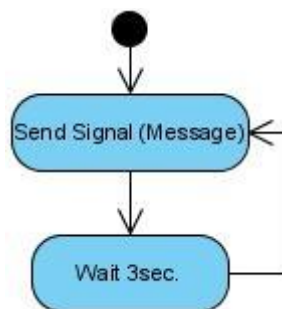


Figure 80: HeartBeat Signaling Activity Diagram of the Sensorial Agent

The Sensorial agent solves the problem of concurrent access to a shared resource since no other agent accesses the physical system to get sensor information therefore, information is more reliable with the disadvantage making the system less robust to failures since the complete system depends on its operation. However heartbeat signalling behaviour and the monitor agent reduce risks.

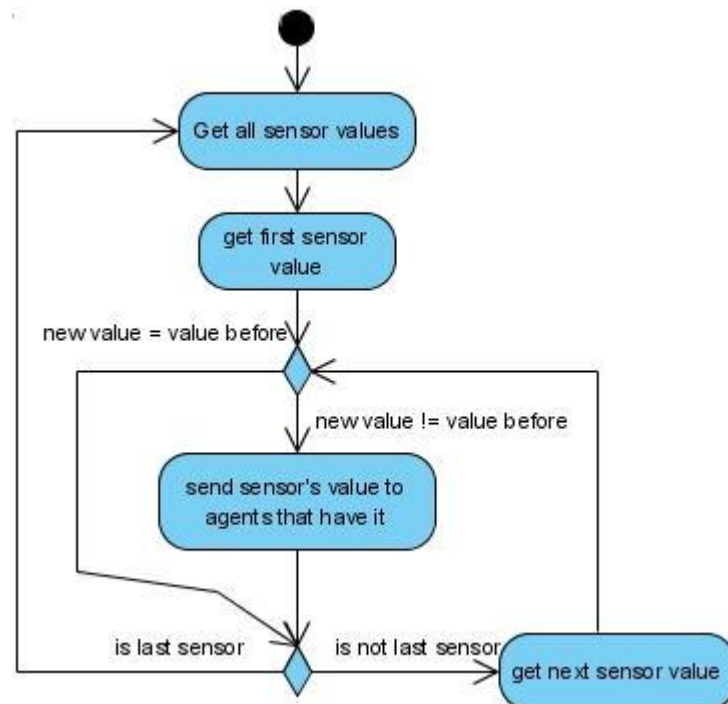


Figure 81: Get sensor info Activity Diagram of the Sensorial Agent

3.5.8 Monitor Agent

The monitor agent is the agent that checks if other agents are blocked or have died, its operation is based in the protocol IPC 2541. The protocol is not implemented but rather its concept of sending liveness messages, on the other side, the entity that receives messages periodically checks the arrival of those messages and once there is no message it means that the agent had a problem, therefore it has to be replaced. Agent's reaction to a missing message consists in killing the agent which message was expected and then launch a new one. The agent is killed since it was not performing as expected when failing to send the message on time whether because it was blocked or for any other reason.

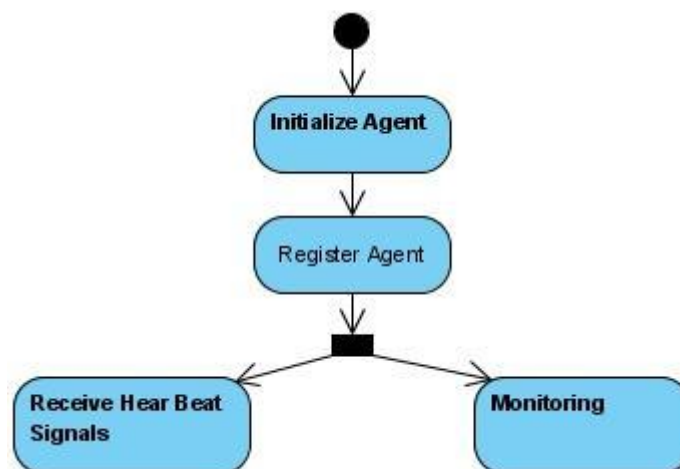


Figure 82: Activity Diagram of the Monitor Agent

The activity diagram that represents agent's behaviour is presented in Figure 82, it starts with agent's initialization where the agent knowledge is analyzed, then it registers itself and the provided services, and finally the two behaviours that represent agent main behaviour are launched, namely receive heartbeat signals and monitoring.

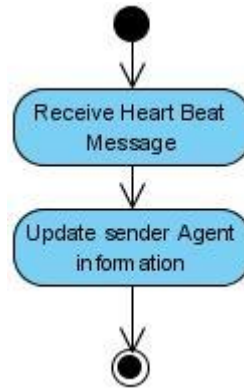


Figure 83: Heart Beat Activity Diagram of the Monitor Agent

Receive heartbeat signals behaviour has the simple task of receiving periodic messages from other agents, specifying that the sender is alive and working, its activity diagram is presented in Figure 83.

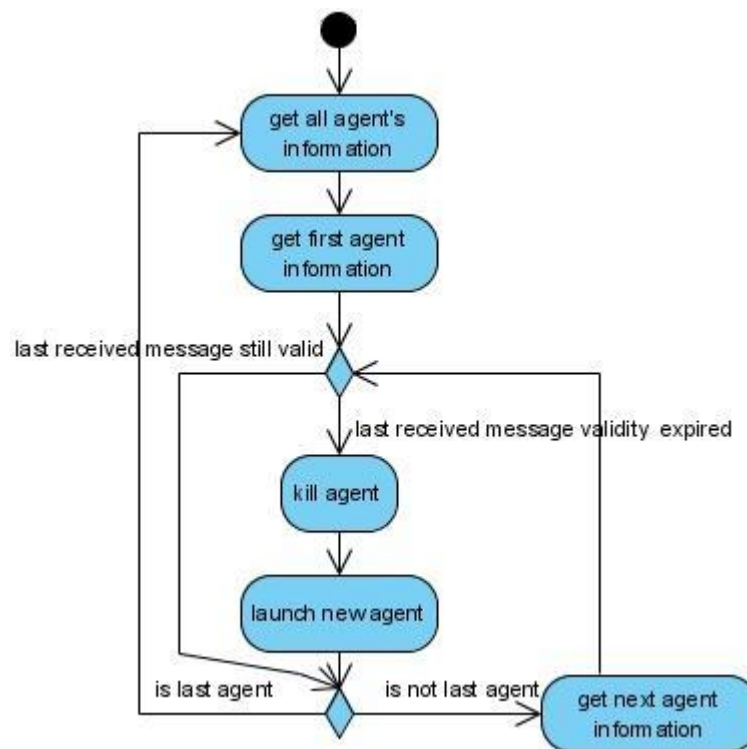


Figure 84: Monitoring Activity Diagram of the Monitor Agent

The monitoring behaviour performs the validation of the last received message, for any out of date message, the respective sender agent is substituted, its activity diagram is present in Figure 84. The behaviour consists in getting all agents information regarding

the previous sent messages and verifying if at the present time the last received message is still valid, if not, the agent is deleted and a new one is launched.

3.6 Interface to physical controller

The physical controller is an Omron PLC CS1G-CPU43H with an Ethernet unit CS1W-ETN01 assigned to each cell. For the system to be controlled, requires handling commands from the MAS, therefore it requires a solution so that both controllers receive agent commands. Of relevance is those commands belong to the available operations present at the class *ControlOperations* from device's ontology so that the MAS knows how to control the physical system. To better understand the position of the required interface in the system verify Figure 85.

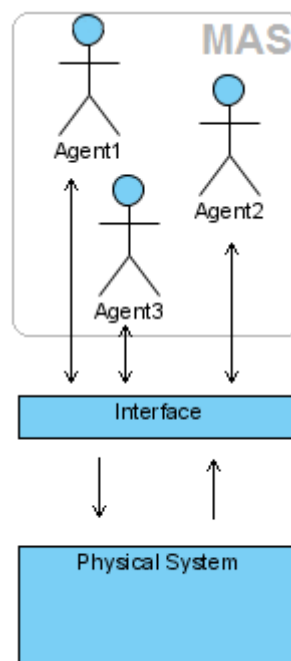


Figure 85: Interaction between agents and the physical system

The solution for agents to send controlling commands is based in a DLL which perform the communication with the PLC, from PLC's point of view it requires a way to receive orders and actuate upon. Actuation consists in writing to specific PLC's memory positions which are assigned to control each device, therefore writing on those memory positions actuate on devices, as such it creates a memory interface that is mapped to the positions that effectively actuate on the system. Memory mapping is presented in Table 5, Table 6 and Table 7 and it is a memory area named CIO.

Table 5 Device's status memory mapping area

		Bit	Description	Maps to
Reading area	Presence sensor area	10.00	S30_BES	3310.00
		10.01	S31_BES	3310.01
		10.02	SM10_Wenglor	3338.00
		10.03	SM11_Wenglor	3338.01
		10.04	SM12_SMC	3338.02
		10.05	SM13_SMC	3338.03
		10.06	SM14_Wenglor	3338.04
		10.07	SM17_Wenglor	3338.07
		10.08	SM18_Wenglor	3338.08
		10.09	SM19_WenglorCell	3338.09
		////////////////////////////////////	////////////////////////////////////	
	Robot area	10.11	Robot (On/Off)	3340.03
		10.12	Robot Initialized	3340.04
		10.13	Robot Performing Task	3340.06

In respect to the memory area to actuate on the devices Table 6 shows it.

Table 6 Actuating memory mapping area

		Bit	Description	Maps to
Actuating area		20.00	YM10_StopperCell2	3238.00
		21.00	YM11_Diverter	3238.01
		22.00	YM12_Lifter	3238.02
		23.00	YM14_Stopper	3238.04
		24.00	YM17_Stopper	3238.07
		25.00	YM18_Stopper	3238.08
		26.00	U151_S1_ServConv2Motor	3244.06
		27.00	U151_S2_ServConv2Speed	3244.07
		28.00	U152_S1_ByPassConv2Motor	3244.08
		29.00	U152_S2_ByPassConv2Speed	3244.09
		30.00	Y21_InterCellConv2Motor	3210.06
		31.00	Y22_InterCellConveyor2Speed	3210.07
		32.00	////////////////////////////////////	////////////////////////////////////
		33.00	Safety Switches	3244.04 and 3244.05
		33.01	Signal Online Mode for Robot	3240.00
		33.02	Select Robot Operation bit 1	3240.01
		33.03	Select Robot Operation bit 2	3240.02
		33.04	Enable Robot Operation	3240.03

In respect to the memory mapping for the RFID devices Table 7 presents it.

Table 7 Sensing and Actuating memory mapping area of RFID

RFID 1 area (Reading and Actuating)	Bit	Description	Maps to
	11.00	Enable ID1 (Trigger)	3233.15
	11.01	Read/Write ('0'/'1')	3233.14
	11.02	Error bit	3333.13
	Address	Description	Maps to
	12	Read value (8bits)	3332
	13	Write value (8bits)	3232

RFID 2 area (Reading and Actuating)	Bit	Description	Maps to
	14.00	Enable ID2 (Trigger)	3235.15
	14.01	Read/Write ('0'/'1')	3235.14
	14.02	Error bit	3335.13
	Address	Description	Maps to
	15	Read value (8bits)	3334
	16	Write value (8bits)	3234

RFID 3 area (Reading and Actuating)	Bit	Description	Maps to
	17.00	Enable ID3 (Trigger)	3237.15
	17.01	Read/Write ('0'/'1')	3237.14
	17.02	Error bit	3337.13
	Address	Description	Maps to
	18	Read value (8bits)	3336
	19	Write value (8bits)	3236

With methods for reading and writing operations based on commands, agents are able to apply the control to the physical system. FINS commands are a proprietary solution from PLC's manufacturer and it allows operations of reading and writing on server side without the need of programming them, therefore FINS commands are the selected solution to communicate with the PLC. Through writing operations it is possible to change device's state and through reading operations it is possible to get device's state. The communication is supported by a local ethernet network over which UDP packets (User Datagram Packets) are exchanged with FINS commands, where the MAS is the client-side and the PLC the server-side. The PLCs applied to control each of the cells is the same therefore are differentiated in terms of the IP address.

With this approach agents have the tools to sense and actuate on the physical system, for a more complete description of FINS commands or other specification of the controller verify manufacturer's manual.

3.7 The Solution Testbed

The objective of the test consists in producing three products with the same characteristics where the MAS cooperate in order to achieve that goal based on the knowledge provided by the ontology. The product consists in three parallel pieces of Lego and the required manufacturing processes consist in changing piece's order, to manufacture products operation require being done in a specific order. Robots present in the production system provide the same services therefore, when one robot is busy the other can provide the service, if this one is also busy the pallet will be will be routed around the system waiting robot's availability. System's entry point is considered to be the Decision Point 2, and the exit point the end of the Inter Cell Conveyor 1, for both cases pallets are added and removed manually.

The MAS is composed by an agent per main device and these are an agent per conveyor, per diverter, per robot and per pallet.

4. The Testbed Results

This chapter presents the results of the testbed executed in order to prove the developed solution.

The controlling MAS starts by launching the agents that compose the MAS, operation done by the launcher agent, and Figure 86 presents its log window. Launcher agent operation starts by registering itself, provided services, then launches all agents, deregisters and ends its operation.

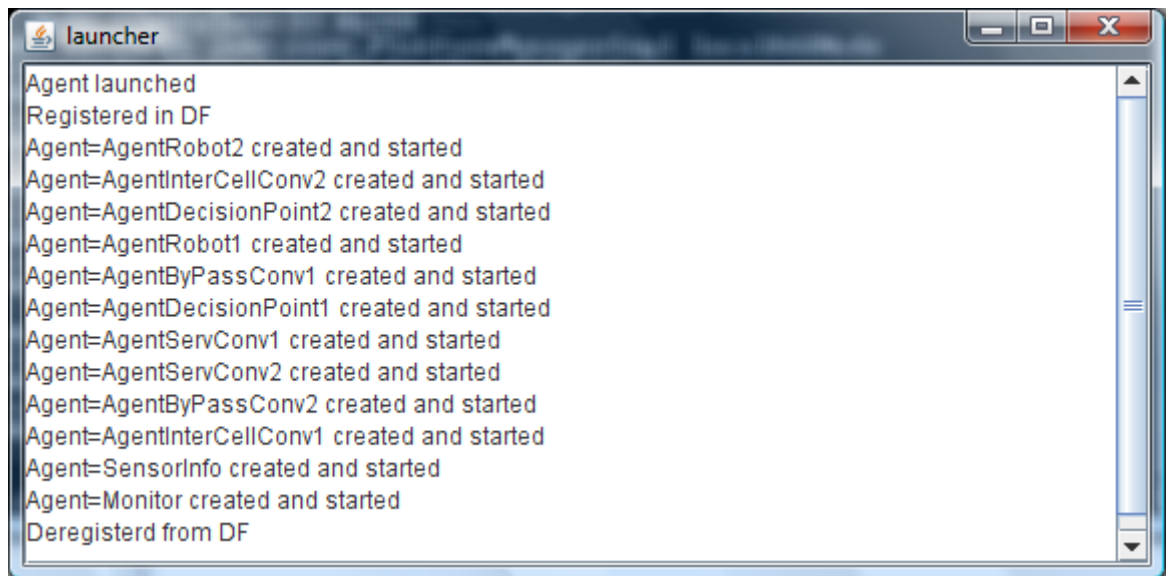


Figure 86: Log Window of the Launcher Agent operation

Once all device agents are launched the system is ready to perform its operation where launched agents are conveyor, decision point, and robot. At this stage, there is no activity in the system since there is no trigger, launched agents are mainly service providers. The trigger is introduced by pallet agent since it requires services from the other agents, an example launching a pallet agent is present in Figure 87 and once again its operation is performed by the launcher agent.

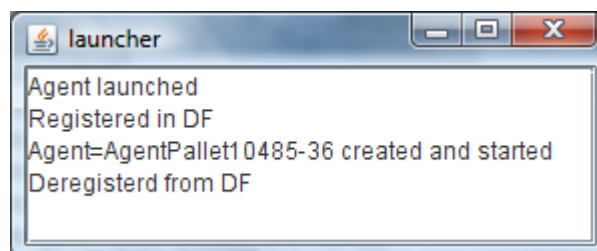


Figure 87: Log Window of the Launcher Agent launching a Pallet Agent

Figure 88 presents the log window information of the operation of a pallet agent, it starts by initializing the agent which consists in registering the agent as well as the provided services, once registered it gets the first required manufacturing process to perform on the product which name is AssemblePart1 and is served by AgentRobot1. With the location information retrieved from the service request, the routing destination is updated. With the updated location destination the agent then performs the requests to the transport agents in order to get to the destination, an example of the service request content is presented in Figure 89.

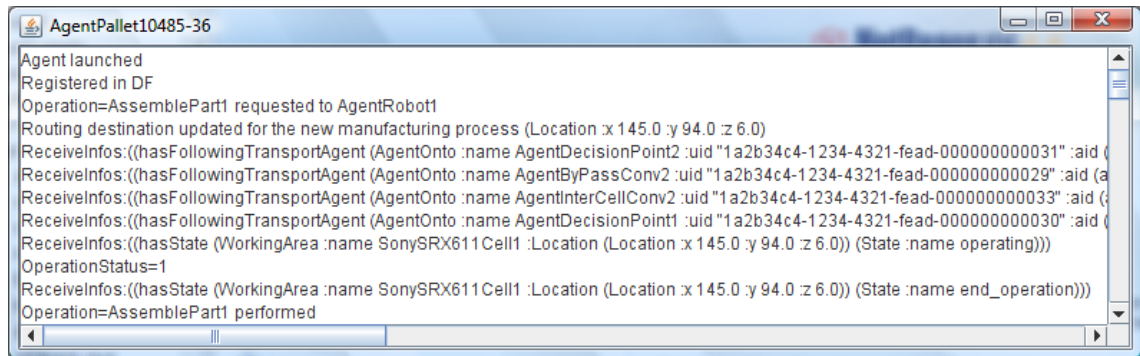


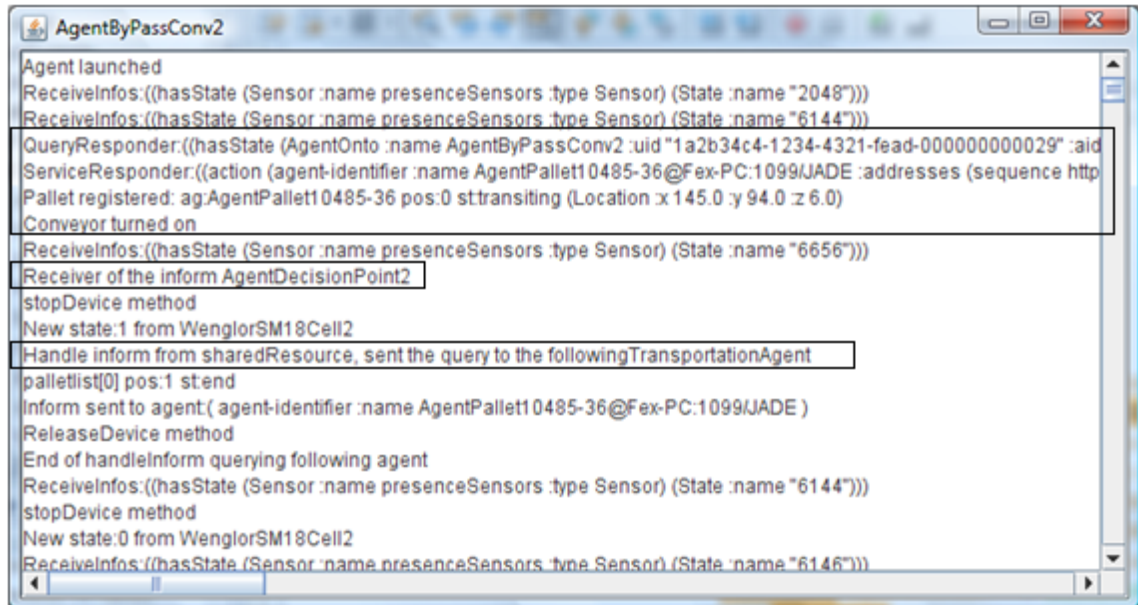
Figure 88: Log Window of a Pallet Agent running

As can be seen by “ReceiveInfos” in Figure 88, it represents the information received by pallet agent with respect to the transport agents which are proving it the transport service. Upon reading desired working area the operation starts to being performed, as such pallet agent and respective transport agent are informed in respect to operation status. This cycle, searching for a service, reaching it provider and consume the service are the fulfilled requirements of one manufacturing process, and it is repeated according to the number of needed manufacturing processes.



Figure 89: Content of a Service Request Message

The operation of a bypass conveyor can be visualized in Figure 90 in terms of the agent log window, its operation starts by receiving a *Query* message from the previous transport agent since it wishes to know its availability. The messages received before the *Query* message are in respect to sensor’s information, and since the conveyor has free places answers it positively and then, it accepts a service *Request* from the pallet that has left the previous conveyor which is confirmed by the log ‘Pallet Registered’. Once the pallet gets on the conveyor there is new sensor information and after it, the previous transport device is acknowledged with an *Inform* message, so confirm that the pallet was correctly delivered.



```

Agent launched
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "2048")))
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
QueryResponder:((hasState (AgentOnto :name AgentByPassConv2 :uid "1a2b34c4-1234-4321-fead-000000000029" :aid
ServiceResponder:((action (agent-identifier :name AgentPallet10485-36@Fex-PC:1099/JADE :addresses (sequence http
Pallet registered: ag:AgentPallet10485-36 pos:0 st:transiting (Location :x 145.0 :y 94.0 :z 6.0)
Conveyor turned on
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6656")))
Receiver of the inform AgentDecisionPoint2
stopDevice method
New state:1 from WenglorSM18Cell2
Handle inform from sharedResource, sent the query to the followingTransportationAgent
palletist[0] pos:1 st:end
Inform sent to agent: ( agent-identifier :name AgentPallet10485-36@Fex-PC:1099/JADE )
ReleaseDevice method
End of handleInform querying following agent
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
stopDevice method
New state:0 from WenglorSM18Cell2
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6146")))

```

Figure 90: Log Window of the ByPass Conveyor 2 Agent running

ByPass conveyor has one working area and since the desired destination is not at this location it has to act in order to deliver the pallet to the following transport agent. The shared area is at its end, due to that the agent that competes for it has to be queried in order to know if the area is free; for a positive answer, the area became reserved, otherwise the procedure is delayed and repeated. Then, the following transport agent is queried to know if is available for one more pallet, at the arrival of a positive answer, the pallet agent is then informed with the following transport agent information to requests it; otherwise the process is delayed and repeated.

For an inter cell conveyor, the operation is very similar since it also has one working area but it do not having a shared area, the process of querying shared area agent does not belong to its operation which is presented in Figure 91 in terms of its log window.

A conveyor that has more than one working area is the service conveyor; its working operation is presented in Figure 92 and Figure 93. The presented log window does respect to the provision of a service to a pallet that requested stopping at its second working area. In Figure 92 it is possible to verify the service request from the pallet agent and the conveyor turning on since it was empty; then, the previous transport agent is informed about a successfully delivery and the pallet is blocked. Since it wishes to stop at the second working area the pallet is released and stopped once again at the second working area, it will be released upon receiving a message specifying an end of operation at that working area. Once the operation is ended, it starts the process of delivering the pallet to the following transport agent since pallet is in the last position.

```

AgentInterCellConv2
Agent launched
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "2048")))
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6656")))
QueryResponder:((hasState (AgentOnto :name AgentInterCellConv2) (State :name empty)))
ServiceResponder:((action (agent-identifier :name AgentPallet10485-36@Fex-PC:1099/JADE :addresses (sequence http://130.230.
Pallet registered: ag:AgentPallet10485-36 pos:0 st:transiting (Location x 145.0 y 94.0 z 6.0)
Conveyor turned on
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6146")))
Receiver of the inform AgentServConv2
Receiver of the inform AgentByPassConv2
stopDevice method
New state:1 from BES-S31 Cell2
palletlist[0] pos:1 st:end
Inform sent to agent ( agent-identifier :name AgentPallet10485-36@Fex-PC:1099/JADE )
ReleaseDevice method
End of handleInform querying following agent
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
stopDevice method
New state:0 from BES-S31 Cell2
ReceiveInfos:((hasState (AgentOnto :name AgentDecisionPoint1 :uid "1a2b34c4-1234-4321-fead-000000000030" :aid (agent-ident
QueryResponder:((hasPallets (AgentOnto :name AgentInterCellConv2 :uid "1a2b34c4-1234-4321-fead-000000000033" :aid (agent-

```

Figure 91: Log Window of the Agent Inter Cell Conveyor 2 running

```

AgentServConv2
QueryResponder:((hasState (AgentOnto :name AgentServConv2 :uid "1a2b34c4-1234-4321-fead-000000000039" :aid (agent-ident
ServiceResponder:((action (agent-identifier :name AgentPallet10485-40@Fex-PC:1099/JADE :addresses (sequence http://130.230.
Pallet registered: ag:AgentPallet10485-40 pos:0 st:transiting (Location x 5.0 y 130.0 z 6.0)
Conveyor turned on
ReceiveInfos:((hasState (Sensor :name ID2Cell2 :type Sensor) (State :name "40")))
Receiver of the inform AgentDecisionPoint2
ReleaseDevice method
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6146")))
ReceiveInfos:((hasState (Sensor :name ID2Cell2 :type Sensor) (State :name "0")))
stopDevice method
ReceiveInfos:((hasState (AgentOnto :name AgentInterCellConv2) (State :name delivered)))
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6400")))
stopDevice method

```

Figure 92: Log Window of the Agent Service Conveyor 2 running 1

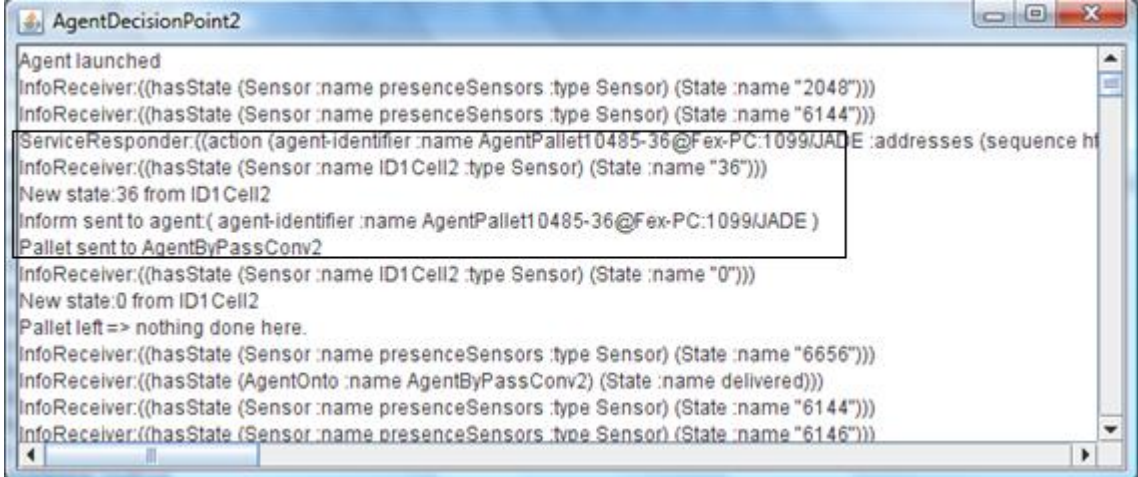
```

AgentServConv2
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "14336")))
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
ReceiveInfos:((hasState (WorkingArea :name SonySRX611Cell2 :Location (Location x 5.0 y 130.0 z 6.0)) (State :name empty)))
end_operation from AgentPallet10485-40
stopDevice method
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6146")))
ReceiveInfos:((hasState (AgentOnto :name AgentInterCellConv2) (State :name delivered)))
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
Handle inform from sharedResource, sent the query to the followingTransportationAgent
palletlist[0] pos:2 st:end
Inform sent to agent ( agent-identifier :name AgentPallet10485-40@Fex-PC:1099/JADE )
ReleaseDevice method
End of handleInform querying following agent
ReceiveInfos:((hasState (Sensor :name ID3Cell2 :type Sensor) (State :name "0")))
stopDevice method
ReceiveInfos:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6146")))
ReceiveInfos:((hasState (AgentOnto :name AgentInterCellConv2) (State :name delivered)))

```

Figure 93: Log Window of the Agent Service Conveyor 2 running 2

Other device that belongs to the transport system is the diverter as well as its agent decision point is a transport agent, its operation is presented in Figure 94.



```


AgentDecisionPoint2
Agent launched
InfoReceiver:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "2048")))
InfoReceiver:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
ServiceResponder:((action (agent-identifier :name AgentPallet10485-36@Fex-PC:1099/JADE :addresses (sequence ht
InfoReceiver:((hasState (Sensor :name ID1Cell2 :type Sensor) (State :name "36")))
New state:36 from ID1 Cell2
Inform sent to agent:( agent-identifier :name AgentPallet10485-36@Fex-PC:1099/JADE )
Pallet sent to AgentByPassConv2
InfoReceiver:((hasState (Sensor :name ID1Cell2 :type Sensor) (State :name "0")))
New state:0 from ID1 Cell2
Pallet left => nothing done here.
InfoReceiver:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6656")))
InfoReceiver:((hasState (AgentOnto :name AgentByPassConv2) (State :name delivered)))
InfoReceiver:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6144")))
InfoReceiver:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "6146")))

```

Figure 94: Log Window of the Agent Decision Point running

Figure 94 presents the log window for the *Request* of a pallet agent that wishes its services, once registered the pallet gets to the diverter and it triggers new sensor's information as can be seen on the log window its RFID value is 36, upon confirming this value, it starts the decision making process to get destination transport agent and when obtained pallet agent is informed.

For the case of a RFID tag value not corresponding to a registered pallet it means that the present pallet is not synchronized or it does not belong to the system, therefore it has to be routed to an exit point. To route the pallet to the exit point the agent exit handler is lunched and its operation is present in Figure 95, if the pallet has an agent associated, this one is deleted and the exit handler substitutes it.



```

AgentDecisionPoint2
Agent launched
Registered in DF
InfoReceiver:((hasState (Sensor :name presenceSensors :type Sensor) (State :name "0")))
InfoReceiver:((hasState (Sensor :name ID1Cell2 :type Sensor) (State :name "40")))
New state:40 from ID1 Cell2
Pallet ID=40, not registered
Unknown pallet agent does not exist in the system
Launching a new agent, name=palletExitHandler40
Agent started
Inform sent to agent:( agent-identifier :name palletExitHandler40@Fex-PC:1099/JADE )
Pallet sent to AgentByPassConv2

```

Figure 95: Log Window of the Agent Decision Point 2 launching an Exit Handler Agent

The operation of an exit handler agent is presented in Figure 96. From the figure is possible to check the path that it had since its log window contains the transport agents to which the agent has been on, and once the exit point is reached, the agent deregisters and deletes itself.

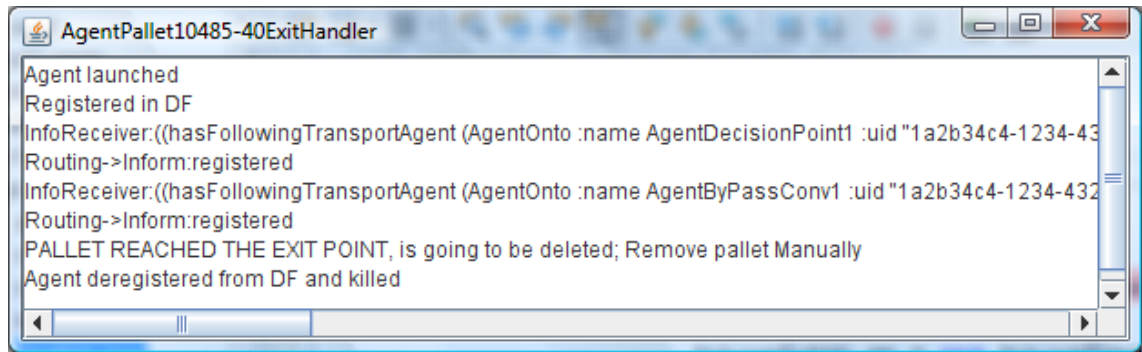


Figure 96: Log Window of the Exit Handler Agent

The operation of the robot agent is divided into several figures, Figure 97 presents initialization, Figure 98 the beginning of operation of a manufacturing process and Figure 99 its end.

In the initialization phase, the robot is physically initialized, once ready it registers itself and the provided services, then it starts its main operation. Upon receiving a service *Request* from a pallet agent if accepted is answered positively and the agent became then committed in proving the requested service, i.e. the request is registered. When the pallet arrives at robot's working area the operation is started. The messages exchanged for this task can be visualized in Figure 98. When the operation is accomplished, an inform message is sent for the pallet and conveyor agent informing both of the end of operation, its log window is presented in Figure 99.

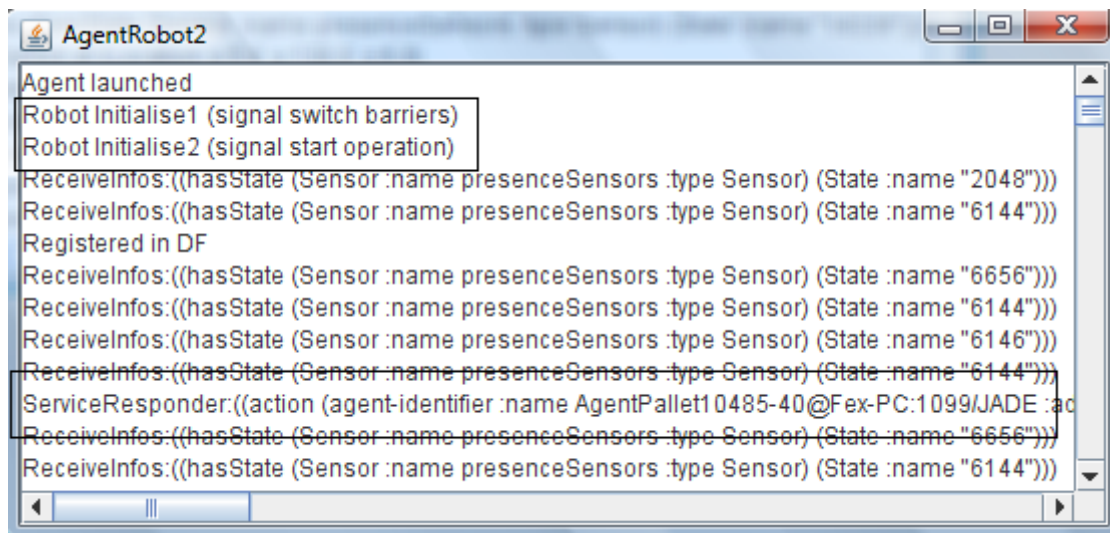


Figure 97: Log Window of robot initialization of the Robot Agent

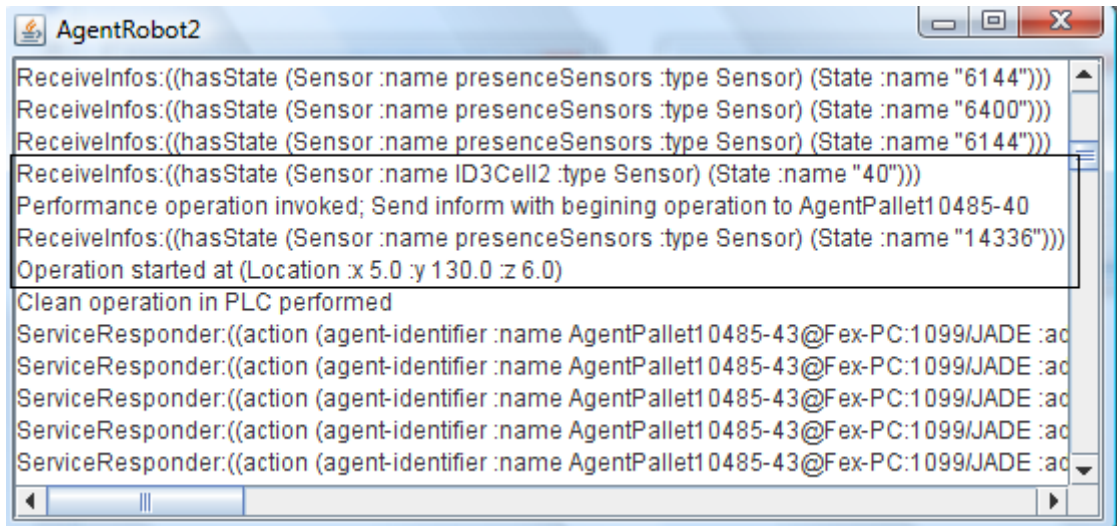


Figure 98: Log Window of a start operation of the Robot Agent

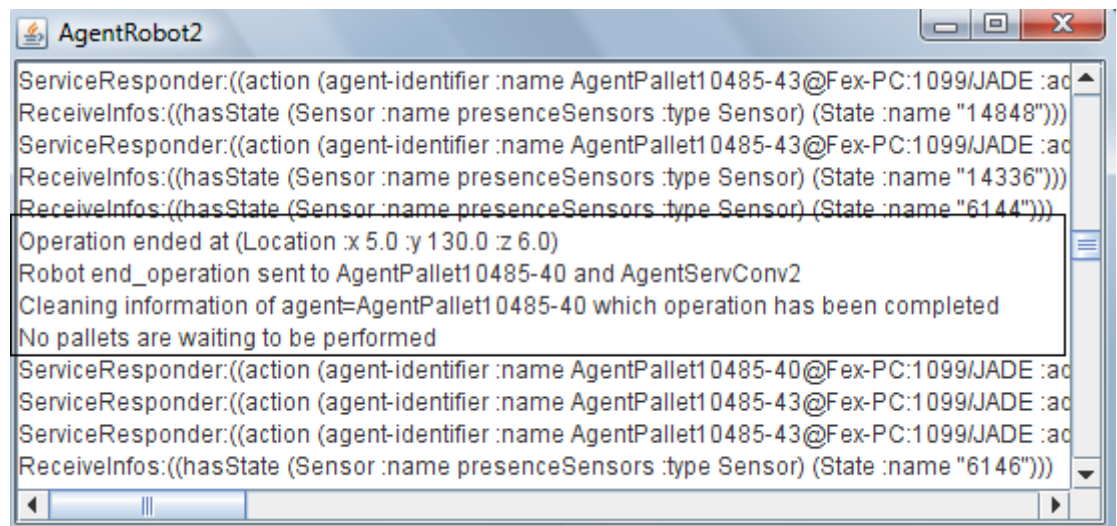


Figure 99: Log Window of an end operation of the Robot Agent

Once the operations are performed on the product, the respective pallet agent reports that it has ended its operation, its log window is presented by Figure 100.

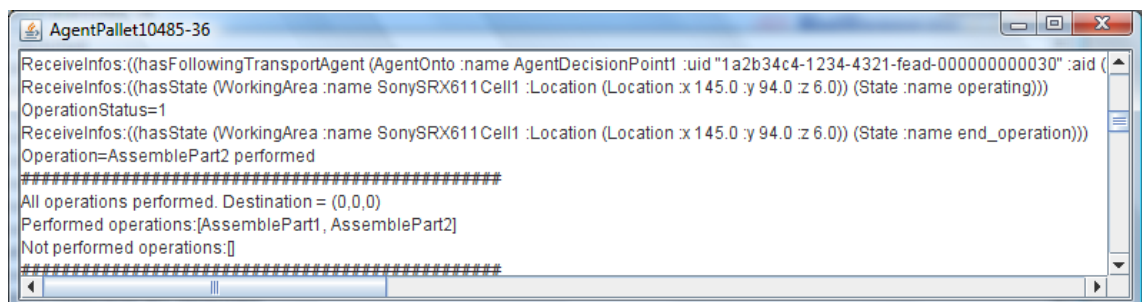
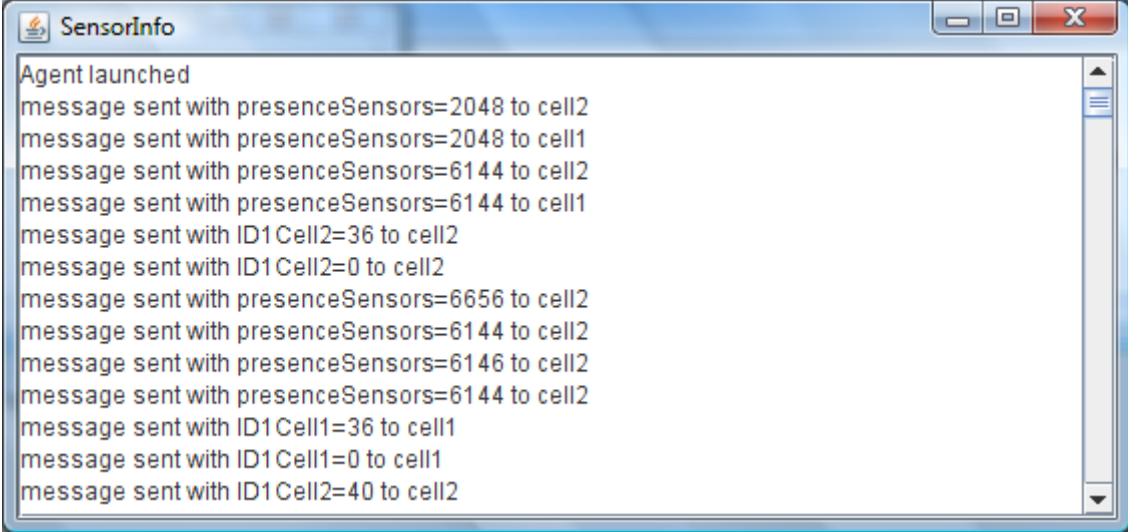


Figure 100: Log Window of Pallet agent when operations completed

In order to sensor data reach each agent, sensorial agent has to be running to get data and distribute information through the MAS, a resume of its operation is presented in Figure 101.



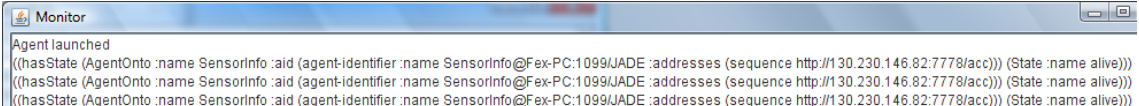
```

Agent launched
message sent with presenceSensors=2048 to cell2
message sent with presenceSensors=2048 to cell1
message sent with presenceSensors=6144 to cell2
message sent with presenceSensors=6144 to cell1
message sent with ID1 Cell2=36 to cell2
message sent with ID1 Cell2=0 to cell2
message sent with presenceSensors=6656 to cell2
message sent with presenceSensors=6144 to cell2
message sent with presenceSensors=6146 to cell2
message sent with presenceSensors=6144 to cell2
message sent with ID1 Cell1=36 to cell1
message sent with ID1 Cell1=0 to cell1
message sent with ID1 Cell2=40 to cell2

```

Figure 101: Log Window of the Sensorial Agent running

The operation performed by the sensorial agent is of great importance because the operation of the complete system depends on it, the agent has to be running without problems, and to guarantee that, the agent monitor is also running and its operation is represented in terms of its log window in Figure 102 and Figure 103; Figure 102 presents the operation of receiving messages; Figure 103 presents the operation of replacing sensorial agent.

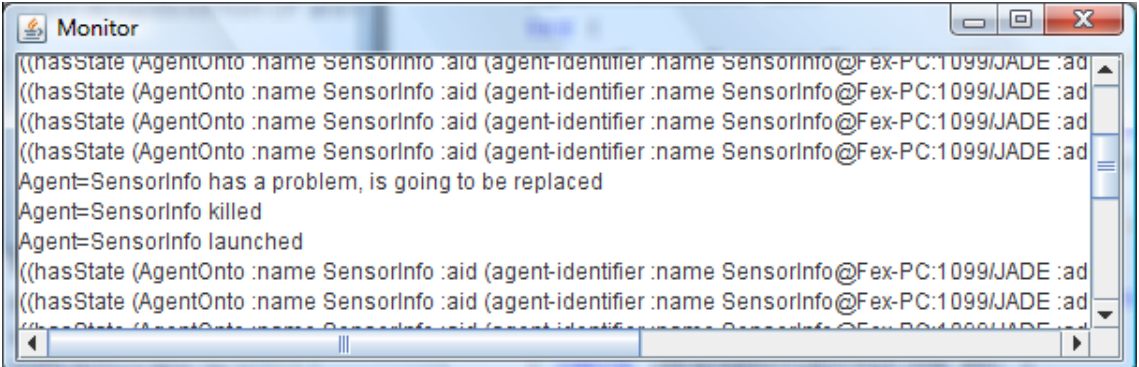


```

Agent launched
((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))
((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))
((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))

```

Figure 102: Log Window of the Monitor Agent receiving messages



```

((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))
((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))
((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))
((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))
Agent=SensorInfo has a problem, is going to be replaced
Agent=SensorInfo killed
Agent=SensorInfo launched
((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))
((hasState (AgentOnto :name SensorInfo :aid (agent-identifier :name SensorInfo@Fex-PC:1099/JADE :addresses (sequence http://130.230.146.82:7778/acc))) (State :name alive)))

```

Figure 103: Log Window of the Monitor Agent substituting Sensorial Agent

5. Conclusions and Future Work

The development of this thesis includes an applied effort in order to integrate different technologies, from the higher level to the lowest with the purpose of:

1. Modelling the ontology and make it accessible and useful at the agents' domain.
2. Model the agents and make them interact with the controller of the physical system
3. Control the physical system with its controller

With respect to the objectives stated in the sub-chapter 1.2.3.1 those have been achieved as follows:

1. Creation of an ontology representing the manufacturing system in terms of its devices and control entities.
2. Integration of the ontology with the control entities of the manufacturing system.
3. Development of a Multi-Agent System that controls the manufacturing system in terms of its devices based on the knowledge obtained from the ontology.
4. Integrate the Multi-Agent System with the physical manufacturing system.
5. Application of the Multi-Agent System in a test case scenario to control the manufacturing system based on the ontology.

Having a control that is independent of the physical system allows it to be applied to different manufacturing systems with a much smaller setup time when compared to re-configuring or changing, existing or legacy systems since the control rely on system's ontology. For agents to actuate on the physical controller an interface is required which is unique for each physical system, creating this interface is time consuming when compared to developing a system's ontology, this is the solution's bottleneck. However, solution's bottleneck require a smaller effort in opposite to the effort required to change current or legacy manufacturing systems, these require a huge amount of reprogramming and reconfiguring.

Other advantage of this solution is that controlling physical systems with agents instead of a single physical controller enhances the level at which physical system's information and control actions are available. With the system's information available at this level it is easier for companies to integrate their Enterprise Resource Planning (ERP) systems with the controlling system, it allows managers to access directly to the shop floor's information, easily adapts to production plans and it also allows remote control and monitoring of the manufacturing system.

Due to the limitation of the physical system it was not possible to prove its scalability, more specifically the decision point agents might need to route pallets from several conveyors and deliver them to one of several possibilities.

In terms of future work there are key points to improve. Adding location information to the pallet when performing requests to service providers can be valuable during the decision making process of choosing the provider to request a service so that the closest provider is requested, therefore increasing efficiency and performance.

Adding traffic information to the transport system could also be a good improvement as well as for the transport system; however it is necessary to consider the amount of network traffic that it adds to make this information available, several protocols from telecommunications are developed and could be used at this point.

Also some changes could be done to the MAS in respect to the transport agents and towards increasing the level of its adaptability to production reconfiguration. A possible solution could be to define behaviours that allow changing the obtained knowledge at launch time so that the transport system is not launched when changes are required. Also the equivalent behaviours could be done to the agents that have devices attached to, so that these could also be changed on the fly.

6. Bibliography

Barata, J., L. Camarinha-Matos, and M. Onori. "A Multiagent Based Control Approach for Evolvable Assembly Systems." *3rd IEEE International Conference on Industrial Informatics*. IEEE, 2005. 478- 483.

—. "A Multiagent Based Control Approach for Evolvable Assembly Systems." *3rd IEEE International Conference on Industrial Informatics*. IEEE, 2005. 478- 483.

Becta. *Emerging technologies for learning*. Becta, 2007.

Bussman, S., N. Jennings, and M. Wooldridge. *Multiagent Systems for Manufacturing Control - A Design Methodology*. Springer Series on Agent Technology, 2004.

Caire, Giovanni, and David Cabanillas. *JADE TUTORIAL - APPLICATION-DEFINED CONTENT LANGUAGES AND ONTOLOGIES*. Tutorial, CSELT S.p.A. & Tilab S.p.A., 2006.

Clark&Parsia. *Pellet Features*. 2004. <http://clarkparsia.com/pellet/features> (accessed July 2009).

Colombo, Armando W., Ralf Neuberg, and Ronald Schoop. "A Solution to Holonic Control Systems."

Corcho, O., M. Fernández-Lopez, and A. Gómez-Pérez. *Ontological Engineering: Principles, Methods, Tools and Languages*. Springer Berlin Heidelberg, 2006.

Dale, Jonathan, Johnny Knottenbelt, and Fujisu Labo. *April Agent Platform*. <http://design-studio.lookin.at/research/relate%20survey/Survey%20Agent%20Platform/April%20Agent%20Platform.htm> (accessed 08 08, 2009).

DAML-OIL-W3C, World Wide Web Consortium. *Annotated DAML+OIL Ontology Markup*. 18 December 2001. <http://www.w3.org/TR/daml+oil-walkthru/> (accessed July 2009).

DIG Standard. <http://dl.kr.org/dig/interface.html> (accessed July 2009).

Elammari, M., and W. Lalonde. "An Agent Oriented Methodology: High-Level and Intermediate Models." *International Workshop on Agent-Oriented Information Systems*. Heidelberg, Germany, 1999.

- . “An Agent Oriented Methodology: High-Level and Intermediate Models.” *International Workshop on Agent-Oriented Information Systems*. Heidelberg, Germany, 1999.
- Finkenzeller, Klaus. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. John Wiley & Sons, Ltd, 2003.
- FIPA, Architecture Board. *FIPA ACL Message Structure Specification*. Standard Specification, FIPA, 2002.
- FIPA, Architecture Board. *FIPA Query Interaction Protocol Specification*. Standard Specification, FIPA, 2002.
- FIPA, Architecture Board. *FIPA Request Interaction Protocol Specification*. Standard Specification, FIPA, 2002.
- FIPA-OS*. 1999. <http://fipa-os.sourceforge.net/index.htm> (accessed July 2009).
- Fraden, Jacob. *Handbook of Modern Sensors: Physics, Designs, and Applications*. Springer, 2004.
- Franklin, S., and A. Graesser. “Is It an agent, or just a program?: A taxonomy for autonomous agents.” In *Intelligent Agents III Agent Theories, Architectures, and Languages*, by Springer, 21-35. Springer Berlin / Heidelberg, 2006.
- Frey, Daniel, Jens Nimis, Heinz Wörn, and Peter Lockemann. “Benchmarking and robust multi-agent-based production planning and control.” In *Engineering Applications of Artificial Intelligence Volume 16, Issue 4*, by G. Morel and B. Grabot, 307-320. Elsevier, June 2003.
- Gasevic, D., D. Djuric, and V. Devedzic. *Model Driven Architecture and Ontology Development*. Springer, 2006.
- Glanzer, K., A. Hammerle, and R. Geurts. “The application of ZEUS agents in manufacturing environments.” *12th International Workshop on Database and Expert Systems Applications*. IEEE, 2001. 628-632.
- Gruber, T. *Ontology (Computer Science)*. 2007. <http://tomgruber.org/writing/ontology-definition-2007.htm> (accessed July 31, 2009).
- Gungui, I., M. Martelli, and V. Mascardi. “DCaseLP: a Prototyping Environment for Multi-Language Agent Systems.” In *Languages, Methodologies and Development Tools for Multi-Agent Systems*, by Springer, 139-155. Springer Berlin / Heidelberg, 2008.
- Gutknecht, O., and J. Ferber. “MadKit: A generic multi-agent platform.” *Fourth international conference on Autonomous agents*. ACM, 2000. 78-79.

- Gutknecht, O., J. Ferber, F. Michel, and S. Mansour. *MadKit*. 2000. <http://www.madkit.org> (accessed July 2009).
- Haarslev, V., R. Möller, and M. Wessel. "Querying the Semantic Web with Racer + nRQL." In *Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04)*. 2004.
- Horrocks, I., et al. *OIL*. 28 November 2008. <http://www.ontoknowledge.org/oil/> (accessed July 2009).
- IBM. *What is ontology? Frequently asked questions*. 2006. <http://www.alphaworks.ibm.com/contentnr/semanticsfaqs> (accessed April 2009).
- Inc, Achronymics. "Reference Manual and User Guide of Agent Builder ." *Reference Manual and User Guide*. Achronymics Inc, 2004.
- Juan, Thomas, Adrian Pearce, and Leon Sterling. "ROADMAP: extending the gaia methodology for complex open systems." *International Conference on Autonomous Agents*. Bologna, Italy: ACM, 2002. 3-10.
- Kaplunova, A., R. Möller, and M. Wessel. "Leveraging the Expressivity of Grounded Conjunctive Query Languages." In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, by Springer - Lectures Notes in Computer Science, 1176-1186. Springer Berlin / Heidelberg, 2007.
- Karp, P., V. Chaudhri, and J. Thomere. *XOL*. 31 August 1999. <http://www.ai.sri.com/pkarp/xol/xol.html> (accessed July 2009).
- Kidd, Paul T. "Agile Manufacturing: A Strategy for the 21st Century." *IEE Colloquium on Agile Manufacturing*, 20 October 1995: 1-6.
- Kinny, David, and Michael Georgeff. "Modelling and design of multi-agent systems." In *Lecture Notes in Computer Science*, by Springer, 1-20. Springer Berlin / Heidelberg, 1997.
- Lastra, J., and A. Colombo. "Engineering framework for agent-based manufacturing control." In *Engineering Applications of Artificial Intelligence*, 625-640. Elsevier, 2006.
- Lastra, José. *Reference Mechatronic Architecture for Actor-based Assembly Systems*. PhD Thesis, Tampere, Finland: Tampere university of Technology, 2004.
- Lemaignan, S., A. Siadat, J.-Y. Dantan, and A. Semenenko. "MASON: A Proposal For An Ontology Of Manufacturing Domain." *Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*. IEEE, 2006. 195 - 200.

- Lohse, N., S. Ratchev, and J. Barata. "Evolvable Assembly Systems - On the role of design frameworks and supporting ontologies." *International Symposium on Industrial Electronics*. IEEE, 2006. 3375-3380.
- López Orozco, Omar J., and José L. Martínez Lastra. "Agent-Based Control Model for Reconfigurable Manufacturing Systems." *IEEE Conference on Emerging Technologies and Factory Automation*. IEEE, 2007. 1233-1238.
- Martínez Lastra, José Luis, Ivan Mateo Delamer, and Fernando Ubiz Lopez. *Domain Ontologies for Reasoning Machines in Factory Automation*. Report number 71, Tampere University of Technology, 2007.
- Masolo, C., S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. "WonderWeb Deliverable D18 - Ontology Library(final)." research project funded by the IST Programme of the Commission of the European Communities as project number IST-2001-33052, 2003.
- Maturana, Francisco P., Raymond J. Staron, and Kenwood H. Hall. "Methodologies and Tools for Intelligent Agents in Distributed Control." *Special Issue of IEEE Journal of Intelligent Systems*, 2005.
- Mehrabi, M., A. Ulsoy, and Y. Koren. "Reconfigurable manufacturing systems: Key to future manufacturing." *Journal of Intelligent Manufacturing*, 2000: 403-419.
- Mizoguchi, R., and Y. Kitamura. "Foundation of Knowledge Systematization: Role of Ontological Engineering." In *Industrial Knowledge Management - A Micro Level Approach*, by Rajkumar Roy Ed. 2000.
- Mönch, L., and M. Stehli. "An Ontology for Production Control of Semiconductor." In *Multiagent System Technologies*, by Springer. Springer, 2004.
- Mönch, Lars, Marcel Stehli, and Jens Zimmermann. "FABMAS: An Agent-Based System for Production Control of Semiconductor Manufacturing Processes." In *Lecture Notes in Computer Science*, 258-267. Berlin: Springer, 2004.
- Muller, J. *The Design of Intelligent Agents: A Layered Approach*. Springer-Verlag New York, Inc., 1996.
- Nguyen, G., T.T Dang, L. Hluchy, M. Laclavik, Z. Balogh, and I. Budinska. *Agent Platform Evaluation and Comparison*. Pellucid 5FP IST-2001-34519, 2002.
- olp.dfki.de. *OntoSelect*. <http://olp.dfki.de/ontoselect?wicket:bookmarkablePage=wicket-0:de.dfki.ontoselect.SearchOntologies> (accessed February 2009).
- Onori, M., J. Barata, and R. Frei. "Evolvable Assembly Systems: Basic Principles." In *Information Technology For Balanced Manufacturing Systems*, by Springer, 317-328. Springer, 2006.

OWL-W3C, World Wide Web Consortium. *Web Ontology Language*. 10 February 2004. <http://www.w3.org/2004/OWL/> (accessed July 2009).

Pan, J.-Y., J. Tenenbaum, and J. Glicksman. "A framework for knowledge-based computer-integrated manufacturing." In *IEEE Transactions on Semiconductor Manufacturing*, 33 - 46. IEEE, 1989.

Pellet Reasoner. <http://clarkparsia.com> (accessed July 2009).

Pouchard, L., N. Ivezic, and C. Schlenoff. "Ontology Engineering For Distributed Collaboration In Manufacturing." *Proceedings of the Artificial Intelligence and Simulation Conference*. 2000.

Pretorius, A. "Ontologies - Introduction and Overview." *Information Visualisation. Eighth International* . IEEE, 2004.

ProtégéWiki. *Protégé Ontologies Library*. 2006. <http://protege.cim3.net/cgi-bin/wiki.pl?ProtegeOntologiesLibrary> (accessed July 2009).

RacerSystems. *RacerPro*. www.racer-systems.com (accessed July 2009).

RacerSystems, GmbH & Co. KG. "Release notes for RacerPro 1.9.2 beta." 2007.

RDF-W3C, World Wide Web Consortium. *RDF*. February 1999. <http://www.w3.org/RDF/> (accessed July 2009).

Ricordel, P.M., and Y. Demazeau. "From Analysis to Development – A Multi-agent Platform Survey." In *Engineering Societies in the Agents World*, by Spring, 93-105. Springer Berlin / Heidelberg, 2000.

SHOE, Parallel Understanding Systems Group, Department of Computer Science, University of Maryland at College Park. *SHOE*. 1999. <http://www.cs.umd.edu/projects/plus/SHOE/> (accessed July 2009).

Sirin, E., and B. Parsia. "SPARQL-DL: SPARQL Query for OWL-DL." *3rd OWLED: Experiences and Directions Workshop (OWLED2007)*. Clark & Parsia, 2007.

Sirin, E., B. Parsia, B. Grau, A. Kalyanpur, and Katz Y. "Pellet: A Practical OWL-DL Reasoner." *Journal of Web Semantics*, 2007.

SPARQL-W3C, World Wide Web Consortium. *SPARQL Query Language for RDF*. 15 January 2008. <http://www.w3.org/TR/rdf-sparql-query/> (accessed July 2009).

Tryllian Agent Development Kit. 1998. <http://www.tryllian.org/> (accessed July 2009).

UMBC, University of Maryland, Baltimore County. *SOOGLE, Semantic Web Search*. 2006. <http://swoogle.umbc.edu/> (accessed July 2009).

Wooldridge. *An Introduction to MultiAgent Systems*. WILEY, 2002.

Wooldridge, Michael, Nicholas R. Jennings, and David Kinny. "The Gaia Methodology for Agent-Oriented Analysis and Design." *Autonomous Agents and Multi-Agent Systems*, 2005: 285-312.

XML-W3C, World Wide Web Consortium. *XML*. 1998. <http://www.w3.org/XML/> (accessed July 2009).

Yusuf, Y.Y., M. Sarhadi, and A. Gunasekaran. "Agile Manufacturing: The drivers, concepts and attributes." *International Journal of Production Economics*, 1999: 33-43.

Zhang, Y., R. Witte, J. Rilling, and V. Haarslev. *Ontological approach for the semantic recovery of traceability links between software artefacts*. IET Software & IEEE, 2007.

ZHANG, Z. *THE ONTOLOGY QUERY LANGUAGES FOR THE SEMANTIC WEB: A PERFORMANCE EVALUATION*. Master of Science Thesis, Graduate Faculty of The University of Georgia, 2005.

Appendix A – Knowledge from the domain

As mentioned in the sub-chapter 2.1.4 - Ontology Building, in order to create an ontology it is first necessary to gain all the knowledge from the domain to be represented. In this case the domain is manufacturing systems, and therefore, it is necessary to get all the specifications in terms of the devices it has.

In the present scenario, the manufacturing system is composed by two robotic cells, a conveyor system, several presence sensors, RFID sensors, stoppers, diverters and pallets.

The first step towards getting good knowledge from the domain is by searching for information from its manufacturer, and then from other sources such as websites, books, standards, etc. The knowledge found then give help in the decision making process regarding the creation of classes, properties, best taxonomy to use and other aspect of an ontology.

Stating this, the following presents the sources that have given help to the decision making process of creating the ontology for each device.

Regarding the identification of useful knowledge and taxonomy in the robots domain, the following was found and used:

- Robot definition:

http://searchcio-midmarket.techtargget.com/sDefinition/0,,sid183_gci519835,00.html

- Manipulator definition

<http://www.robots.com/faq.php?question=robot+manipulator>

- SCARA definition:

<http://www.motionnet.com/cgi-bin/search.exe?a=showdefinition&no=3626>

- Specifications of the robot (Datasheet)

<http://www.flexmont.hu/eng/fooldal.php?page=sony>

<http://www.exapro.eu/uk/produit-20658-sony-scara-robot-axis.html>

<http://www.lasermotion.com/sony611.html>

- Types of robots

Appendix 9 – Robot Domain Ontology from (Martínez Lastra, Delamer and Ubiz Lopez 2007)

- General descriptions regarding robots

ISO 8373 – 1994

<http://prime.jsc.nasa.gov/ROV/types.html>

<http://www.eod.gvsu.edu/eod/mechtron/mechtron-417.html>

With respect to the identification of useful knowledge and taxonomy in the sensors domain, the following was found and used:

- Sensor Properties

Chapter 2 from (Fraden 2004) or at

<http://books.google.com/books?hl=en&lr=&id=SB7glOc4VlAC&oi=fnd&pg=PR8&dq=Handbook+of+Modern+Sensors&ots=6U8juLho5Z&sig=6aXwXJdLd9B3EN1K0jZZf4QL44#PPP1,M1>

- Sensing Resource

From (J. Lastra 2004)

- Sensor definition

<http://dictionary.reference.com/browse/sensor>

- Inductive Sensor Definition:

<http://www.nationmaster.com/encyclopedia/Inductive-sensor>

- Information for the RFID

From (Finkenzeller 2003)

- Manufacturer's datasheets:

- BES inductive sensor

<http://www.balluff-china.com/PDF/en/datenblaetter/BES%20516-325-G-E4-Y-PU-05.pdf>

- Wenglor inductive sensor (a compatible model for general descriptions)

http://www.wenglor.com/wenglor.php?Sprache=US&Land=USA&Start=Produktdaten.php&P=Artikel_NR=IB040BM46VB8;;Kategorie=IN;;Benennung_Gesamt=Inductive%20Proximity%20Switch;;sort=Bauform

- RFID sensor (V600-HS51)

[http://www.omron247.com/marcom/pdfcatalog.nsf/PDFLookupByUniqueID/78C2C296F68DC2F08525702E0051AACA/\\$File/M09Z129E101A0505.pdf?OpenElement](http://www.omron247.com/marcom/pdfcatalog.nsf/PDFLookupByUniqueID/78C2C296F68DC2F08525702E0051AACA/$File/M09Z129E101A0505.pdf?OpenElement)

- SMC inductive sensor (D-A93 page 46)

http://www.smc.eu/portal/WebContent/resources/docs/atex/cat_en.pdf

Considering the identification of useful knowledge and taxonomy for the conveyors, pallets and diverter domain, the following was found and used:

- Pallet definition

<http://dictionary.reference.com/browse/pallet>

- Manufacturer's datasheet

http://www.paro.ch/uploads/media/Flyer_transfers_e_02.pdf

- General Information

<http://www.flexlink.com/wps/public/s/10000/c/1061714>

<http://www.flexlink.com/wps/public/s/10000/c/1061820>

In order to obtain useful knowledge and taxonomy for the tool's domain, the following was found and used:

- General information

http://whatis.techtarget.com/definition/0,,sid9_gci521693,00.html

Related with the identification of useful knowledge and taxonomy for the stopper domain, the following was searched and used:

- General information

<http://www.ferret.com.au/c/Festo/Stopper-cylinders-for-conveyor-systems-n681939>

Finally, taking into account getting useful knowledge and taxonomy for the location domain, the following was found and used:

- Definition by (J. Lastra 2004)

Other useful references are a dictionary referenced by (Martínez Lastra, Delamer and Ubiz Lopez 2007) which link is

<http://www.merriam-webster.com>

and an acronym finder at

www.acronymfinder.com