

Otto-von-Guericke-Universität Magdeburg



**FAKULTÄT FÜR
MASCHINENBAU**

Agent-based mechanism for smart distributed dependability and security supervision and control of Cyber-Physical Production Systems

Dissertation zur Erlangung des akademischen Grades

Doktoringenieur

(Dr.-Ing.)

von M.Sc. Hessamedin Bayanifar

geb. am 09.09.1988 in Babol

genehmigt durch die Fakultät für Maschinenbau
der Otto-von-Guericke-Universität Magdeburg

Gutachter: apl. Prof. Dr.-Ing. habil. Arndt Lüder

Jun. Prof. Dr. Sebastian Zug

Promotionskolloquium am 18.12.2017

Preface

Writing this thesis sent me through an adventure that in the end has provided me with many lessons to learn. First, I want to thank Professor Kühnle for making this adventure possible and for providing me with the opportunity to join the IAF institute in the Otto von Guericke Universität Magdeburg.

Here, I would like to express my gratitude to my friends and colleagues at IAF institute for supporting me throughout this journey. I need to show my special appreciation for all the friendly and professional supports of Ronald who had always been there for me. I also need to thank my friend and colleague Ambra for all her inspirations and friendly presence through this hard path, and Yahia with whom I shared many parts of my journey.

Moreover, I want to acknowledge the supports and helps of Professor Lüder for supervising my work and providing me with his valuable comments.

I also would like to thank my friends in my home country, specially Sheyda, Mohsen, Anahita, and Hengame for never letting their sense of presence fade away because of the long distance between us.

In the end I want to express my deepest love for my family and my parents who gave me life and supported me through all the challenges I have faced.

Magdeburg, 26.10.2017

Hessamedin Bayanifar

Declaration of Honor

I hereby declare that I produced this thesis without prohibited external assistance and that none other than the listed references and tools have been used. I did not make use of any commercial consultant concerning graduation. A third party did not receive any nonmonetary perquisites neither directly nor indirectly for activities which are connected with the contents of the presented thesis.

All sources of information are clearly marked, including my own publications.

In particular I have not consciously:

- Fabricated data or rejected undesired results
- Misused statistical methods with the aim of drawing other conclusions than those warranted by the available data
- Plagiarized data or publications
- Presented the results of other researchers in a distorted way

I do know that violations of copyright may lead to injunction and damage claims of the author and also to prosecution by the law enforcement authorities. I hereby agree that the thesis may need to be reviewed with an electronic data processing for plagiarism. This work has not yet been submitted as a doctoral thesis in the same or a similar form in Germany or in any other country. It has not yet been published as a whole.

Magdeburg, 26.10.2017

Hessamedin Bayanifar

Abstract

Information and Communication Technologies, or for short ICTs, are growing and advancing remarkably fast and along with this growth, their applications in various areas of human life and activities is being radically extended. Likewise, industrial and productions systems have been no exception in embracing these technologies to provide the organisations with better performance and efficiency and as a result bringing strategic advantages.

The most recent contribution of ICTs in manufacturing is aiming at ever increasing convergence of virtual world and the physical world by introducing the concepts of Cyber-Physical Systems (CPS) and Internet of Things (IoT) into the manufacturing era, and giving rise to the fourth industrial revolution, which is commonly known as the Industry 4.0. However, despite all the potentials and capabilities, there are still major challenges to address. One of the main challenges here is the dependability and security of these new systems, given their vast exposure to the cyber space and internet, and increased complexity and versatility of their functionality. Accordingly, this study intends to address this issue and make a contribution by proposing a robust mechanism that enables distributed components to fulfil their own dependability and security objectives by constantly and autonomously supervising and controlling their condition and actions with regards to these two aspects (dependability and security).

Therefore, first the concept and the model for such mechanism has been modelled by taking into account the features and attributes of the systems on which the mechanism is going to applied. Consequently, the mechanism has been designed in the way that can inherently demonstrate the same properties to assure highest effectiveness and performance. The main properties considered were interoperability, scalability, modularity, autonomy, heterogeneity, and context-awareness. For the next step which was the implementation of the mechanism, Multi-Agent System approach was selected, given all the capabilities of these systems and perfectly matching properties that this toolset enables.

As a result, and agent-based mechanism for autonomous and intelligent distributed supervision and control of smart cyber-physical systems' components has been developed and implemented, and various use cases to test the mechanism were defined. And finally, the results have been demonstrated.

Contents

Chapter One.....	1
INTRODUCTION.....	1
1.1. Motivation.....	1
1.2. Research Hypotheses and Questions.....	3
1.3. Thesis Structure	8
Chapter Two.....	11
FUNDAMENTALS AND STATE OF THE ART	11
2.1. Smartness of Objects	11
2.1.1. Smart Objects.....	11
2.1.2. Cyber Physical Systems	12
2.1.3. Cyber Physical Systems Properties	17
2.2. Smartness of Manufacturing Systems	22
2.2.1. Towards the new Industrial Revolution.....	22
2.3. Multi-Agent Systems (MAS) as the toolset for Model Implementation	27
2.4. The DACS methodology	29
2.5. Jade platform	30
2.6. Dependability and Security of Smart Manufacturing Systems	33
2.6.1. Aspects of Dependability and Security	33
2.6.2. Dependability and Security issues of Cyber Physical Production Systems	38
2.6.3. State of the Art approaches to model and tackle Dependability and security Issues of cyber physical production systems.....	42
2.7. Implementation approaches for dependability and security	52
Chapter Three	59
THE DEPENDABILITY AND SECURITY MODEL	59
3.1. The Dependability and security model	59
3.2. Contribution of smart system’s properties in the dependability and security mechanism	68
Chapter Four	77
Model Implementation and Use Cases.....	77
4.1. Using DACS to design the mechanism	77
4.2. The use cases for testing the mechanism.....	87

4.2.1.	Constituent of the model production system.....	88
4.2.2.	Failures associated with the case (conveyor)	91
4.3.	The failure cases in this study	94
4.3.1.	Case one: Scheduled maintenance	94
4.3.2.	Case two: Item (wood block) is not present on time.....	96
4.4.	Agents, interfaces, and their interactions.....	97
4.5.	Case three: A machine breakdown from a shopfloor (to test the ability of negotiation over resources)	103
Chapter Five	107
Conclusion and discussion	107
5.1.	Conclusion.....	107
5.2.	Outlook	109
References	111
Appendix (codes for agents)	121

List of Figures

Figure 1. A conveyor - pusher system for sorting wooden blocks.....	7
Figure 2. Wood blocks with magnets on them	7
Figure 3. Thesis Structure	9
Figure 4. Structure of Smart Systems (EPoSS)	11
Figure 5. Breakdown of hierarchical systems into heterarchy	15
Figure 6. redraw. CPS maturity model (from cps in manufacturing).....	17
Figure 7. Compatibility Scale against device features (IEC 65/290/DC)	19
Figure 8. How ICTs grow to bring cyber world and real world together	22
Figure 9. Industry 4.0 components in comparison with legacy systems components.....	23
Figure 10. Advancement of the use of ICTs in manufacturing, from: a) Computer Integrated Manufacturing (CIM), to b) Cyber physical systems (CPS), to c) Cloud-based manufacturing systems (CMfg).....	26
Figure 11. Steps of DACS methodology	30
Figure 12. The overall structure of the JADE platform	31
Figure 13. JADE's main container	32
Figure 14. Eclipse as the compiler used to develop and run JADE agents	32
Figure 15. Dependability and security tree.....	34
Figure 16. Dependability and Security attributes and correlations.....	34
Figure 17. fault classification according to [7]	37
Figure 18. Elements of Fault Tree Analysis (FTA).....	42
Figure 19. An example of a risk level matrix.....	44
Figure 20. Sample of failure taxonomy.....	45
Figure 21. Example of Petri Net	46
Figure 22. Two-dimensional data sets and anomalies	48
Figure 23. DoS Attack anomaly	49
Figure 24. SQC example of anomaly	49
Figure 25. Steps in anomaly detection	49
Figure 26. Breakdown of automation hierarchy at MES level	56

Figure 27. From traditional automation centralized and hierarchical structure towards PABADIS and then to PABADIS'PROMISE.....	56
Figure 28. The decision cycle as shown in	60
Figure 29. Components to form the database in the core of the model	62
Figure 30. The Dependability and Security Model	63
Figure 31. Agents structure for dependability and security mechanism development.....	80
Figure 32. Failure Data as categorized in the database.....	81
Figure 33. Monitoring Agents	84
Figure 34. Negotiation over utilization of other entities resources	86
Figure 35. agents in the main container	86
Figure 36. The Fischer model.....	87
Figure 37. Objects and sections associated with the case (Fischer model): a) objects and sensors for the production system, and b) parts (wood blocks) and magnets	89
Figure 38. The structure of the control system	90
Figure 39. Failures database interface for adding, removing, editing failures and their data. One the right, information for failures can appear	97
Figure 40. List of failures at any time.....	98
Figure 41. A monitoring agent's general behaviour	98
Figure 42. Analyser Agent's general behaviour	99
Figure 43. Analyzer agent's activities for the case of conveyer breakdown.....	100
Figure 44. Analyzer agent's activities for the case of pusher robot's maintenance due	100
Figure 45. Decision Maker agent's behaviour	101
Figure 46. Decision Maker agent output for case two, the conveyer belt failure.....	102
Figure 47. Decision Maker Agent activities dealing with the case pusher robot maintenance due	102
Figure 48. Hypothetical example for negotiation case.....	104
Figure 49. Offer Providers A, B, and C	105
Figure 50. decision maker agent's activities for the case three.	105
Figure 51. Negotiation over a failed resource	106

List of Tables

Table 1. Information to form the core of the model 62

Table 2. Control loop steps and the relationship with smart manufacturing systems' properties
..... 69

Table 3. Table of agents and their behaviour, decisions, and interactions 77

Table 4. Components of the conveyor system, and their behaviour 89

Table 5. Failure information and action data 91

ABBREVIATIONS

ACL	Agent Communications Language
BDI	Belief-Desire-Intention agents
CAD	Computer-Aided Design
CIM	Computer-Integrated Manufacturing
CPPS	Cyber Physical Production Systems
CPS	Cyber Physical Systems
DACS	Design of Agent-based Control Systems
HMS	Holonic Manufacturing System
I 4.0	“Industrie 4.0” paradigm
IA	Intelligent Agent
IEC	International Electrotechnical Commission
IIoT	Industrial Internet of Things
IOT	Internet of Things
JADE	Java Agent DEvelopment framework
MAS	Multi-Agent Systems
PABADIS	Plant Automation Based on Distributed Systems
PLC	Programmable Logic Controller
RAMI 4.0	Reference Architecture Model for Industrie 4.0
VDE	“Verband der Elektrotechnik Elektronik Informationstechnik e.V.” (Europe’s technical-scientific associations)
VDI	“Verein Deutscher Ingenieure” (Assotiation of German engineers)
ZVEI	“Zentralverband Elektrotechnik- und Elektronikindustrie e.V.” (German Electrical and Electronic Manufacturers' Association)

CHAPTER ONE

INTRODUCTION

In the past few decades, the rise of information and communication technologies (ICT) and the growth of artificial intelligence, and their increasing expansion over almost every aspect of human life has been conspicuous and tangible. From medical sciences, to everyday transportation, to children’s education systems, to humans’ endeavour in exploring the heavens and other planets. Manufacturing systems too have not been an exception. Catching up with the fast pace of advancements in higher degrees of machines’ smartness and ICTs technologies, manufacturing systems in recent years are trying to step into an era where intelligence and connectedness are almost ubiquitous. In this marathon of competing for benefiting from vast underlying potentials and opportunities, as well as gaining strategic advantages, companies are struggling to overcome various challenges. Among these challenges, one of the main questions to be addressed has been “how to assure security and dependability of these systems given the rather extreme dispersity, functional versatility, and exposure of the components to cyber space?”

After contemplating this question and studying the issue, the second question formed which has inspired my work in this thesis: “can smart components use this smartness to tackle with their own security and dependability concerns?”.

This chapter is framing this thesis by first proving the motivation for this work, then forming the research question and hypotheses accordingly, and finally providing an outline for rest of this work.

1.1. Motivation

Humankind have always had the passion for transforming things into more valuable goods, and this passion led them to develop industries. As time went by, industries witnessed several turning points. Started with harnessing steam and water energy and developing production tools as the first revolution in around 1784, followed by the introduction of manufacturing as mass production by assigning labours specific repetitive tasks and using electricity and bringing about the second revolution in around 1870, and continuing through the third revolution by summoning electronics and IT techniques to start automation in manufacturing processes in

1969, the industries have evolved significantly. Now, according to the Federal Ministry of Education and Research, Germany (BMBF), industry is on the verge of its fourth revolution by merging Internet of Things (IoT), and Internet of Service (IoS) into manufacturing, and enabling an ever-increasing collaboration of virtual and physical world by introducing Cyber-Physical Systems (CPSs) [1-2].

Equipped with smart objects and embedded systems, manufacturing components demonstrate self-controllability and self-organizability, and are now capable of intelligently cooperating and collaborating together to fulfil their tasks. Sensors and actuators are now more extensively being used to assist physical components to provide a network of intelligent objects, which is now being referred to as the Next Generation Network (NGN), and Wireless Sensor/Actuator Networks (WSAN) assisted by IoT and Cloud technology, altogether playing pivotal roles in enabling the abovementioned fourth industrial revolution, known as the Industry 4.0. This name genuinely came from “Zukunftsprojekt Industrie 4.0” as a project in 2011 in Germany to develop high-tech methods in manufacturing [3]. Since then, it has gone up the chart in research priorities, e.g. in Horizon 2020, The EU Framework Programme for Research and Innovation, by the European Commission [4]. Also, a report in 2014 from Deutsche Bank announced that an estimate from a joint research carried out by Fraunhofer institute and Bitkom company suggests that through opportunities and applications of the new industrial revolution (I4.0), the gross value added of the country can tremendously rise by aggregate of 267 Billion euro by the year 2025 [5].

Industry 4.0 systems, which can also be referred to as smart distributed manufacturing systems, or cyber-physical production systems (CPPSs), often consist of a large number of widely dispersed loosely-coupled yet collaborating heterogeneous components, that are vastly connected to and communicating with cyber spaces. To enhance their capabilities, these systems try to exploit smart properties through enhancing their own intelligence and processing power, or via accessing the internet and its vast options to enhance these properties. On the one hand, using these properties and enhancing capabilities can provide a plethora of opportunities and strategic advantages to manufacturing enterprises. On the other hand, however, vast dispersity and exposure to cyber spaces as well as versatility of processes and systems’ structures raise major vulnerabilities [6]. Hence, dependability and data security issues may diminish the readiness to rely on such enormous capabilities. Therefore, to harness all capabilities, not just through implementation, but also in real-time dependabilities and utmost security must be

assured.

Moreover, an adaptable, learning tool to analyse and steadily improve all conditions that accommodate all dependability and security needs, would represent an important contribution to make smart manufacturing units' applications more attractive.

This thesis accordingly aims at proposing a generic model for intelligent and autonomous distributed dependability and security supervision and control that is broken down throughout the system, and is to be carried out by smart components themselves in Industry 4.0 environments. To this aim, it adopts and harnesses smart manufacturing systems' capabilities and properties (i.e. interoperability, autonomy, scalability, modularity, heterogeneity, and context-awareness), for maximizing its performance and versatility.

1.2. Research Hypotheses and Questions

Given the versatility and dynamic characteristic of the smart systems, and accordingly the need for a powerful mechanism to assure the dependability and security in real-time, the following research question and hypotheses are proposed:

Question: *Can an agent-based intelligent distributed dependability and security supervision and control mechanism that inherits smart system properties enhance the overall dependability and security of the system?*

Before introducing the hypotheses, giving proper definitions and explanations might be necessary to understand what the mechanism mentioned above is.

From different uses of the word "intelligent" in the CIRP Encyclopedia of Production Engineering, in the context of this thesis, "intelligence" refers to as the ability of learning, reasoning, and acting upon the environment. The word "distributed" here suggests that this mechanism is to be performed by intelligent components themselves (i.e. the task of supervision and control is broken down among entities down to the levels of details). That accordingly requires a clear definition of intelligent components. This thesis takes a manufacturing unit as its case of an intelligent component, as shall later be described further in more details.

Then it comes to the two attributes that in my work are expected to improve by the mean of the proposed mechanism. Dependability as defined in [7], is the ability to deliver service that can justifiably be trusted. A dependable system should be robust and stable, and perform without interruption in the environment that due to its dynamic character may cause frequent disturbances. The security objectives (with regards to information systems) on the other hand, as determined by Federal Information Security Management Act (FISMA) [8], takes into account the three major attributes known as CIA triad, that are namely: Confidentiality, Integrity, and Availability. Confidentiality here means that no information should be revealed or disclosed by an unauthorised entity. Integrity suggests that the information shall not undergo any undesirable and unauthorised alteration. Lastly, availability refers to the readiness of the information at the right time to the right entity. Together, dependability and security are to assure smooth, flawless, and timely delivery of services within satisfactory specifications.

The second part of the question refers to smart systems' properties (as appeared and explained in [9]) that are thoroughly elaborated in the next chapter and are shortly defined bellow within the scope and purpose of this thesis:

Context-Awareness: is the ability of components to understand the context in which they are working. It can refer to the components' environment, collaborations, physical status, or tasks, goals etc.

Heterogeneity: refers to the diversity of structure or elements, as in smart systems components of different format and structure or functionality are connected to the cyber space and communicate with other components to fulfil their tasks.

Interoperability: this is an attribute for systems of heterogeneous and diverse components in which components can easily collaborate and inter-operate. Depending on the component definition, highly heterogeneous sets of constituents or collaborators may involve in interactions which may require high interoperability.

Autonomy: as another intrinsic feature of smart units, autonomy is referred to as the ability of components to fulfil their tasks and objectives without the need for other components or human intervention. As in the perspective of this thesis, the proposed mechanism is to be exercised by the components themselves.

Scalability: is generally referred to as the ability of components to expand on or shrink their resources and constituents the way that no significant effort as structural or application changes or technological updates are demanded. As the dynamic characteristics of smart systems require scalability, the dependability and security supervision and control mechanism would also see scalability of its constituents as imperative to its proper and satisfactory functioning.

Modularity: components and systems can be called modular, if their constituent components can be broken down and rearranged in different setups or replaced to serve various services and enable various functionalities.

The suggested mechanism in this thesis and what the question is referring to must inherit these properties and reflect them in its behaviour. After clarifying the question, the following hypotheses are proposed:

H1: *Additional properties and enhanced capabilities and constant connectedness to cyber space increase risks in playing manufacturing on the base of smart manufacturing units. Dependability and security may increase and risks may be lowered when given components continuously monitor their behaviour and autonomously take proper actions through harnessing and maturing the properties. Maturing these properties, in addition to reduce the need for human intervention in the process of monitoring, reasoning, and action taking, may enhance flexibility, stability, and responsiveness.*

H2: *A powerful tool for monitoring and analysing dependability as well as maturities of properties and units can be established on the base of dynamic modelling. The model architecture may be specified by expected contributions and resulting requirements. The desired features for tool implementations may be provided by Multi-agent design pattern.*

H3: *A decision table for failure risks, disturbance cases, decisions for actions and inputs for learning may be established. Using agents, adequate architecture may be implemented into a learning experimentation environment for simulation or manufacturing scenarios and evaluations of risks.*

H4: *The implemented architecture can perform independent of input heterogeneity, reasoning techniques in decision making, and knowledgebase generation and analysing approaches.*

To test these hypotheses first the anticipated deliverables must be clearly defined, and accordingly the expected mechanism must be developed. Beside reducing human intervention which is on a par with higher autonomy (one of the properties), flexibility can be perceived as the capacity of a system to adjust itself in response to changing requirements without significant expense in terms of time, effort, cost, or performance [10], and responsiveness means sufficiently timely reactions to disturbances and environmental stimuli.

Stability can generally be referred to as the property of an equilibrium in a dynamic system [11], which in the scope of this work means the ability of the system to run flawless in the face of disturbances. That implies, if due to any reason a failure (whether intentionally or unintentionally) occurred, the system should not break down. To reach these enhancements the mechanism then must enable the properties mentioned in the question section. This need triggers the second and third hypothesis where using Multi-Agent Systems' capabilities implementing the desired mechanism is intended.

Performing independent of input heterogeneity, and techniques or approaches to construct knowledgebase or analyse disturbances and react upon them suggest the universality and applicability of the proposed mechanism regardless of the smart component type and the environment or context in which it is performing.

Accordingly, this work takes into account developing a generic dependability and security model plus an agent-based structure, and will test it against various types of failure modes to subject the proposed mechanism to scrutiny to see if the hypotheses stand up.

The case component for this study (figures (1, and 2)) to demonstrate and analyse the functionality and performance of the mechanism as mentioned above is a manufacturing unit with a set of machines and conveyors, controlled by a programmable logical control (PLC), and equipped with OPC UA server to receive sensor data. Monitoring input data is to be provided via OPC UA server through a client agent.

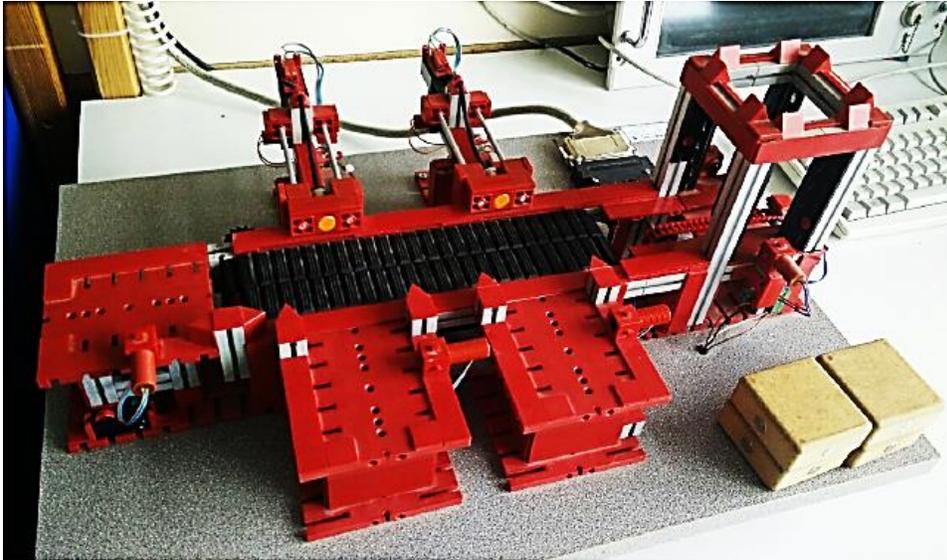


Figure 1. A conveyor - pusher system for sorting wooden blocks

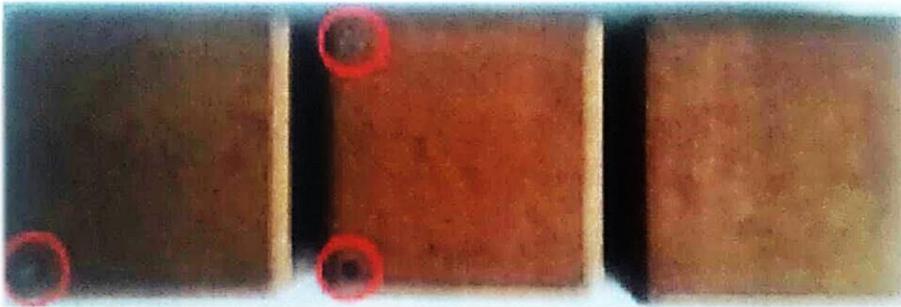


Figure 2. Wood blocks with magnets on them

Failure cases will be generated both physically (i.e. making physical unapproved changes in the system) and via simulation (i.e. by introducing anomalies in data communication or data that come from sensors in the field) as different risk modes and type. Failure simulation will be used to test the capability of negotiation over utilising another component's resources.

One case for this study would be to introduce delays in the process. There would be accepted and unaccepted delays according to the time intervals between sequences of actions. This case represents a failure that can occur on the field level and actions are to be taken accordingly. Delays can be caused by a problem in conveyor (the part sticks on the way or the conveyor does not work) or the pusher robot. Sensors are to be used to constantly provide data for monitoring and as previously mentioned these data are gained by subscribing the OPC UA server that reads the sensor data.

The second case considers a KPI (Key Performance Indicators) related to the life cycle of the assets. For some components in a system, there might be a limited number of operational actions before every preventive maintenance call to maximize the performance and reliability of that given component and avoiding a breakdown. In this case, for machines in our system (pusher robots), a limited number of pushes will be defined before maintenance and when the number is reached, the machine will send an anomaly not caused by a failure, but to call for maintenance to prevent a possibly imminent failure.

To set up a knowledgebase for disturbance cases and identification thereof, as well as a table of suggested sets of actions and decisions, this work adopts and modifies FMEA technique. FMEA among the several techniques recommended by International Standard for Dependability Management [12] is one of the most common and suggested techniques which in modified form suits the requirements of this thesis adequately (as shall be described thoroughly in the following chapters). Other techniques, according to the cases and policies, as well as dependability objectives for specific components can be merged into the mechanism in for instance dependability analysis as it can be inferred from the fourth hypothesis of this thesis and it will accordingly be discussed how the model can perform irrespective of the system context, scale, and approaches in forming its modules (for supervision, analysis, decision making, etc.).

Next section of this chapter outlines the thesis sections and structure.

1.3. Thesis Structure

Considering the research question and hypotheses, and the cases for concept implementation, the rest of thesis will break down into following chapters:

Chapter two, which takes into account the fundamentals on this subject and the latest works and studies done in this literature. It provides information on the movement of industries on the path of gaining higher levels of smartness and accordingly the rise of dependability and security concerns, as well as the potential approach in tackling these concerns which leads the thesis to the third chapter.

In the chapter three, the concept and the structure of the proposed mechanism will be addressed. Contribution of the smart systems properties into the model behaviour will be defined along with the requirements to enable them. The dependability and security model will be developed and elaborated on. A modified FMEA approach to develop the core database of the model will be specified. Multi-Agent Systems will be introduced and discussed as the adopted

tool to deliver the desired features of the model. Subsequently, the architecture for the collaboration of different parts (Agents) of the model and a structure for implementing it will be established.

Chapter four demonstrates the implementation procedure. In this chapter, the process of programming and detailed structure and functions of agents that are collaborating with each other will be presented.

In chapter five, the outcomes and results of the experiments will be shown and discussed as well as the improvements that the proposed mechanism provides upon existing approaches for dealing with dependability and security issues. In addition, suggestions for extending the functionalities and the performance of the model will be presented.

The following diagram (figure (3)) summarized the thesis structure.

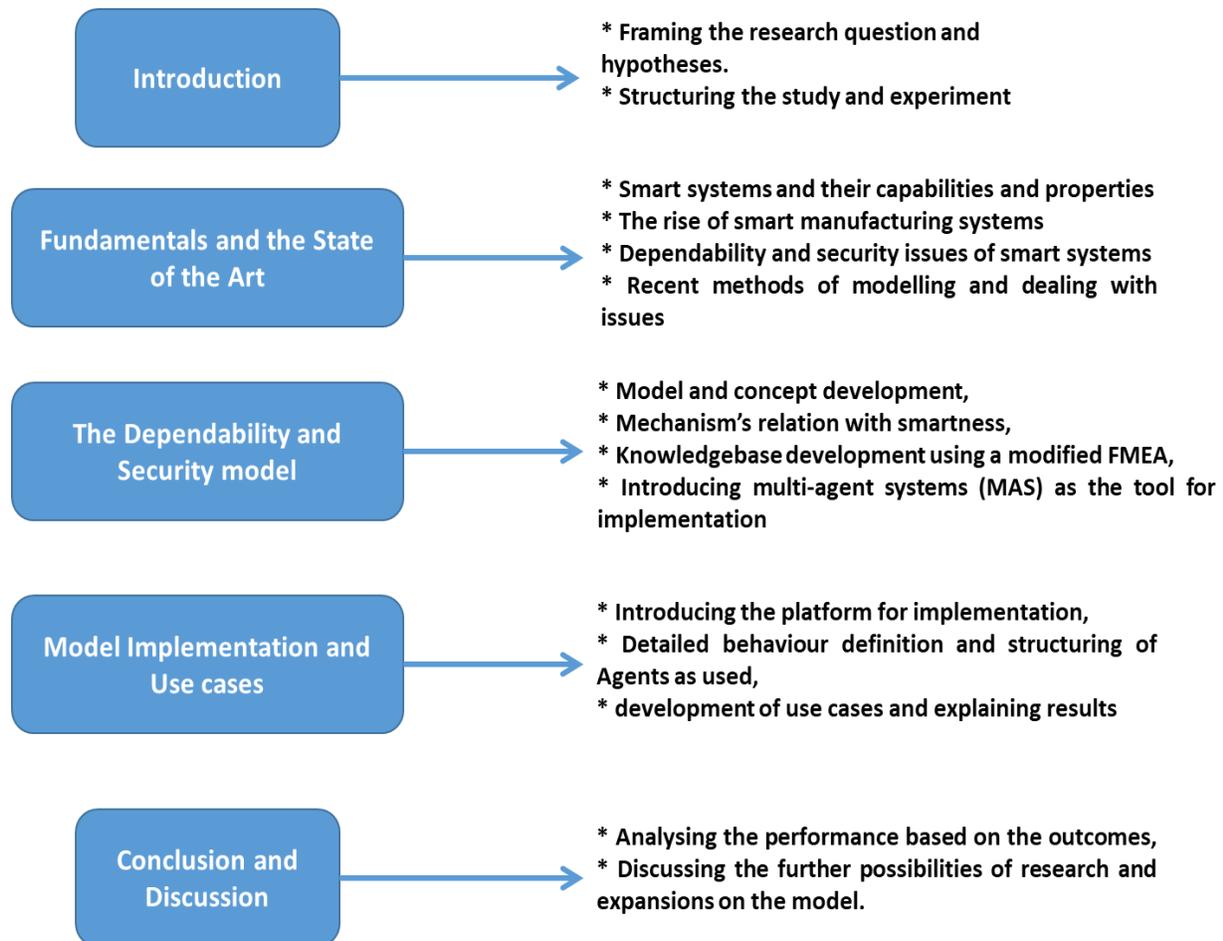


Figure 3. Thesis Structure

CHAPTER TWO

FUNDAMENTALS AND STATE OF THE ART

2.1. Smartness of Objects

2.1.1. Smart Objects

Smart systems generally refer to devices that are ubiquitously connected and communicating with one another and are capable of demonstrating degrees of intelligence and autonomy [13]. Their functionalities include collecting data from the environment, transmitting the acquired data to their controlling unit to perform decision making process and develop instructions based on the information received, and finally reacting on the environment according to the commands provided by the controlling unit (figure (4)). Accordingly, the main constituent components of smart systems are for the most part sensors for data acquisition, controllers for data processing, reasoning and decision making, actuators for performing action, and means of communication and data transmission between these entities [14]. These systems normally also have a power source and are generally very energy efficient or even totally energy autonomous.

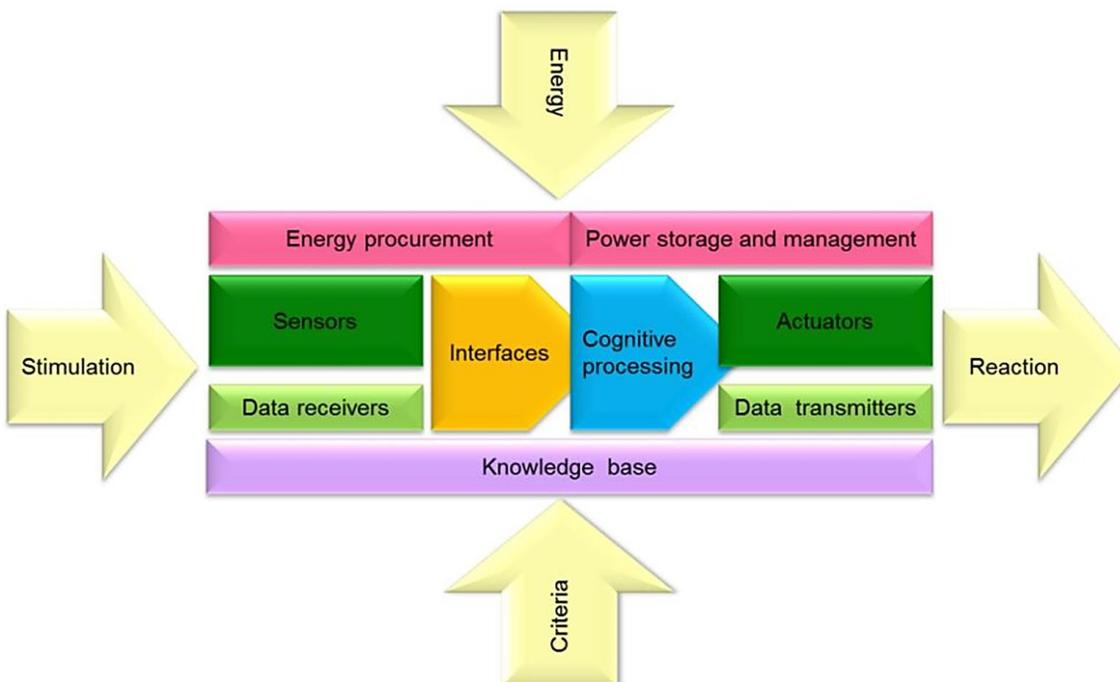


Figure 4. Structure of Smart Systems (EPoSS)

2.1.2. Cyber Physical Systems

The term Cyber-Physical systems (CPS) was first invented by Helen Gill in the year 2006, at National Science Foundation (NSF) [15]. It may be that the notion of “cyber space” rose from this term (cyber physical systems), given that CPS was coined earlier and carries broader roods and concepts. However, it would be fair to associate both the terms “cyber space” and “cyber physical systems” to the word “cybernetics”, which dates back to much older time, to about 70 years ago in 1948 by Norbert Wiener [16], who himself took remarkable part in stablishing the theory of control systems.

Edward Ashford Lee, defines Cyber Physical Systems as “*an integration of computation with physical processes whose behaviour is defined by both cyber and physical parts of the system*”. In these systems, the behaviour of physical world and processes are influenced through the cyber space in which supervision and control of physical activities are being carried out, and vice versa. Such bilateral interactions are made possible typically via feedback loops. As Edward Lee puts, Cyber Physical Systems deals more with the intersection of the virtual and real world, and not the union of the two [17]. From the definitions given, CPSs and Smart systems have broad similarities where cyber physical systems main focus is on the interactions of nodes and smart entities.

Beside their structural similarities the behaviours of these systems, due to their extreme heterogeneity, can vary significantly. However, smart systems share two behaviours as their main commonality; firstly, they perform interactions with physical world, and secondly, they communicate with each other. Such attributes enable them for performing in vast variety of applications from as simple as controlling temperature, light or air pressure, to dealing with significant environmental issues such as air or water pollution or business exercises. The federal Networking and Information Technology Research and Development (NITRD) CPS Senior Steering Group developed a list of the main practical uses of cyber physical systems in their vision statement released in 2015 [18] which along with other studies are shortly described as follow:

Agriculture

Cyber physical systems can be harnessed to improve on responsiveness towards ever increasing issues that agriculture industry is facing or will encounter in not very distant future. The world’s population is estimated to exceed nine billion in just over 30 years. In addition to that are other factors such as global warming and climate change as well as uncertainty of weather as a more immediate factor. Beside these more general drivers are rather shocking

figures maintaining that in many cases a large amount of losses is being observed in the process of reaching to consumers from production. All of these show the need for bringing higher efficiency and smartness into this industry throughout the value chain and product life cycle from bringing smartness in dealing with climate [19], to control of irrigation system [20,21], to monitoring and analysis of the farm animals' conditions [22]. This should ultimately diminish the carbon footprint of this industry and aid it with moving towards higher sustainability and environmental friendliness.

Building controls

cyber physical systems can enhance features related to buildings with regards to a variety of factors ranging from the experiences of people who dwell in them to the environment in general, by affording buildings higher levels of smartness. Considering inhabitants experience smart safety systems, improved energy consumption and in case of net-zero energy buildings improved energy generation to energy consumption ratio (can be gained through the use of solar energy or other sources of energy generation and controlling and minimizing energy loss through for instance smart heating and air conditioning, windows and shades control, timing of the use of household appliances, etc.), features to assist aged people, as well as provision of better comfort are some features to name [23-25].

Defence

with regards to defence and military practices, advance ICT technologies are considered essential and sophisticated network of interacting components and entities play a heavy role in an effective defence system. Among the areas of high importance with regards to the defence section and associated technologies specified by The Department of Defence, smart and cyber physical systems are to be seen as important enablers to at least three, according to the NITRD CPS senior group [18]. These areas include engineering of resilient systems, areas of cyber capabilities, and enhancing and advancing autonomy within defence systems throughout the military operations.

Emergency response

another place that cyber physical systems can play a pivotal share deals with catastrophic phenomena as natural disasters, etc. Hundreds of such cases occurs early which lead to tens of

billions of dollars of costs to be imposed on nations. Using smart sensor-actuator networks, and CPS technologies, related agencies can be more responsive with regards to predicting and reacting to emergency situations whether environmental, such as earthquakes, storms, or floods, or caused by humans, for instance terrorist or cyber attacks [26,27]. By utilizing data acquired through smart systems sensor networks and software application to analyse received data, awareness towards the environment increases which increases the predictability of events that may arise and provides times to generate more optimum and timely decisions to tackle or prevent disasters.

Energy

As they are becoming more affordable and less costly to implement and due to changes in the climate as a result of human activities like burning vast amount of fossil fuels, renewable sources of energies, more specifically electricity, such as from the sun using solar panels and wind or wave energy are becoming more broadly considered. This trend towards cleaner energy provision and consumption has also led to more creative electric products such as electric cars and electricity supply mechanisms for them.

Numerous studies are being published in this area [28-32] from implementation, to security provision of smart grids, to their capabilities and future development, given the importance and opportunities of bringing smartness into power systems. Using cyber physical systems and through sensor networks and smart devices and approaches like smart metering infrastructure or in short (SMI), power industry is moving towards harnessing the most out of integration of these sources with hardly predictable behaviour.

In addition to smart metering system are smart and robust power grids using cyber physical systems aided by multiagent systems, and smart distribution of electricity to consumers. In this movement towards smartness, cyber physical systems are fair to be referred to as the best enabler available from ICTs which can provide resilient and robust self-optimizable infrastructure to generate and deliver appropriate services.

Healthcare

Likewise, health care systems are growing in smartness using ICT [33-35]. Sensors are being extensively used for gathering patients' health information as Electronic Patient Records (EPR) as mentioned in International Telecommunication Union (ITU) [36] or Electronic Health Records (HER), and redundant analysis of data can be more easily avoided [37].

The support of the smart systems also goes beyond the diagnosing procedure and can offer functions for treatment process as well for instance by directing patients to right health specialists or entities timely and conveniently via any devices. Medication is now also able to be personalized and accessible where patients can take the appropriate amount of the appropriate type of medication at the right time.

Manufacturing and industry

One of the areas that is tremendously developing the use of ICTs and is seeking the opportunities these technologies can afford is industry and more specifically manufacturing section. Cyber-physical Production Systems (CPPSs) which drives Industry 4.0 is attracting so much of researchers' attention that has high on the agenda and funding list of many industrialized countries.

That has triggered an enormous number of research studies, standardizations, etc. to be carried out and published focusing on enabling, implementing, and taking into account the possible opportunities of cyber physical production systems [38-46]. The use of Cyber Physical Systems in this area will be discussed in much further details in following sections.

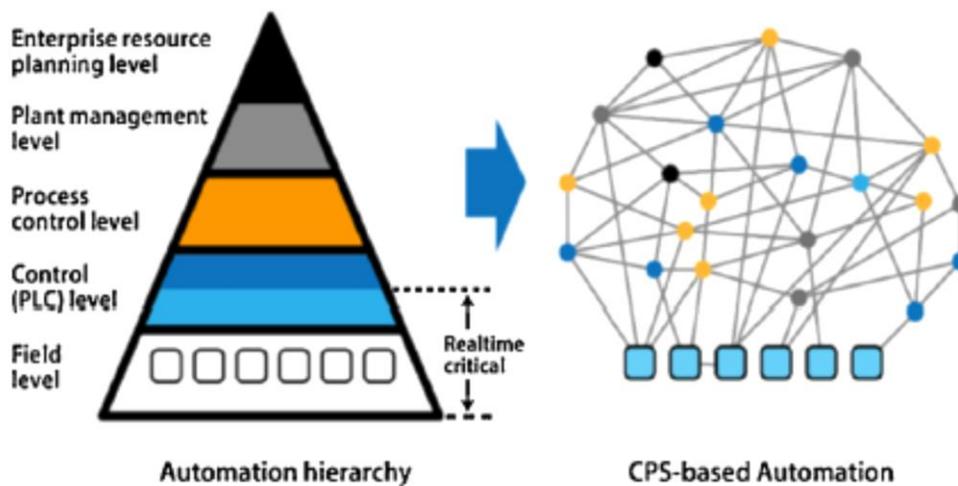


Figure 5. Breakdown of hierarchical systems into heterarchy [47]

Figure [5] shows the way cyber physical system can affect the hierarchical structure of legacy automation systems. It is doing so by breaking down the top-down hierarchical architecture to heterarchically communicating entities that are working towards common or shared goals.

Society

Water systems are harnessing smart systems to improve and provide solutions for water quality and distribution management considering location of buildings and environmental contingencies such as climate change, flood and other natural disasters, etc. and their prediction and management. Along with water management, waste management and environmental sustainability is also touched and considered by smart systems to control the entire waste cycle from their collection to transportation, depositing and processing, and to monitor and control the pollutions or environmental effects caused by various wastes.

Transportation

Cyber Physical Systems (CPS) and ICT technologies are being extensively used in scheduling and safety of public transportation system and vehicles as well as smart parking systems such as Parking Guidance Information Systems (PGI) or smart and E-Parking, etc. [48-50] all together giving rise to what is commonly known as Intelligent Transportation Systems (ITS) [37].

Beside public transportation personal vehicles are extensively enjoying ICT technologies and smart systems in their different parts such as engines, transmission system, suspension system, their stability program, or in their safety features. Moreover, cars with autonomous driving capabilities are vastly advancing by harnessing the capabilities of smart systems and their interactions and collaborations.

All abovementioned areas as well as other aspects of smart systems' broad applications such as in education systems, economy, as well as in government, have led to the emergence of the concept of Smart Cities which is targeted by increasing number of cities all around the world.

Bellow (figure (6)), shows the maturity model of a cyber physical system, according to [51]. Cyber physical systems' maturity stages can be noted as basics, enabling transparency, improving the level of understanding, enhancing decision making capabilities, and eventually the ability of self-optimisation. The basics requires simply the general conditions within the organization and its structures. Then moving upward in maturity levels, the necessities and requirements include processing and comprehension of the acquired data and information, as well as communicational and interoperability factors. Acquiring data and information accordingly demands real-time generation of data and information with regards to the functions and behaviours of that cyber physical system. On the other hand, information processing suggests the ability to derive and cumulate knowledge and understanding from sets of data. Then it moves to information linking

which is about process adaptation on the basis of cooperation. The highest degree of maturity can be obtained by independent cyber physical entities with the ability of making decisions, solving problems, and interoperation and collaboration.

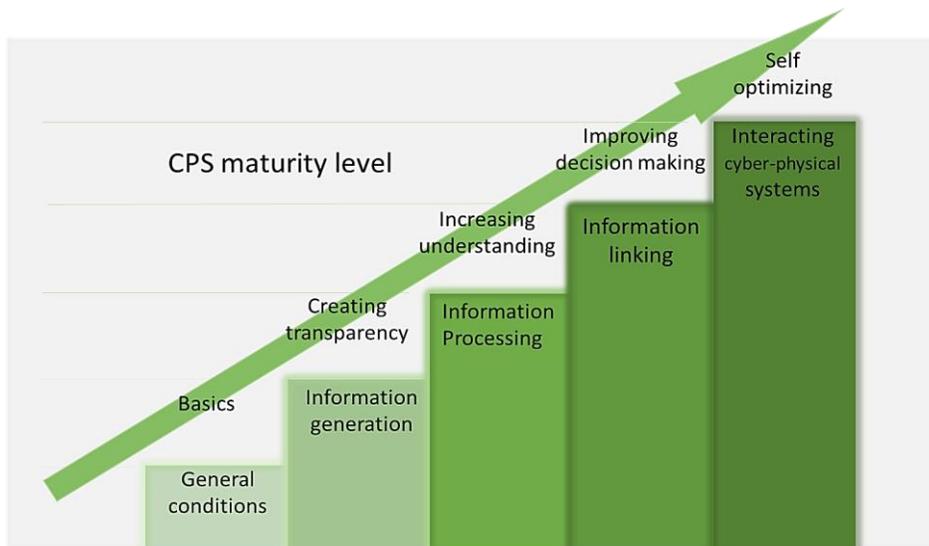


Figure 6. CPS maturity model (CPS in manufacturing)

2.1.3. Cyber Physical Systems Properties

Cyber Physical Systems along with their structure similarities, generally share a set of properties to different degrees. To deliver their potentials, CPSs mature these properties that they inherit thanks to advancements of ICTs. In general, New technological applications equipped by cyber physical systems and Cloud technologies, are demonstrating higher level of agility, better flexibility, and broader ubiquity and manifests a score of certain characteristics and properties [9,52,53] that being discussed in this section.

First thing to see as the main property of cyber physical systems is that they enable heterarchy and horizontal integration and collaboration along with hierarchical top-down and vertical structure. This for sure demands much higher level of interoperability which is defined as *the ability of two or more systems or components to exchange information and to use the information that has been exchanged* [54], among components of any level of detail (LoD). This requirement arises following the intrinsic heterogeneous characteristic entities that are interacting in CPSs, and due to the existence of various sources of information and the involvement of a variety of data formats. and a seamless cooperation of these entities, demands greater degrees of interoperability which is going to be discussed further in details later in this section.

The next behaviour to be found in cyber physical systems is Decentralization, which in manufacturing context is regarded as one of the outstanding features of the new industrial revolution. For decentralization to become achievable presence of adequate autonomy is imperative along with sufficient degrees of self-awareness and context-awareness, that are also going to be discussed in more details within this section. In general, with autonomy components can react on their own through what they receive from the environment and with self/context-awareness, they can know and express themselves in the system. By exploiting these features smart entities are enabled to make decisions on their own by interacting with other entities and the environment where they perform.

With regards to the intentions of this study a number the most important of these properties which are going to be used within this thesis are further elaborated below:

Context-awareness

ITU recommendation for Smart Ubiquitous Networks (SUN) defines context as “*the information that can be used to characterize the environment of a user*”. Accordingly, it further defines Context-Awareness as “*a capability to determine or influence a next action in telecommunication or process by referring to the status of relevant entities, which form a coherent environment as a context*”. [55] Context data can range from information about the environment where an object is operating (e.g. temperature, noises, vibrations, humidity et.), object’s communications and collaborations (e.g. the entities and devices with which the object is connected or cooperates, data in communication, bandwidth used, etc.), or information about the operation and the process, etc. These items can vastly vary in type or scale based on objects’ functionalities and structure. Cyber physical systems are generally equipped with smart ICT devices and technologies to gather context information. Examples can be obtaining data from various sensors, or via cloud. Data acquired will then be processed and translated into a meaning full information based on which analysis, reasoning and decision making can be carried out.

Heterogeneity

Heterogeneity is another intrinsic characteristic of cyber physical systems as these systems are networks of communicating components of generally vastly diverse type and structure with different constituent pieces of hardware of different make and computational power as well as dissimilar software platform, communication mechanism, or programming languages used in their architecture.

Such diversity is one of the major challenges in making interactions and collaborations between these entities possible. Establishment of standards and proper semantics and ontologies help overcoming heterogeneity.

Interoperability

Interoperability is defined by the Institute of Electrical and Electronics Engineers standards glossary as “the ability of a system or a product to work with other systems or products without special effort on the part of the customer” [56] and implementation of standards is the way to enable interoperability. That suggests that heterogeneous entities, regardless of their manufacturer would be capable of cooperating and collaborating in one or various distributed applications, according to IEC 65/290/DC standard [57].

That definition puts interoperability high in the compatibility scale presented in IEC 65/290/DC [57] as shown in figure below. That requires ICT devices to be capable of sharing information and using the exchanged information for intended applications.

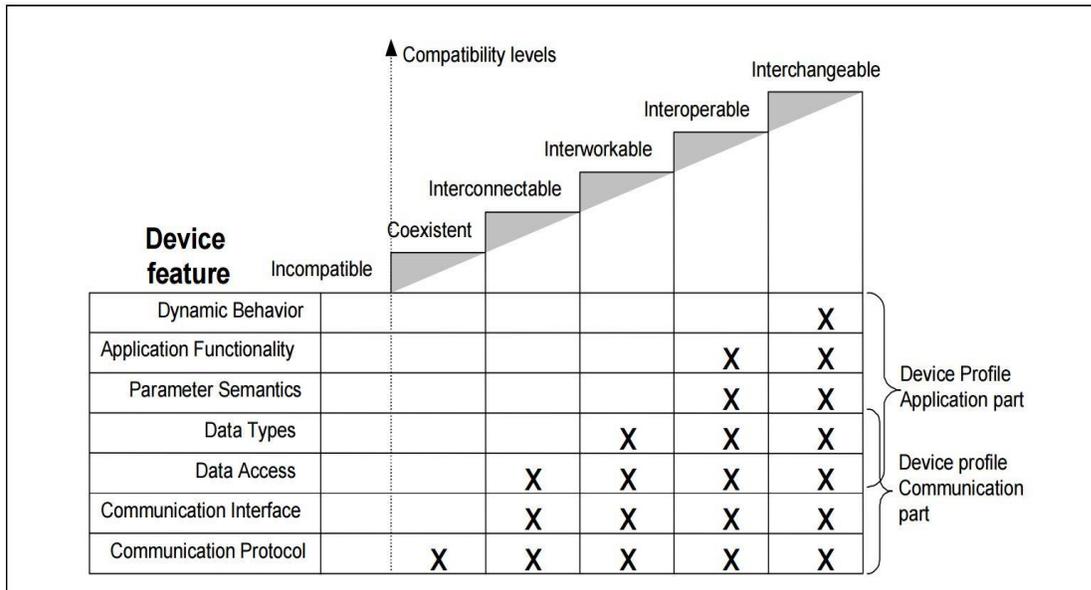


Figure 7. Compatibility Scale against device features (IEC 65/290/DC)

In order to reach interoperability, definition of ontologies as well as determining a model for semantics to be share by interacting components, on the basis of approaches like defining key performance indicators (KPIs) can be important.

Autonomy

Autonomy is another feature of cyber physical systems that is manifested by smart components in various levels. It can be basically defined as the ability of an entity to struct and perform its actions and behaviours without the need for human or other entity's intervention. Among aspects of autonomy the ability perceiving the environment through sensors or by any means, reasoning on the data acquired from the environment and to react accordingly. In other words, autonomy refers to the notion of collaboration with or to self-organisation to react with regards to external stimuli, through setting up self-fed loop with the context in which the given component performs.

Objects with high level of autonomy would be able to carry out self-reconfiguration and self-optimization along with self-control to deliver higher expectations and more complex objectives that match new industrial revolution's criteria.

Scalability

Another property of Cyber physical systems is scalability in the sense that adding or removing resources to and from the system should not require major effort. Scalability can cover a broad range of attributes according to the system under consideration as the target of scalability.

It can be computational resources such as processing power or storage, software capabilities and applications (for instance in cloud-based systems), addition or reduction of a set of features or behaviours with regards to the dynamic requirements of the time and accordingly changes in tasks and objectives of the system, pieces of equipment and expansion or shrinkage of the capacity, range of coverage as in the case of monitoring and supervision of processes or other components, updating databases in real-time and the ability of making use of the updated knowledge, or changes in volume or number of variables in data acquired from the context via additional sensors or cloud, etc.

Due to the dynamic nature of cyber physical systems and the growth of the desire for higher levels of flexibility and adjustability in response to the need for either intentional strategic changes for gaining more advantages, or unintentional ones such as in cases of breakdowns, scalability becomes one of the main features of CPSs.

Modularity

A modular system consists of loosely coupled elements that when required, can be reorganized in new setups or be interchanged with functionally similar elements irrespective of manufacturer,

or changed with elements of different functionality with little or no effort. CIRP encyclopaedia of production engineering defines modularity as *the compartmentalization of production functions and requirements into operational units that can be manipulated between alternate production schemes to achieve the most optimal arrangement to fit a given set of needs* [58].

With regards to system and software engineering, ISO/IEC 12207 [59] calls modularity the minimum coupling and maximum cohesion in a system, and in ISO/IEC/IEEE 24765 [60] modularity has been defined as *the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components*. Modularity is an important supporting feature for enabling reconfigurability, and customizability. In the language of industry 4.0, modularity is a necessary feature to possess, which affords the system greater degrees of flexibility that is essential in ever increasingly changing and dynamic context and environment. Modularity in parts of the system and its resources (e.g. pieces of production equipment, transportation mechanism such as conveyors, software applications, controlling systems, etc.) can bring better reconfigurability in the face of stimuli. It also enhances the agility of the production system in responding to changes in the product or service should it require.

Another main attribute of importance in the manufacturing era which stands out among the characteristics of cyber physical systems is Real-time capability. This ability is to be enabled via real time gathering and monitoring of data from sensors (which requires real-time availability and accessibility of required data), analysing them, making decisions accordingly, and applying these control decisions to the physical world using actuators. Service orientedness is the property that should not be missed when it comes to smart and cyber physical systems, more specifically with regards to the movement towards new industrial and manufacturing era. Vendor independent and loosely coupled services provided via applications and networks is provided to customers of these services in an ad hoc way. Provision of services via applications on the internet and the concept of IoT (internet of things) along with other technologies make possible the emergence of service oriented architecture (SOA) in manufacturing industries.

Within their concepts, cyber physical systems and Industry 4.0 have goal of bringing together the virtual world and the physical world [51]. Accordingly, the fourth industrial revolution and smart manufacturing has virtualization and representation of physical entities in cyber space as one of its main attributes. Using sensor data collected from physical processes and producing models in cyber world, associated engineers in process control and optimization, product designers, managers, etc. can customize and optimise models and interactions between digital

(or virtual) twins of manufacturing resources and components in the virtual world by changing and testing models via simulations and improving on them while the physical process are running. This way manufacturing processes and parts' design can be modified and improved upon faster and with much lower costs, and responses to changes can be applied in a more efficient and timely manner.

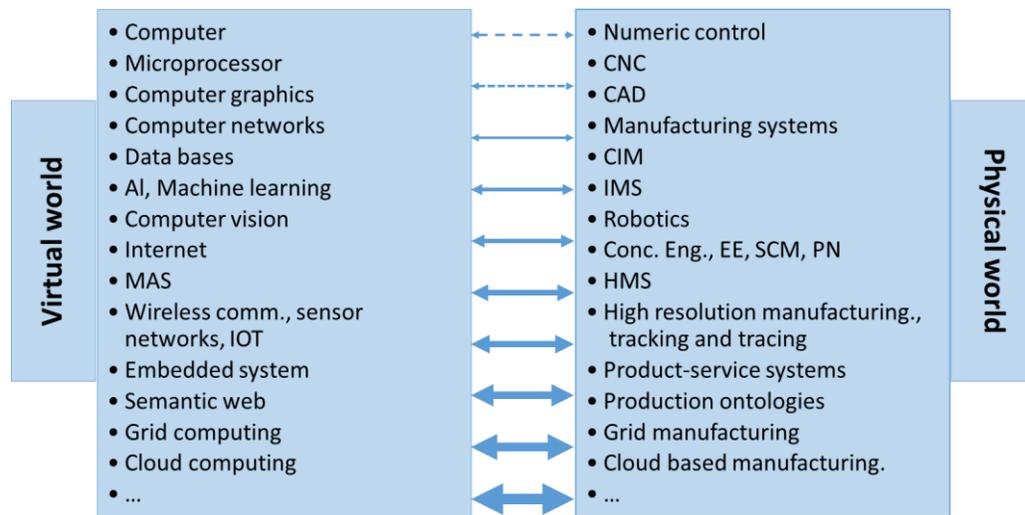


Figure 8. How ICTs grow to bring cyber world and real world together [51]

2.2. Smartness of Manufacturing Systems

To raise their strategic advantages and in order to gain higher competitiveness, manufacturing systems are aiming at exploiting the latest and cutting-edge information and communication technologies that are growing rapidly following the Moor's law which indicates that computing would exponentially increase in power and decline in their cost. This trend enabled manufacturing systems to move into the era of automation and harness computers and logical controllers to aid management, supervision and control of manufacturing processes through approaches such as CIM (Computer Integrated Manufacturing), CAD/CAM (Computer Aided Design/Manufacturing), etc.

2.2.1. Towards the new Industrial Revolution

In recent years, through the introduction of smart objects and embedded systems into manufacturing enterprises and production practises, this industry is taking a revolutionary step into a new era and paradigm known as Smart Manufacturing or as it is called in Germany, Industry 4.0 (which indicates the 4th industrial revolution). This new paradigm is about the convergence

of existing manufacturing systems and advanced ICT technologies to enable more flexible processes, more advanced decision-making capabilities and in overall the manifestation and continuous maturation of the previously mentioned properties within the manufacturing organizations. NIST (National Institute of Standards and Technology), defines Smart Manufacturing as “fully-integrated and collaborative manufacturing systems that respond in real time to meet the changing demands and conditions in the factory, supply network, and customer needs.” According to the Implementation Strategy Industry 4.0 established by industry associations VDMA, ZVEI, and Bitkom [39], real-time availability of information through the networking of all entities in value chain and enhancement of value stream based on provided real-time data is of the fundamentals of this new paradigm. Such networks of interacting entities from humans to objects and systems enables a dynamic value network which is capable of performing real-time self-optimization with regards to desired criteria from resource utilization to cost control or optimum time.

The main distinguishable difference between the legacy system and Industry 4.0 (for short I4.0), is that in I4.0, objects instead of communicating with a centralized database, communicate with their digital twins and models in the cyber world, where all these digital twins and virtual objects (VOs) are communicating together. The figure bellow, shows the way cyber physical objects work in I4.0 structure. Explained in Rami 4.0 (Reference Architecture Model Industrie 4.0) [38], an object is an I4.0 component when it has communication ability, and an “administrative shell” that represents the object in virtual world and carries its functionality model.

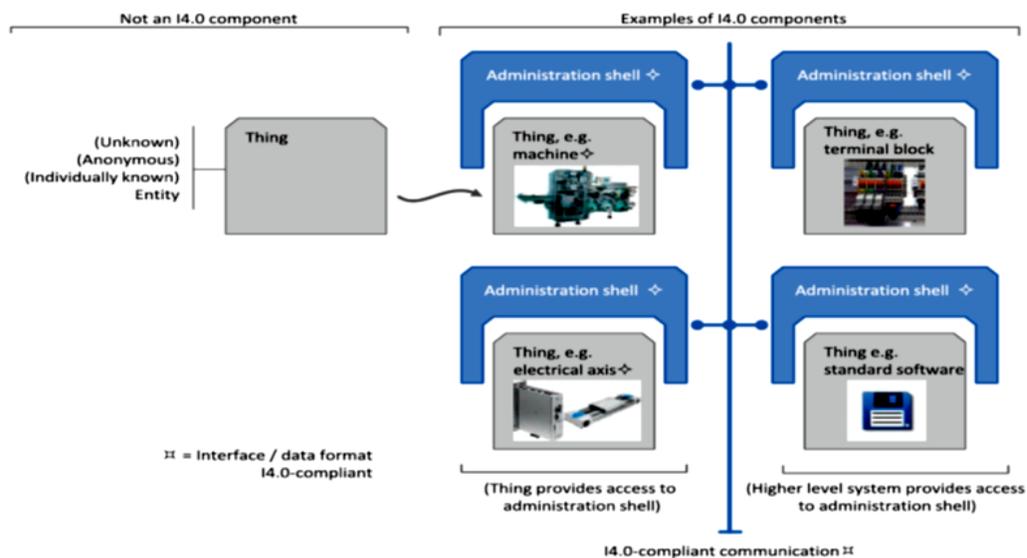


Figure 9. Industry 4.0 components in comparison with legacy systems components

Major countries are now working on developing and integrating key enablers of this industrial revolution. The main technologies based on which smart manufacturing is developing are cyber physical systems, as well as Internet of Things (IoT), Big Data, Cloud Services, industrial Wireless Sensor Actuator Networks (WSAN), etc. IoT usually refers to a network of interconnected heterogeneous objects (sensors, actuators, smart devices, smart objects, RFID, embedded computers, and so forth) uniquely addressable (for instance by IP), based on standard communication protocols. It was further defined as a loosely coupled, decentralized system of cooperating Smart Objects [61].

In Industrial Internet of Things (IIoT), IoT provides an internet-based integration for cyber-physical objects that in this case are manufacturing resources and applications. It connects machines and facilities embedded with sensors and actuators, RFID tags and readers, positioning systems etc. and affords the industrial systems higher observability over the entire supply chain and resources as well as operations, more autonomy and awareness of components, improved Key Performance Indicators (KPIs), enhanced levels of maintenance, and way better level of data collaboration and real-time reconfigurations triggered by dynamic environments.

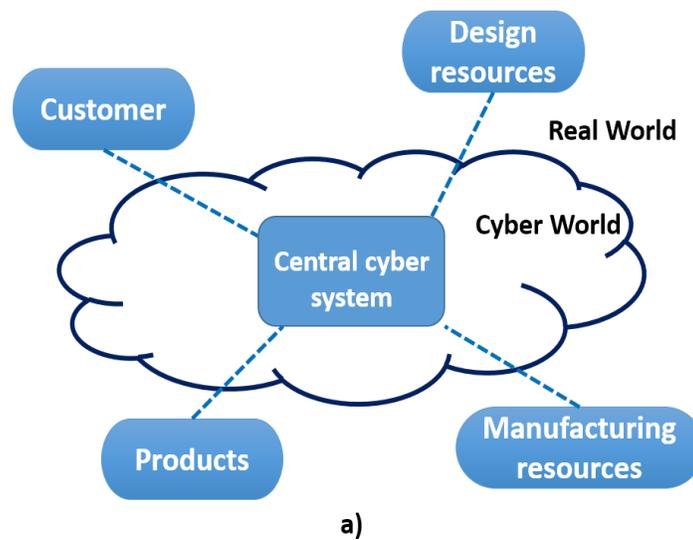
Inferred from the abovementioned, a large amount of data is in transition within this system that makes industrial Big Data to be frequently heard along with IIoT. The term industrial Big Data suggests a great Volume of data of Various sources to be gathered at high Velocity (in real-time), giving industrial Big Data its 3 dimensions (3Vs), and raising the need for decent analytical methods to deal with this heterogeneity and largeness of data acquired in real-time. It also requires large processing power, massive storage, and network capabilities [62].

Speaking of these requirements, cloud service with its ginormous capabilities is now considered to be summoned to answer industries' needs. Accordingly, the concept of Cloud Manufacturing (CMfg) has emerged. One definition of Cloud manufacturing was brought in [63], as a customer-centric manufacturing model that exploits on-demand access to a shared collection of diversified and distributed manufacturing resources to form temporary and reconfigurable production lines that enhance efficiency, reduce product lifecycle costs, and allow for optimal resource loading in response to various and changing demands of customers by flexibly defining and reconfiguring operational tasks.

CMfg is the application of cloud computing, Business Process Management (BPS), and IoT in manufacturing, and considers CPS objects and shop floors [64] in a Service Oriented Architecture (SOA). Cloud-based manufacturing deals with generally four types of services delivered to cloud service customer typically by cloud providers and brokers:

Hardware as a service (HaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) i.e. a set of hardware provided normally by service provider as requirements of cloud service implementation, an infrastructure and networking system in which components (things) can connect, communicate, and collaborate (IaaS), a platform for e.g. dealing with large amount of data (big data) and information processing, search engines (PaaS) in the middleware, and various applications for enhancing the efficiency, and software packages like CAD/CAM, ERP etc. (SaaS) [64].

Putting all in a nutshell, it can be perceived that CIM was more of an enabler for third industrial revolution, whereas, CPS, IIoT, and CMfg are key enablers are Industry 4.0, illustrating that manufacturing system is now going to be more decentralized, more service-centred, and relied more on smart objects [65]. In figures below the structural differences between computer integrated manufacturing, cyber physical systems, and cloud manufacturing is presented [66]. The evolution from centralized systems to being more decentralized and service oriented is observable.



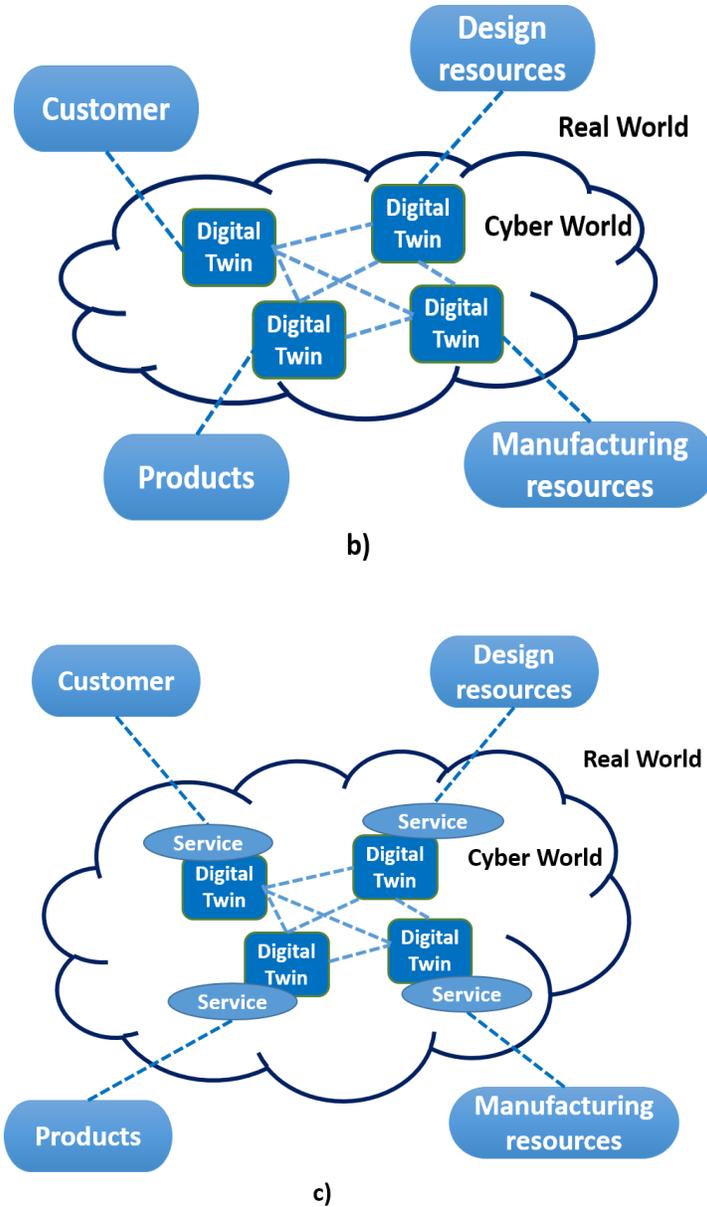


Figure 10. Advancement of the use of ICTs in manufacturing, from: a) Computer Integrated Manufacturing (CIM), to b) Cyber physical systems (CPS), to c) Cloud-based manufacturing systems (CMfg)

The mechanism to be presented in this thesis can be considered as an addition to the intelligence and awareness of the digital twins of the industry 4.0 objects.

2.3. Multi-Agent Systems (MAS) as the toolset for Model Implementation

Intelligent Agents (IA) have emerged and become a point of interest for researchers in early 1990s. Due to their capabilities these agents and systems on the basis of them have grown significantly, which led to the raise in their attraction specifically in the fields of computer science. Soon the interest in agents started to spread way beyond the borders of computer science and became a multi-disciplinary research area to be seen in engineering fields, medical science research and applications, even psychological studies, and so forth. Agents are entities equipped with computational power that are able to demonstrate degrees of autonomy, goal-oriented decision making and the ability to find solutions for various problems they encounter.

To achieve their purposes, they may show reactive or even pro-active conduct in a dynamic and hardly predictable environment they might be working. Despite their vast capabilities, often due to rather inevitable limitations of individual agents, complex tasks and real-world objectives are sought and carried out through collaboration of multiple intelligent agents. This community of interacting and cooperating agents which might individually perform semi or fully autonomously to fulfil their own objectives and to serve system's global objectives is called a Multi-Agent System or for short (MAS) [67].

Agents in a multi-agent system approach their goal by harnessing their smartness and rationality ability to communicate and negotiate with one another in order to better collaborate and perform responsibilities towards intended goals. Obviously, such behaviours from agents demands them to possess sufficient understanding of the real world and the capability of rational decision making. With this goal in mind, researchers introduced a system around late 1980s, which is called Belief, Desire, Intention (BDI) and has ever since become the most common approach in developing agents involved in multi-agent systems.

Agent's beliefs are its knowledge and comprehension of the section of the real world it is dealing with, and can be unique to individual agents, meaning that various agents might have diverse beliefs about a context. On the other hand, is agent's desire which tells the objectives of that agent and all it aims to fulfil. These desires can be put into two groups: short-term desires and long-term desires, where the latter provides the main structure of agent's behaviours and is more abstract than the former, short-term desires, which are derived from long-term goals and are to deal with specific problems of any specific moment. Intentions are desires that the agent has decided to achieve, and to achieve them it performs a series of actions that themselves might

be a part of a plan, which along with other plans are being performed by the agent to serve its intentions. Intrinsically, intelligent agents (IA) demonstrate following properties [68]:

Autonomy: smart agents can control their actions and perform independently without a need for help of other agents whether real world agents like humans or other smart agents in a system.

Responsiveness: using sensors to cumulate knowledge of the states and behaviours of its environment and through actuators by which it can act upon its environment, intelligent agents can respond to environmental stimuli with regards to the goals and desires they are programmed to achieve.

Proactiveness: more advanced agents can be proactive in the sense that they might be capable of estimating or predicting the state of their environment should they perform a specific action or they may act upon what they have foreseen about their environment.

Goal-orientation: agents' behaviours are goal driven which means their actions are planned to let it reach its intentions and desires.

Smart behavior: here smart behaviour suggests that the agent has adequate knowledge of a specific area as well as its environment and hence it is able to solve problems of that very field in which it is an expert.

Social ability: as intelligent entities, agents are able to interact and collaborate with others, including humans or other agents towards their goals and fulfilling their tasks and providing expected services. This way they can use one another's aid in predicting conditions they are limited to predict solely, or accomplish tasks they could not otherwise have accomplished on their own. That may require them to have an abstract knowledge of their counterparts or components they cooperate with.

Learning capabilities: another interesting feature of the agents is that they are learning capable which means they are adaptive and can incrementally cumulate and evolve their knowledge and optimize themselves possibly without human intervention.

After determining the tool, the model is translated into an architecture composed of interacting agents. To this aim a design methodology can be used for building the intended architecture.

2.4. The DACS methodology

Stands for Design of Agent-based Control Systems, DACS method is one of the most common and thought after techniques which being used in developing multi-agent systems and mainly focuses on the acting agents and their internal communications [69,70]. This technique can use decision making approaches to produce a model from which agents can be characterized and chosen. It takes the behaviours and features of agent systems into account and helps developing agents and their interactions based on each specific cases and functionalities they need to serve. This technique therefore can accommodate the needs in tools for developing an agent based system by taking all the concepts of agent control systems into its consideration. Accordingly, in order to deliver its goal, the DACS methodology follows three steps that are described as follow [71]:

1. Analysing the decisions to be made. First step is to specify and analyse the controlling decision that must be made throughout the processes by the agents. In addition, the necessary decisions, their interdependencies are also important to be specified, since agents' interactions can be determined and designed based upon a constructed model of these relations between decisions.
2. determination of agents. In this step, the agents required for delivering the decisions specified in the last step will be identified to design an agent oriented system. Furthermore, the decisions each of them should make will be assigned to them along with all their deliverables and services they need to provide, as well as the requirements for their interactions (i.e. when to interact, what to interact or share, with which other agents, etc.) shall be specified.
3. Choosing interaction protocols. In the third step means of interaction between agents will be defined and interaction protocols will be selected among the available agent-oriented protocols.

Following these three steps results in structured design of interacting agents each of which has sets of controlling decisions to take and objectives and responsibilities as well as various interactions with other agents. The following figure shows the steps of DACS methodology and outcomes thereof.

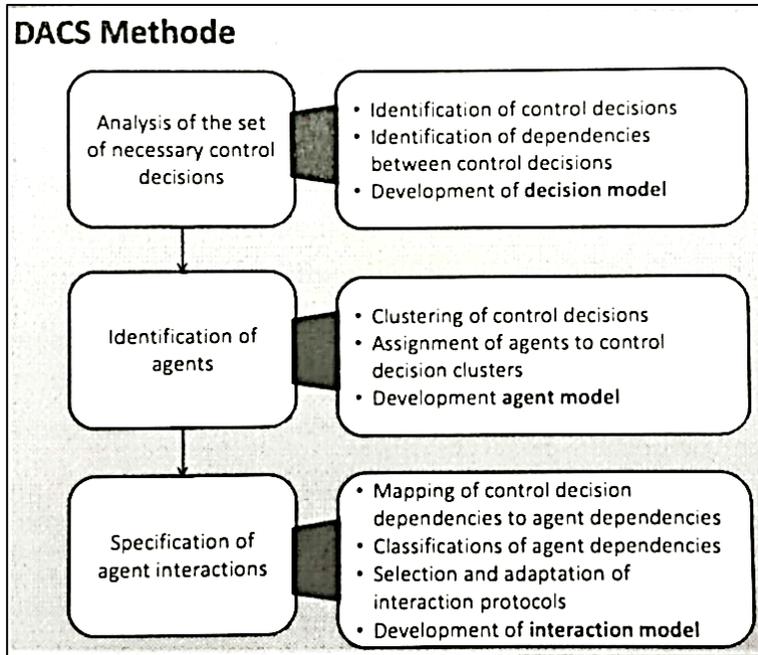


Figure 11. Steps of DACS methodology

2.5. Jade platform

First introduced by Research & Development department of Telecom Italia, JADE (Java Agent DEvelopment framework) can be referred to as the most common agent-based middleware that is being utilized these days. It provides a totally distributed platform with a flexible structure which also makes extension and add-on installations easily possible [72]. Using this platform enables developing fully agent-based real-time capable applications which can be used in any environment. Being developed in Java, JADE has the advantage of enjoying all the capabilities of this language as well as rich third-party libraries that are available, which makes using JADE and agent programming easier for less experience people.

The following figure (12) depicts the structure of the JADE platform. As can be seen, it gives a main container which dwells the Directory Facilitator agent and Agent Management system using which agents can register themselves to the system, perform activities, and offer and provide services. Their life cycle can also be easily managed as they can be stopped or terminated, paused, move, etc.

The main container also has a Container Table (CT) which is the list of registered containers to the system and Global Agent Descriptor Table (GADT) which provides information about all the registered agents in the platform.

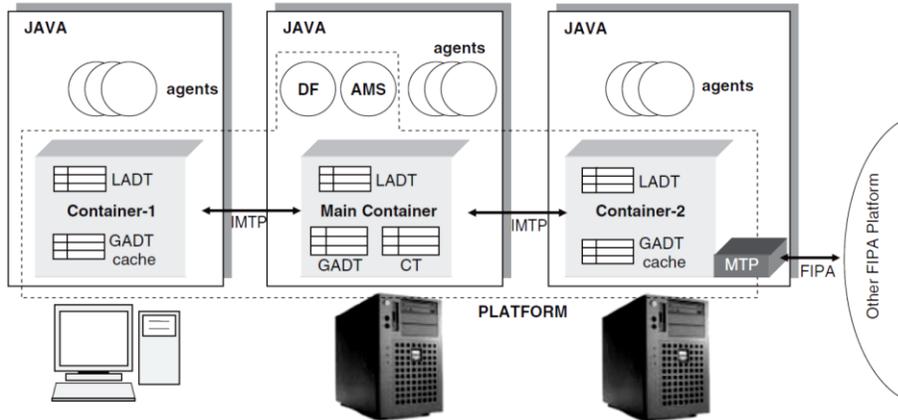


Figure 12. The overall structure of the JADE platform

JADE platform can afford the programmers and agent-based system developers with following capabilities and functionalities. It needs to be mentioned that JADE's features are not limited to the following list.

Firstly, JADE is fully distributed system with graphical tool. Agents can live on this platform, running as different threads and located on various machines and they can communicate through provided means. Also, the graphical tools can help with debugging and monitoring events or communications that take place among agents. Secondly, it is in compliance with the FIPA specifications. Since developers of this platform themselves contributed in FIPA standards development, this platform also conforms to its specifications such as identification, interoperability, etc. Implementations of both white pages and yellow pages is another feature in this platform. White page and yellow page services are provided by Agent management system (AMS) agent and Directory Facilitator (DF) agent respectively. Agents can register and offer their services easily through the yellow page service by DF agent. There is also an effective, agent life-cycle management. Agents when registered to the white page service of their platform can easily be managed to start, stop, pause, die, migrate, etc.

Subscription mechanism exists for agents and other applications to receive notification about events that occur in the platform. Moreover, supporting ontologies and content languages, agents' migration, wireless environments, etc. are some other features provided by JADE platform. The following figure shows JADE's interface and the main container.

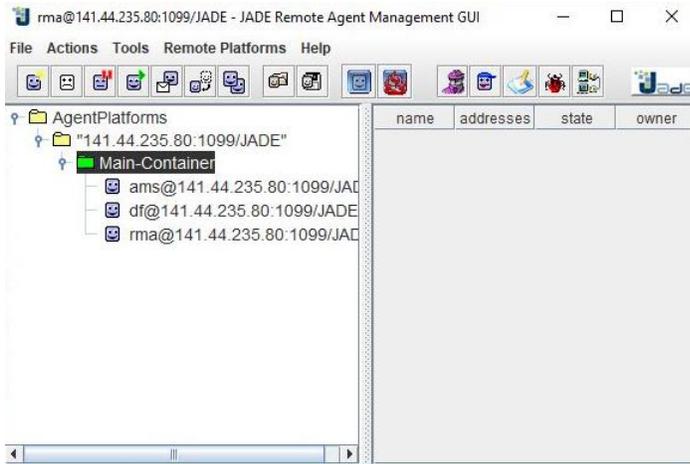


Figure 13. JADE’s main container

To run JADE agents for this thesis, Eclipse has been chosen as the compiler in which programming codes were written. It is considered as an integrated development environment (IDE) and is known to be one of the most commonly and frequently used application for developing JAVA programs. Eclipse offers a user friendly and customizable workspace easily extendable using various plug-ins. This application itself is mostly written in JAVA and can be used for developing various different programming languages including JAVA. Below is a screenshot of the Eclipse workspace while it was being used for the purpose of this thesis.

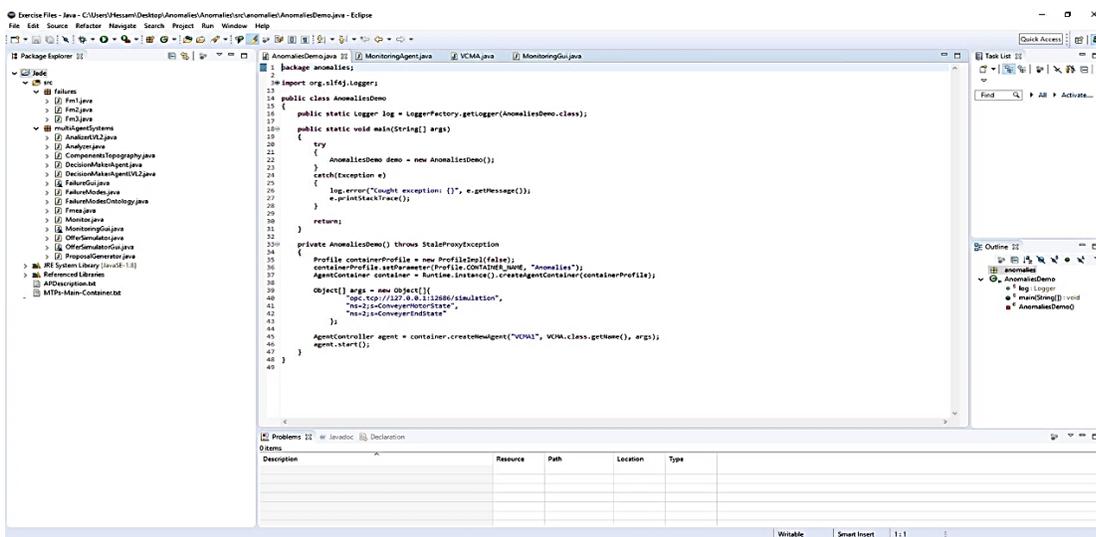


Figure 14. Eclipse as the compiler used to develop and run JADE agents

2.6. Dependability and Security of Smart Manufacturing Systems

Dependability and security have always been major challenges for ensuring a flawless and robust performance in a system, and as far as these two aspects are concerned, with the introduction of more complexity and transmission and exchange of information and data, the system becomes increasingly vulnerable towards dependability and security issues. Accordingly, with regards to the fact that smart manufacturing systems enabled by cyber physical systems are generally comprised of a large number of heterogeneous and interconnected components and are dealing with substantial amount of data flow among them as well as vast exposure to the cyber space and internet, dependability and security gain exceedingly higher significance [73]. This is because inefficiency of these two aspects or the lack thereof may lead to costly or in cases disastrous breakdowns of the system due to ineffective or tardy responses to failures and risks, disclosure of critical information or loss of data integrity, incompetence in predicting coming failures, etc.

2.6.1. Aspects of Dependability and Security

Dependability is a concept that according to the system in which it is considered, can cover a variety of attributes such as reliability, stability, availability, safety, integrity, maintainability, etc. It is in overall to be matured to assure that a given system is going to work flawless and without unacceptable interruptions and will fulfil and deliver its objectives and services within satisfactory specifications, costs, as well as timeliness.

The concept of security (concerning ICT security) can be perceived as *the collection of tools, policies, security concepts, security safeguards, guidelines, risk management approaches, actions, training, best practices, assurance and technologies that can be used to protect the cyber environment and organization and user's assets* [74], on the other hand generally focuses on the three main attributes known as CIA triad (i.e. Confidentiality, Integrity, and Availability).

[7] in IEEE Transactions on Dependable and Secure Computing, demonstrates the main dependability and security attributes and their general (not definite) interrelations that are summarized in the following figure.

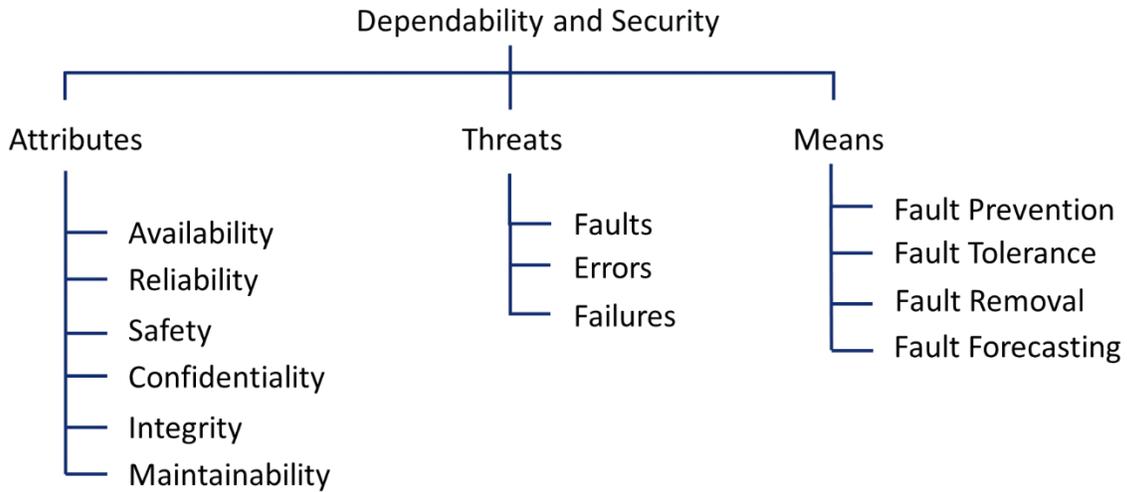


Figure 15. Dependability and security tree

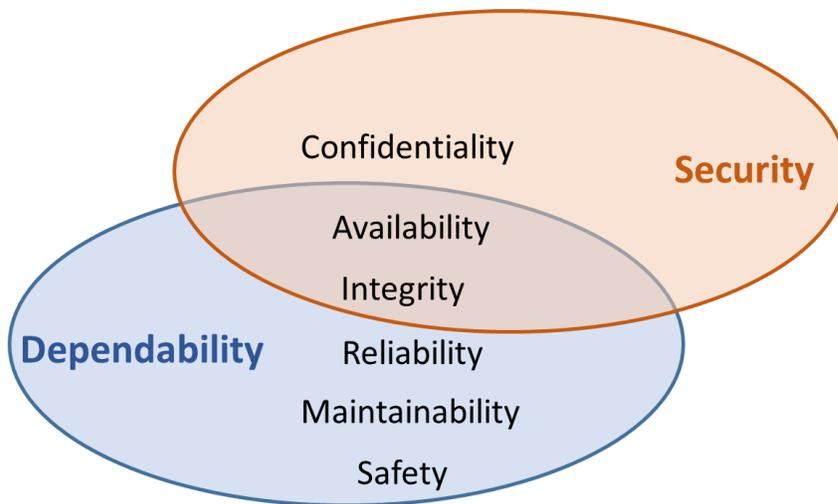


Figure 16. Dependability and Security attributes and correlations

These attributes can be defined as follow:

- **Availability:** suggests that services, resources, information, etc. must be available to the right entity at the right time.
- **Reliability:** is defined as the ability of the entity to perform with consistency within the desired condition and specification for an expected period of time.
- **Safety:** is to assure that assets (humans, equipment, etc.) or the environment are protected from harms and hazards.

- **Confidentiality:** to protect and preserve the confidentiality of information means to ensure that it is not made available or disclosed to unauthorized entities. In this context, entities include both individuals and processes [ISO IEC 27000: 2014] [75].
- **Integrity:** it refers to accuracy and completeness. That means there should be no unauthorized alteration.
- **Maintainability:** degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers [ISO/IEC 25010: 2011] [76]. Modifications can be repairs, corrections, updates and upgrades, enhancements and adaptations (e.g. of software according to changes), etc.

Dependability and security, beside the abovementioned attributes, based on the system under consideration can have broader objectives and are not limited to these aspects. For instance, International Telecommunication Union (ITU) lists some of the main security goals of the ICT networks as follow, and suggests careful attention to security considerations in all phases of design, implementation, and in real-time while the system is working [77]:

- Only righteous entities should receive access to data and the network
- Each legal entity should have access to what it is authorized for
- Confidentiality of a given network should stand up to what is defined as expected level within its policy objectives
- Components in a network should be only responsible for their own actions and not those of others
- Adequate protection should be guaranteed for networks with regards to standing against activities that unsought and/or are unauthorized
- All information related to security of any sort should be only accessible by authorized entities;
- Network must be provided with plans with regards to tackling security related incidents;
- Network must have strategies available for restoring its normal behaviour and processes should any security issue occurred and dealt with; and
- The structure of the network should allow a variety of security strategies and approaches of any type and intensity to be applicable.

The concept and purpose of dependability and security is to assure acceptable delivery of intended services. Ergo, the mechanism for guaranteeing dependability and security should protect the component from anything that may pose risk to the delivery of adequate services. In figure 15, these causes of service inadequacy are categorized as threats, that are failures, faults, and errors. In general, faults are reasons for the occurrence of errors and one or more errors may lead to failures. Failures eventually may cause the system to fail in delivering the required output. Understood thus, in order to maintain the consistency of satisfactory fulfilment of services in a system, vulnerabilities of the system that may be compromised by faults, the associated errors, and consequent failures should be well understood.

To understand faults, failures, and errors, [7] provides thorough classifications that come very helpful both in categorizing them in almost any given system, and to determine them accordingly. For instance, it can be seen that faults can be classified based on various criteria such as phases of the system i.e. when it is performing or at the time of development (e.g. maintenance); whether it is originated within the system or it is caused from a change outside the boundaries of the system; if it is intentional, accidental, hardware or software related (i.e. physical devices, or programs and data), etc.

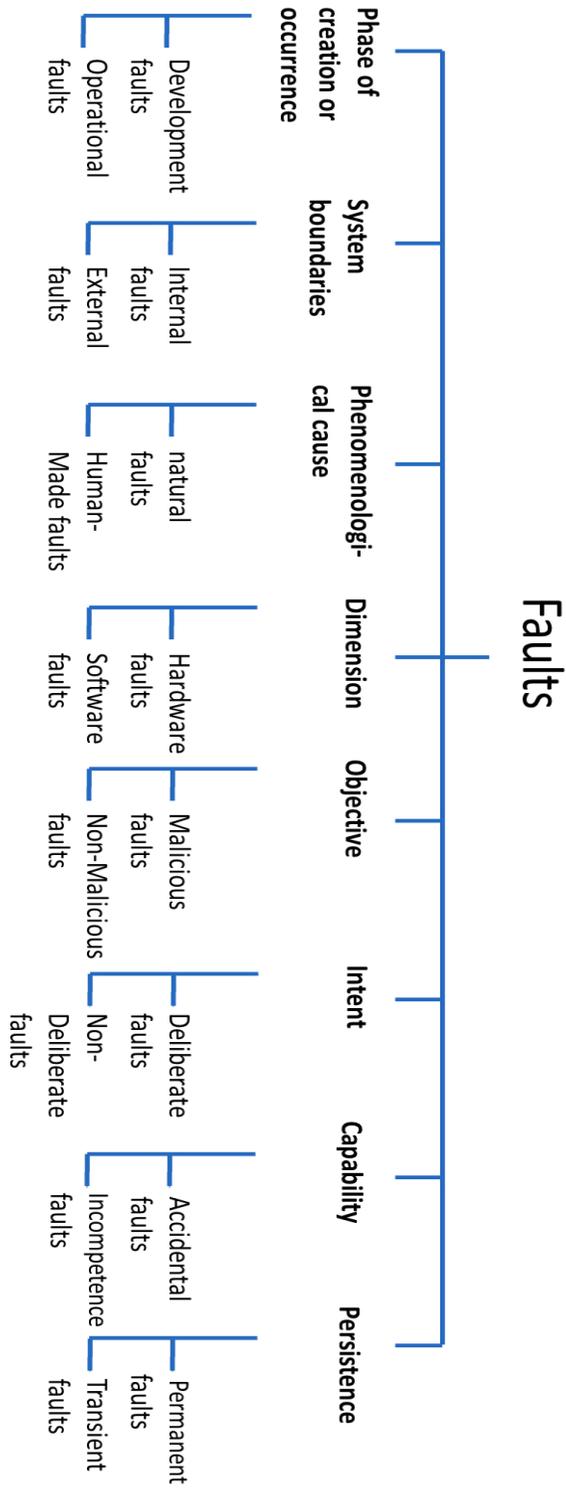


Figure 17. fault classification according to [7]

To tackle these threats the dependability and security mechanism should first be capable of detecting faults and errors should any occur. Also, predictable failures and associated faults and error should be foreseen to be dealt with adequate preventive exercises. Furthermore, the mechanism should be ready to perform proper actions in a timely manner in cases of any fault occurrence. The right action should provide the system with stability and should remove the failure and its causes. The feasibility of these actions relies on the two major properties in the system:

- ✓ Firstly, the system must be observable
- ✓ Secondly, the system must be controllable and reconfigurable

The former suggests that components of the system must be observable to dependability and security mechanism so that should anything out of the approved context occurred, it can be detected within acceptable time. That includes factors that can affect the performance of the system both internal (from within the system, e.g. components and communications among them), and external (outside the boundaries of the system, e.g. the environment). Observability can be fulfilled using proper sensor networks, accessing and reading data communication flow between entities, etc.

The latter property, which is controllability and reconfigurability refers to the ability of being changed, replaced, or fixed when an undesirable phenomenon takes place. Alteration should be easy to make by the controller. For instance, compromised computers or codes must be reconfigurable and physical components must be changeable. This property plays a crucial role in making system stabilization feasible when due to a failure, a constituent component has become unstable.

2.6.2. Dependability and Security issues of Cyber Physical Production Systems

Cyber physical production systems are manifestations of cyber physical systems in manufacturing and production environment and context. Hence, in order to find dependability and vulnerability of these systems it would be reasonable to investigate risks that are posed to cyber-physical systems.

Cyber physical systems as it can be perceived from their name, generally consist of three layers [78-80] that are namely: physical layer, cyber layer, and the integration layer or communication layer. Accordingly, each of these layers can have their own concerns and susceptibilities with

respect to dependability and security. Due to the enormous potential advantages of these systems, numerous studies have recently been dedicated to delve into finding CPS and Smart systems' vulnerabilities [74, 78, 79, 81-95].

Physical layer

Physical layer often consists of physical naturally tangible assets like pieces of equipment, machines, transportation vehicles, parts, materials, etc. (in general various sources of information generation), as well as sensors and actuators. Due to any type of fault (e.g. intentional, accidental, etc.) for example some sensors may stop working, creating a coverage hole. In some cases, one or more sensors may be influenced by attackers to generate ill data, or may be replaced by a malicious one to send erroneous data to the system, causing interruption in decision making processes and bringing damages to the system.

Sometimes sensors are connected to a sink node or base and these nodes are the victim of faults which may lead to more serious conditions, especially if for instance due to higher data flow, they are spotted by adversaries and be compromised for intentional undesirable influences or for eavesdropping confidential data. Sensor can also stop providing data if their power source runs out. Though extremely energy efficient, some smart objects or sensor may run out of battery or power due to various reasons. An intentional reason can be Denial of Sleep (DoS, not to be confused with the other DoS (Denial of Service) attack) attack in which an adversary keeps the device unnecessarily working until its power source depletes and is no longer capable of performing. In general, the higher the geographical dispersity and exposure, the more the vulnerability of the physical infrastructure. However, due to the distributed, disperse, and ubiquitous nature of cyber physical systems, "security by obscurity" or in other word isolation of properties in designing the system, may not be the optimum way of protecting it from threats anymore [96]. Therefore, to exploit the advantages of smart systems and CPSs more efficient approaches must be sought. Physical properties can also be subject to vandalism (in deliberate cases) or environmental undesirable changes (in undeliberate cases) as examples of external faults. Internal faults on the other hand may include breakdown of constituent components, human errors, equipment malfunction, etc. which can vary vastly based on a system's functionality, constituents, or scale. Failures may occur in electronics and electrical facilities such as in electronic circuits, power system (e.g. loss of power supply, or changes in voltage and/or in electric current), mechanical failures due to inadequate or no maintenance or harsh working environment such as high pressure or temperature, etc.

Communication layer

Next area of vulnerability in cyber physical systems is the connection or integration layer. This layer's issues are more distinctively concerns of CPSs, and widely include dependability and security attributes of the flow of information between physical entities and cyber space (applications, database, decision making unit, etc.).

Maintaining dependability and security in this layer can become even more critical when the heterogeneity of the components and service providers rises. A reason for this can be due to for instance different standards used by different manufacturers for integrated components of a CPS. That means every component can have its own concerns with regards to dependability and security when deployed and integrated in a cyber physical system. In addition, given that mostly involved interacting heterogeneous components' behaviours are like black boxes, there is a possibility of encountering unexpected behaviours when they are summoned. Taking smart cars as an example, the largest number of issues and bugs that further exploited to make attacks possible, could be traced back to the communication layer between smart entities that had been produced by dissimilar manufacturing companies and vendors.

Integration layer generally uses open standard protocols for communication that despite being implemented and widely used for years, there are still security issues and concerns in using them.

Denial of Service (DoS), or Distributed DoS (DDoS) attacks are of most common types of security attacks in CPSs. In these cases, the attacker by consuming the bandwidth, blocks the data from being delivered to righteous destination.

Eavesdropping compromises confidentiality by getting into the system (e.g. as a result of IP spoofing). One of the well-known types of attacks that may compromise confidentiality of the data and may involve eavesdropping is Man-in-the-Middle attack or for short (MITM) in which the attacker secretly compromises the communication between two entities that perceive that are communicating directly with one another.

Masquerading is to deceive an agent to get data from it. Cloning happens when a malicious entity replicates a righteous one and gets access to the system or even takes the place of the right entity. Then a replaced malicious entity can deny interactions or service after negotiations with other agents and causes conflicts in resource allocations etc. This type of risk is called repudiation.

Or as in the risk of manipulation, agents and data may be misleadingly manipulated to send ill data on behalf of for instance, an agent or a sensor. Social Engineering and Phishing is another common risk that is on the top of the security and vulnerability list of top 10 threats and

countermeasures of Industrial Control Systems (ICSs) according to Federal Office for Information Security recommendation: IT in production. Social Engineering involves disclosure of confidential information to an unauthorised entity. An example of such attacks is phishing mails that contain malwares like trojans etc. in their content that will be installed once opened or direct the target to malicious web addresses.

Another issue that can undermines the dependability of Cyber Physical Systems would touch the essentiality of real-timing in some systems. Delay in communication and transmission of important data or loss of data packages may introduce severe undesirable effects at considerable expenses to the system. Exposure of the topology of the system also may pose threats to its dependability and security since more important nodes can be founds and be targeted by adversaries. One way of learning about an important node (e.g. sink node) is the connections and data traffic on this node, and a way that to some extent may help to prevent the conspicuousness of such node is to use aggregate nodes or intermediate nodes to connect to less important nodes, filter the data received from them, and communicate it to the terminal node [79,97].

Cyber layer

As far as the cyber space is concerned various vulnerabilities with regards to information and application are generally involved. Cyber entities in cyber physical systems may include digital twin and cyber representatives of physical entities, applications for analysing large amounts of data acquired from various sources and providing control and decision-making support, data storages, resources for computational supports, etc.

Similar to the integration layer, due to multi-vendor characteristic of cyber physical systems, each cyber entity may have its own short comings in terms of security and dependability. Improper integration for presence of vast heterogeneity is one of the vulnerabilities. Applications' bugs may be another factor to pose risk to dependability of the system or to be exploited by adversaries and bring security challenges and cause significant costs.

Furthermore, various security risks that were discussed in previous layer (communication and integration) can also compromise the security of information and data flow in this layer. Unauthorized access to information and data bases, DoS attacks through for instance keeping control and computational resources engaged and overloaded by making repetitive requests so that the provision of appropriate service to the right entities are prohibited or delayed, viruses and malwares to control and compromise devices, are some major examples of security risks.

2.6.3. State of the Art approaches to model and tackle Dependability and security Issues of cyber physical production systems

So far, the structure of cyber physical and cyber physical production systems has been shown and dependability and security issues associated with these systems were discussed. Given the vast potential advantages of using CPSs and the move into the fourth industrial revolution which as was shown is based on CPSs, numerous studies have been dedicated to propose solutions for coping with issues and assuring dependability and security of CPSs and CPSs in manufacturing systems and also international standards and recommendations are being released every year by various international organizations and unions to provide a thorough insight into what issues are and what might help in identifying, and dealing with them [89,98-111].

There are also methods for modelling and analysis of dependability and security in a system. The following are some of the main techniques suggested by IEC 60300-3-1 international standard [12]:

Fault Tree Analysis (FTA)

FTA is a top-down approach that analyses faults and undesirable states in a given system through the propagation of a set of lower-level errors. It helps finding logical connections between events that can lead to an unwanted condition or failure. The tree is normally developed using a set of symbols to indicate the types of *events* and *gates* (relationships).

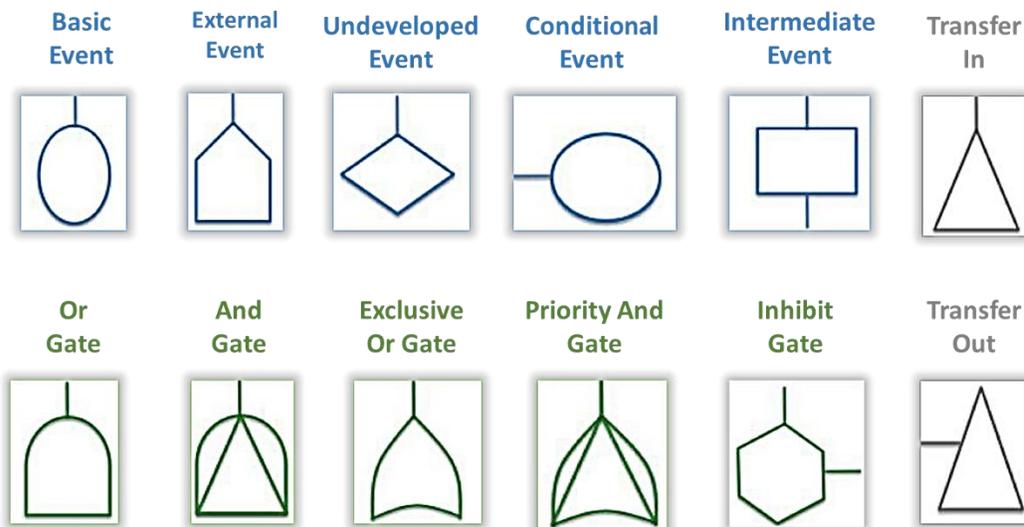


Figure 18. Elements of Fault Tree Analysis (FTA)

Basic events are generally errors that occur in an entity within a given system. External events are those type of phenomena which are expected to happen. Undeveloped ones are not considered important or there is not sufficient data about them. Conditioning events are restrictions influencing the logic gates. Intermediate events are for inclusion of extra information for events.

For gates, and/or gates respectively means the occurrence of an event depends on the occurrence of all inputs, or one or more (any) of them. Exclusive Or on the other hand suggests that an event happens only if one, and only of the inputs happen. Priority-And Gate comes when the order of inputs matters in order for an output to occur. Inhibit Gate ties the occurrence of an output to the occurrence of an input with a conditional input event.

Transfer “In” and “Out” are used to connect events of two related tree.

Event Tree Analysis (ETA)

ETA is a method that investigates the consequences stemmed from a starting event, and assesses the probabilities corresponding to each possible resulting event that may occur subsequently [107]. This bottom-up inductive technique also identifies the degree of successfulness of mitigations and evaluates various scenarios with different probabilities of happening. Some of the advantages of this method are the ease of use, functionality at various levels of details, computerizability and the existence of commercial supporting software applications, simultaneous consideration of various failures and success/unsuccess scenarios, and simple mathematical approach, which is a set of multiplications in estimation of probabilities in every path:

Probability of a path = $\prod_{i=1}^n Event_i$, Where $event_i$ denotes the events in that path.

This technique can go hand in hand with Fault Tree Analysis as a combined method to increase performance. This joint method that is named by [112] as *ET with boundary conditions*, can have each Event Tree branch represented by a Fault Tree.

Markov Analysis

Markov technique suits well with dependability evaluation of complex systems. It considers the current state of the given system irrespective of the previous events that occurred, and from that derives a model of the next state of the failure or repair and maintenance strategies dealing with

each component. This property of Markov model which is also known as Markov Property, can be mathematically shown as:

$$P\{S(t_n) = s_n | S(t_{n-1}) = s_{n-1}, S(t_{n-2}) = s_{n-2}, \dots, S(t_1) = s_1\} \\ = P\{S(t_n) = s_n | S(t_{n-1}) = s_{n-1}\}$$

Which indicates that the state of the system at time t_n only depends on the state of the state of the system at $t = t_{n-1}$, and not the previous states. Markov Chain is the simplest form of Markov Model that uses Markov Property that suggests in a system with a random variable, finding this variables distribution only requires knowing the distribution of the previous state.

Failure Mode and Effect Analysis (FMEA)

Developed at late 1950s by reliability engineers of NASA for aerospace design in military context [113], FMEA is one of the most utilized techniques to investigate possible failure modes that can arise in a system, and their effects therein.

This bottom-up method studies various failure mode by taking into account parameters namely: probability, severity, detectability, and associates a risk level to each failure accordingly as well as the Risk Priority Number or for short (RPN) which is generally the multiplication of the quantified values of the abovementioned three parameters.

$$RPN = Probability * Severity * Detectability$$

A sample of a risk level matrix is shown below:

		Severity					
Low	Moderate	1	2	3	4	5	6
High	Unacceptable						
Probability	1						
	2						
	3						
	4						
	5						

Figure 19. An example of a risk level matrix

The probability ranges from extremely unlikely when the chance is almost next to zero for that failure to happen, to frequent where the failure is virtually inevitable. Similarly, severity falls into the range from no relevant influence on the system, to almost catastrophic with regards to safety, the output of the system, etc.). Detectability as the third parameter goes from certainly detectable to undetectable (failure might be latent for a period of time that can be included in this analysis).

In this technique, failures can be classified in various forms based on the type of object under consideration (a sample of a failure taxonomy can be seen in figure bellow), and a worksheet is also being developed in which there are information about lists of failure modes, their potential causes, the components to which the failures are related, the possible effects of the failures, ways of detection, risk level, actions and recommendation in mitigating or removing the failure, etc. using which failures are documented.

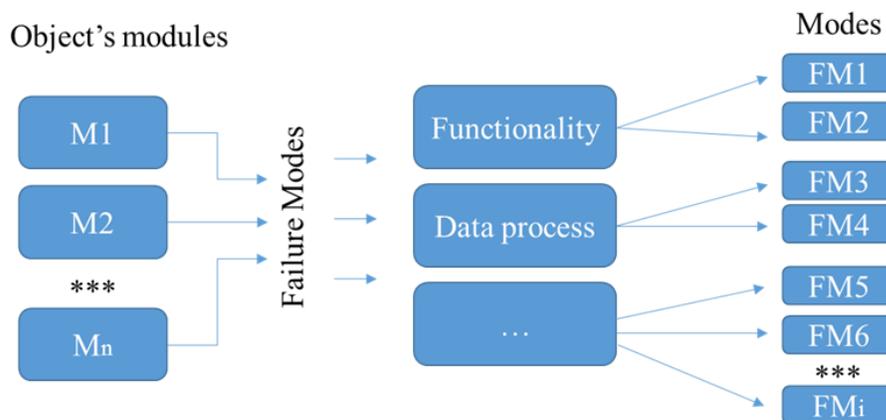


Figure 20. Sample of failure taxonomy

Failure Mode, Effect, and Criticality Analysis (FMECA)

FMECA is an extension of FMEA technique which includes criticality analysis as well. In this technique, the probability and the severity of the consequences of a failure are also brought into the spotlight in order to help focusing mitigating actions on where they have highest values and outcomes.

Functional Failure Analysis (FFA)

Used in early design stages of development, this technique takes into account the constituent elements and functionalities of the component under consideration to recognize and analyse the possible hazards and failures. In other words, FFA intends to specify those system activities and

functions that may lead to failures or threats. It accordingly categorizes those functions based on their degree of seriousness and severity. Using this technique, failures can fall into three categories which are as follow:

- 1) the functionality is not delivered when asked for
- 2) the functionality is delivered when it is not asked for; and
- 3) faulty functioning.

Similar to FMEA technique, this method too gives tables as its output with required information about various actions and functions, associated threats and errors and their description, the operations involved, the effects that each failure might have in addition to methods of dealing with them. It also can provide the methods of analysis approved by policies with regards to the system.

Petri Net analysis

Introduced in 1962 by Carl Adam Petri in his dissertation, Petri Nets are mathematical modelling techniques utilized for describing distributed systems. It is a bipartite graph including sets of events and states, represented respectively as bars and circles. Bars are called *transitions*, and circles are called *places* (sometimes Petri Nets are called *Place-Transition Nets*). There are also *Arcs* which connect places and bars. No two places or two bars can directly connect, and place before and after each transition are respectively called *Input* and *Output* places.

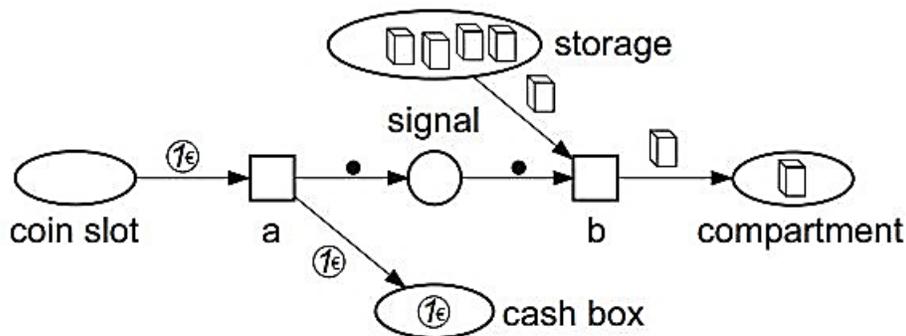


Figure 21. Example of Petri Net [114]

Places can contain a number of *tokens* (called *marks* and are generally represented as black dots) and distribution of tokens in places is called *marking*. After each transition from a state (place) to another, a number of tokens can move from input place to the output place (called

firing, causes a change in markings and means the state of the system has changed) if the transition is possible. Given that firing in Petri Net is nondeterministic and tokens may happen to be anywhere in the net, this tool can be seen as a powerful tool for modelling concurrent behaviours.

Often one or more of these modelling techniques, along with other approaches for prevention, detection, mitigation, removal, etc. of risks and failures have been adopted in dealing with dependability and security issues of cyber physical systems. The following are some risk detection approaches with regards to cyber physical systems.

Anomaly Detection

Anomalies in data-mining are deviations in data patterns from what is considered normal, and anomaly detection is the process of finding these unexpected patterns using various methods and approaches [115, 116]. In anomaly detection, the first step then would be to know what is expected and approved in a given system and then to monitor its behaviour to find any deviation from this approved normal state or data pattern. Below are some examples of anomalies found within two different sets of data, two of which are one dimensional and the other one is a two-dimensional data set.

In the first one, anomalies are shown for a two-dimensional data set where variables on horizontal and vertical axes are related and their values against one another are expected to fall into expected regions (shown whit circles) to be considered as normal. There are also observed samples and patterns that are far enough from the approved regions to be called anomalies. The second one (figure (23), from [117]) shows anomalies in data volume in the form of unreasonable escalation of bandwidth use which in this case denotes the occurrence of DDoS attack to the system. In the last example anomalies are distinguished as data samples that fall out of control limits which can be defined statistically to assure that the observed values can demonstrate correctly the behaviour of the system under control.

There are several factors that are important in anomaly detection. One of them is the type of input data or in other term, data instances that are being monitored. Instances can be patterns, events, vector etc. and can have attributes of various types such as binary, or continuous. There can also be interrelated instances of data with multiple attributes or single attributes.

The other important factor is the type of anomaly. [116] gives three different classes to anomalies to distinguish their nature:

3. Point anomalies

When a single data instance is to be seen as anomaly when compared to the other part of data, it can be called a point anomaly, which is the most common type used in analysing components behaviour.

4. Contextual anomalies

These types of anomalies refer to instances that are regarded as anomalous in some contexts and not others. To specify these anomalies there are two attributes to consider in defining the problem.

- Contextual attributes, which is to be defined to plot the context; for instance, values to define a location.
- Behavioural attributes, that is not the context but happens within the context; for instance, the average temperature of each location (context).

Here, a temperature can be acceptable for one context and not the other.

5. Collective anomalies

As can be implied from the term, this type of anomaly refers to occasions where a collection of values is seen as anomaly among all data acquired. In this case a single data instance may not be regarded as an anomaly.

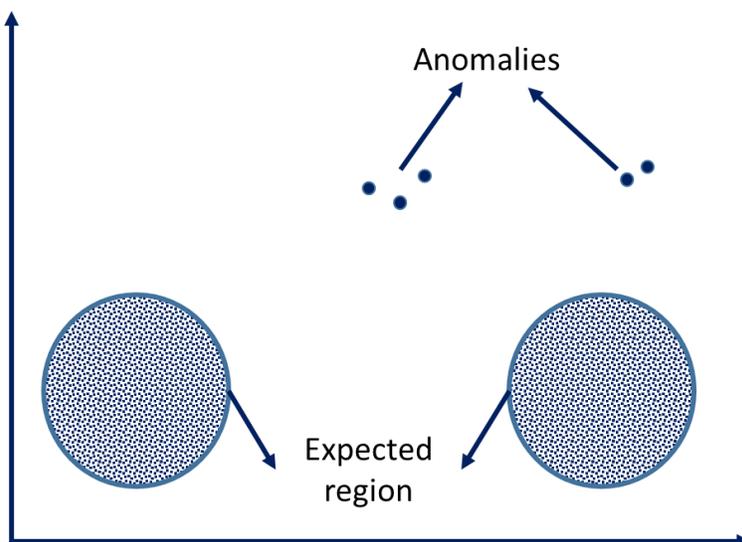


Figure 22. Two-dimensional data sets and anomalies

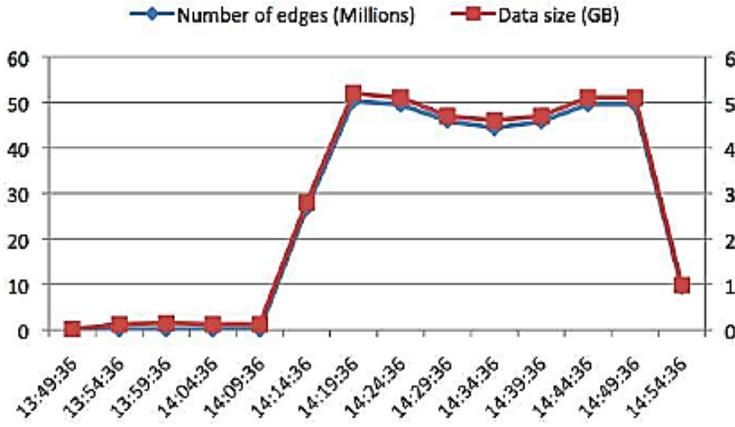


Figure 23. DoS Attack anomaly [117]

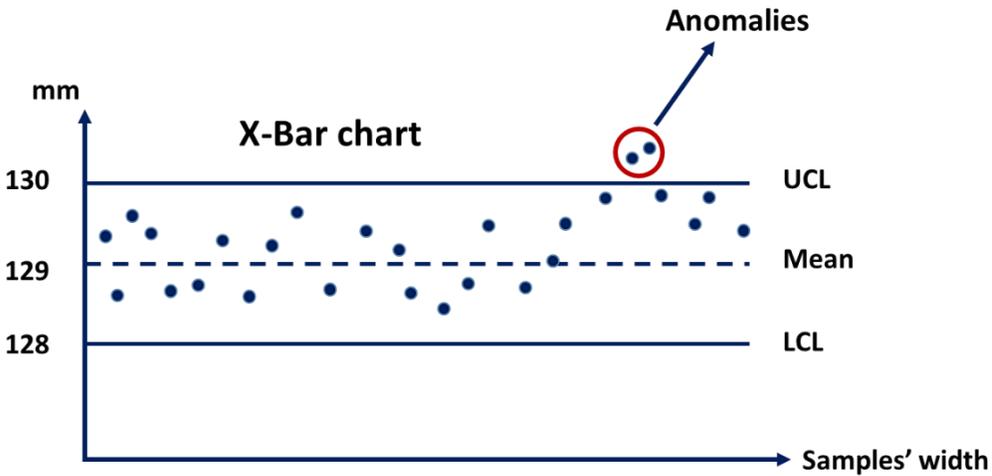


Figure 24. SQC example of anomaly

Various reasons and causes can lead to such deviation which might be a malicious attack or of an accidental cause, which can be investigated and categorized using the approaches shown in section (2.6.1). The steps in this approach can be shown as follow:

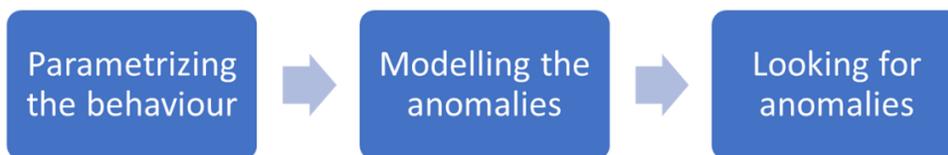


Figure 25. Steps in anomaly detection

Parametrizing the behaviour: there is a need for understanding the behaviour of the system that is being monitored and parametrizing the values that are to be considered.

Modelling the anomalies: this step is to build model of approved and unapproved states of the system, expected data patterns, and to define the thresholds.

The third step, *looking for anomalies*, is to compare the real-time state of the system with the approved state and detects if an anomaly occurred in observed data patterns or if a threshold exceeded in the defined values.

Anomaly detection can target a broad range of issues with respect to both dependability and security. Some examples are intrusion detection, fault or damage detection, or other non-industrial cases might be anomalies in medical information received from a patient, or fraud detection.

As was said above, Intrusion detection is one of the applications of anomaly detection techniques and is referred to as spotting and finding out adversaries' activities in breaking into a computer system or network, etc. [118]. What makes anomaly detection techniques particularly useful for detection of intrusions into the system is that the systems under intrusion attacks behave differently when compared to the same systems in their regular conditions, and anomaly detection is all about finding deviations in systems behaviours and data patterns from what is considered a normal and approved status. However, there are some main challenges for this technique to overcome, the most important of which is an enormous amount of data being poured in for analysis. With this matter in consideration, sufficient computational resources can be imperative to successful analyses and reliable outcomes. Another issue to overcome is online real-time capability since data is usually being streamed.

Next challenge for anomaly detection in intrusion detection techniques is to deal with the rate of false alarm. This is due to the immense volume of input data which can make even small percentages of false alarm an intimidation deal for the analyser agent. Various data labelling can be used depending on whether the anomaly detection is going to be unsupervised, semi-supervised or supervised. But given that in many cases "normally behaving" data's label is available, ill data and intrusions stand out as unlabelled and that thus makes semi-supervise and unsupervised methods as more preferable when it comes to intrusion detection.

Another area where anomaly detection comes in hand, and is also the focus of this study, is dealing with failures associated with industrial units and systems, pieces of equipment, machinery, networks, etc. Failures in manufacturing units can vary. For example, exploitation of

industrial equipment for longer time than they can afford without running maintenance can cause machines to deviate from delivering expected specification. This deviation in behaviour, if not detected and treated, may deteriorate into further damages which may be tremendous in cost or possible hazards that may arise. Data here is usually acquired through sensors that are placed throughout the system to provide a model of how components are behaving. These data will then be analysed to detect failures and damages. Anomaly detection techniques are indeed very common in finding failures and errors in industrial systems and to identify how healthy system is performing.

In case of fault detection, mechanical units should be constantly monitored for their performance. Instances of these units can be motors, pipelines, valves, ventilation systems wind turbines etc. which may demonstrate defect functioning due to partial breakdowns or wears, or a variety of unexpected phenomena. On the other hand, defects (as anomalies) might occur in the structure an equipment, i.e. rusts or cracks may appear on a surface or distortion may occur to metallic components due to heat or pressure.

Wireless Sensor Networks (WSN) is another area of technology which has found its way into many aspects of human life, especially in industries. Accordingly, duo to its characteristics and large data flow of a variety of formats and type, they have gone under the spotlight of researchers and data analysts. As far as sensor networks are concerned, anomalies found in sensor data can have several connotations. They are either demonstrative of adversary activities or failures (or such phenomena) in the network, or are caused by sensors malfunction itself, or in some cases both of the above.

Like before, various challenges exist when it comes to anomaly detection approaches in wireless sensor networks that should be taken into account in analysing streamed data that are being received from sensors. One of these challenges is noises in data that can be associated with the environment in which the network is working, or limitations or bugs in communication infrastructure, etc. Similar to noises in data, for the same reasons data or package loss may occur while network is operating. Noises and data losses also create a challenge for anomaly detection mechanism to understand which anomaly is actually stemmed from a failure and is to be taken seriously, and what anomaly is indeed come only due to some noises for instance.

Furthermore, given the computational resource limitations the detection approach should be designed and programmed in way that requires least resources, since they should perform online

similar to sensor data that are being streamed. Another matter to consider is that often sensor data are acquired from distributed sources. This fact implies that analysers should be capable of dealing with this issue for instance by summoning data mining techniques suited for such conditions [119].

Sequential/Change Detection

In this technique of detection input and output sequences are the keys to determine whether the system is compromised. This is being done by making a comparison between output sequence observed with the anticipated output corresponding to a certain input sequence. In order to implement this method accordingly there are two steps required [120]:

- A model describing system's functionality and characteristics
- A proper algorithm to be used for detection

Sequential detection and change detection rely on two slightly different approaches. And are based on different hypotheses [121]. In change detection technique, as inferred from its name the presumption is that the starting point is what is approved and changes are to be spotted and caught in any time. In the sequential detection on the other hand a given sequence of time can be observed as either normal or as at risk. Under the assumption that false alarms' probability is constant and not changing, Sequential Probability Ratio Test (SPRT) technique can be used to solve the problem.

There are also some other approaches that might be of interest of organizations to be used for modelling dependabilities and security of the given system among which Probabilistic dependence graph for understanding faults and their location [122], HAZard and OPerability studies (HAZOP) [101] for evaluating safety threats and hazards for employees based on pieces of equipment and functionalities or operations, or Reliability Block Diagram (RBD) [123], which studies the components and the connections between them.

2.7. Implementation approaches for dependability and security

Various implementation approaches have been adopted to guarantee dependability and security of cyber physical systems to different degrees and in different context. In this section some of the major approaches are presented and discussed and the shortcomings of these

current approaches that are going to be addressed by the mechanism presented in this thesis will be pointed out. In additions potential underlying opportunities and capabilities that the proposed approach may afford are going to be presented.

Methods of implementation usually adopt one or more of the approaches of identification and detection in combination with modelling and categorization techniques presented above in this chapter.

In [89] a sensor actuator network was used along with an information system by which predefined security-related context data and sensor data are kept. Accordingly, based on predefined sets of anticipated reactions actuators will get involved in reacting to a security threat. To build the context information Wan and Alagar suggest five main questions which require answer to almost sufficiently grasp the right context. However, based on the application more questions can be asked to include more dimensions of the context. Question number one is that WHO is accessing the information, then the second question is that WHAT is being accessed or is asked for by an entity or a client. Next question is from WHERE the request or attempt is made for accessing information, and WHEN it is taking place. And the last question is WHY is the information being accessed.

Each question has a type of value as its answer and therefore, context can be represented as ordered pairs of (D_i, V_i) , where $D_i \in$ dimensions, and $V_i \in$ values. For instance, context c can be described as:

$$c = [\text{Where: city name, Who: ID, When: date/time}],$$

which can describe a client and the place and time of its access. Such context labelling can detect and restricts unauthorized accesses to information and resources.

To construct a dependability model at run-time, authors in [105] develop an agent-based approach along with wireless sensor networks. In doing so, they combine and use failure mode and effect analysis (FMEA) for analysing dependability and agent ontology to analyse dependability of the system in real-time. In addition, environmental monitoring is to be performed using wireless sensor network over UDP and agent system. Accordingly, five sets of agents were developed to form the dependability model in real-time:

- Agents to collect and store sensed data

- Agents to negotiate if needed to eliminate conflicts between received data
- Agents to process cumulated data for data validation and processing
- Agents to diagnose the systems ontologically with regards to failures, and
- Agents to manage ontology

Another agent-based approach which came earlier in 2010 [124] adopts EPS/EAS, that are the shortened versions of Evolvable Production System and Evolvable Assembly System concepts; the former expands on the latter to encompass a broader framework within the system. The approach tries to enable adaptability of the components to the changing environment as well as its evolution in making the system more resilient. That of course require the system to be evolvable which means it should be capable of addressing the need for changes in requirements should the situation requires.

The use of agent-based architecture, which is stemmed from the domain of Distributed Artificial Intelligence (DAI), due to their features and capabilities that are going to be more thoroughly discussed in following chapters (as the approach of this thesis is also based on multi-agent systems), have also been seen in developing decentralized control system which also show certain degrees of improvement in robustness and dependability. For instance, in [98] agent-based approach for developing a distributed sensor-controller coordination was suggested in conjunction with Adaptive Distributed Optimization algorithm (ADOM) to provide more stability and robustness in the system (with more focus on micro grid systems) to achieve the following features:

- completeness, which points out the ability to reach globally optimum solution,
- low overhead, to arrive to the solution by locally available data,
- and the capability of finding solutions with limited delay.

Also, in industrial systems various methods have been adopted in implementing agent-based distributed control architecture. One of the earliest attempts in this area using agents, indeed the earliest attempt, to transform hierarchy to heterarchy was presented by Duffie andPiper in 1986 in [125] by suggesting agents for resources, work pieces, humans, and control units to carry out scheduling. Another early approach to be named is Holonic Manufacturing (HM) which using agents, provides a network of widely distributed and highly heterogeneous collaborative and intelligent autonomous components, that are capable of sharing ontology knowledge in a service

oriented architecture supporting big data analysis, altogether enabling the integration of enterprise hierarchy (vertical integration from shop floor to managerial systems), Value chain (vertical integration throughout the supply chain), the life cycle from ideas and design to manufacturing, delivering, using and remanufacturing of products, and the integration of virtual and physical world [126].

This idea broadens the environment to a collaborative condition in which various factories' facilities as represented by agents (holons) contribute in negotiate upon scheduling and delivering customer orders as a multi-factory system [127], which gives rise to more flexibility. Furthermore, due to high scalability, multi-agent systems provide easy integration, auto-configuration, and detachment of components (controlling and controllable entities) without the need of any major changes in the system. This ability also enables ubiquitous monitoring, control and decision-making ability. Another feature of holonic manufacturing systems (HMS) and MAS is the ability of reducing complexity through the support of self-similarity that is demonstrated both horizontally and vertically by the well-known architecture of the holonic manufacturing, "Product-Resource-Order-Staff Architecture (PROSA) [128]. The holonic architecture then further improved by the introduction of ADACOR [129] that was designed to responds to agile and adaptive manufacturing needs, that later revised and evolved through ADACOR2 [130], inspired by biological and evolutionary theories to provide higher self-organizability using MAS. The evolution paradigm of HMS architecture is providing more linkages with the concept of CPS and intelligent products [131].

Another agent-based approach which was presented in 2001 by Sauter and Massotte [132] based on a European research project is called PABADIS (Plant Automation Based on Distributed Systems). This system which adopts CMU approach (which stands for Cooperative Manufacturing Units) attempts to expand the presence of distributed intelligence at all levels of controlling process, and to break down the deeply centralized and offline hierarchical structure of traditional automation and control systems. It was done through using mobile software agents within agent platforms connected with control devices.

The focus of PABADIS unlike holonic manufacturing systems (which mostly target control level) is on MES level (Manufacturing Execution System) in the way that it breaks it into two parts; a centralized part integrated with the ERP level (enterprise resource planning), and a decentralized part comprised of mobile agents.

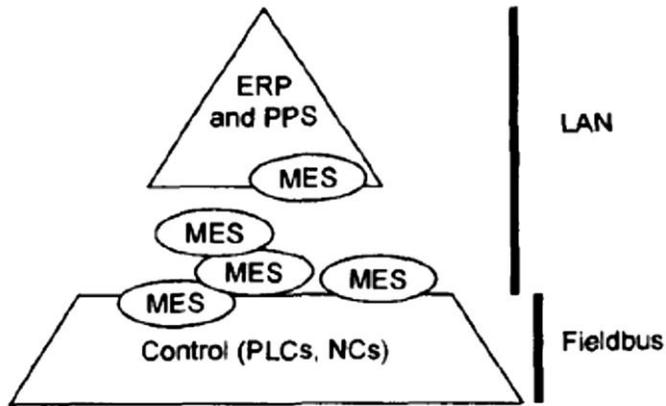


Figure 26. Breakdown of automation hierarchy at MES level [132]

Following the PABADIS approach, later another approach called PABADIS'PROMISE (PABADIS based Product Oriented Manufacturing Systems for Re-Configurable Enterprises) stemmed from and is added to it which uses mobile agents as well as Radio Frequency Identification (RFID) technologies at the field level [133].

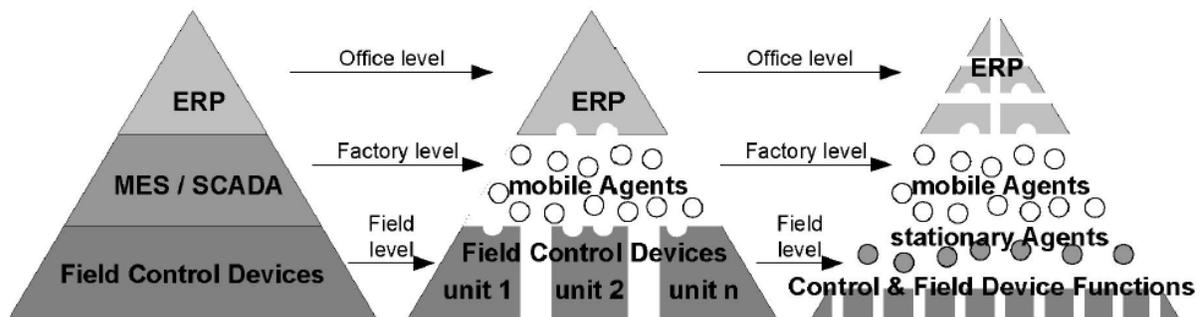


Figure 27. From traditional automation centralized and hierarchical structure towards PABADIS and then to PABADIS'PROMISE

The aim is to take the aspects of manufacturing processes to their maximum flexibility and application versatility and efficiency, and broaden the boundaries of the enterprise value chain to encompass other collaborating companies as well. Furthermore, all these goals put the focus on order-based manufacturing to the extent that PABADIS'PROMISE forms and calls its vision as "The Order is the Application".

As can be seen, various approaches have targeted the enhancement of production systems with regards to their robustness towards dynamic behaviours and changes in the environment, as well as dependability issues like possible failures, and security threats and risks. However,

there is still a lack of a coherent, flexible, and smart mechanism for assuring dependability and security issues which is scalable, autonomous, and interoperable throughout the process of inspection and reaction. The following is the list of objectives and features that mechanism proposed in this thesis aims to accomplish:

- A) using agent-based design the mechanism will be broken down into independent yet collaborating modules that are scalable. Meaning that the mechanism can be expanded or shrunk at will with no considerable effort. Furthermore, as an emergent property, it should demonstrate the potential to dwell various approaches for various issues that may arise in a component.
- B) Each module performs as a black box the behaviour of which is to be chosen based on the preferences and policies of the organization and developer. That means the mechanism should be flexibly used in heterogeneous systems and inherit heterogeneity from monitoring process by monitoring entities, to identification approaches, to measurement, and finally to decision making algorithms and reactions.
- C) The mechanism should be potentially capable of integrating layers of automation pyramid through learning capabilities and reconfigurability.
- D) The mechanism should demonstrate inherit features of industry 4.0

In this chapter, the state of the art for understanding the smart and cyber physical systems and their behaviours and properties, as well as their capabilities and possible applications have been broadly presented. More thoroughly, the contribution of the advancement of ICTs and smart cyber physical systems in manufacturing area was discussed. Furthermore, dependability and security features and aspects in general and in particular for these systems have been classified, the possible issues and risks in every layer of CPSs were shown, and the ways of dealing with them or modelling them have been introduced.

The next chapter provides the concept and structure of the intelligent and distributed dependability and security mechanism proposed and presented by this thesis and discusses its behaviours and properties.

CHAPTER THREE

THE DEPENDABILITY AND SECURITY MODEL

To assure maximum dependability throughout an enterprise, the adopted approach must cover all incorporated components, all information flows among them as well as the addressed cyber areas, networks, databases and servers. In the previous chapter it was shown that some methods have been suggested for classifying dependability and security issues and threats, approaches were suggested for modelling dependability in cyber physical systems, and ways of dealing with various types of dependability and security risks were taken into account either application specific or to some extent general.

Here our purpose is to introduce a mechanism which intends to tackle dependability and security issues by harnessing the properties of cyber physical and smart systems. A mechanism that can encompass the previously discussed techniques and approaches within its modules and adopt them based on the application and system for which it is summoned. The suggested approach is to provide bed and a platform for collaboration of various techniques in an intelligent and automated fashion in enhancing the dependability and security of cyber physical systems, specifically in manufacturing context.

Accordingly, this chapter will first introduce the proposed dependability and security model and its constituents. Subsequently, the contribution of the properties (as discussed in the previous chapter) of smart and cyber physical systems will be discussed. Afterwards, Multi-Agent Systems (MAS) as the proposed toolset for enabling this mechanism is to be introduced. The chapter will then end with proposing the agent based dependability and security mechanism and its modules along with the approaches used for this thesis's use.

3.1. The Dependability and security model

To the aim of this thesis, a distributed Dependability and Security Model is proposed that can be an instance of the encapsulated decision cycle (elaborated in [134] shown in the figure (28)) for covering the entire network, every unit and component down to all levels of detail (LoD). As can be seen from the figure, the first step is network strategy and related objectives within all levels. These levels go from network design and operations. Then next is monitoring of these processes and the information flow within the network. Behaviours are being analysed and

undesirable activities that are found will be treated and dealt with as the loop reaches improvement step. If improvement (level one reaction) did not satisfactorily solve the issue, the next step which is adaptation will be deployed where restructuring (level two reaction) takes place. Through each step decision would vary from an action among categorized actions or no action.

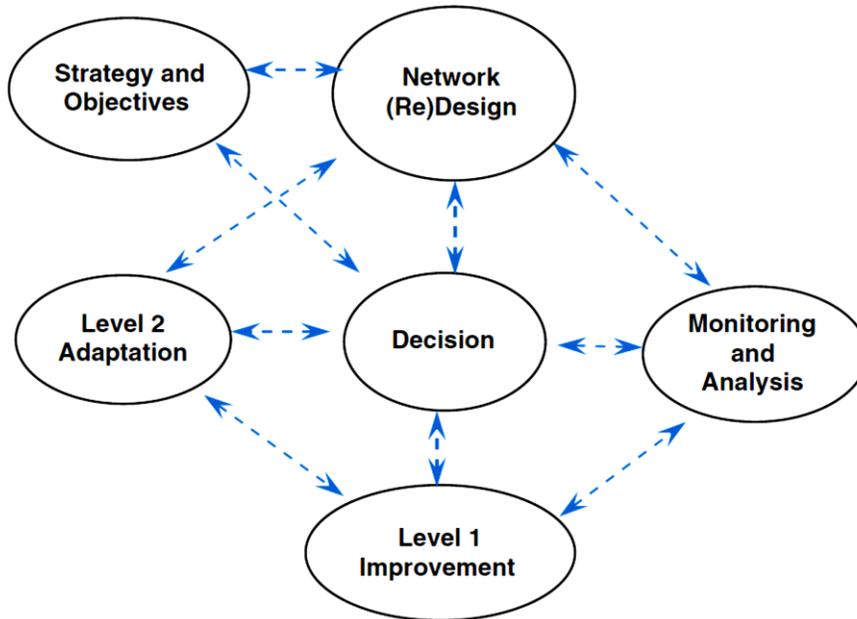


Figure 28. The decision cycle as shown in

Derived from the decision cycle shown and explained above, the dependability and security model (figure (30)), is developed to perform the intended task of supervision and control of smart systems. It contains a **core**, a **control loop** [135], and is communicating with the cyber space. In fact, the model is placed in the virtual space as a part of components intelligence. For this reason, it should be equipped with proper information about the component, the context in which it is performing, as well as its dependability and security objectives. The model's core consists of two main parts [136]:

- Object description, and
- Risk model

Object description, is to give the component information about itself and the environment. In other word, it gives the component adequate self-awareness and context-awareness. The former

is about what elements and modules constitute the component's structure, what are its tasks and expected behaviours going to be, what are the desirable specifications of the services they are ought to deliver, how the given task is going to be performed stepwise, how much data should be generated or sent, acceptable internal heat, vibration, etc. On the other hand, the latter takes into account component's external-awareness or context-awareness which can refer to the environment's characteristics i.e. noise, heat, moisture, etc. or the collaborations and type of interactions that the component has with other components of any type or constituent, the anticipated and approved data flow, the operations that it is taking part fulfilling, etc.

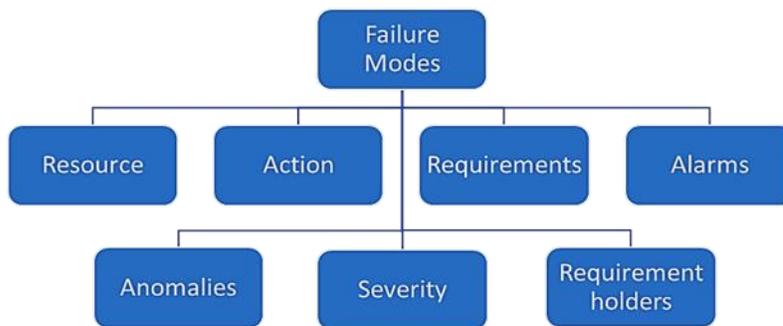
Risk model as another main inclusion within the model's core deals with required data with regards to failures and errors i.e. how to detect them should anything occur, how to identify and measure their possible effects on the system and their criticality and severity, what are the countermeasures and how to react to and mitigate them, and bring system back to its normal behaviour. The risk model is to be chosen by the developer based on the component type and constituents, its structure and type of tasks it is performing, as well as the dependability and security objectives and standards it needs to hold up to with regards to the organization's policies. Some of the main risk and dependability modelling techniques and approaches, along with detection methods and mechanisms were presented and explained in the previous chapter (section 2.6.3) which can provide the appropriate intended input for the core model. This study is going to use Failure Mode and Effect Analysis (FMEA) technique and modify its output table based on the use cases (which are going to be presented in the next chapter) to provide necessary risk information, and anomaly detection is going to be the chosen method for monitoring and detecting the errors and failures.

The model's core is to support the improvement process which is going to be carried out through the control loop. The required data to build and develop the core of the mechanism are to be imported from the cloud/ cyber space or sensed through implemented sets of sensors, together forming the component's self-awareness and context-awareness. The data can also be edited, added or removed manually in real-time through established interfaces and be utilized by the component as shall be described thoroughly later in development and implementation section. Furthermore, in this mechanism, self-optimization can occur via sharing knowledge with all other smart units, and updating one's own structure and database through continuous feedbacks as it provides a loop to update on its performance and knowledge.

Table 1. Information to form the core of the model

Objects Description	Risk Model
<ul style="list-style-type: none"> ▪ <i>Objectives</i> ▪ <i>Task description</i> ▪ <i>Structure and modules</i> ▪ <i>Environment and position</i> ▪ <i>Collaborations</i> ▪ ... 	<ul style="list-style-type: none"> ▪ <i>What are the Dependability and security objectives?</i> ▪ <i>What are the vulnerabilities and their probabilities?</i> ▪ <i>How can they be detected?</i> ▪ <i>What are the effects and their severity?</i> ▪ <i>Who must be alarmed at occurrence</i> ▪ <i>How can they be terminated?</i> ▪ <i>How can they be prevented in the future?</i>

For this thesis following information are to form the core database from which the control loop will be fed. These components are also to be scaled and updated. Later in the next chapter the thesis with elaborate more on each of these components.

**Figure 29. Components to form the database in the core of the model**

The Control Loop of the mechanism in direct communication and to be fed by the core is to carry out the task of supervision and control. This process is being done constantly and repeatedly and consists of two main parts:

- Inspection process (i.e. monitoring, detecting anomalies, and identifying the failure mode, and measuring the risk), and
- Reaction process (i.e. providing alarms, taking action, and doing the Reconfiguration and reporting subsequently)

These steps are to be done in real-time. moreover, all of them shall be performed fully- or semi-

autonomously through smart components themselves using sensors and actuators and with constant access and communication to the model's core, which is dwelled in the virtual world and can be in communication in other components' models. In this way components will be capable of sharing and caring for dependability and security objectives of the whole system beside objectives of their own. Figure (30) shows the dependability and security mechanism's components. The diagram may invoke the idea of PDCA (Plan-Do-Check-Act) concept which targets continuous improvement of the production process starting with planning a change, to implementing it, checking the results and acting accordingly. Whereas the mechanism here constantly looks for failures and anomalies to identify and act upon them. Therefore, despite having similarities, the two mechanisms serve different purposes.

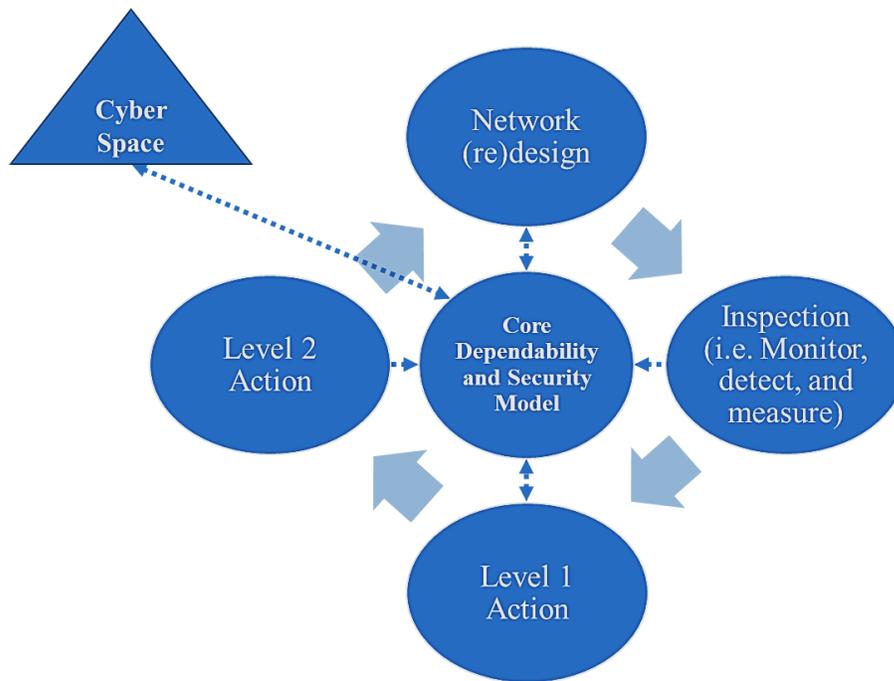


Figure 30. The Dependability and Security Model

Monitoring:

The starting point in this smart distributed dependability and control mechanism is monitoring. This functionality is to be carried out in a semi/fully autonomous manner, by exploiting the data acquisition resources such as sensors, etc. and the information available in the model's core that the smart entities are harnessing. Intelligent objects continuously and within real-time observe the present and existing status of parameters to guarantee that all parameters are conforming to and following the accepted determined values. The variables are to be specified and

parametrised based upon the context in which the object is performing and the component's description itself, and hence the components' self/ and context-awareness should be updated in real-time. Components will have the knowledge of the deliverables (i.e. tasks and services, etc.) they are ought to provide and the satisfactory outcomes with desirable specifications. One the other hand they are context-aware and know their environment, interactions and the expected behaviour of the entities they work with, the righteous and authorized entities and the type of access they are limited to have (whether its accessing data or administrative rights to change or modify anything in the system), the data flow and bandwidth that is anticipated to be consumed and a given time, the ambient parameters such as vibration, temperature, moisture, etc. and many other variables that can vastly defer based on the component's type and constituent, or the openness and exposure and possible accessibility of physical or cyber parts of the component to possible adversaries, and so on.

As mentioned before the technique that is going to be used for detecting issues and failures is anomaly detection. Monitoring phase looks for anomalies in every relative layer of cyber physical system (i.e. virtual space, real world object, and integration layer) related to the component. Taking the inherent heterogeneity of cyber physical system's components into account (being physical and/or virtual (a software application, a machine, a sensor actuator network, etc.), communication and data transition mechanism (platform and infrastructure, type data being communicated, protocols, language, etc.), type of service (physical or computational), scale and level of details, etc.), models and approaches for monitoring the values can vary remarkably. The application of finding anomalies can also be broaden to perform preventive (scheduled and done routinely) and even predictive maintenance which is more cost effective. Predictive approaches and proactive mitigation or protection can be carried out for instance through monitoring physical parameters such as vibration or surface wear etc. or slight shifts in output specification (e.g. suggesting the need for recalibration), or through using data mining techniques to find anomalies or deviations in patterns which themselves are not denoting a failure but suggest or help to foresee an imminent risk.

A simple example can be that a sink sensor node and a rolling machine (though can be on the same shop floor), a CRM datacentre in the cloud, a casting line, and an inventory of dairy products, have different parameters to supervise and monitor. Components can also interoperate in performing the act of monitoring. They can collaborate in monitoring subsystems and subcomponents, or groups of sensors and actuators that together constitute a WSAN. Also

in a group of components, an entity's monitoring agent can also find anomalies in its collaborative component, e.g. by receiving no, or a series of broken data from it. In addition, due to the modular nature of the system, resources and methods (models, sensors, etc.) used in monitoring a component or system (e.g. actuator, machine, unit, etc.) can be used in monitoring other systems of related more or less complex components, or in various setups. That gives rise to the need for the scalability of the monitoring mechanism. The dynamic nature of the distributed manufacturing systems provokes changes in structure, setups, addition or removal of resources, etc. that requires the monitoring mechanism to be accordingly scalable.

Detecting:

Using the risk model and information available, anomalies found through sensors and monitoring resources that continuously and in real-time make observation over the component's constituents behaviours, will be identified by comparing the stream of received anomalies from monitoring entities with the list of anomalies and associated failure modes. In cyber-physical systems, due to the often extreme heterogeneity of entities, risks characteristics, security and dependability issues, threats and failures can and often are accordingly significantly different. Hence, approaches for detection and identification of the issues and anomalies varies as well. These methods can include data-mining techniques to monitor object's behaviour and data pattern to for instance predict if it is going to cause a failure, can be thresholds defined as maximum/ minimum acceptable values. They can be specification limits to statistically control the calibration of a machine or equipment, or might simply be checking some Boolean variables. There can also be various statistical approaches (e.g. statistical quality control using specification limits) to assist with detecting changes and whether or not they are acceptable.

Components can be constituted from, or themselves be a constituent of other components. They also collaborate in order to deliver various services. This fact suggests components to have high interoperability to collaborate in detecting risks, should one or more arise. Moreover, detection resources and databases, can work and be used in a modular way. For instance, databases for various detection methods for various risks can be put together to be applied in new setups and situations or for difference components.

Measuring:

To be carried out autonomously by the components, the process of measuring is done using identified failure mode in detection step and information available in the database. There might

be a need for collaboration with other entities should the information for measuring a specific failure mode was not adequate. To perform the measuring process, there can be various approaches to model risks and failures within components' dependability and security model. Risk assessment and modelling techniques such as Fault Tree Analysis (FTA), FMEA, any other models (more common of which have been presented and described earlier in chapter two, section (2.6.3), etc. can be used solely or in conjunction with one another to deliver the measurement purposes. Measurement can be of the severity of the risk, possible side-effects to the system and to the other components in collaboration with the compromised or failed component. Likewise, the component which the compromised component is a constituent of should also be taken into the account, since the failure may propagate through other components.

Similar to other steps, this phase too is sensitive to the heterogeneity of components, as well as the context in which they work, the scale of the system and how elaborate it is, etc. which implies that for example the criteria based on which the measurement is taken or the parameters might change from one component to the other, or the risk model given that for various failures or components different risk models might be preferable, may be different. Accordingly, the measurement mechanism should match its decisions and parameters in real-time with the changing scale or dynamic behaviour of the system. That means the risk data base, and context information should be also updated in real-time.

Alarming:

After detection, identification and measuring of the failure that has taken place, it then comes the reaction steps. The compromised component, having measured the severity and likely side-effects of the failure (i.e. the possible damages to other entities and components e.g. those with which this component is collaborating or components of higher level), will start providing alarms for other relevant entities. Here, the relevant entities could be other components in direct communication with failed object, entities related to mitigation of the failure e.g. maintenance unit or inventory, or sets of transportation components like automatic guide vehicles etc. or any other entity that might in some ways be related to the issue.

For components to know the relevant entities the first essential requirement would be an adequate degree of context-awareness, and adequate is some cases depending on the component under consideration and its tasks and type might go well beyond its immediate

environment. For instance, in cases where fixation or replacement of resources are involved. In addition, the process can get more complicated when an elaborate network or even networks of interacting components are considered (e.g. a complicated wireless sensor actuator networks with many sink nodes). This consequently may raise the need for interoperability of heterogeneous component higher given that complex propagation of alarms among entities may demand it to keep the alarming mechanism flawless. Moreover, as a result of a flexible and dynamic nature of cyber physical systems, the alarming mechanism should allow flexibility and scalability as well as context-awareness to perform satisfactorily throughout the component's lifetime.

Taking action:

Taking action as one of the most important steps in intelligent dependability and security mechanism is about making decisions and sending commands, requests, or informative messages to the actuators and right entities accordingly. The decisions are to be made based on the stream of information received from analyser and measurement section of the mechanism and by retrieving appropriate information from the database according to the risk model and possibly suggested countermeasure appropriate to the occurred failure mode which might have already been categorized and listed. In some cases, there might be a need for negotiations with other components to find the globally optimum solution for a given failure. Actions might also involve cooperation of several components in mitigating an issue. For instance, in order to provide higher robustness in a distributed control system, multiple controllers can cooperate in making decisions in predicting the next state of the system based on current or previous states should noise or package loss occurred since one controller's prediction might not be reliable in some cases [98,137,138].

The act of decision making and taking action shall be carried out in a semi/fully autonomous manner, which suggest that the component should have sufficient degree of self/ and context-awareness, knowledge about risks and failures and a model to how to deal with them, as well as information about entities that are relevant in mitigating a problem. In addition, interoperability among all these entities play a key role in more effective decisions, solutions, and reactions. This step also requires the components to be aware of and flexible to changes that happen within the system. These changes might be the size of the system e.g. new components that are added to

or removed from the system, applications and tasks that are being performed, change in eligible entities to have access to any data, etc.

There might also be a need for the capability of negotiation to a sufficient extent e.g. for resource substitution, which is going to be discussed further in details in the phase of implementation in the next chapter.

Reconfiguration:

The last phase in the control loop would be “reconfiguration” which following the actions taken as countermeasure and for mitigation of a failure, deals with updating the model’s database with reports and feedbacks, and also preparing the failed or broken-down component to be reconfigured into the system to continue its tasks or be reused in other configurations. An example can be of a wireless sensor network with several sink nodes where one node is compromised and infected by a malicious activity of an adversary to eavesdrop on confidential data or to interrupt a functionality. After detection of such ill activity and blocking it and disinfecting the node or component, it can be reloaded with appropriate codes for going back into the system and be reused. The feedbacks and reports provided to the database can give the component’s model the capability of self-optimization with regards to dealing with future failures or foreseeing them and preventing them, or provide more accurate detections and identifications, measurements, and reactions.

The dependability and security model with the properties shown, requires a toolset capable of backing such properties. Accordingly, for reasons that are going to be discussed, Multi-Agents Systems (MAS) are top candidates.

3.2. Contribution of smart system’s properties in the dependability and security mechanism

It has been noted that the dependability and security model is adopting smart systems’ properties to enhance its performance. Now we are going to discuss how the influence of these properties are going to be demonstrated in the behaviour of the proposed dependability and security mechanism. The following table shows how each property contributes in each step of the control loop from inspection to reaction, and what might be the requirements for making these contributions feasible.

Table 2. Control loop steps and the relationship with smart manufacturing systems' properties

	Contribution	Requirements / method
Monitoring: constantly checking the related parameters defined in the core model to find risks		
Context-awareness	<ul style="list-style-type: none"> Parameters are derived from the context. The component knows its context and what is approved within the context. The component also knows its constituents and its own proper behaviour (internal context-awareness) 	<ul style="list-style-type: none"> Values are to be defined based on components type and constituents and its tasks Data is to be acquired via sensors/ cloud
Interoperability	<ul style="list-style-type: none"> To be seen both hierarchically and heterarchically through collaborations of modules of the monitoring unit of the mechanism irrespective of type, parameters, internal structure (e.g. codes or language), etc. Components' interoperation does not require tight coupling between them 	<ul style="list-style-type: none"> Using standards and shared semantics Using proper ontologies
Autonomy	<ul style="list-style-type: none"> Performance of each monitoring operator is not depended on other entities. Each component autonomously performs the monitoring procedure using its tools 	<ul style="list-style-type: none"> Connection to the core (database, topography) to know what to monitor. Sensors, servers, to receive data from Intelligent Agents to perform the monitoring and communication
Modularity	<ul style="list-style-type: none"> Monitoring resources to be used in various setups to modify and enhance monitoring task Monitoring unit is composed of independent interacting modules that may be interacting 	<ul style="list-style-type: none"> To be carried out by intelligent agents

Scalability	<ul style="list-style-type: none"> Monitoring resources (modules) can be added or removed Expansion or shrinkage of the system can easily be supported by extension or reduction of monitoring entities 	<ul style="list-style-type: none"> Parameters and values to be monitored by intelligent agents Each agent can be in charge of one or a group of related values, so changing parameters can be supported by adding or removing agents
Heterogeneity	<ul style="list-style-type: none"> Monitoring modules can be utilized for objects of any form and type consisting heterogeneous constituents Each monitoring agent can observe values from totally different sources and communicate the anomalies to analyser 	<ul style="list-style-type: none"> Agents communication language Semantic definition
Detecting: Finding anomalies/risks by comparing real-time status with approved parameters		
Context-awareness	<ul style="list-style-type: none"> Real time data from context is to be compared with the expected context 	<ul style="list-style-type: none"> Definition of anomalies and upgrading them for use of analyser agent has access to real-time context's approved parameters' values Sensors, cyber space etc. to acquire data
Interoperability	<ul style="list-style-type: none"> Irrespective of heterogeneity of constituents each monitoring agent communicates with detection unit where with collaboration of database agent failures will be detected 	<ul style="list-style-type: none"> Shared semantic/ontology Definition of anomalies Updated database
Autonomy	<ul style="list-style-type: none"> Detection mechanism is autonomous through interactions between monitoring, analyser, and database agents 	<ul style="list-style-type: none"> Context-awareness Agents interaction Updated Database

Modularity	<ul style="list-style-type: none"> • Anomaly data are stored in a modular and tabular way • Sources of data are work as independent modules 	<ul style="list-style-type: none"> • Modular risk Data-base • Modular monitoring agents
Scalability	<ul style="list-style-type: none"> • Database for anomalies and failures is scalable in real-time • Analyser agents with different adopted detection approaches can be added or removed 	<ul style="list-style-type: none"> • Independent analyser agents • Updatable database agent
Heterogeneity	<ul style="list-style-type: none"> • Failures of heterogeneous sort can be parametrised and detected, based on components and their constituents • Detection methods can differ or several detection approaches can be used for various failures within one component 	<ul style="list-style-type: none"> • Use of independent intelligent agents for detection process • Proper definition of failures and detection methods in the database
<p>Measuring: When detected, identifying the risk type, its severity, impacts, occurrence frequency, using assessments methods such as FMEA/FTA</p>		
Context-awareness	<ul style="list-style-type: none"> • Using the context data and topography of the context (internal structure and external collaborations) failures effects will be measured 	<ul style="list-style-type: none"> • Updating context data • Updating topography of the system
Interoperability	<ul style="list-style-type: none"> • In cases where the measurement cannot be done by one agent, components can collaborate over estimating a more accurate measurement 	<ul style="list-style-type: none"> • Via agents' collaboration • Semantic definition
Autonomy	<ul style="list-style-type: none"> • Done by the component itself using one or more agents through the risk model/s used by various agent for different failures or sub-components 	<ul style="list-style-type: none"> • Developing risk models in the model core
Modularity	<ul style="list-style-type: none"> • Measurement resources/ agents may be shared or reused in various or new setups • Risk data are to be accessible in modular way 	<ul style="list-style-type: none"> • Risk categories to be modularly saved • Assessment and modelling methods to

		be used by various agents
Scalability	<ul style="list-style-type: none"> • Risk measurement models and criteria can be updated or replaced • Measurement agents can be added or removed 	<ul style="list-style-type: none"> • Scalable database • Various agents for various assessment approaches
Heterogeneity	<ul style="list-style-type: none"> • Various objects or failures may require different techniques that can be supported by agents carrying out the measurement 	<ul style="list-style-type: none"> • Risk model definition • Using Intelligent Agents
Alarm: Alarming right entities, i.e. people, or components that may be affected by the risk		
Context-awareness	<ul style="list-style-type: none"> • Right entities to be alarmed are to be known through component's knowledge of its topography 	<ul style="list-style-type: none"> • Information about component's internal constituents as well as external collaboration should be updated in real-time
Interoperability	<ul style="list-style-type: none"> • Sending or receiving alarms between entities does not require tight coupling among them and can occur irrespective of their type • Alarms may flow through various entities and to reach their aimed destination 	<ul style="list-style-type: none"> • Proper semantic definition • Proper communication language
Autonomy	<ul style="list-style-type: none"> • Done by agents after measuring risks and finding the right entities from the context 	<ul style="list-style-type: none"> • Understanding the context • Agents' collaboration on steps that comes beforehand (finding, identifying, database, etc.)
Modularity	<ul style="list-style-type: none"> • Alarming resources can be modular and reused elsewhere in other setups 	<ul style="list-style-type: none"> • Modular alarming resource (database, etc.)

Scalability	<ul style="list-style-type: none"> • methods/agents to be added or removed • database for alarming will be scalable to support dynamic changes in the component 	<ul style="list-style-type: none"> • Scalable alarm resource (database, etc.)
Heterogeneity	<ul style="list-style-type: none"> • Alarm messages communication operates irrespective of components type, objective, etc. 	<ul style="list-style-type: none"> • Proper semantic definition
Action: Making globally optimum decisions and defending against/fixing measured risks.		
Context-awareness	<ul style="list-style-type: none"> • Decision is aimed at being more globally optimum context-wise • Knowing the context, negotiations or requests can take place for performing or fulfilling the reactions or for fixation process 	<ul style="list-style-type: none"> • Updated context and topography data • Communication tools • Proper decision models
Interoperability	<ul style="list-style-type: none"> • Component can collaborate with other components in performing a reaction or send requests to other entities for fixation process • Component's constituents can collaborate in performing actions • Components can negotiate over free/available resources they possess to fulfil tasks of the failed resource 	<ul style="list-style-type: none"> • Proper semantic and ontology definition • Context and self-awareness • Proper communication language tools (e.g. agent communication language)
Autonomy	<ul style="list-style-type: none"> • Decisions are taken autonomously by the component (or it negotiates if necessary) • Actions are done through commands that are the outcome of decision maker agent to actuators • Requests and negotiations can be carried out by the components 	<ul style="list-style-type: none"> • Enough understanding of the context by the component • Stablishing proper communication platform (agents) • Definition of proper semantics
Modularity	<ul style="list-style-type: none"> • There can be one or more decision maker agent with various decision-making approaches that can be used or reused in new setups 	<ul style="list-style-type: none"> • Intelligent agents for decision making and acting process

	<ul style="list-style-type: none"> • Database for decision making and acting/ reacting process will be set up in modular way according to components, or their constituents type, etc. 	<ul style="list-style-type: none"> • Updatable database agent in real-time
Scalability	<ul style="list-style-type: none"> • Decision making and acting resources can be scalable as in terms of agents that perform the actions, actuators that carry out the commands, as well as the database that provides data about context, failures and countermeasures, entities to negotiate, etc. to match the dynamics nature of the components 	<p>Scalable database</p> <p>Registration of decision making agents of various required approaches or functionalities</p>
Heterogeneity	<ul style="list-style-type: none"> • Various types of strategies, approaches or kind of countermeasure can be applied according to the components and their constituents. Different agents harnessing different resources can be used for one component, controlling its various, perhaps heterogeneous parts 	<ul style="list-style-type: none"> • Proper semantic definition • Intelligent agents carrying out decision making and acting process
Reconfiguration: providing feedback to the model and preparing the component to be reused		
Context-awareness	<ul style="list-style-type: none"> • To be done based on context requirements 	<ul style="list-style-type: none"> • Via updating the risk model • Via updating the other parts of the system (adding more monitoring resource, etc.)
Interoperability	<ul style="list-style-type: none"> • Providing understandable feedbacks to others/ receiving required data 	<ul style="list-style-type: none"> • Via model and semantic definition
Autonomy	<ul style="list-style-type: none"> • Semi/fully autonomous and done by agents or humans (if necessary) 	<ul style="list-style-type: none"> • Standard model development • Agents/ humans, etc.
Modularity	<ul style="list-style-type: none"> • Mechanism's modules may change or reconfigure for reuse or improvement after a failure occur 	<ul style="list-style-type: none"> • Modular components

Scalability	<ul style="list-style-type: none"> • Based on the needs and decisions, the mechanism may shrink in size or expand in different parts to address failures better or to prevent them, etc. • The database accordingly can expand or shrink 	<ul style="list-style-type: none"> • Scalable model and database • Agents in different sections of the mechanism to be added to removed
Heterogeneity	<ul style="list-style-type: none"> • Reconfiguration process and objectives can vary vastly based on the component type and its constituents 	<ul style="list-style-type: none"> • Model development

In the next chapter the implementation of the agent based structure of the mechanism is going to be shown and discussed along with the platform and applications used for developing agents and their behaviours. However before doing so, using DACS technique, agents' identification, decisions and behaviours, as well as their responsibilities and interactions will be specified. The cases of failure will be elaborated and the mechanism's behaviour in monitoring and detecting, identifying, and reacting to these failures will be demonstrated. In addition, an extra case will be simulated to show the negotiations and interactions over using a resource offered by some other components.

CHAPTER FOUR

MODEL IMPLEMENTATION AND USE CASES

In this chapter, the model implementation using multi agent systems (MAS) in JADE platform will be presented in details and the system under consideration and the failure case scenarios thereof which are going to be tested will be discussed.

4.1. Using DACS to design the mechanism

Even though the DACS methodology is designed originally for production control systems, it is abstract and powerful enough to encompass almost all multi-agent systems design, given that decision making and communication can be seen as two of the main features of almost all multi agent systems.

Accordingly, the following table can be formed to help us design the dependability and security mechanism in terms of intelligent agents and their behaviours, decisions, and interactions. The table structure is borrowed from the survey on agent design patterns for productions system control by Lüder et. Al. [139]

Table 3. Table of agents and their behaviour, decisions, and interactions

Decisions	Intended behaviour	Supervision and control of the dependability and security of the component under consideration	
	Decisions	Reacting to observed anomalies	
Agents	Database Agent (Risk Model, info) (DBA)	Behaviour	<ul style="list-style-type: none"> • Storing and modifying risk data via a user interface/ automatically • Storing and modifying reactions' suggestions and requirements via a user interface/ automatically
		Decisions	•
	Monitoring agent/s (MA)	Behaviour	• Receiving values and comparing them with the approved pattern
		Decisions	• Whether a value or pattern is considered an anomaly

Analyzer agent level one (AZ1)	Behaviour	<ul style="list-style-type: none"> • Receiving the anomaly and identifying it • Receiving alarms from other components
	Decisions	<ul style="list-style-type: none"> • What should be sent to decision maker
Decision Maker Agent level one (DM1)	Behaviour	<ul style="list-style-type: none"> • React on the identified failure or alarm received
	Decisions	<ul style="list-style-type: none"> • What is the immediate action • Who to alarm • How to mitigate or fix the issue
Assessment agent (ASA)	Behaviour	<ul style="list-style-type: none"> • Measuring the severity of the risk
	Decisions	<ul style="list-style-type: none"> • What factors should be considered
Topography agent (TPG)	Behaviour	<ul style="list-style-type: none"> • Provides and updates a dataset of network of connections of components
	Decisions	<ul style="list-style-type: none"> •
Analyzer agent level two (AZ2)	Behaviour	<ul style="list-style-type: none"> • Receiving the anomaly and identifying it • Receiving alarms from other components • Tackle anomalies that are unknown or negotiate over identifying failures
	Decisions	<ul style="list-style-type: none"> • What should be sent to decision maker
Decision Maker Agent level two (DM2)	Behaviour	<ul style="list-style-type: none"> • React on the identified failure or alarm received
	Decisions	<ul style="list-style-type: none"> • What is the immediate action • Who to alarm • How to mitigate or fix the issue • What to negotiate about • Who to negotiate with and what are the constraints

4. MODEL IMPLEMENTATION AND USE CASES

	MA – Server Client (OPC-UA client)/ field etc.	Protocol	
		Exchanged info	Real-time sensor data
Interactions	MA – AZ1	Protocol	Inform
		Exchanged info	The anomaly/ies found
	AZ1 – DBA	Protocol	
		Exchanged info	Accessing the risk data (anomaly/ies: failure)
	AZ1 – DM1	Protocol	Inform
		Exchanged info	Identified failure
	AZ1 – AZ2	Protocol	Inform
		Exchanged info	Anomaly data for identification
	AZ2 – middleware (negotiation)	Protocol	Request/Inform
		Exchanged info	Negotiation over anomalies' identification
	AZ2 – DM1	Protocol	
		Exchanged info	Identified failure
	DM1 – DBA	Protocol	
		Exchanged info	Requirements Resources Actions Alarms Units to call
	DM1 – ASA	Protocol	Inform
		Exchanged info	Risk severity
	DM1 – DM2	Protocol	Inform
		Exchanged info	The failure to deal with
	DM1 – (AZ1 other componenets)	Protocol	Inform
		Exchanged info	Sending alarm (failure)
	DM2 – DBA	Protocol	
		Exchanged info	Requirements Resources Actions Alarms Units to call

	DM2 – Middleware (negotiation)	Protocol	CFP/Inform/ ...
		Exchanged info	Negotiation over resources Negotiation over optimum decision

Doing so, the following lists the agents that are defined to specify the model, and their task description. Figure (31), shows their overall structure and collaborations' relations [6]. It needs to be mentioned that the number of agents may vary according to the case. For instance, monitoring task can be split among agents each of which monitor one variable or a group of related variables. On the other hand, data base is more likely to be presented by only one agent. The number of Analyser and decision-making agents can also be based on the number of different approaches or algorithms used for each phase (which itself might be based on the heterogeneity of the issues).

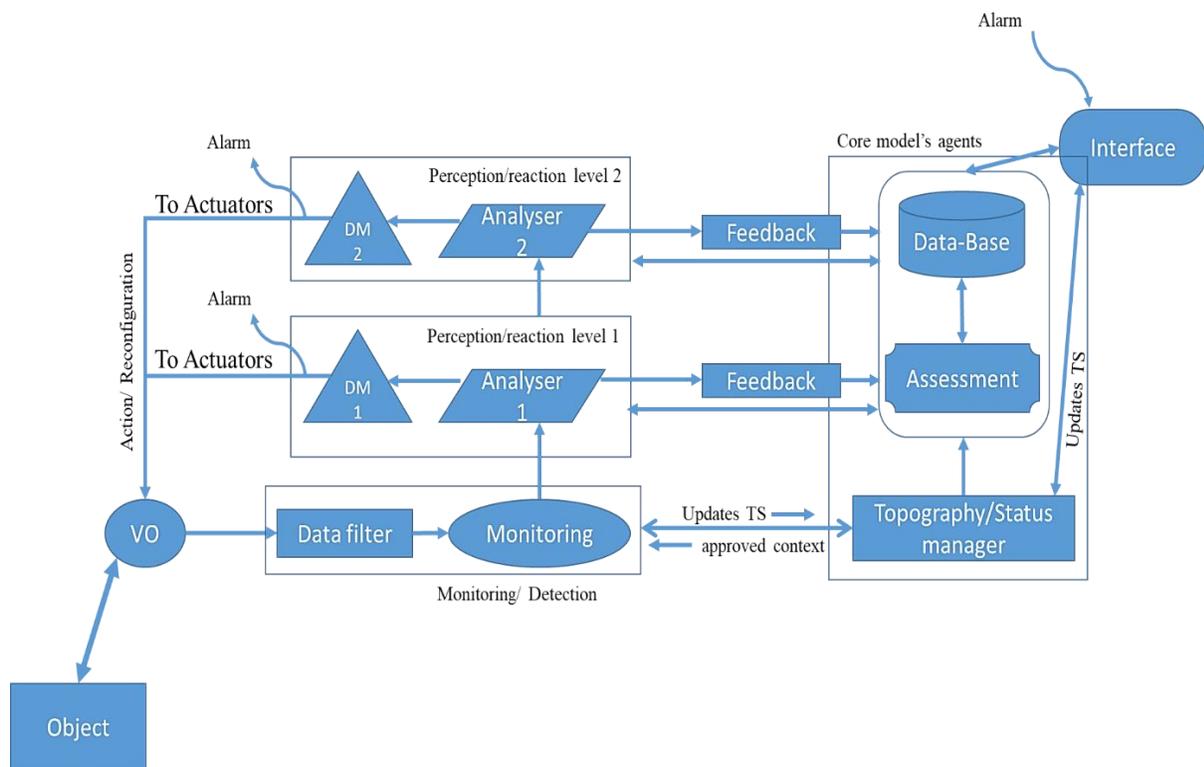


Figure 31. Agents structure for dependability and security mechanism development

Agents supporting the model core

Status/topography manager: Updates the status of components as main part of context-awareness. Knows all approved contexts, authorities, topography of the system, etc. The

component should know what are its constituents and what is to be monitored. In addition, it should know with whom it is collaborating. That is to let the component be aware of other components that must be alarmed when it is necessary.

Database: Stores components' models and risk data. Data are stored in modules for each type of risk to be accessed by analysers and assessment agents. Other components when authorized, have access to some of the data during negotiation or when they are new to the system, to get updated with vulnerabilities and measures, etc.

Risk data here are all the information that might be important for the control loop of the dependability and security mechanism. That means, the database should have data required for identifying risks based on anomalies detected. It should also have information associated with risk measurement, and countermeasure suggestions, or in other word data that assist decision making process. These data include: information about resources that will breakdown are necessary to replace in order to avoid or mitigate delays in delivering the expected services; information about fixation requirements and other units that might need to be involved in fulfilling those requirements.

Various methods can be used based on applications and/ or the interest of the developer to organize and form the database. Some of these methods have been introduced in chapter two. This study forms failure information in the database by developing a table (which is a modified instance of FMEA table) of failures, and data about their detection and identification, their effects and severity (data useful for calculating severity), requirements, etc. which are shown (figure (32)) and described below:

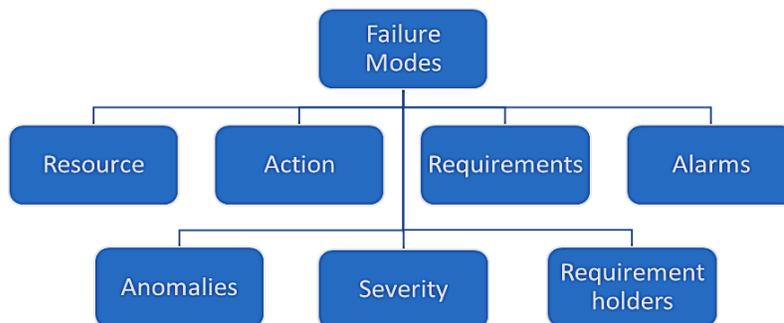


Figure 32. Failure Data as categorized in the database

- **Anomalies:** a list of expected anomalies associated with each failure is to be stored and updated to feed the identifier agent with the required data.
- **Actions:** possible suggestions for actions to be taken should a failure or error have occurred. For some problems, some actions can be definite. For instance, a machine in a line breaks down and one of the actions might be to turn of the conveyor that takes parts to it for processing. Complicated actions may require more complicated decision-making procedures that may involve negotiations to other intelligent components or human intervention in cases that no proper action was found by the system.
- **Requirements:** some of the failures may contribute in breakdowns which the fixation procedure may require spare parts, or depending on the type and constituents of the component may have other requirements. These requirements are being listed and categorized for individual failures.
- **Alarms:** updatable list of entities to be alarmed in cases of different failure occurrences is required to be stored in the database to be accessed by decision maker agent or other possible authorized entities.
- **Resources:** the tasks that were to be fulfilled by failed component might be urgent in some cases and not sufficient time might be available to wait until the failed component is repaired or replaced. Some component may be unjustifiably costly to replace and may also be timely to repair. For such or similar scenarios there should be information about the type of resource needed to fulfil the halted task to avoid possible delays in service delivery. This information can then be accessed by for instance decision maker to find a temporary substitute or make a right decision about it.
- **Requirement holders:** are possible entities or components that may have the requirements discussed above. It can be inventory unit, maintenance unit, etc.
- **Severity:** as described in chapter two, FMEA or some other risk modelling technique also considers the severity of failures in assessing them. Depending on the application etc. the technique can be different.

Assessment Agent: Receives data from the analyser and assesses the risks (if necessary, by negotiating with other components' assessors), and by receiving context data from TS manager. Here the risk should be assessed after it is identified by analyser. Assessment and level of the risk

can depend on its effect on the component itself and its modules, as well as other components with which it is working or it is a constituent of.

Effects are defined based on the topology data, i.e. which higher level components might be affected, or what other object might be compromised, given their relationship with the failed component. The severity accordingly can be determined as follow:

```
probability [ ]   p = {P0, P1, ..., Pi};
Severity [ ]     s = {S0, S1, ..., Si};
Double          totalSeverity = 0;
for (int i = 0; i < p.length; i++) {
totalSeverity = totalSeverity + p [i] * s [i];
}
```

Where P_i and S_i respectively indicate the probability and the severity of losses to each of the other components, that are collaborating with the compromised component (in cases where there is a possibility of risk propagation or a failure causing other failures to other components in collaboration with it). Other information to record after each risk it the time, and its new occurrence value (OccurrenceCount ++).

Agents supporting the control loop:

Data filter: Filtering out redundancies. Looking for useful data among loads of data. In cases, it might happen that a large amount of data is available and not all of these data are desirable for monitoring and analysis or storing.

Monitoring: Looking for anomalies and risks, by comparing the data of the current-state with what is approved for the current-state's contexts. Then sends the detected cases to the analyser. There can be any number of monitoring agents based on the system under consideration. Each agent can be checking one or more values that are parametrized to describe a state of a process or one or more constituents of the component etc. changes in these values if considered as anomalies such as if a threshold exceeds, or an approved trend is violated.

Agent-based monitoring makes it possible that without any significant effort, monitoring mechanism become scalable and breaks down into level of details and be performed totally

heterogeneously. That means scalability of the system and dynamic changes in the size of the system or its constituents will not affect performance of the monitoring mechanism, given that agents can be easily added and removed for checking any new parameter from any new constituent that has been added to or removed from the system or has changed or replaced by a component of different manufacture or parameters.

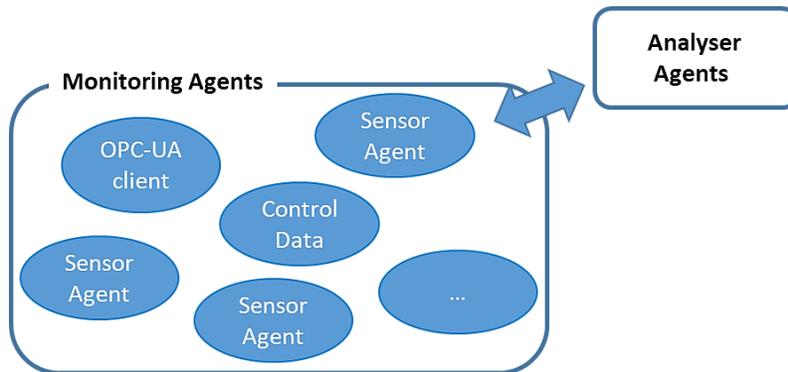


Figure 33. Monitoring Agents

In cases, where the system has OPC-UA (Open Platform Communication-Unified Architecture) implemented, the monitoring agent subscribes to it as a client to receive the sensor or control data required. OPC UA provides an information transition standard which enables a dependable and secure communication of data among industrial components. One of the benefits of this standard is the ability to perform independent of the platform and to make possible the data exchange and interaction of components of different developers irrespective of operating systems used. This standard has been made on the basis of specifications defined in collaboration with manufacturers is based on specifications that were developed in close cooperation between manufacturers, people and organizations which performs relative research studies and actual utilizers of this standard to reach the best outcome in its development. OPC UA was introduced as a more capable successor for the already popular OPC which was commonly used in various areas, more specifically industrial automation. Among its new capabilities scalability, platform-independence, and service-orientedness of its architecture are some to stand out.

OPC UA has an event-based alarming mechanism that can inform the client (i.e. here the client can be monitoring agent or various monitoring agents each of which might be responsible for one or more values) if a certain condition in a certain part of the system is not met. This mechanism in OPC-UA is configurable by the client, and is scalable. Another feature to note is

that on OPC UA server there might be innumerable amount of data and values associated to numerous variables. However, the client can subscribe to receive and monitor only values that are of the interest of the dependability and security mechanism. The rest of the data which is unnecessary will be easily filtered out.

Data-analyser: The analyser task can be broken into simpler and more complex problems. More complication may arise from lack of sufficient anomaly data or lack of rich risk information in database. That can, as shown in the figure (31), give rise to developing two separate analysers one for simpler and one for more complicated issues. In the abovementioned figure, they are called respectively analyser level one and analyser level two. For simpler problems and provision of quicker responses, data analyser level one does the identification and measurement of risks. It communicates with the assessment agents and the database to provide proper input for *decision maker agent*.

Data analyser level two on the other hand is for more complicated problems where analyser level one is incapable of identifying and measuring them and hence it hands the problem to the level two analyser with more abilities. If needed this analyser performs negotiations with other components' agents to provide best global data of the risk to feed the decision maker agent for right decisions and actions.

Decision-making agent: similar to data analyser agent, as shown in figure (31), decision making can also be separated into two levels corresponding to simpler and more complicated issues, performed by respectively level one and level two decision maker agents. For simpler reactions and quicker responses, the former will be responsible. After supplied with proper risk information, decision maker agent releases alarm to right entities and send proper commands to actuators to apply right corrective actions, and reconfiguration when needed. Feedback is then being provided to the core model.

To provide more advanced decisions, higher level reactions and alarming, decision-making agent level two is to be summoned. If needed, it asks for collaboration of other agents and components' resources to solve a problem. These collaborations can be in terms of asking for fixation, requesting spare parts from inventories, etc. It can also make inquiry from other units for making use of perhaps the same resource that has broken down within this component to make sure of meeting the deadlines for proper service delivery (figure (34)). Feedback is then being provided to the core model.

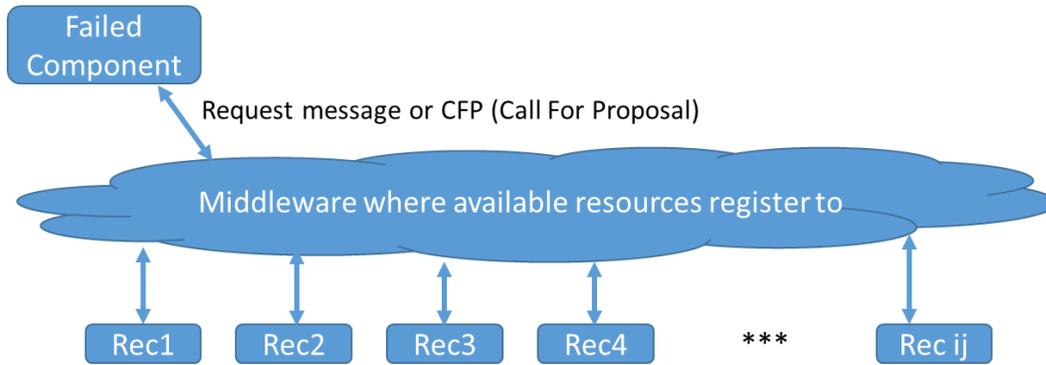


Figure 34. Negotiation over utilization of other entities resources

The following figure shows the agents developed for the dependability and security mechanism in the main container of the JADE platform. The offer provider agents are later going to be discussed further in more details. These agents are designed to provide negotiation over resources as previously mentioned.

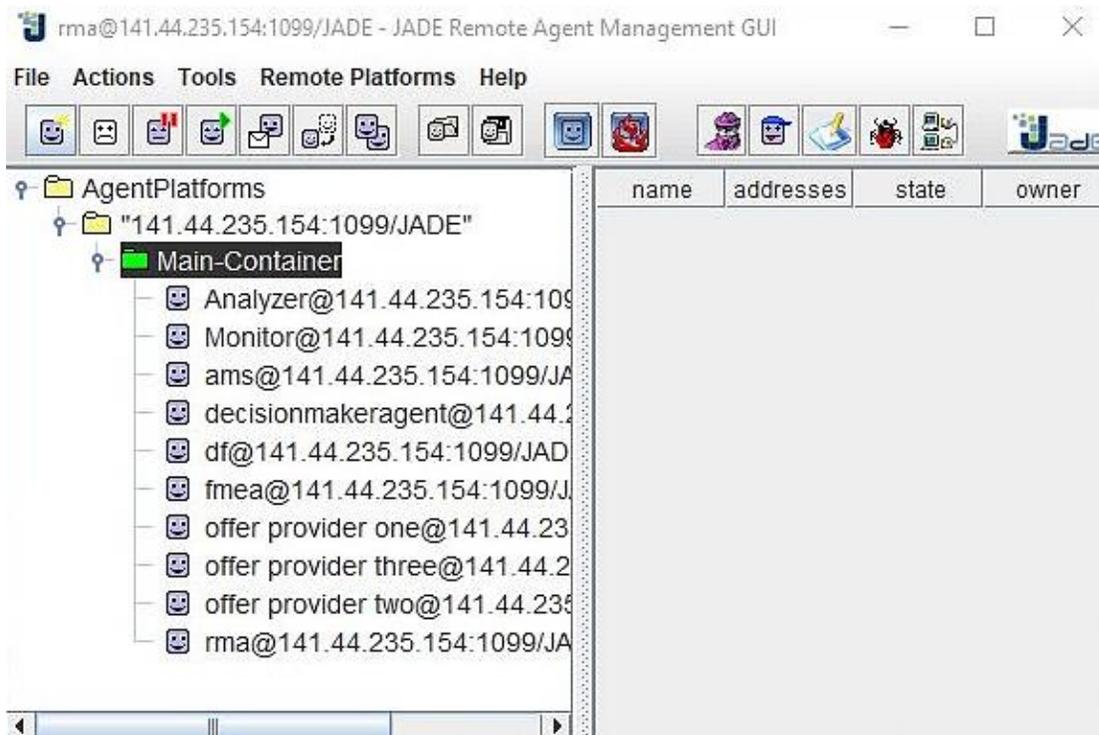


Figure 35. agents in the main container

4.2. The use cases for testing the mechanism

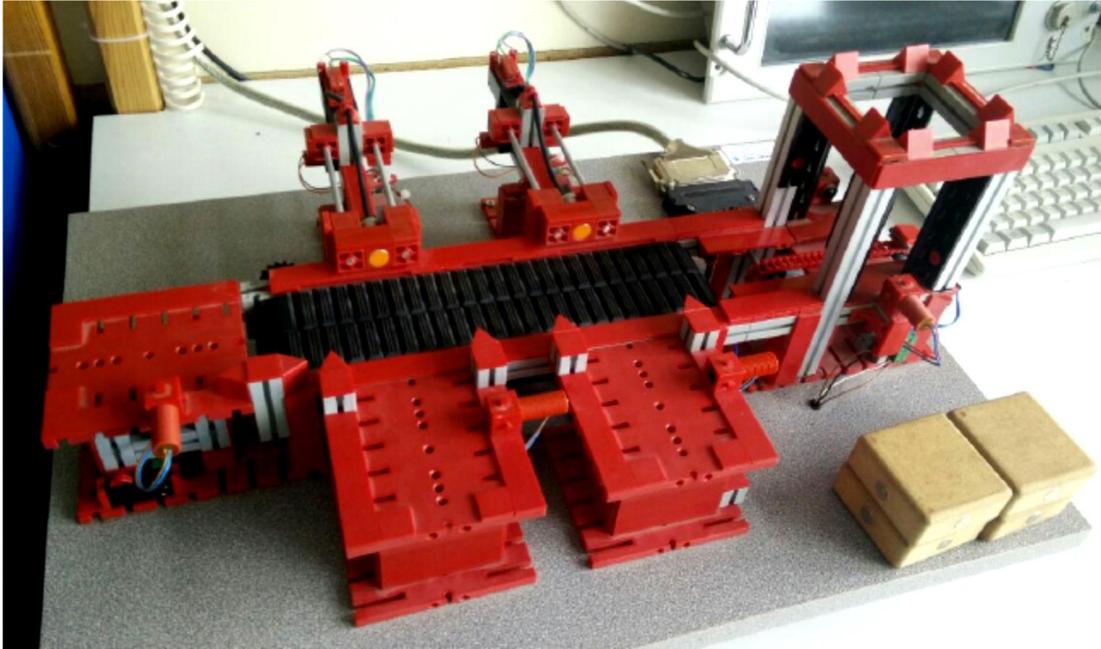


Figure 36. The Fischer model

The case chosen to demonstrate the functionality of dependability and security mechanism is the Fischer conveyor model as shown in the figure (36). The system consists of a loading station where objects (wood blocks) arrive, and tables where objects are destined. It also contains two pusher robots and conveyor belts as well as a number of sensors to provide data.

The process within this system runs as follow: wood blocks can be a representative of a pallet of products to be transported to different stations based on the type of the products. And different tables (stations) can represent machines or work stations where the received products will be processed. The pallets (wood blocks) will be loaded into the loading station and will then reach the conveyor belt via a small band, and then after being identified (to know that which stations, here table, is its determined destination), it moves towards its destined table via conveyor belts and pusher robots that will push the wood block to the specified table. The system (i.e. Fischertechnik model) represents order-based products as its throughput. And the outcome might be delivered products to destination, or products that are going to be processed or customized to the will of the end users and customers.

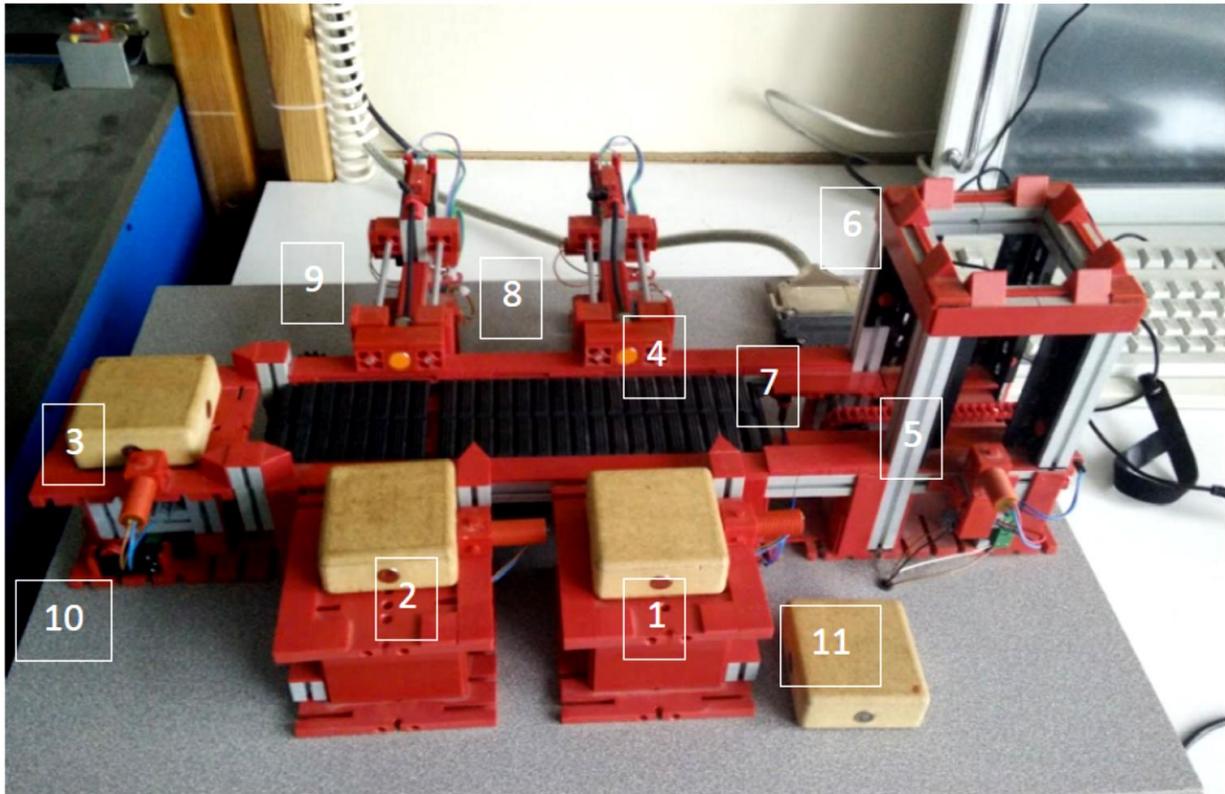
It must be added that the system conforms to Industry 4.0 specifications. The system is also capable of providing self-optimization with regards to productivity increase by e.g. optimizing the

speed of conveyor belts. Furthermore, agent-based control has been previously developed for this system which provides flexibility in its control of processes, resource management and order organization.

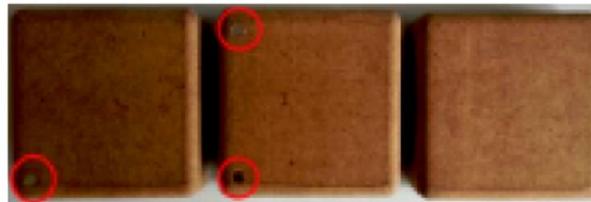
Along this process various cases of disturbances and interruptions may occur in the case of failures, errors and malfunctions. These cases will be categorized and arranged in a table to form the information needed to start and develop the database for the mechanism. It will be discussed further and the cases will be derived from the table of failures. Here the goal of this modelled production system is to have maximum productivity and to have a satisfactory level of robustness. It will further be shown how the dependability and security mechanism can deal with issues associated with this case to provide a distributed and intelligent autonomous supervision and control whose behaviour matches with smart systems properties.

4.2.1. Constituent of the model production system

Within this section the different parts and sections of the system and their functionality will be shown and presented through figures and tables below. Moreover, since the system is equipped with various sensor and identification magnets, they are also going to be specified, and in the next section these sensors will be used as means of data provision for monitoring agents for detecting anomalies.



a)



b)

Figure 37. Objects and sections associated with the case (Fischer model): a) objects and sensors for the production system, and b) parts (wood blocks) and magnets

Table 4. Components of the conveyor system, and their behaviour

number	component	sensors	functionality
1	Table one	One sensor	Taken/ free
2	Table two	One sensor	Taken/ free
3	Table three	One sensor	Taken/ free
4	Conveyor belt	Motor	Moving/ not moving

5	Loader belt	Motor, one sensor and one magnet	Moving/ not moving
6	Loading station	Two sensors	With parts/ without parts
7	Object identifier	Two sensors for magnet	Identifies the object
8	Pusher robot one	Two sensors Motor	Pushing the item to the destined table
9	Pusher robot two	Two sensors Motor	Pushing the item to the destined table
10	Button	Either on, or off	-
11	Wood block	Four marks, and One/two magnets	Going to specified table

the following figure shows various part of the control system of our model production system.
The items specified in the photo are as follow:

1. BUS coupler
2. Raspberry Pi programable logical controller (PLC)
3. Switch
4. Power source

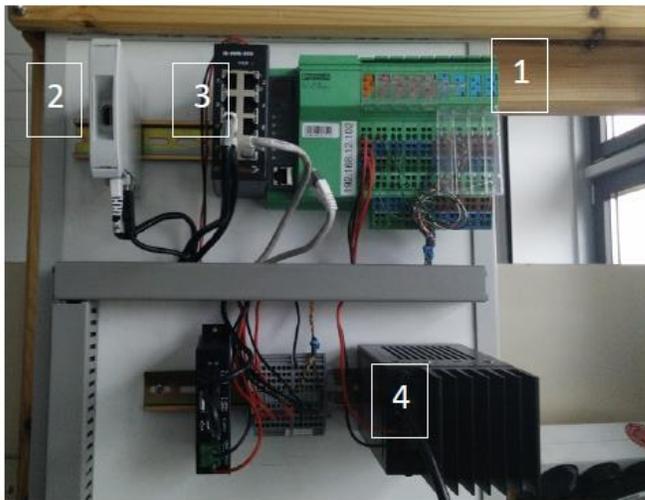


Figure 38. The structure of the control system

4.2.2. Failures associated with the case (conveyor)

In order to specify the failures that are possible to occur within this system, using categorization explained in chapter two, we pointed out some of the risks and by adopting FMEA technique a table of risk information (on the basis of FMEA table and modified to the needs and necessities of this thesis and the proposed model) was formed to feed the system with required data that will be used by developed agents within the dependability and security mechanism designed to monitor and control the system.

As previously discussed, the required information in the table are: the anomalies (for monitoring, and identifier agent), failure modes (for identifier), and information that are necessary for decision making that include (but not limited to): Possible immediate actions, corrective or mitigating actions, fixation requirements, resources to look for and negotiate upon, involved entities for fixation process (transportation, inventory, maintenance unit, etc.), and entities to be alarmed (such as those that may need to alter their behaviours according to this failure, etc.). depending on the system under consideration or methods adopted for different parts of the mechanism, the information needed might change which all can be included in the database (risk model agent) and be altered through the user interface.

To develop this table, we can categorise the failures according to level of details and modules that are included in the system and shown in the figure (37) and listed in table (4).

Table 5. Failure information and action data

Failures	information	items
Conveyor belt has delay	Object	Conveyor belt
	Description	Item does not reach to the specified pusher on time and timely arrival to the desired table may be violated
	Anomalies	Pusher sensor $0 < \text{Time violation} < a$ ($a = \text{buffer time}$)
	Severity	•
	Immediate Actions	no action
	Corrective Actions	Call for maintenance team to check the belt/ motor
	Resource Failed	none
	Alarms	Maintenance unit (request), staff in charge of this unit (inform)

	Involved entities	Maintenance unit
Conveyor belt stops working	Object	Conveyor belt
	Description	Item does not reach the specified pusher on time and it didn't get to the destination table. The buffer time between two items is violated
	Anomalies	Pusher sensor Table sensor Time violation > a (a = buffer time)
	Severity	•
	Immediate Actions	Stop the conveyor
	Corrective Actions	Reschedule the process Call for maintenance Call the inventory for necessary parts
	Resource Failed	None for negotiation
	Alarms	Control unit (inform), maintenance unit (request), inventory unit, staff in charge of this unit (inform)
	Involved entities	Maintenance inventory
Pusher robot p_i does not work	Object	Pusher p_i
	Description	Object arrives at pusher but the pusher does not push it to the right table where it is expected
	Anomalies	Table sensor Time violation
	Severity	•
	Immediate Actions	Stop the process
	Corrective Actions	Send Request to inventory unit (parts) and maintenance unit (repair)
	Resource Failed	•
	Alarms	Control unit
	Involved entities	Inventory Maintenance Control unit
	Object	Pusher p_i

Pusher maintenance call due p_i	Description	The machine requires maintenance after a period of work to maximizes its productivity and satisfies certain related KPIs
	Anomalies	Number of pushes = a
	Severity	•
	Immediate Actions	Stop the line
	Corrective Actions	Call for maintenance
	Resource Failed	none
	Alarms	Maintenance (request), staff in charge of this unit (inform)
	Involved entities	Maintenance unit Control unit
Intrusion Attack (DoS) to control data	Object	Controller unit
	Description	The control data may be compromised by an adversary or an attack. That may pose threat to the entire system as partial or total breakdowns, or cause information leakage
	Anomalies	Data traffic > t (expected bandwidth consumption at a specific moment)
	Severity	•
	Immediate Actions	Trace the compromised channel Report the compromised channel
	Corrective Actions	Change the channel Fix the compromised entity
	Resource Failed	•
	Alarms	Control unit (inform) Control unit (request) Staff in charge of this unit (inform)
Involved entities	Control unit IT unit	
Part is not in the loading section on time	Object	Loading station
	Description	Blocks have not been loaded in the loading station and therefore are not ready to go on the conveyor towards the specified tables
	Anomalies	Loading station sensors
	Severity	•
	Immediate Actions	Stop the line

		Call the unit staff
	Corrective Actions	Inform control unit Reschedule the process
	Resource Failed	•
	Alarms	Control unit Staff in charge of this unit
	Involved entities	Control unit Staff in charge of this unit

4.3. The failure cases in this study

Out of the failures listed above, the cases below are chosen to demonstrate some different type of cases that can be considered.

4.3.1. Case one: Scheduled maintenance

One of the essential practices as a basis of management is to assess the performance of the system and/or the pieces of equipment. The importance of assessing the performance lays within the fact that it reveals how much the current performance needs to be enhanced to reach the desired performance levels. This therefore suggests that for assessing the performance KPIs (or Key Performance Indicators) must be chosen wisely to help focusing on the appropriate actions to improve the performance with respect to the system and its objectives. Some of the most important KPIs can be related to asset performance and asset reliability. Examples of some of these indicators can be as follow:

- OEE (Overall Equipment Effectiveness): Availability * performance * quality
- TEEP (Total Effective Equipment Performance): measures OEE against loading time
- MTBF (Mean Time Between Failures): (Total up time) / (number of breakdowns)
- MTTR (Mean Time to Repair): (Total down time) / (number of breakdowns)

One of the major ways of assuring the performance of machines and pieces of equipment is to make sure that they are in good health. This is where the concept of preventive maintenance can get importance since it aims at avoiding failures and breakdowns to take place. Preventive maintenance is a scheduled program and routine guarantee maximum availability of machines and devices, and minimum errors and failures. To do so, this program regularly looking after machines and equipment.

Preventive maintenance might bring the company some costs. However, the gains outweigh the losses remarkably. Below are listed some of the main costs and benefits of practising preventive maintenance [140, 141].

Expenses:

- Cost of personnel for managing and timing
- Cost of personnel for doing the maintenance procedure
- The time at which the machine or device is not performing due to its maintenance procedure which leads to loss of production for that period.
- Expenses related to the material used for this procedure

Benefits:

- Less deviations from acceptable service will occur
- Fixation expenditures which are often remarkably costly will be avoided
- Higher percentage of machines' outputs will conform to required specifications, which means lower costs of redoing jobs
- Higher level of safety can be obtained

Comparing costs and benefits of preventive maintenance, first thing to notice is that the costs of performing maintenance are usually much lower than fixation costs that may arise by lack of taking sufficient care. A rather clear example could be the comparison between the costs of lubrication for a machine and the cost of replacing a motor that can fail otherwise which may propagate and cause more damages and harms to other sections, or pose safety threats to operators and personnel in its vicinity. It can be suggested that in most case the same result can be extrapolated.

The way preventive maintenance performances can help avoiding defects might for instance be to prevent cumulative and gradual wears, misalignments, calibration issues, etc. that may come about by intense exploitation of that device or by environmental factors like moisture, heat, vibration, to name a few.

Performing preventive maintenance is normally in the form of scheduled maintenance activities. This scheduled routine can be planned on the basis of various factor that can depend on the type and functionality of the equipment, as well as the conditions under which it is working and performing. The most common of these factors are as follow:

- Aggregate number of hours the device operates

- The number of parts produced or number of operations performed
- The number on/ offs

In this case the criteria for the pusher robots are defined as the number of pushes at which the preventive maintenance of for the system shall be called.

4.3.2. Case two: Item (wood block) is not present on time

Another common issue that may arise is delay in the execution of processes where it is supposed to be at a specific time. Delays may be caused by various different failures, and in the case of this thesis where the part should start in the loading section and then reach to specified destination table via conveyors and pushers, delays can be as follow:

- Delay in reaching the loading station
- Delay in reaching the pusher
- Delay in reaching the destination table

Each of these cases can occur due to failures specified in the table above (table (5)) and can be distinguished via specific types of anomalies associated with them. These anomalies are to be detected using sensors at each step to identify the presence of wood blocks with magnets placed on them.

The experiment takes two level of delays:

1. Delay that is less than the buffer time (time till the next work piece be placed on the conveyor)
2. Delay that exceeds buffer time

The monitoring agent which is reading sensors as a client of OPC UA server and is responsible for variables related to such failure, is programmed with the timings required for the process under consideration. In this case the buffer time is set to 10 seconds and the normal expected time should be 5 second after the start of the process.

In the case of the acceptable delay within the buffer time, the process does not need to stop and only rescheduling the process might be necessary to keep the buffer time. So, when the item (wood block) does not appear on time, the monitoring agent sends a message to analyser reporting the delay to reschedule the timing of the process. Also, the unit staff will be called to check the possible cause of this delay.

When the delay consumes the accepted delay time, the process will stop following the anomaly message generated and sent by the monitoring agent, and based on the anomaly the failure will be found and dealt with accordingly.

4.4. Agents, interfaces, and their interactions

As previously demonstrated in this chapter, the agents were specified using DACS methodology and their behaviours, responsibilities, objectives, communications, etc. were determined. For the sake of practicality interfaces have been designed to demonstrated their behaviours in dealing with the failure cases described above within the system under consideration (the conveyor). Agents' communications will also be presented.

The first step after developing a table of required data for failures (which can be developed through any known or preferable technique, among which FMEA was preferred for this study), is to provide the model with the ability to read and store these pieces of information as well as to edit, add or to remove them. To that aim, main ontology class was developed for all the failures to share including the variables defined for the failures which are namely: name (mode), anomalies (to help with detection), actions to take, resources that is failed (for substitution and negotiation with others over this failed resource), requirements (which might be needed for fixation). This way every failure class extending failure mode ontologies can easily be added to the list. In addition, to make it easier for real-time changes in failure modes data base, interfaces were developed as figures bellow:

Figure 39. Failures database interface for adding, removing, editing failures and their data. One the right, information for failures can appear



Figure 40. List of failures at any time

All the failures added to the database through any means can be seen in the interface shown in figure (40), and using the interface in figure (39) failures can be modified or scaled in real-time. Also, information about any given failures can be shown and checked in the panel on the right side of the figure (39). Components can be saved with their collaborations using which failure propagation and alarm signals can be accordingly generated.

The monitoring agent for this case experiment as said before is a client of OPC-UA server which reads the sensor and control data. Each monitoring agent, responsible for one or a group of related variables and parameters, will prepare a “INFORM” message to analyser once they found an anomaly in the data, and the message ID will be “anomaly report” so that the analyser knows what is the purpose of this message and distinguish it from alarm messages.

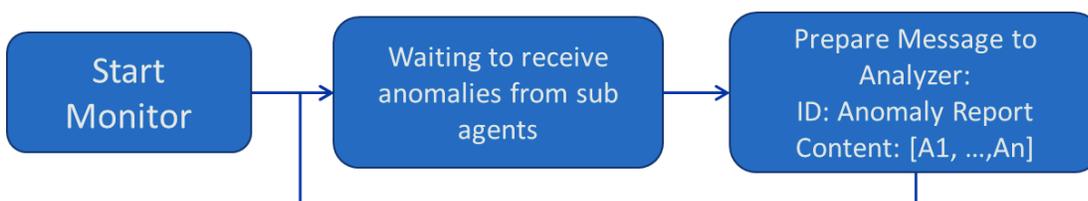


Figure 41. A monitoring agent's general behaviour

Analysers are constantly looking for messages as “ID: Alarm” from other components’ decision maker agent, or as “ID: anomaly report” from the component’s own monitoring system. Once it

receives the message from monitoring agent, using the capability of having access to risk data, it looks for the type of failure (i.e. failure mode) according to the array of anomalies it received. In the next step, it prepares an “INFORM” message with failure data as its content for the decision maker agent.

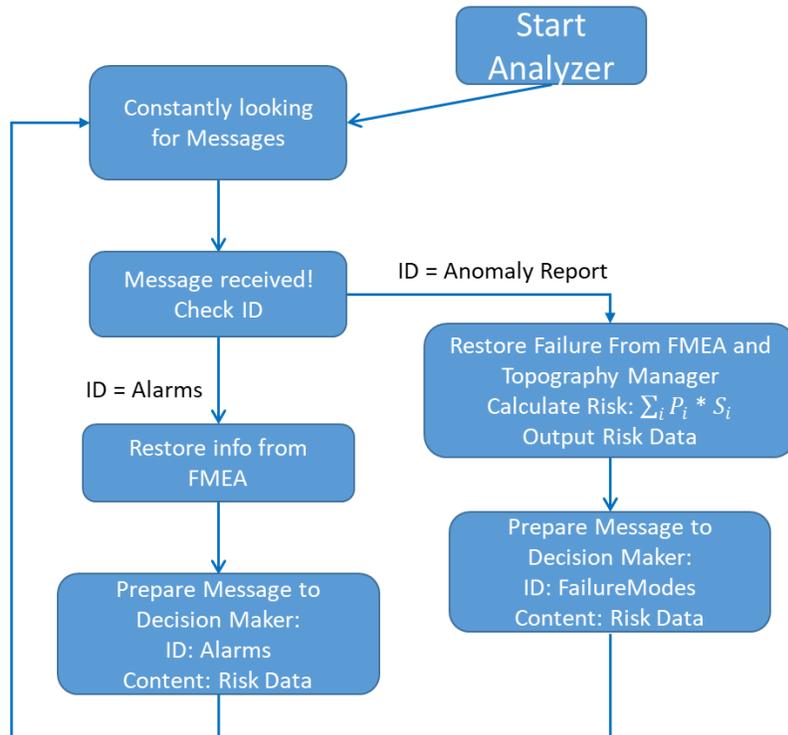


Figure 42. Analyser Agent's general behaviour

As can be seen from the figures below (figures (43 and 44)) analyser agent reacts to the set of anomalies received from the monitoring agent for each case (Conveyer breakdown, and pusher robot's maintenance due), the way that is shown in the diagram above (figure (42)) Anomalies are specified for monitoring agents based on the risk model so that they know what each deviation from the approved behaviour means as an anomaly. On the other hand, analyser agent should figure out what each set or array of anomalies can mean as a failure. The results are then sent to the decision maker agent with the proper conversation ID (failuremodes), for further decisions on actions and countermeasures.

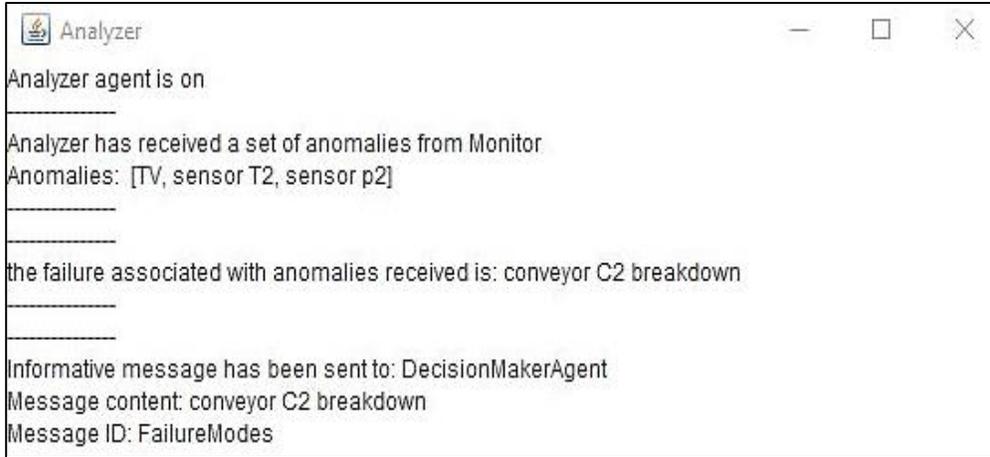


Figure 43. Analyzer agent's activities for the case of conveyor breakdown

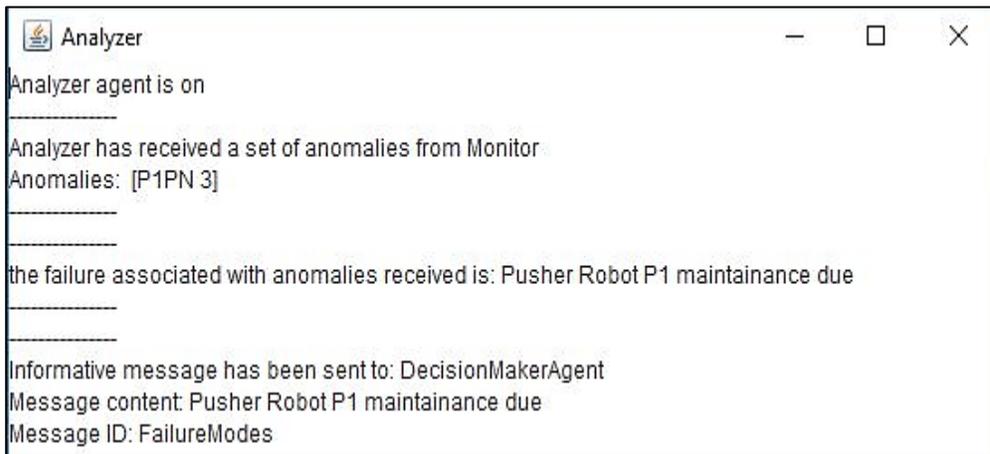


Figure 44. Analyzer agent's activities for the case of pusher robot's maintenance due

Next agent to involve is the Decision-Making agent that is constantly looking for a signal of a failure to react to it. It retrieves the required information from the failure database on how to react, who to alarm, what to negotiate on (if there is a resource to negotiate on) and who to negotiate with. The behaviour of the decision-making agent is shown in the following figure. It constantly looks for messages to react to. Messages can generally be either from analyser to with “INFORM” template to notify the decision maker of a failure, or they can be “CFP” (call for proposal) messages from negotiations over resources.

In case of messages from analyser, the message ID can be either “alarm”, or “failure modes”, where the two may trigger different sets of behaviour. In case of the former, there might be actions needed to avoid receiving harm from the compromised entity or to stop the running process since the compromised collaborator is not capable of providing service at the expected

time. The latter on the other hand, requires more considerations such as fixations, requirements and requirement holders (e.g. spare part and inventory unit), resource substitutions and negotiations, alarm provisions, etc.

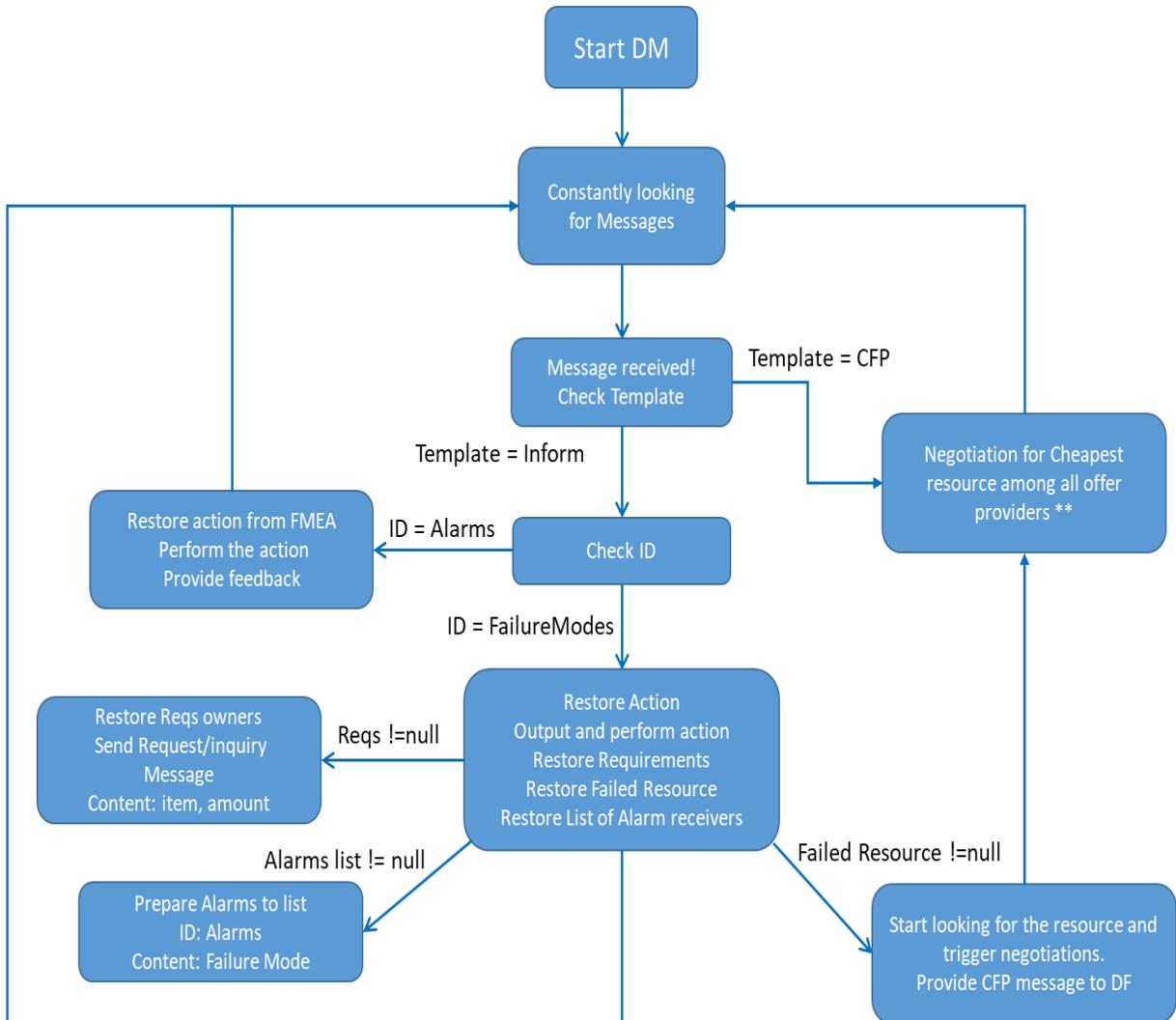


Figure 45. Decision Maker agent's behaviour

When negotiation is required, in our case, the decision maker looks into directory facilitator (DF) agent which provides the yellow page service to see if any service (in this case free resource of the same specification to that of the failed one) is available for use. After finding the list of service providers negotiation can start based on various criteria such as finding the lowest cost, fastest service, or quality, or using more sophisticated algorithms to find an optimum solution in a multi-criteria decision-making (MCDA) context, where there might various and possibly

conflicting criteria of which an optimum answer is desired and aimed. In the next case negotiation will be taken into more detailed account.

In the two examples related to the conveyer system the activities of the decision maker agent are shown in the figures below (figures (46 and 47)). Once the message from analyser has been received (identifying the failure that took place), activities of decision maker starts to mitigate the issue and provide solutions if there are any available.

```

DecisionMakerAgent
Decision Making agent is on
-----
Message was received from Analyzer@localhost:1099/JADE : conveyor C2 breakdown
-----
Action to be taken is: Stop the line
-----
Involved Entities Are: [inventory, maintenance]
-----
Request message has been sent to: inventory
Content: Component: The Conveyer System--- Failure Mode: conveyor C2 breakdown
Request message has been sent to: maintenance
Content: Component: The Conveyer System--- Failure Mode: conveyor C2 breakdown
-----
Requirements for fixation are: [belt, motor]
-----
Request message has been sent to: inventory
Content: [belt, motor]

```

Figure 46. Decision Maker agent output for case two, the conveyer belt failure.

```

DecisionMakerAgent
Decision Making agent is on
-----
Message was received from Analyzer@192.168.220.113:1099/JADE : Pusher Robot P1 maintainance due
-----
Action to be taken is: Stop the line
-----
Involved Entities Are: [control unit, maintenance]
-----
Request message has been sent to: control unit
Content: Component: The Conveyer System--- Failure Mode: Pusher Robot P1 maintainance due
Request message has been sent to: maintenance
Content: Component: The Conveyer System--- Failure Mode: Pusher Robot P1 maintainance due
-----
Requirements for fixation are: [null]

```

Figure 47. Decision Maker Agent activities dealing with the case pusher robot maintenance due

4.5. Case three: A machine breakdown from a shopfloor (to test the ability of negotiation over resources)

An example to demonstrate this mechanism is a job-shop unit as an intelligent component with several automated robots, along with multi-agent systems implemented for controlling the processes. The monitoring agent, through its subscribed agents that monitor various sensors data, constantly checks the behaviour of the sub-components and the process. A possible failure can be an accidental breakdown of one of the robot's arms, causing it to stop working properly. Taking the assumption that the robot was being monitored through 5 sensors namely S1, ..., S5, and an anomaly is detected in sensors S2 and S4 indicates that the arm of that robot is broken-down, the analyser identifies the issue after checking anomaly data from the monitoring agent against its library of failure modes and their associated anomaly cases. The next step would then be to measure the total severity of this failure in the system.

The arm causes the robot not to work well, and malfunctioning of the robot may bring delay to production process for the unit in which it is performing. Also, in its collaboration with other entities this failure may cause damage to components next to it. Based on the severity and probability of each of these cases, the total severity would be calculated as the sum of the product of the probability of each case and its severity if that case occurs. The next step would be to trigger the decision-making process. The Analyzer sends the failure ID and the severity to decision-making agent which sends a request to the assessment agent and database to retrieve the required data for releasing alarms to right entities (e.g. higher level component, other robots to stop if necessary, and so on), performing countermeasure/fixation (e.g. requirements and other components that provide them), in this example parts are needed to change the broken arm, inventory units that have them, maintenance teams, and perhaps transportation units. If several options are available, the decision maker choses the most favourable one (i.e. cheapest, fastest, etc. depending on the priorities).

The next step is to find a substitute (robot/ a unit that has the same robot or can perform the same task). A request as a call for proposal (CFP) will be sent to get offers among available components with desirable resources to perform the task. When found, the decision-making agent will start negotiations with those components to make decisions on the best deal seen its priorities. After locating the alternative resource, ways of transportation (e.g. conveyors, AVGs, trucks, etc.), will be chosen to send the parts (that had to be processed by the broken robot) to the alternative resource. A Feedback of this process will be sent back for updating the model.

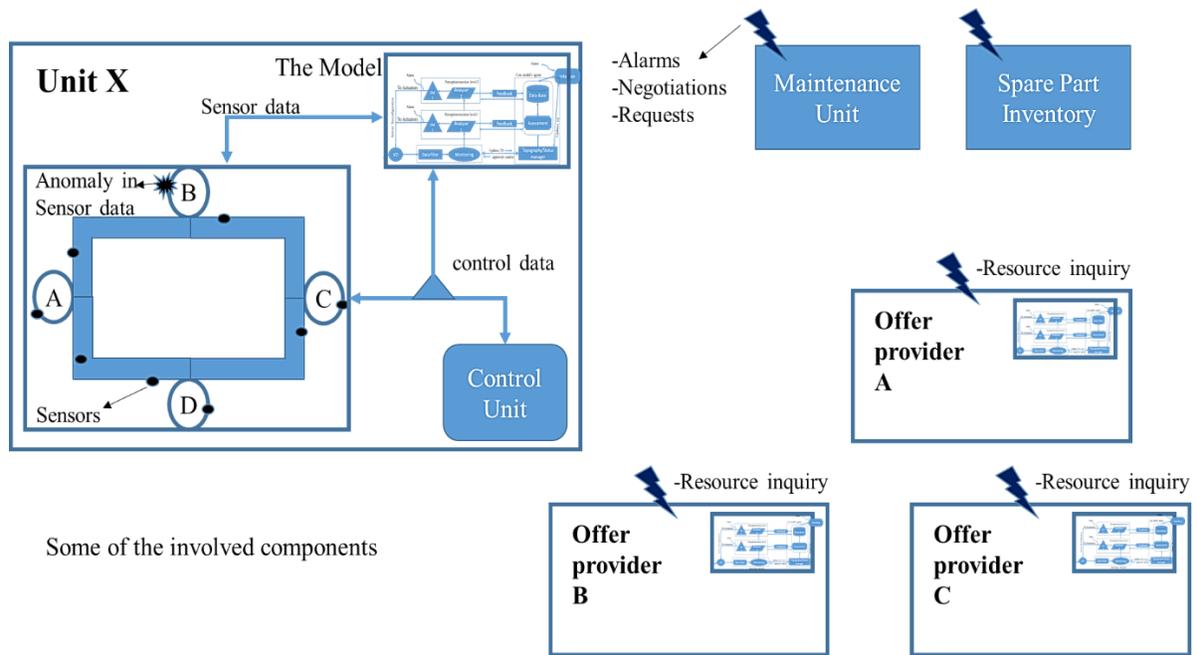
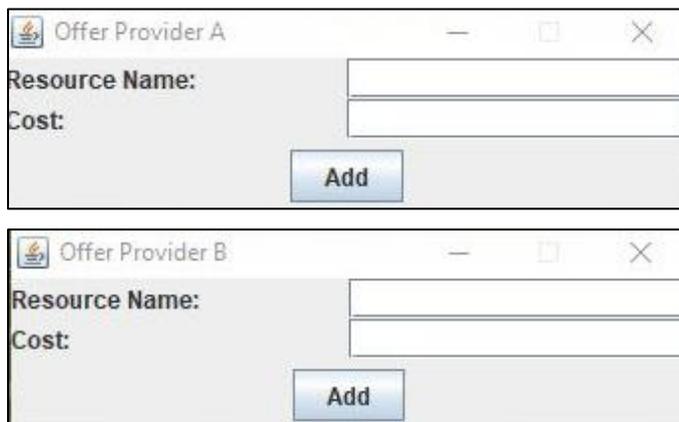


Figure 48. Hypothetical example for negotiation case

To perform the negotiations after detections and identification an extra agent was developed as “offer simulator” through with desired number of offer providers could be simulated. For this example, three offer providers were created as shown in the figure (49) that are namely: Offer Provider A, B, and C. These providers will register their service to the directory facilitator agent where yellow page service is being provided. For this case, offers are only compared based on the cost of substitution offered by the provider and the decision is to be made based on the lowest cost.



Offer Provider C

Resource Name:

Cost:

Add

Figure 49. Offer Providers A, B, and C

The prices offered by these offer providers are set to be respectively 640, 550, and 780 unit. The following figure shows the activities of the decision maker agent when receiving the message from analyser agent and its final decision and negotiation results. Furthermore, figure (51) demonstrates the process of negotiation between the decision maker and offer providers to pick the provider with the best price among all offers. The diagram is provided via sniffer tool available in JADE platform.

DecisionMakerAgent

Decision Making agent is on

Message was received from Analyzer@141.44.235.91:1099/JADE : Robot B Breakdown]

Action to be taken is: Stop the line

Involved Entities Are: [inventory, maintenance]

Request message has been sent to: inventory
Content: Component: The Conveyer System— Failure Mode: Robot B Breakdown

Request message has been sent to: maintenance
Content: Component: The Conveyer System— Failure Mode: Robot B Breakdown

Requirements for fixation are: [robot b arm, motor]

Request message has been sent to: inventory
Content: [robot b arm, motor]

Failed resource is: machine B

— Negotiation Results —
Available offerces were: [640, 550, 780]
machine B is to be substituted from:
Offer Provider B@141.44.235.91:1099/JADE
price = 550

Figure 50. decision maker agent's activities for the case three.

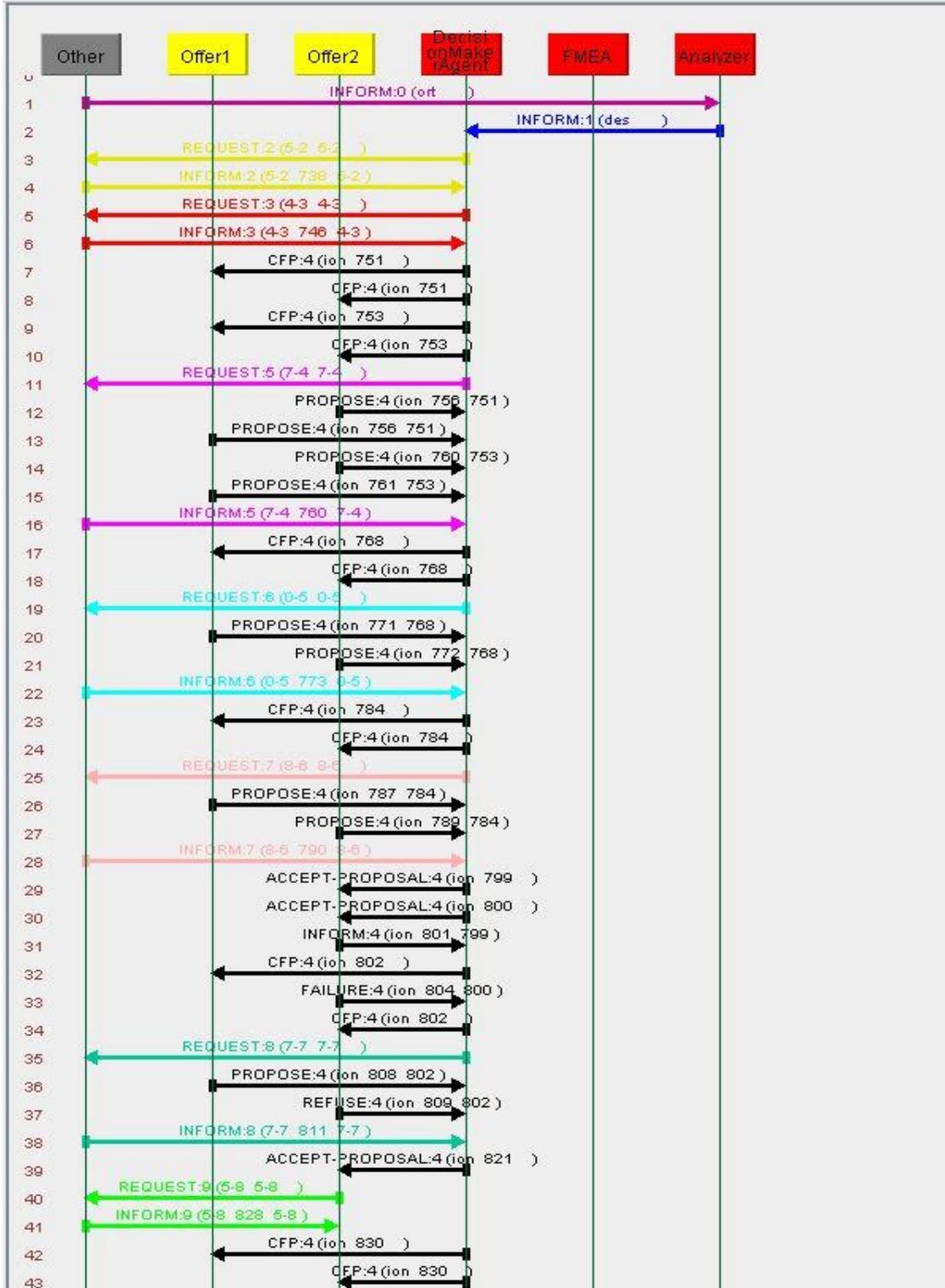


Figure 51. Negotiation over a failed resource

CHAPTER FIVE

CONCLUSION AND DISCUSSION

5.1. Conclusion

The convergence of ICTs technologies and industrial practices in recent decades has led to tremendous advancements and innovations in industries including manufacturing industries. The last movement that is pushing the boundaries in bringing smartness and autonomy into the realm of machines and industrial systems. In this path, the engagement of Cyber-Physical Systems, networks of loosely coupled smart semi/fully autonomous communicating entities, and considerably higher exposure to cyber space and internet requires industrial organizations to equip their systems with sufficiently capable security and dependability mechanisms to assure smooth and seamless performance within such dynamic and open context. Lack of a mechanism to catch up with such changing environment, and application and behavioural flexibility has created concerns and posed a hindrance to fully enjoy the intrinsic and potential capabilities of new industrial and manufacturing era.

This thesis has accordingly aimed at addressing this issue by developing a mechanism which is capable of handling dependability and security problems of systems irrespective of their flexibility and changing behaviours, their dispersity and distributed nature, their heterogeneity, their level of exposure, etc. In other words, a mechanism which inherits and is capable of demonstrating the behaviours and properties of smart systems to possibly fit best with the characteristics and needs of them. This idea, which was driven from the lack a generic approach in addressing the abovementioned issues, led to the research question of this thesis.

Question: *Can an agent-based intelligent distributed dependability and security supervision and control mechanism that inherits smart system properties enhance the overall dependability and security of the system?*

Following the question four hypotheses were presented to for implementing and testing mechanism to reach an answer for the research question. First step was to develop the model

and defining the properties it had to show and therefore as an instance of the decision cycle (explain in chapter three), the model and the concept was formed, and subsequently the contribution of each properties in every functional behaviour of the mechanism was determined. The established model could enable the mechanism to encompass the entire system as it was capable being broken down and applied into all levels of details within a given system where each smart component can be constituted from or be a constituent of other smart components and therefore the model needed both vertical and horizontal integration.

Subsequent to the model development was an implementation method capable of delivering the model's requirements. Hence, multi-agent systems were used as a perfect candidate to provide the solution to implement the model and develop the mechanism. This tool was opted given all its capabilities which made them fit perfectly for this mechanism's needs. Accordingly, using DACS methodology agents to carry out the decisions and responsibilities, as well as their communications were identified and defined, and the agent-based structure of the dependability and security mechanism was developed.

The next hypothesis points out the capability of the mechanism to be applied irrespective of the heterogeneity of components and the changing scale of the components, or the approaches summoned to establish each module of the mechanism. That suggests from the starting point, any component of heterogeneous type and variety of scale can be added or removed and the rest of the mechanism will perform without a need for major changes. The identification approach used by analyser can be flexibly scaled or chosen based on the requirements and policies at the developer's choice. Database development and risk models to be used for classification of threats and failures can be flexibly chosen based on the system's needs, and last but not least is the decision-making process which again can adopt decision-making approaches appropriate to the policies or economical approaches of the system under consideration, the scale of the system, objectives of the system or the developer's preferences. Ergo, this hypothesis too can be proven given the modular nature of the mechanism makes it sufficiently flexible to be used in any context and not just in manufacturing enterprises.

As it has been shown and discussed, through the collaboration of intelligent agents, the processes of inspection and reaction, from monitoring, detecting, identification, to making decisions and performing necessary reactions are done autonomously and constantly through the methods and behaviours used in designing these agents.

In the end, by fulfilling the requirements to prove proposed hypotheses, the answer to the research question is positive since the mechanism can afford given systems various enhancements listed as follow:

- A) Distributed components can carry out their own dependability and security supervision and control constantly and autonomously in a self-optimizing manner through a feedback loop to update the database.
- B) The mechanism is flexible and responsive towards changes, which means it can be easily scaled without the need to apply major changes. That means system can grow in size or shrink, and irrespective of the type of components added or removed the mechanism can also flexibly be scaled.
- C) The mechanism works as the collaboration of autonomous entities and modules, and the approaches in developing the behaviours of each module does not affect the mechanisms overall behaviour or other modules' behaviours. Each module can work as a black box which suggests its internal algorithms are not of other modules' concern.
- D) The mechanism can bring together various techniques for different cases of failures which empowers it in dealing with elaborate components made of heterogenous sub-components and constituent.

5.2. Outlook

In this section, a number of suggestions for improving and expanding the model will be presented. As noted before, various parts of the model are designed in a modular way independent of other modules. That means, different monitoring agents which work independently checking completely different parameters and looking for anomalies in data patterns from different sources of data, detected anomalies can be sent to any analyser designed through various preferred algorithms for the type of system or in accordance to dependability and security objectives. Here, data mining methods can help with improving the capability of monitoring agents and analysers to find anomalies in data pattern or to predict possible or likely future failures to act in a preventive manner as well as reactive.

Also, more sophisticated algorithms than the ones used in the cases shown here for decision making and negotiations. Various methods like Multi-Criteria Decision Analysis (MCDA) or Multi-Attribute Decision Making (MADM) can support and aid decisions maker agents with reaching to

solutions closer to globally optimum answers for complex cases in real world where conflicting criteria such as cost and time, etc. may be involved. Furthermore, methods for agents' collaboration in finding failures associated with uncategorized anomalies in a network of interacting components can be studied further as well as methods for decision making agents' collaborations of various entities for finding solutions over complex failure case where decision maker agent of the compromised component is not capable of finding solution on solving the issue due to lack of sufficient data, or resources to act upon the occurred failure.

REFERENCES

- [1] DIN/DKE/VDE, "GERMAN STANDARDIZATION ROADMAP: Industry 4.0," DIN, Germany2016.
- [2] Germany Trade And Invest, "INDUSTRIE 4.0: smart manufacturing for the future."
- [3] BMBF, "Zukunftsbild „Industrie 4.0“,“ Bundesministerium für Bildung und Forschung, Germany2013.
- [4] B. Bagheri, S. Yang, H.-A. Kao, and J. Lee, "Cyber-physical Systems Architecture for Self-Aware Machines in Industry 4.0 Environment," *IFAC-PapersOnLine*, vol. 48, pp. 1622-1627, 2015.
- [5] S. Heng, "Industry 4.0: Huge potential for value creation waiting to be tapped," Deutsche Bank Research, Germany2014.
- [6] Hermann Kuhnle and Hessamedin Bayanifar, "Enhancing Dependabilities of Smart Units in Industry 4.0 Contexts: A Multiagent System Approach," *International Journal of Advances in Science, Engineering and Technology(IJASEAT)*, vol. 5, pp. 38-44, 2017.
- [7] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11-33, 2004.
- [8] "E-GOVERNMENT ACT OF 2002," ed. United State of America, 2002, p. 72.
- [9] H. Kühnle and G. Bitsch, "Smart Manufacturing Units," pp. 55-70, 2015.
- [10] "Flexibility," in *CIRP Encyclopedia of Production Engineering*, L. Laperrière and G. Reinhart, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 524-524.
- [11] M. Colledani, "Statistical Process Control," in *CIRP Encyclopedia of Production Engineering*, L. Laperrière and G. Reinhart, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 1150-1157.
- [12] International Electrotechnical Commission, "IEC 60300-3-1:2003 " in *Dependability management - Part 3-1: Application guide - Analysis techniques for dependability - Guide on methodology*, ed, 2003, p. 126.
- [13] J.-P. Vasseur and A. Dunkels, "Chapter 1 - What Are Smart Objects?," in *Interconnecting Smart Objects with IP*, ed Boston: Morgan Kaufmann, 2010, pp. 3-20.
- [14] EPoSS – The European Technology Platform on Smart Systems Integration, "ETP Self assessment," 2013.
- [15] Edward A. Lee, "The Past, Present and Future of Cyber-Physical Systems: A Focus on Models," *Sensors*, vol. 15, p. 33, 2015.
- [16] Norbert Wiener, *Cybernetics: Or Control and Communication in the Animal and the Machine*: MIT Press, 1948.

-
- [17] S. A. S. Edward Ashford Lee, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*, second edition ed., 2015.
- [18] Networking and Information Technology Research and Development (NITRD), "Cyber Physical Systems (CPS SSG)," 2015.
- [19] J. Tissier and J.-Y. Grosclaude, "What About Climate-Smart Agriculture?," in *Climate Change and Agriculture Worldwide*, E. Torquebiau, Ed., ed Dordrecht: Springer Netherlands, 2016, pp. 313-324.
- [20] A. Rabadiya Kinjal, B. Shivangi Patel, and C. Chintan Bhatt, "Smart Irrigation: Towards Next Generation Agriculture," in *Internet of Things and Big Data Analytics Toward Next-Generation Intelligence*, N. Dey, A. E. Hassanien, C. Bhatt, A. S. Ashour, and S. C. Satapathy, Eds., ed Cham: Springer International Publishing, 2018, pp. 265-282.
- [21] C. Goumopoulos, "An Autonomous Wireless Sensor/Actuator Network for Precision Irrigation in Greenhouses," in *Smart Sensing Technology for Agriculture and Environmental Monitoring*, S. C. Mukhopadhyay, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1-20.
- [22] A. Sieber, A. Nafari, R. Konrad, P. Enoksson, and M. Wagner, "Wireless Platform for Monitoring of Physiological Parameters of Cattle," in *Smart Sensing Technology for Agriculture and Environmental Monitoring*, S. C. Mukhopadhyay, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 135-156.
- [23] S. Das, "Technology for SMART HOME," in *Proceedings of International Conference on VLSI, Communication, Advanced Devices, Signals & Systems and Networking (VCASAN-2013)*, V. S. Chakravarthi, Y. J. M. Shirur, and R. Prasad, Eds., ed India: Springer India, 2013, pp. 7-12.
- [24] S. J. Lee, H. C. Kim, S. M. Ko, and Y. G. Ji, "A Study of Properties and Services of a Smart Home for the Elderly," in *HCI International 2013 - Posters' Extended Abstracts: International Conference, HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part II*, C. Stephanidis, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 656-660.
- [25] P. Stieninger, "The Smart Building in the Smart City," in *Integration of Nature and Technology for Smart Cities*, P. E. R. L. B. D. C. C. A. Ahuja, Ed., ed Cham: Springer International Publishing, 2016, pp. 333-349.
- [26] N. Bansal, M. Mukherjee, and A. Gairola, "Smart Cities and Disaster Resilience," in *From Poverty, Inequality to Smart City: Proceedings of the National Conference on Sustainable Built Environment 2015*, F. Seta, J. Sen, A. Biswas, and A. Khare, Eds., ed Singapore: Springer Singapore, 2017, pp. 109-122.
- [27] G. L. Foresti, M. Farinosi, and M. Vernier, "Situational awareness in smart environments: socio-mobile and sensor data fusion for emergency response to disasters," *Journal of Ambient Intelligence and Humanized Computing*, vol. 6, pp. 239-257, 2015/04/01 2015.
-

-
- [28] I. Friedberg, K. McLaughlin, and P. Smith, "Towards a Cyber-Physical Resilience Framework for Smart Grids," in *Intelligent Mechanisms for Network Configuration and Security: 9th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security, AIMS 2015, Ghent, Belgium, June 22-25, 2015. Proceedings*, S. Latré, M. Charalambides, J. François, C. Schmitt, and B. Stiller, Eds., ed Cham: Springer International Publishing, 2015, pp. 140-144.
- [29] M. U. Tariq, S. Grijalva, and M. Wolf, "A Service-Oriented, Cyber-Physical Reference Model for Smart Grid," in *Cyber Physical Systems Approach to Smart Electric Power Grid*, K. S. Khaitan, D. J. McCalley, and C. C. Liu, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 25-42.
- [30] *Smart Power Systems and Renewable Energy System Integration*: Springer, Cham, 2016.
- [31] I. Akkaya, Y. Liu, and E. A. Lee, "Modeling and Simulation of Network Aspects for Distributed Cyber-Physical Energy Systems," in *Cyber Physical Systems Approach to Smart Electric Power Grid*, S. K. Khaitan, J. D. McCalley, and C. C. Liu, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 1-23.
- [32] B. B. Gupta and T. Akhtar, "A survey on smart power grid: frameworks, tools, security issues, and solutions," *Annals of Telecommunications*, vol. 72, pp. 517-549, 2017/10/01 2017.
- [33] D. Sonntag, S. Zillner, C. Schulz, M. Weber, and T. Toyama, "Towards Medical Cyber-Physical Systems: Multimodal Augmented Reality for Doctors and Knowledge Discovery about Patients," in *Design, User Experience, and Usability. User Experience in Novel Technological Environments: Second International Conference, DUXU 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part III*, A. Marcus, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 401-410.
- [34] A. Ariani and S. Soegijoko, "The Development of Cyber-Physical System in Health Care Industry," in *Computational Intelligence for Decision Support in Cyber-Physical Systems*, H. Z. Khan, S. A. B. M. Ali, and Z. Riaz, Eds., ed Singapore: Springer Singapore, 2014, pp. 107-148.
- [35] M. I. Pramanik, R. Y. K. Lau, H. Demirkan, and M. A. K. Azad, "Smart health: Big data enabled health paradigm within smart cities," *Expert Systems with Applications*, vol. 87, pp. 370-383, 2017/11/30/ 2017.
- [36] International Telecommunication Union, "Overview of smart sustainable cities infrastructure," 2015.
- [37] L. G. Anthopoulos, "The Smart City in Practice," in *Understanding Smart Cities: A Tool for Smart Government or an Industrial Trick?*, L. G. Anthopoulos, Ed., ed Cham: Springer International Publishing, 2017, pp. 47-185.
- [38] ZVEI, "Referenzarchitekturmodell Industrie 4.0 (RAMI 4.0)," ed, 2016.
- [39] Bitkom / VDMA / ZVE, "Implementation Strategy Industrie 4.0," 2016.
-

-
- [40] D. P. F. Möller, "Digital Manufacturing/Industry 4.0," in *Guide to Computing Fundamentals in Cyber-Physical Systems: Concepts, Design Methods, and Applications*, ed Cham: Springer International Publishing, 2016, pp. 307-375.
- [41] J. Lee, B. Bagheri, and H.-A. Kao, "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18-23, 2015.
- [42] H. Kühnle, "Distributed Manufacturing (DM) - Smart Units and Collaborative Processes," *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering*, vol. 9, pp. 1229-1241, 5/ 2015.
- [43] G. B. Hermann Kühnle, *Foundations & Principles of Distributed Manufacturing: Elements of Manufacturing Networks, Cyber-Physical Production Systems and Smart Automation*: Springer International Publishing, 2015.
- [44] H. Kuehnle, "Smart units in Distributed Manufacturing (DM)–Key properties and upcoming abilities," in *18th Annual Cambridge International Manufacturing Symposium*, 2014.
- [45] H. Kühnle, "Distributed Manufacturing: Paradigms, Concepts, Solutions and Examples," in *Distributed Manufacturing: Paradigm, Concepts, Solutions and Examples*, H. Kühnle, Ed., ed London: Springer London, 2010, pp. 1-9.
- [46] Alasdair Gilchrist, *Industry 4.0: The Industrial Internet of Things*: Apress, 2016.
- [47] VDI/VDE, "Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation," 2013.
- [48] P. Tomar, G. Kaur, and P. Singh, "A Prototype of IoT-Based Real Time Smart Street Parking System for Smart Cities," in *Internet of Things and Big Data Analytics Toward Next-Generation Intelligence*, N. Dey, A. E. Hassanien, C. Bhatt, A. S. Ashour, and S. C. Satapathy, Eds., ed Cham: Springer International Publishing, 2018, pp. 243-263.
- [49] M.-F. Tsai, Y. C. Kiong, and A. Sinn, "Smart service relying on Internet of Things technology in parking systems," *The Journal of Supercomputing*, 2016/09/19 2016.
- [50] Z. Xiong, H. Sheng, W. Rong, and D. E. Cooper, "Intelligent transportation systems for smart cities: a progress review," *Science China Information Sciences*, vol. 55, pp. 2908-2914, 2012/12/01 2012.
- [51] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, *et al.*, "Cyber-physical systems in manufacturing," *CIRP Annals - Manufacturing Technology*, 2016.
- [52] A. Gilchrist, "Introducing Industry 4.0," in *Industry 4.0: The Industrial Internet of Things*, ed Berkeley, CA: Apress, 2016, pp. 195-215.
- [53] B. V.-H. a. D. Hess, "Guest Editorial Industry 4.0; Prerequisites and Visions," *IEEE Transactions on Automation Science and Engineering*, vol. 13, pp. 411-413, 2016.
- [54] Institute of Electrical and Electronics Engineers, "IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries," *IEEE Std 610*, pp. 1-217, 1991.
- [55] International Telecommunication Union, "Smart ubiquitous networks – Context-awareness framework," 2013.
-

-
- [56] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," 1990.
- [57] INTERNATIONAL ELECTROTECHNICAL COMMISSION, "TC65: INDUSTRIAL PROCESS MEASUREMENT AND CONTROL," in *ISO/IEC 65/290/DC*, ed, 2002.
- [58] "Changeable Manufacturing Systems," in *CIRP Encyclopedia of Production Engineering*, L. Laperrière and G. Reinhart, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 163-163.
- [59] International Electrotechnical Commission, "ISO/IEC 12207:2008; Systems and software engineering -- Software life cycle processes," ed, 2008, p. 123.
- [60] I. E. Commission, "ISO/IEC/IEEE International Standard - Systems and software engineering--Vocabulary," in *ISO/IEC/IEEE 24765:2017(E)*, ed, 2017, pp. 1-541.
- [61] *Internet of Things Based on Smart Objects*: Springer International Publishing, 2014.
- [62] R. F. Babiceanu and R. Seker, "Big Data and virtualization for manufacturing cyber-physical systems: A survey of the current status and future outlook," *Computers in Industry*, vol. 81, pp. 128-137, 2016.
- [63] H. S. Kang, J. Y. Lee, S. Choi, H. Kim, J. H. Park, J. Y. Son, *et al.*, "Smart manufacturing: Past research, present findings, and future directions," *International Journal of Precision Engineering and Manufacturing-Green Technology*, vol. 3, pp. 111-128, 2016.
- [64] O. Skarlat, M. Borkowski, and S. Schulte, "Towards a methodology and instrumentation toolset for cloud manufacturing," *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)*, pp. 1-4, 2016.
- [65] B. Esmailian, S. Behdad, and B. Wang, "The evolution and future of manufacturing: A review," *Journal of Manufacturing Systems*, vol. 39, pp. 79-100, 2016.
- [66] C. Yu, X. Xu, and Y. Lu, "Computer-Integrated Manufacturing, Cyber-Physical Systems and Cloud Manufacturing – Concepts and relationships," *Manufacturing Letters*, vol. 6, pp. 5-9, 2015.
- [67] "Agent Systems," in *CIRP Encyclopedia of Production Engineering*, L. Laperrière and G. Reinhart, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 33-33.
- [68] R. Unland, "Chapter 1 - Software Agent Systems A2 - Leitão, Paulo," in *Industrial Agents*, S. Karnouskos, Ed., ed Boston: Morgan Kaufmann, 2015, pp. 3-22.
- [69] S. Bussmann, N. R. Jennings, and M. Wooldridge, "The DACS Methodology for Production Control," in *Multiagent Systems for Manufacturing Control: A Design Methodology*, S. Bussmann, N. R. Jennings, and M. Wooldridge, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 117-186.
- [70] A. Lüder and M. Foehr, "Chapter 10 - Identification and Implementation of Agents for Factory Automation Exploiting Mechatronical Concepts for Production System Structuring A2 - Leitão, Paulo," in *Industrial Agents*, S. Karnouskos, Ed., ed Boston: Morgan Kaufmann, 2015, pp. 171-190.
-

-
- [71] P. N. R. J. Dr. Stefan Bussmann, Professor Michael Wooldridge, *Multiagent Systems for Manufacturing Control: A Design Methodology*: Springer Berlin Heidelberg, 2004.
- [72] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*: John Wiley & Sons, 2007.
- [73] H. Kühnle and H. Bayanifar, "An approach to develop an intelligent distributed Dependability and Security supervision and control for industry 4.0 systems," *Asian Journal of Information and Communications*, vol. 9, p. 8, 2017.
- [74] international telecommunication Union, "Overview of cybersecurity," 2008.
- [75] International Electrotechnical Commission, "Information technology -- Security techniques -- Information security management systems -- Overview and vocabulary," in *ISO/IEC 27000:2014*, ed, 2014.
- [76] international Electrotechnical Commission, "Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models," in *ISO/IEC 25010:2011*, ed, 2011, p. 34.
- [77] International Telecommunication Union, "Security in Telecommunications and Information Technology: An overview of issues and the deployment of existing ITU-T Recommendations for secure telecommunications," 2015.
- [78] L. Zhang, Q. Wang, and B. Tian, "Security threats and measures for the cyber-physical systems," *The Journal of China Universities of Posts and Telecommunications*, vol. 20, Supplement 1, pp. 25-29, 8// 2013.
- [79] M. E. Karim and V. V. Phoha, "Cyber-physical Systems Security," pp. 75-83, 2014.
- [80] A. Giordano, G. Spezzano, and A. Vinci, "A Smart Platform for Large-Scale Cyber-Physical Systems," in *Management of Cyber Physical Objects in the Future Internet of Things: Methods, Architectures and Applications*, A. Guerrieri, V. Loscri, A. Rovella, and G. Fortino, Eds., ed Cham: Springer International Publishing, 2016, pp. 115-134.
- [81] G. Wu, J. Sun, and J. Chen, "A survey on the security of cyber-physical systems," *Control Theory and Technology*, vol. 14, pp. 2-10, 2016.
- [82] F. O. f. I. Security, "Industrial Control System Security: Top 10 Threats and Countermeasures 2016," 2016.
- [83] Z. DeSmit, A. E. Elhabashy, L. J. Wells, and J. A. Camelio, "Cyber-physical Vulnerability Assessment in Manufacturing Systems," *Procedia Manufacturing*, vol. 5, pp. 1060-1074, // 2016.
- [84] S. Huang, C.-J. Zhou, S.-H. Yang, and Y.-Q. Qin, "Cyber-physical system security for networked industrial processes," *International Journal of Automation and Computing*, vol. 12, pp. 567-578, 2015.
- [85] G. Sabaliauskaite and A. P. Mathur, "Aligning Cyber-Physical System Safety and Security," pp. 41-53, 2015.
- [86] T. S. S. O. ITU, "Security framework for cloud computing," ed, 2015, p. 30.
-

-
- [87] S. Karnouskos, "Chapter 6 - Industrial Agents Cybersecurity," in *Industrial Agents*, ed Boston: Morgan Kaufmann, 2015, pp. 109-120.
- [88] L. J. Wells, J. A. Camelio, C. B. Williams, and J. White, "Cyber-physical security challenges in manufacturing systems," *Manufacturing Letters*, vol. 2, pp. 74-77, 4// 2014.
- [89] K. Wan and V. Alagar, "Context-Aware Security Solutions for Cyber-Physical Systems," *Mobile Networks and Applications*, vol. 19, pp. 212-226, 2014.
- [90] M. Burmester, E. Magkos, and V. Chrissikopoulos, "Modeling security in cyber-physical systems," *International Journal of Critical Infrastructure Protection*, vol. 5, pp. 118-126, 12// 2012.
- [91] H. L. Huiyingh and X. Zhou, "Study on Security Architecture for Internet of Things," in *Applied Informatics and Communication*, D. Zeng, Ed., ed: Springer Berlin Heidelberg, 2011, pp. 404-411.
- [92] R. Dorociak and J. Gausemeier, "Methods of Improving the Dependability of Self-optimizing Systems," in *Dependability of Self-Optimizing Mechatronic Systems*, J. Gausemeier, F. J. Rammig, W. Schäfer, and W. Sextro, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 37-171.
- [93] B. Vogel-Heuser and S. Rösch, "Integrated Modeling of Complex Production Automation Systems to Increase Dependability," pp. 363-385, 2014.
- [94] R. Dorociak, T. Gaukstern, J. Gausemeier, P. Iwanek, and M. Vaßholz, "A methodology for the improvement of dependability of self-optimizing systems," *Production Engineering*, vol. 7, pp. 53-67, 2012.
- [95] M. Missbach, T. Staerk, C. Gardiner, J. McCloud, R. Madl, M. Tempes, *et al.*, "Securing SAP on the Cloud," pp. 75-120, 2016.
- [96] A. H. a. J. L. a. F. L. a. B. Luo, "Cyber-Physical Systems Security -- A Survey," *IEEE Internet of Things Journal*, vol. PP, p. 30, 2017.
- [97] Q. Z. a. C. R. a. T. Başar, "A hierarchical security architecture for cyber-physical systems," *2011 4th International Symposium on Resilient Control Systems*, pp. 15-20, 2011.
- [98] F. Hu, Y. Lu, A. V. Vasilakos, Q. Hao, R. Ma, Y. Patil, *et al.*, "Robust Cyber-Physical Systems: Concept, models, and implementation," *Future Generation Computer Systems*, vol. 56, pp. 449-475, 2016.
- [99] International Electrotechnical Commission, "Dependability management – Part 3-1: Application guide – Analysis techniques for dependability – Guide on methodology," in *IEC 60300-3-1*, ed, 2003.
- [100] I. O. f. S. International Electrotechnical Commission, "Information technology -- Security techniques -- Information security management systems -- Overview and vocabulary," in *ISO/IEC 27000:2016*, ed, 2016, p. 34.
- [101] International Electrotechnical Commission, "Hazard and operability studies (HAZOP studies) - Application guide," in *IEC 61882:2016*, ed, 2016.
-

-
- [102] International Electrotechnical Commission, "Application of Markov techniques," in *IEC 61165:2006*, ed, 2006, p. 67.
- [103] international Electrotechnical Commission, "Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA)," in *IEC 60812:2006*, ed, 2006, p. 93.
- [104] International Electrotechnical Commission, "Fault tree analysis (FTA)," in *IEC 61025:2006*, ed, 2006, p. 103.
- [105] T. Sanislav, G. Mois, and L. Miclea, "An approach to model dependability of cyber-physical systems," *Microprocessors and Microsystems*, vol. 41, pp. 67-76, 2016.
- [106] C. A. Boano, K. Römer, R. Bloem, K. Witrisal, M. Baunach, and M. Horn, "Dependability for the Internet of Things—from dependable networking in harsh environments to a holistic view on dependability," *e & i Elektrotechnik und Informationstechnik*, vol. 133, pp. 304-309, 2016/11/01 2016.
- [107] B. P. Douglass, "Chapter 6 - Dependability Architecture," in *Real-Time UML Workshop for Embedded Systems (Second Edition)*, ed Oxford: Newnes, 2014, pp. 149-177.
- [108] I. Silva, R. Leandro, D. Macedo, and L. A. Guedes, "A dependability evaluation tool for the Internet of Things," *Computers & Electrical Engineering*, vol. 39, pp. 2005-2018, 2013/10/01/ 2013.
- [109] P. Barger, J.-M. Thiriet, M. Robert, and J.-F. Aubry, "Dependability Study in Distributed Control Systems Integrating Smart Devices," *IFAC Proceedings Volumes*, vol. 37, pp. 67-72, 2004/06/01/ 2004.
- [110] C. W. Axelrod, "Managing the risks of cyber-physical systems," *2013 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1-6, 2013.
- [111] C. Schmittner, T. Gruber, P. Puschner, and E. Schoitsch, "Security Application of Failure Mode and Effect Analysis (FMEA)," in *Computer Safety, Reliability, and Security: 33rd International Conference, SAFECOMP 2014, Florence, Italy, September 10-12, 2014. Proceedings*, A. Bondavalli and F. Di Giandomenico, Eds., ed Cham: Springer International Publishing, 2014, pp. 310-325.
- [112] R. F. Stapelberg, *Handbook of Reliability, Availability, Maintainability and Safety in Engineering Design*: Springer London, 2009.
- [113] D. Natarajan, "Failure Mode and Effects Analysis," in *Reliable Design of Electronic Equipment: An Engineering Guide*, D. Natarajan, Ed., ed Cham: Springer International Publishing, 2015, pp. 31-44.
- [114] W. Reisig, "An Example," in *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*, W. Reisig, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 3-12.
- [115] S. Agrawal and J. Agrawal, "Survey on Anomaly Detection using Data Mining Techniques," *Procedia Computer Science*, vol. 60, pp. 708-713, 2015/01/01/ 2015.
-

-
- [116] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, pp. 1-58, 2009.
- [117] C. Joslyn, S. Choudhury, D. Haglin, B. Howe, B. Nickless, and B. Olsen, *Massive scale cyber traffic analysis: a driver for graph database research*, 2013.
- [118] V. V. Phoha, "Internet Security Dictionary," V. V. Phoha, Ed., ed: Springer New York, 2002.
- [119] V. C. a. S. P. a. M. G. a. B. Maglaris, "Hierarchical Anomaly Detection in Distributed Large-Scale Sensor Networks," *11th IEEE Symposium on Computers and Communications (ISCC'06)*, pp. 761-767, 2006.
- [120] C. K. a. W. L. a. I. Hwang, "Security analysis for Cyber-Physical Systems against stealthy deception attacks," *2013 American Control Conference*, pp. 3344-3349, 2013.
- [121] X. Y. a. J. L. a. P. M. a. W. Y. a. X. F. a. W. Zhao, "A Novel En-route Filtering Scheme against False Data Injection Attacks in Cyber-Physical Networked Systems," *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pp. 92-101, 2012.
- [122] G. K. Baah, A. Podgurski, and M. J. Harrold, "The Probabilistic Program Dependence Graph and Its Application to Fault Diagnosis," *IEEE Trans. Softw. Eng.*, vol. 36, pp. 528-545, 2010.
- [123] S. Bernardi, J. Merseguer, and D. C. Petriu, "Dependability Analysis Techniques," in *Model-Driven Dependability Assessment of Software Systems*, S. Bernardi, J. Merseguer, and D. C. Petriu, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 73-90.
- [124] L. Ribeiro, J. Barata, G. Cândido, and M. Onori, "Evolvable Production Systems: An Integrated View on Recent Developments," in *Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology*, G. Q. Huang, K. L. Mak, and P. G. Maropoulos, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 841-854.
- [125] N. A. Duffie and R. S. Piper, "Nonhierarchical control of manufacturing systems," *Journal of Manufacturing Systems*, vol. 5, p. 141, 1986/01/01/ 1986.
- [126] *Service Orientation in Holonic and Multi-Agent Manufacturing* vol. 640: Springer International Publishing Switzerland, 2016.
- [127] I. Badr, "Integrated Scheduling for Make-to-Order Multi-factory Manufacturing: An Agent-Based Cloud-Assisted Approach," in *Service Orientation in Holonic and Multi-Agent Manufacturing*, T. Borangiu, D. Trentesaux, A. Thomas, and D. McFarlane, Eds., ed Cham: Springer International Publishing, 2016, pp. 277-284.
- [128] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters, "Reference architecture for holonic manufacturing systems: PROSA," *Computers in Industry*, vol. 37, pp. 255-274, 11// 1998.
- [129] P. Leitão and F. Restivo, "ADACOR: A holonic architecture for agile and adaptive manufacturing control," *Computers in Industry*, vol. 57, pp. 121-130, 2// 2006.
- [130] J. Barbosa, P. Leitão, E. Adam, and D. Trentesaux, "Dynamic self-organization in holonic multi-agent manufacturing systems: The ADACOR evolution," *Computers in Industry*, vol. 66, pp. 99-111, 1// 2015.
-

-
- [131] D. Trentesaux and P. Millot, "A Human-Centred Design to Break the Myth of the "Magic Human" in Intelligent Manufacturing Systems," in *Service Orientation in Holonic and Multi-Agent Manufacturing*, T. Borangiu, D. Trentesaux, A. Thomas, and D. McFarlane, Eds., ed Cham: Springer International Publishing, 2016, pp. 103-113.
- [132] T. S. a. P. Massotte, "Enhancement of distributed production systems through software agents," *ETFA 2001. 8th International Conference on Emerging Technologies and Factory Automation. Proceedings*, 2001.
- [133] J. P. a. A. L. a. H. Kuhnle, "The PABADIS' PROMISE architecture - a new approach for flexible manufacturing systems," *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, p. 6, 2005.
- [134] H. Kühnle and G. Bitsch, "Core Models, Principles, and Networks' Structuring," in *Foundations & Principles of Distributed Manufacturing: Elements of Manufacturing Networks, Cyber-Physical Production Systems and Smart Automation*, ed Cham: Springer International Publishing, 2015, pp. 27-53.
- [135] H. Kühnle and H. Bayanifar, "Smart Manufacturing – Expanding the Systems Approach onto Complex Networks," presented at the 13th International Conference on Industrial Engineering (IIEC 2017), Iran, 2017.
- [136] H. Kühnle, U. Bergmann, M. Heinicke, G. Wagenhaus, H. Bayanifar, I. Muhammed, *et al.*, *Lecture notes in manufacturing systems design and manufacturing process organisation - selected chapters from factory operations, factory planning, manufacturing enterprise organisation & cyber physical production*: Göttingen Cuvillier Verlag, 2017.
- [137] F. Z. a. K. S. a. W. W. a. V. Mooney, "Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems," *2008 Real-Time Systems Symposium*, pp. 47-56, 2008.
- [138] Z. Wei, M. S. Branicky, and S. M. Phillips, "Stability of networked control systems," *IEEE Control Systems*, vol. 21, pp. 84-99, 2001.
- [139] A. C. Lüder, Ambra. Zawisza, Jacek. Rosendahl, Ronald, "Design Pattern for Agent Based Production System Control – a Survey (I)," in *13th Conference on Automation Science and Engineering*, China, 2017, p. 16.
- [140] S. O. Duffuaa and A. Raouf, "Preventive Maintenance, Concepts, Modeling, and Analysis," in *Planning and Control of Maintenance Systems: Modelling and Analysis*, S. O. Duffuaa and A. Raouf, Eds., ed Cham: Springer International Publishing, 2015, pp. 57-94.
- [141] D. F. Percy, "Preventive Maintenance Models for Complex Systems," in *Complex System Maintenance Handbook*, K. A. H. Kobbacy and D. N. P. Murthy, Eds., ed London: Springer London, 2008, pp. 179-207.

APPENDIX (CODES FOR AGENTS)

Agent: FailureModes

```
package multiAgentSystems;

import jade.content.Concept;

public class FailureModes implements Concept {

    private static final long serialVersionUID = 1L;

    public String name;
    public String identification;
    public String requirements;
    public String action;
    public String resourceNeeded;
    public String entitiesToCall;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getIdentification() {
        return identification;
    }
    public void setIdentification(String identification) {
        this.identification = identification;
    }
    public String getRequirements() {
        return requirements;
    }
    public void setRequirements(String requirements) {
        this.requirements = requirements;
    }
    public String getAction() {
        return action;
    }
    public void setAction(String action) {
        this.action = action;
    }
    public String getResourceNeeded() {
        return resourceNeeded;
    }
    public void setResourceNeeded(String resourceNeeded) {
        this.resourceNeeded = resourceNeeded;
    }
    public String getEntitiesToCall() {
        return entitiesToCall;
    }
    public void setEntitiesToCall(String entitiesToCall) {
        this.entitiesToCall = entitiesToCall;
    }
}
}
```

Agent: FailureModesOntology

```

package multiAgentSystems;

import jade.content.onto.BasicOntology;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.schema.ConceptSchema;
import jade.content.schema.ObjectSchema;
import jade.content.schema.PrimitiveSchema;

public class FailureModesOntology extends Ontology {

    public static final String OntologyName = "FailureModesOntology";

    public static final String Failure = "failure";
    public static final String Failure_Name = "name";
    public static final String Failure_Identification = "identification";

    public static final String Failure_Actions = "actions";
    public static final String Failure_Requirements = "requirements";

    // The singleton instance of this ontology
    private static Ontology theInstance = new FailureModesOntology();
    // Retrieve the singleton failuremodes ontology instance
    public static Ontology getInstance() {
        return theInstance;
    }

    // Private constructor
    private FailureModesOntology() {
        // The FailureModes ontology extends the basic ontology
        super(OntologyName, BasicOntology.getInstance());
        try {
            add(new ConceptSchema(Failure), FailureModes.class);

            // Structure of the schema for the Failure concept
            ConceptSchema cs = (ConceptSchema) getSchema(Failure);

            cs.add(Failure_Name, (PrimitiveSchema) getSchema(BasicOntology.STRING));

            cs.add(Failure_Identification, (PrimitiveSchema) getSchema(BasicOntology.STRING), 0,
                ObjectSchema.UNLIMITED);

            cs.add(Failure_Requirements, (PrimitiveSchema)
                getSchema(BasicOntology.STRING), 0, ObjectSchema.UNLIMITED);

        }

        catch (OntologyException oe) {
            oe.printStackTrace();
        }
    }

    private static final long serialVersionUID = 1L;
}

```

Agent: FailureModes (fm1)

```

package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm1 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm1() {
        // failure mode
        this.name = "convayor belt C1 delay";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();
        String a2 = new FailureModes().getIdentification();

        a1 = "sensor p1";
        a2 = "delay";

        anomalies.add(a1);
        anomalies.add(a2);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Call maintenance";

        // adding requirements to the list
        java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
        String r1 = new FailureModes().getRequirements();
        String r2 = new FailureModes().getRequirements();

        r1 = "Belt";
        r2 = "motor";

        requiredParts.add(r1);
        requiredParts.add(r2);

        // Resource to look for
        this.resourceNeeded = null;

        /**
         sending data to include each of this failure's
         properties in the list to be used by other agents
         */

        Fmea.reqList.put(this.name, requiredParts);
        Fmea.names.add(this.name);
        Fmea.actionList.put(this.name, this.action);
        Fmea.failure_Resource.put(this.name, this.resourceNeeded);
    }
}

```

Agent: FailureModes (fm2)

```
package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm2 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm2() {

        // failure mode
        this.name = "convayor belt C2 delay";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();
        String a2 = new FailureModes().getIdentification();

        a1 = "sensor p2";
        a2 = "delay";

        anomalies.add(a1);
        anomalies.add(a2);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Call maintenance";
        // adding requirements to the list
        java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
        String r1 = new FailureModes().getRequirements();
        String r2 = new FailureModes().getRequirements();

        r1 = "Belt";
        r2 = "motor";

        requiredParts.add(r1);
        requiredParts.add(r2);

        // Resource to look for

        this.resourceNeeded = null;

        /**
         sending data to include each of this failure's
         properties in the list to be used by other agents
         */

        Fmea.reqList.put(this.name, requiredParts);
        Fmea.names.add(this.name);
        Fmea.actionList.put(this.name, this.action);
        Fmea.failure_Resource.put(this.name, this.resourceNeeded);

    }
}
```

Agent: FailureModes (fm3)

```

package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm3 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm3() {
        // failure mode
        this.name = "conveyor C1 breakdown";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();
        String a2 = new FailureModes().getIdentification();
        String a3 = new FailureModes().getIdentification();

        a1 = "sensor p1";
        a2 = "sensor T1";
        a3 = "TV"; // time violation

        anomalies.add(a1);
        anomalies.add(a2);
        anomalies.add(a3);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Stop the line"; // add the rest in a hash map

        // adding requirements to the list
        java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
        String r1 = new FailureModes().getRequirements();
        String r2 = new FailureModes().getRequirements();

        r1 = "belt";
        r2 = "motor";

        requiredParts.add(r1);
        requiredParts.add(r2);

        // Resource to look for
        this.resourceNeeded = null;

        /**
         sending data to include each of this failure's
         properties in the list to be used by other agents
         */

        Fmea.reqList.put(this.name, requiredParts);
        Fmea.names.add(this.name);
        Fmea.actionList.put(this.name, this.action);
        Fmea.failure_Resource.put(this.name, this.resourceNeeded);
    }
}

```

Agent: FailureModes (fm4)

```
package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm4 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm4() {
        // failure mode
        this.name = "conveyor C2 breakdown";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();
        String a2 = new FailureModes().getIdentification();
        String a3 = new FailureModes().getIdentification();

        a1 = "sensor p2";
        a2 = "sensor T2";
        a3 = "TV"; // time violation

        anomalies.add(a1);
        anomalies.add(a2);
        anomalies.add(a3);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Stop the line"; // add the rest in a hash map

        // adding requirements to the list
        java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
        String r1 = new FailureModes().getRequirements();
        String r2 = new FailureModes().getRequirements();

        r1 = "belt";
        r2 = "motor";

        requiredParts.add(r1);
        requiredParts.add(r2);

        // adding entities to the list
        java.util.ArrayList<String> invEnt = new
java.util.ArrayList<String>();
        String i1 = new FailureModes().getEntitiesToCall();
        String i2 = new FailureModes().getEntitiesToCall();

        i1 = "inventory";
        i2 = "maintenance";

        invEnt.add(i1);
        invEnt.add(i2);

        // Resource to look for
```

```

        this.resourceNeeded = null;

        /**
         sending data to include each of this failure's
         properties in the list to be used by other agents
         */

        Fmea.reqList.put(this.name, requiredParts);
        Fmea.names.add(this.name);
        Fmea.actionList.put(this.name, this.action);
        Fmea.failure_Resource.put(this.name, this.resourceNeeded);
        Fmea.involvedentities.put(this.name, invEnt);

    }
}

```

Agent: FailureModes (fm5)

```

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm5 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm5() {
        // failure mode
        this.name = "Pusher Robot P1 stopped working";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();
        String a2 = new FailureModes().getIdentification();

        a1 = "sensor T1";
        a2 = "TV";

        anomalies.add(a1);
        anomalies.add(a2);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Stop the line";

        // adding requirements to the list
        java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
        String r1 = new FailureModes().getRequirements();
        String r2 = new FailureModes().getRequirements();

        r1 = "pusher arm";
        r2 = "motor";

        requiredParts.add(r1);
    }
}

```

```

        requiredParts.add(r2);

        // Resource to look for

        this.resourceNeeded = null;

        /**
         * sending data to include each of this failure's
         * properties in the list to be used by other agents
         */

        Fmea.reqList.put(this.name, requiredParts);
        Fmea.names.add(this.name);
        Fmea.actionList.put(this.name, this.action);
        Fmea.failure_Resource.put(this.name, this.resourceNeeded);
    }
}

```

Agent: FailureModes (fm6)

```

package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm6 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm6() {

        // failure mode
        this.name = "Pusher Robot P2 stopped working";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();
        String a2 = new FailureModes().getIdentification();

        a1 = "sensor T2";
        a2 = "TV";

        anomalies.add(a1);
        anomalies.add(a2);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Stop the line";
        // adding requirements to the list
        java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
        String r1 = new FailureModes().getRequirements();
        String r2 = new FailureModes().getRequirements();
    }
}

```

```

    r1 = "pusher arm";
    r2 = "motor";

    requiredParts.add(r1);
    requiredParts.add(r2);

    // Resource to look for

    this.resourceNeeded = null;

    /**
     * sending data to include each of this failure's
     * properties in the list to be used by other agents
     */

    Fmea.reqList.put(this.name, requiredParts);
    Fmea.names.add(this.name);
    Fmea.actionList.put(this.name, this.action);
    Fmea.failure_Resource.put(this.name, this.resourceNeeded);
}
}

```

Agent: FailureModes (fm7)

```

package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm7 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm7() {
        // failure mode
        this.name = "Pusher Robot P1 maintenance due";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();

        a1 = "P1PN 3";

        anomalies.add(a1);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Stop the line";

        // adding requirements to the list
        java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
        String r1 = new FailureModes().getRequirements();
    }
}

```

```

r1 = null;

requiredParts.add(r1);

// adding entities to the list
java.util.ArrayList<String> invEnt = new java.util.ArrayList<String>();
    String i1 = new FailureModes().getEntitiesToCall();
    String i2 = new FailureModes().getEntitiesToCall();

    i1 = "control unit";
    i2 = "maintenance";

invEnt.add(i1);
invEnt.add(i2);

// Resource to look for

this.resourceNeeded = null;

/**
sending data to include each of this failure's
properties in the list to be used by other agents
*/

Fmea.reqList.put(this.name, requiredParts);
Fmea.names.add(this.name);
Fmea.actionList.put(this.name, this.action);
Fmea.failure_Resource.put(this.name, this.resourceNeeded);
Fmea.involvedentities.put(this.name, invEnt);
}
}

```

Agent: FailureModes (fm8)

```

package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm8 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm8() {
        // failure mode
        this.name = "Pusher Robot P2 maintainance due";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();

        a1 = "P2PN 3";

        anomalies.add(a1);

        Fmea.anomalyList.put(anomalies, this.name);
    }
}

```

```

// suggested countermeasure required
this.action = "Stop the line";

// adding requirements to the list
java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
String r1 = new FailureModes().getRequirements();

r1 = null;

requiredParts.add(r1);

// Resource to look for

this.resourceNeeded = null;

/**
sending data to include each of this failure's
properties in the list to be used by other agents
*/

Fmea.reqList.put(this.name, requiredParts);
Fmea.names.add(this.name);
Fmea.actionList.put(this.name, this.action);
Fmea.failure_Resource.put(this.name, this.resourceNeeded);

    }
}

```

Agent: FailureModes (fm9)

```

package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm9 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm9() {

        // failure mode
        this.name = "Intrusion DOS";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();

        a1 = "DTV"; //data traffic violation

        anomalies.add(a1);

        Fmea.anomalyList.put(anomalies, this.name);
    }
}

```

```

// suggested countermeasure required
this.action = "change the channel"; //add

// adding requirements to the list
java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
String r1 = new FailureModes().getRequirements();

r1 = null;

requiredParts.add(r1);

// Resource to look for

this.resourceNeeded = null;

/**
sending data to include each of this failure's
properties in the list to be used by other agents
*/

Fmea.reqlist.put(this.name, requiredParts);
Fmea.names.add(this.name);
Fmea.actionList.put(this.name, this.action);
Fmea.failure_Resource.put(this.name, this.resourceNeeded);
}
}

```

Agent: FailureModes (fm10)

```

package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm10 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm10() {
        // failure mode
        this.name = "Part is not loaded";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();

        a1 = "sensor L";

        anomalies.add(a1);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Stop the line"; // add the rest in a hash map
    }
}

```

```

// adding requirements to the list
java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
String r1 = new FailureModes().getRequirements();

r1 = null;

requiredParts.add(r1);

// Resource to look for

this.resourceNeeded = null;

/**
sending data to include each of this failure's
properties in the list to be used by other agents
*/

Fmea.reqlist.put(this.name, requiredParts);
Fmea.names.add(this.name);
Fmea.actionList.put(this.name, this.action);
Fmea.failure_Resource.put(this.name, this.resourceNeeded);

}
}

```

Agent: FailureModes (fm11)

```

package failures;

import multiAgentSystems.FailureModes;
import multiAgentSystems.Fmea;

public class Fm11 extends FailureModes {

    private static final long serialVersionUID = 1L;

    public Fm11() {
        // failure mode
        this.name = "Robot B Breakdown";
        // adding anomalies to the list for identification
        java.util.ArrayList<String> anomalies = new java.util.ArrayList<String>();
        String a1 = new FailureModes().getIdentification();
        String a2 = new FailureModes().getIdentification();

        a1 = "sensor a1";
        a2 = "sensor m1";

        anomalies.add(a1);
        anomalies.add(a2);

        Fmea.anomalyList.put(anomalies, this.name);

        // suggested countermeasure required
        this.action = "Stop the line"; // add the rest in a hash map
    }
}

```

```

// adding requirements to the list
java.util.ArrayList<String> requiredParts = new java.util.ArrayList<String>();
String r1 = new FailureModes().getRequirements();
String r2 = new FailureModes().getRequirements();

r1 = "robot b arm";
r2 = "motor";

requiredParts.add(r1);
requiredParts.add(r2);

// adding entities to the list
java.util.ArrayList<String> invEnt = new java.util.ArrayList<String>();
String i1 = new FailureModes().getEntitiesToCall();
String i2 = new FailureModes().getEntitiesToCall();

i1 = "inventory";
i2 = "maintenance";

invEnt.add(i1);
invEnt.add(i2);

// Resource to look for

this.resourceNeeded = "machine B";

/**
sending data to include each of this failure's
properties in the list to be used by other agents
*/

Fmea.reqList.put(this.name, requiredParts);
Fmea.names.add(this.name);
Fmea.actionList.put(this.name, this.action);
Fmea.failure_Resource.put(this.name, this.resourceNeeded);
Fmea.involvedentities.put(this.name, invEnt);

}
}

```

Agent: FMEA (database)

```

package multiAgentSystems;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.HashMap;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;

```

```

import javax.swing.JPanel;
import javax.swing.JTextArea;

import failures.*;
import jade.content.lang.Codec;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;

public class Fmea extends Agent {

    private Codec codec = new SLCodec();
    private Ontology ontology = FailureModesOntology.getInstance();

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    // list of the modes
    public static jade.util.leap.ArrayList names = new jade.util.leap.ArrayList();
    // anomalies and their modes in a list (for identification)
    public static HashMap<java.util.ArrayList<String>, String> anomalyList= new
HashMap<java.util.ArrayList<String>, String>();
    // list of failure modes and fixation requirements
    public static HashMap<String, ArrayList<String>> reqList = new HashMap<String,
ArrayList<String>>();
    // list of failure modes and fixation requirements
    public static HashMap<String, ArrayList<String>> involvedentities = new HashMap<String,
ArrayList<String>>();
    // list of failure modes and solution actions
    public static HashMap<String, String> actionList= new HashMap<String, String>();
    // resource to look for when a failure occur
    public static HashMap<String, String> failure_Resource= new HashMap<String, String>();
    // entities in collaborations for alarms and risk calculation
    static jade.util.leap.ArrayList collaborations = new jade.util.leap.ArrayList();

    String list;

    // Failures as subclasses to be instantiated here
    FailureModes[] fm = {new Fm1(), new Fm2(), new Fm3(),
        new Fm4(), new Fm5(), new Fm6(),
        new Fm7(), new Fm8(), new Fm9(),
        new Fm10(), new Fm11()};

    private FailureGui myGui;
    private FModes myfmealist;

    public void setup() {

        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(ontology);

        myGui = new FailureGui(this);
        myGui.showGui();

        myfmealist = new FModes(this);
        myfmealist.showGui();

```

```

System.out.println("-----");
System.out.println(names);
System.out.println(reqList);
System.out.println(anomalyList);
System.out.println(actionList);
System.out.println(failure_Resource);
System.out.println(involvedentities);
System.out.println("-----");

}

public void updateCatalogue(final String fName, final String fAction,
final String fResource, final ArrayList<String> anomalies,
final ArrayList<String> reqs) {
addBehaviour(new OneShotBehaviour() {
/**
*
*/
private static final long serialVersionUID = 1L;

public void action() {

names.add(fName);
anomalyList.put( anomalies, fName);
reqList.put(fName, reqs);
actionList.put(fName, fAction);
failure_Resource.put(fName, fResource);
System.out.println("-----");
System.out.println("New failureMode has been added: ");
System.out.println("Name: " + fName);
System.out.println("Anomalies: " + anomalies);
System.out.println("Requirements: " + reqs);
System.out.println("Resource to seek: " + fResource);
System.out.println("Action: " + fAction);
System.out.println("-----");
}
} );
}

public void removeFromCatalogue(final String fName) {
addBehaviour(new OneShotBehaviour() {
/**
*
*/
private static final long serialVersionUID = 1L;

public void action() {
names.remove(fName);
reqList.remove(fName);
actionList.remove(fName);
failure_Resource.remove(fName);

for (java.util.ArrayList<String> checkKey : anomalyList.keySet()) {

if (anomalyList.get(checkKey).equalsIgnoreCase(fName)) {
anomalyList.remove(checkKey);
}
}

System.out.println(fName + " has been removed from FMEA database ");

```

```

        } );
    }

    public void printCatalogue() {
        addBehaviour(new OneShotBehaviour() {
            /**
             *
            */
            private static final long serialVersionUID = 1L;

            public void action() {

                System.out.println("-----");
                System.out.println(names);
                System.out.println(list);
                System.out.println(reqList);
                System.out.println(anomalyList);
                System.out.println(actionList);
                System.out.println(failure_Resource);
                System.out.println("-----");
            }
        } );
    }

    class FModes extends JFrame {

        /**
         *
        */
        private static final long serialVersionUID = 1L;

        private JTextArea fminfo;
        private JLabel label;
        private JLabel label2;

        FModes(Fmea fmea) {
            super(fmea.getLocalName());

            JPanel p = new JPanel();
            p.setLayout(new GridLayout(2,1));

            label = new JLabel("  Failure Modes");
            label2 = new JLabel("-----");
            p.add(label);
            p.add(label2);

            getContentPane().add(p, BorderLayout.NORTH);

            fminfo = new JTextArea(20,20);
            p.add(fminfo);
            getContentPane().add(fminfo, BorderLayout.CENTER);

            JButton refButton = new JButton("Refresh");
            p.add(refButton);
            getContentPane().add(refButton, BorderLayout.SOUTH);
        }
    }

```

```

refButton.addActionListener( new ActionListener() {
    public void actionPerformed(ActionEvent ev) {
        try {
            fminfo.setText("");
            list = "";
            for (int i = 0; i < names.size(); i++) {
                list = list + names.toArray()[i] + "\n";
            }
            fminfo.setText(list);
        }
        catch (Exception e) {
            JOptionPane.showMessageDialog(FModes.this, "Invalid values. "+e.getMessage(), "Error",
            JOptionPane.ERROR_MESSAGE);
        }
    }
});
}

public void showGui() {
    pack();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int centerX = (int)screenSize.getWidth() / 2;
    int centerY = (int)screenSize.getHeight() / 2;
    setLocation(centerX - getWidth() / 2, centerY - getHeight() / 2);
    super.setVisible(true);
}
}

```

Agent: FMEA (interface)

```

package multiAgentSystems;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;

class FailureGui extends JFrame {
    /**
     *
     */
}

```

```

private static final long serialVersionUID = 1L;

private Fmea myAgent;
java.util.ArrayList<String> anomalies = new java.util.ArrayList<>();
java.util.ArrayList<String> reqs = new java.util.ArrayList<>();
private JTextField nameField, anomalyField, reqField, resField, actionField;
private JTextArea info;
java.util.ArrayList<String> anomaly;

FailureGui(Fmea fmea) {
    super(fmea.getLocalName());

    myAgent = fmea;

    JPanel p = new JPanel();
    p.setLayout(new GridLayout(6, 4));

    p.add(new JLabel(" Failure Mode: "));
    nameField = new JTextField(20);
    p.add(nameField);

    p.add(new JLabel(" Anomalies List (as):  a1, a2, ..., an"));
    anomalyField = new JTextField(20);
    p.add(anomalyField);

    p.add(new JLabel(" Requirements List (as):  r1, r2, ..., rn"));
    reqField = new JTextField(20);
    p.add(reqField);

    p.add(new JLabel(" Failed Resource: "));
    resField = new JTextField(20);
    p.add(resField);

    p.add(new JLabel(" Action: "));
    actionField = new JTextField(20);
    p.add(actionField);

    getContentPane().add(p, BorderLayout.CENTER);

    info = new JTextArea(5,25);

    p.add(info);
    getContentPane().add(info, BorderLayout.EAST);

    JButton addButton = new JButton("Add");
    JButton removeButton = new JButton("Remove");
    JButton printButton = new JButton("Print Info");

    addButton.addActionListener( new ActionListener() {
        public void actionPerformed(ActionEvent ev) {
            try {
                String inputAnomalies = anomalyField.getText().trim();
                String[] anomalyessplit = inputAnomalies.split("\\, ");

                for (int i = 0; i < anomalyessplit.length; i++) {
                    anomalies.add(anomalyessplit[i]);
                }
            }
        }
    });

```



```

info.setText("Actions to take: " + Fmea.actionList.get(fName).toString() + "\n"
+ "Requirements: " + Fmea.reqList.get(fName).toString() + "\n"
+ "Anomalies list: " + anomaly.toString() + "\n"
+ Fmea.collaborations.toString() ); //+

anomalyField.setText("");
nameField.setText("");
resField.setText("");
reqField.setText("");
actionField.setText("");

        }
        catch (Exception e) {
JOptionPane.showMessageDialog(FailureGui.this, "Invalid values. "+e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
} );

p = new JPanel();
p.add(addButton);
p.add(removeButton);
p.add(printButton);
getContentPane().add(p, BorderLayout.SOUTH);

// Make the agent terminate when the user closes
// the GUI using the button on the upper right corner
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        myAgent.doDelete();
    }
} );

setResizable(false);
}

public void showGui() {
    pack();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int centerX = (int)screenSize.getWidth() / 2;
    int centerY = (int)screenSize.getHeight() / 2;
    setLocation(centerX - getWidth() / 2, centerY - getHeight() / 2);
    super.setVisible(true);
}
}
}

```

Agent: Analyzer

```

package multiAgentSystems;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.util.Collections;

```

```

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class Analyzer extends Agent {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    static String failure;
    String failureMode;
    private AnGui myGui;
    private StringBuilder stringBuilder;

    public void setup(){

        //says hello!
        System.out.println("-----");
        System.out.println("Analyzer agent" + getAID().getLocalName() + " is on");

        myGui = new AnGui(this);
        myGui.showGui();

        String interfacepreview = "Analyzer agent is on";
        stringBuilder = new StringBuilder (interfacepreview);
        myGui.info.setText(stringBuilder.toString());

        // Analyzer Agent reads the messages from Monitoring and does its tasks
        addBehaviour(new ReceiveAnomalies() );

        System.out.println("-----");

    }

    /**
     System.out.println("Sorted as compared with FMEA database" + monReport);
     failure = Fmea.anomalyList.get(monReport);
     System.out.println("the failure associated with received anomalies is: " + failure);
     */

    private class ReceiveAnomalies extends CyclicBehaviour {
        /**
         *
         */
        private static final long serialVersionUID = 1L;

        @Override
        public void action() {

```

```

        // Checking for messages from Monitoring Agent or others (Alarms)
        MessageTemplate mt = MessageTemplate.MatchPerformative( ACLMessage.INFORM );

        jade.lang.acl.ACLMessage msg = myAgent.receive(mt);

        if (msg != null) {

            // Message received. Process it
            String cId = msg.getConversationId(); //

            if (cId.equalsIgnoreCase("AnomalyReport")) {

                String monitoringReport = msg.getContent();

                System.out.println("Analyzer agent has received set of anomalies from "
                    +
                    msg.getSender().getLocalName() + ": " + monitoringReport);

                stringBuilder.append("\n" + "-----"
                    + "\n" + "Analyzer has received a set of anomalies from "
                    + msg.getSender().getLocalName() + "\n" + "Anomalies: " + monitoringReport
                    + "\n" + "-----" );
                myGui.info.setText(stringBuilder.toString());

                // comparing the anomaly set received with the list available in FMEA database

                for (java.util.ArrayList<String> check : Fmea.anomalyList.keySet()) {
                    failure = Fmea.anomalyList.get(check);

                    // sets are sorted to make sure the strings are found equal even if the items order in them
                    // are different

                    Collections.sort(check);

                    if (monitoringReport.equalsIgnoreCase(check.toString()) ) {

                        failureMode = failure;
                        System.out.println("the failure associated with anomalies received is: "
                            + failureMode);
                        System.out.println("-----");

                        stringBuilder.append("\n" + "-----"
                            + "\n" + "the failure associated with anomalies received is: " + failureMode
                            + "\n" + "-----" );

                        myGui.info.setText(stringBuilder.toString());

                        // Sending message to Decision Maker level one
                        ACLMessage informDM = new ACLMessage(ACLMessage.INFORM);
                        informDM.addReceiver(new AID("DecisionMakerAgent", AID.ISLOCALNAME));
                        informDM.setContent(failureMode);
                        informDM.setConversationId("FailureModes");
                        informDM.setOntology("FailureModesOntology");
                        send(informDM);

                        stringBuilder.append("\n" + "-----"
                            + "\n" + "Informative message has been sent to: " + "DecisionMakerAgent"
                            + "\n" + "Message content: " + failureMode
                            + "\n" + "Message ID: " + informDM.getConversationId()
                            + "\n" + "-----" );

```

```

myGui.info.setText(stringBuilder.toString());
        }
    }

    if (failureMode == null) {
        System.out.println("Send message to analyzer LVL 2");

        //Send message to Analyzer LVL2
    }

    else if (cId.equalsIgnoreCase("Alarms")) {
        // Restore information about alarms and send message to DM
    }

    else {
        //output the message information
        System.out.println("message from an unknown source!");
        System.out.println("the sender of the message is: " + msg.getSender());
        System.out.println("the content of the message is: " + msg.getContent());
        System.out.println("-----");
    }
}

class AnGui extends JFrame {

    private static final long serialVersionUID = 1L;

    private JTextArea info;

    AnGui(Analyzer an) {
        super(an.getLocalName());

        JPanel p = new JPanel();
        p.setLayout(new GridLayout(2,1));

        p.add(new JLabel(" Analyzer agent output "));
        info = new JTextArea(10,50);
        p.add(info);
        getContentPane().add(p, BorderLayout.CENTER);
        getContentPane().add(info, BorderLayout.CENTER);
    }

    public void showGui() {
        pack();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        int centerX = (int)screenSize.getWidth() / 2;
        int centerY = (int)screenSize.getHeight() / 2;
        setLocation(centerX - getWidth() / 2, centerY - getHeight() / 2);
        super.setVisible(true);
    }
}

```

Agent: DecisionMakingAgent

```

package multiAgentSystems;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.util.ArrayList;
import java.util.HashMap;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class DecisionMakerAgent extends Agent {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    static String action;
    static ArrayList<String> involvedEntities;
    static ArrayList<String> reqsForThisFailure;
    static HashMap<String, Integer> offerPrice = new HashMap<String, Integer>();
    String fmReport;
    StringBuilder stringBuilder;
    //The name of the resource to look for
    String failedResource;
    // The list of components who may have them
    private AID[] offerProviders;

    private DmGui myGui;

    public void setup(){

        //says hello!

        System.out.println("-----");
        System.out.println("Decision Making agent" + getAID().getLocalName() + " is on");

        /**
         * This behaviour receives the message from Analyzer Agent and
         * finds the solutions and requirements from FMEA database
         */

```

```

myGui = new DmGui(this);
myGui.showGui();

String interfacepreview = "Decision Making agent is on";
stringBuilder = new StringBuilder (interfacepreview);
myGui.info.setText(stringBuilder.toString());

addBehaviour(new ReceiveFailureMode());

//This behaviour starts searching and negotiating over available resources
addBehaviour(new CyclicBehaviour(){

private static final long serialVersionUID = 1L;

@Override
public void action() {

failedResource = Fmea.failure_Resource.get(fmReport);
if (failedResource != null) {
System.out.println(failedResource);
//Beginning of negotiation
addBehaviour(new OneShotBehaviour(){

private static final long serialVersionUID = 1L;

@Override
public void action() {

System.out.println("Target resource is "+ failedResource);

System.out.println("Trying to access "+ failedResource);
// Update the list of seller agents
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("Offer-Providing");
template.addServices(sd);
try {
DFAgentDescription[] result = DFService.search(myAgent, template);
System.out.println("Found the following seller agents:");
offerProviders = new AID[result.length];
for (int i = 0; i < result.length; ++i) {
offerProviders[i] = result[i].getName();

System.out.println(offerProviders[i].getName());
}
}
catch (FIPAException fe) {
fe.printStackTrace();
}

// Perform the request
myAgent.addBehaviour(new RequestPerformer());
}

});

}
});
});

```

```

        System.out.println("-----");
    }

    private class ReceiveFailureMode extends CyclicBehaviour {

        /**
         *
         */
        private static final long serialVersionUID = 1L;

        public void action() {

            MessageTemplate mt =
                MessageTemplate.and(
                    MessageTemplate.MatchPerformative( ACLMessage.INFORM ),
                    MessageTemplate.MatchSender( new AID( "Analyzer",
                                                            AID.ISLOCALNAME) ) );

            jade.lang.acl.ACLMessage msgFromAnalyzer = myAgent.receive(mt);

            if (msgFromAnalyzer != null) {
                // Message received. Process it
                String cId = msgFromAnalyzer.getConversationId();

                if (cId.equalsIgnoreCase("FailureModes")) {

                    fmReport = msgFromAnalyzer.getContent();

                    action = Fmea.actionList.get(fmReport);
                    reqsForThisFailure = Fmea.reqList.get(fmReport);
                    involvedEntitiZ = Fmea.involvedentities.get(fmReport);
                    System.out.println("Decision Maker Received a failure mode From Analyzer: " + fmReport);

                    // Write on the interface
                    stringBuilder.append("\n" + "-----"
                                       + "\n" + "Message was received from " +
msgFromAnalyzer.getSender().getName() + " : " + fmReport
                                       + "\n" + "-----" );
                    myGui.info.setText(stringBuilder.toString());

                    //prepare alarms

                    for (int i = 0; i < Fmea.collaborations.size(); i++) {

                        String receiver = (String) Fmea.collaborations.get(i);
                        String msgContent = myAgent.getAID().getLocalName() + " has failed. The failure is: " +
fmReport;

                        // Sending alarms to others in collaboration
                        ACLMessage alarmCollaborators = new ACLMessage(ACLMessage.INFORM);
                        alarmCollaborators.addReceiver(new AID(receiver, AID.ISLOCALNAME));
                        alarmCollaborators.setConversationId("Alarms");
                        alarmCollaborators.setContent(msgContent);
                        alarmCollaborators.setOntology("FailureModesOntology");
                        send(alarmCollaborators);

                        stringBuilder.append("\n" + "-----"
                                           + "\n" + "Alarm Message has been sent to: " + receiver
                                           + "\n" + "-----" );
                    }
                }
            }
        }
    }

```

```

        myGui.info.setText(stringBuilder.toString());
    }

    if (action != null) {

        System.out.println("countermeasure for identified failure is: " + action);

        stringBuilder.append("\n" + "-----"
            + "\n" + "Action to be taken is: " + action
            + "\n" + "-----" );
        myGui.info.setText(stringBuilder.toString());

        }
        else {

        System.out.println( "no action is identified for this agent. Message will be sent to
        DecisionMaker level 2" );
        stringBuilder.append("\n" + "-----"
            + "\n" + "Action to be taken is: none"
            + "\n" + "-----" );
            myGui.info.setText(stringBuilder.toString());

        }

        if (involvedEntitiZ != null) {

            stringBuilder.append("\n" + "-----"
                + "\n" + "Envolved Entities Are: " + involvedEntitiZ
                + "\n" + "-----" );

            myGui.info.setText(stringBuilder.toString());

            for (int i = 0; i < involvedEntitiZ.size(); i++) {

                // The failed unit ( The use case: The conveyer system)
                String failedUnit = "The Conveyer System";
                /**
                Since there is only one component involved in this example
                there is no list of components to identify specifically for
                a given failure, to be chosen and sent to locate the failure.
                */

                String receiver = (String) involvedEntitiZ.get(i);
                String msgContent = "Component: " + failedUnit + "--- Failure Mode: " + fmReport;

                // Sending alarms to others in collaboration
                ACLMessage askForHelp = new ACLMessage(ACLMessage.REQUEST);
                askForHelp.addReceiver(new AID(receiver, AID.ISLOCALNAME));
                askForHelp.setConversationId("Fixation");
                askForHelp.setContent(msgContent);
                askForHelp.setOntology("FailureModesOntology");
                send(askForHelp);

                stringBuilder.append("\n" + "Request message has been sent to: " + receiver
                    + "\n" + "Content: " + msgContent);

                myGui.info.setText(stringBuilder.toString());
            }
        } else {

```

```

stringBuilder.append("\n" + "-----"
    + "\n" + "Involved Entities Are: none"
    + "\n" + "-----" );

myGui.info.setText(stringBuilder.toString());

    }

    if (reqsForThisFailure != null) {

System.out.println("Requirements for fixation of the identified failure is: " +
    reqsForThisFailure );

        stringBuilder.append("\n" + "-----"
            + "\n" + "Requirements for fixation are: " + reqsForThisFailure
            + "\n" + "-----" );
        myGui.info.setText(stringBuilder.toString());

        //a hypothetical inventory as a receiver
        String receiver = new String("inventory");
/**
This is a hypothetical inventory to show the behaviour since
there is no real inventory in the test, otherwise instead of this
there had to be the address (agent ID) of the inventory unit.
*/

        // Sending alarms to others in inventory
        ACLMessage askForRequirements = new ACLMessage(ACLMessage.REQUEST);
        askForRequirements.addReceiver(new AID(receiver, AID.ISLOCALNAME));
        askForRequirements.setConversationId("Fixation");
        askForRequirements.setContent(reqsForThisFailure.toString());
        askForRequirements.setOntology("FailureModesOntology");
        send(askForRequirements);

        stringBuilder.append("\n" + "-----"
            + "\n" + "Request message has been sent to: " + receiver
            + "\n" + "Content: " + reqsForThisFailure.toString()
            + "\n" + "-----");

        myGui.info.setText(stringBuilder.toString());

/**
there might be a group of inventories or more requirement providers
which can demand a loop code through a list which each item matching
specific requirements
*/

    }

    else {

System.out.println("no requirements are identifies for this failure mode");

}

        System.out.println("-----");

    }

    else if(cId.equalsIgnoreCase("Alarms")) {

```

```

        // retrieve action from FMEA Alarm Actions
        }

        else {

//output the message information
System.out.println("message from an unknown source!");
System.out.println("the sender of the message is: " + msgFromAnalyzer.getSender());
System.out.println("the content of the message is: " + msgFromAnalyzer.getContent());
System.out.println("-----");
        }
    }
}

private class RequestPerformer extends Behaviour {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private AID bestProvider; // The agent who provides the best offer
    private int lowestCost; // The best offered price
    private int repliesCnt = 0; // The counter of replies from seller agents
    private MessageTemplate mt; // The template to receive replies
    private int step = 0;

    public void action() {
        switch (step) {
            case 0:

// Send the cfp to all sellers
ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
for (int i = 0; i < offerProviders.length; ++i) {
    cfp.addReceiver(offerProviders[i]);
}
        cfp.setContent(failedResource);
        cfp.setConversationId("Resource-Utilization");
        cfp.setReplyWith("cfp"+System.currentTimeMillis()); // Unique value
        myAgent.send(cfp);
        // Prepare the template to get proposals
        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Resource-Utilization"),
            MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
        step = 1;

break;
case 1:
// Receive all proposals/refusals from seller agents
ACLMessage reply = myAgent.receive(mt);
if (reply != null) {
    // Reply received
    if (reply.getPerformative() == ACLMessage.PROPOSE) {
        // This is an offer

        int price = Integer.parseInt(reply.getContent());
        String sendername = reply.getSender().toString();
        offerPrice.put(sendername, price);

```

```

if (bestProvider == null || price < lowestCost) {
// This is the best offer at present
    lowestCost = price;
    bestProvider = reply.getSender();
}
}
repliesCnt++;
if (repliesCnt >= offerProviders.length) {
// We received all replies
step = 2;
}
}
else {
    block();
}
}
break;
case 2:
    // Send the purchase order to the seller that provided the best offer
    ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
    order.addReceiver(bestProvider);
    order.setContent(failedResource);
    order.setConversationId("Resource-Utilization");
    order.setReplyWith("order"+System.currentTimeMillis());
    myAgent.send(order);
    // Prepare the template to get the purchase order reply
    mt = MessageTemplate.and(MessageTemplate.MatchConversationId("Resource-Utilization"),
MessageTemplate.MatchInReplyTo(order.getReplyWith()));
step = 3;
break;
case 3:
    // Receive the purchase order reply
    reply = myAgent.receive(mt);
    if (reply != null) {
// Purchase order reply received
if (reply.getPerformative() == ACLMessage.INFORM) {
// Purchase successful. We can terminate
System.out.println(failedResource + " successfully purchased from agent "
+ reply.getSender().getName());
System.out.println("Price = " + lowestCost);

stringBuilder.append("\n" + "Failed resource is: " + failedResource
+ "\n" + " ----- "
+ "\n" + " --- Negotiation Results --- "
+ "\n" + " Available offerces were: " + offerPrice.values()
+ "\n" + failedResource + " is to be substituted from: "
+ "\n" + reply.getSender().getName()
+ "\n" + "price = " + lowestCost);

myGui.info.setText(stringBuilder.toString());

}
else {
System.out.println("Attempt failed: requested resource is not available anymore.");
}

step = 4; // TODO double check this
fmReport = null;
failedResource = null;
repliesCnt = 0;

```

```

        }
        else {
            block();
        }
        break;
    }
}

public boolean done() {
    if (step == 2 && bestProvider == null) {
        System.out.println("Attempt failed: "+failedResource+" not available for sale");
    }
    return ((step == 2 && bestProvider == null) || step == 4);
}
} // End of inner class RequestPerformer

class DmGui extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private JTextArea info;

    DmGui(DecisionMakerAgent dm) {
        super(dm.getLocalName());

        JPanel p = new JPanel();
        p.setLayout(new GridLayout(2,1));

        p.add(new JLabel(" Negotiation Result "));
        info = new JTextArea(10,50);
        p.add(info);
        getContentPane().add(p, BorderLayout.CENTER);
        getContentPane().add(info, BorderLayout.CENTER);
    }

    public void showGui() {
        pack();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        int centerX = (int)screenSize.getWidth() / 2;
        int centerY = (int)screenSize.getHeight() / 2;
        setLocation(centerX - getWidth() / 2, centerY - getHeight() / 2);
        super.setVisible(true);
    }
}
}

```

Agent: Monitoring agent

```

package multiAgentSystems;

import java.util.Collections;

```

```

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;

public class Monitor extends Agent {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    static java.util.List<String> errors;
    private MonitoringGui myGui;

    public void setup(){

        System.out.println("-----");
        System.out.println("Monitoring agent " + getAID().getLocalName() + " is on");

        errors = new java.util.ArrayList<String>();
        myGui = new MonitoringGui(this);
        myGui.showGui();
        System.out.println("-----");
    }

    public void monitoredValues(final java.util.List<String> anomalies) {
        addBehaviour(new OneShotBehaviour() {
            /**
             *
             */
            private static final long serialVersionUID = 1L;

            public void action() {
                errors = anomalies;
                System.out.println("Anomalies found as: " + anomalies );
                if (errors != null) {

                    Collections.sort(errors);
                    String report = errors.toString();
                    System.out.println("Monitoring agent found " +
errors.size() + " anomalies: " + errors);

                    ACLMessage inform = new ACLMessage(ACLMessage.INFORM);
                    inform.addReceiver(new AID("Analyzer", AID.ISLOCALNAME));
                    inform.setContent(report);
                    inform.setConversationId("AnomalyReport"); // to distinguish
anomaly inform message from alarms
                    inform.setOntology("FailureModesOntology");
                    send(inform);
                    System.out.println("message sent to Analyzer: " +
inform.getContent());

                }
            }
        });
    }
}

```

Agent: Monitoring agent (interface)

```

package multiAgentSystems;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.Arrays;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

class MonitoringGui extends JFrame {
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private Monitor myAgent;

    private JTextField anomalyField;

    MonitoringGui(Monitor monitor) {
        super(monitor.getLocalName());

        myAgent = monitor;

        JPanel p = new JPanel();
        p.setLayout(new GridLayout(1, 2));
        p.add(new JLabel("Anomalies List"));
        anomalyField = new JTextField(20);
        p.add(anomalyField);

        getContentPane().add(p, BorderLayout.CENTER);

        JButton addButton = new JButton("Add");
        addButton.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent ev) {
                try {
                    String input = anomalyField.getText().trim();
                    String[] titlesplit = input.split("\\, ");
                    java.util.List<String> title = Arrays.asList(titlesplit);

                    myAgent.monitoredValues(title);
                    anomalyField.setText("");
                }
                catch (Exception e) {
                    JOptionPane.showMessageDialog(MonitoringGui.this, "Invalid values. "+e.getMessage(), "Error",
                    JOptionPane.ERROR_MESSAGE);
                }
            }
        });
    }
}

```

```

        }
    }
} );
p = new JPanel();
p.add(addButton);
getContentPane().add(p, BorderLayout.SOUTH);

// Make the agent terminate when the user closes
// the GUI using the button on the upper right corner
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        myAgent.doDelete();
    }
});

setResizable(false);
}

public void showGui() {
    pack();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int centerX = (int)screenSize.getWidth() / 2;
    int centerY = (int)screenSize.getHeight() / 2;
    setLocation(centerX - getWidth() / 2, centerY - getHeight() / 2);
    super.setVisible(true);
}
}
}

```

Agent: Offer simulator agent

```

package multiAgentSystems;

import jade.core.Agent;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.domain.DFService;
import jade.domain.FIPAAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;

import java.util.*;

public class OfferSimulator extends Agent {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    // saving offers
    private Hashtable<String, Integer> catalogue;
    // The GUI for offer providers
    private OfferSimulatorGui myGui;

    protected void setup() {

        catalogue = new Hashtable<String, Integer>();

        myGui = new OfferSimulatorGui(this);
    }
}

```

```

myGui.showGui();

// Register the offer in the yellow pages
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("Offer-Providing");
sd.setName("Resource-Negotiation");
dfd.addServices(sd);
try {
    DFService.register(this, dfd);
}
catch (FIPAException fe) {
    fe.printStackTrace();
}

// Add the behaviour for queries from decisionmaker
addBehaviour(new OfferRequestsServer());

// Add the behaviour for requests from decisionmaker
addBehaviour(new PurchaseOrdersServer());
}

protected void takeDown() {

    try {
        DFService.deregister(this);
    }
    catch (FIPAException fe) {
        fe.printStackTrace();
    }
    // Close the GUI
    myGui.dispose();

System.out.println("Offer Simulator-agent "+getAID().getName()+" is terminating.");
}

public void updateCatalogue(final String resName, final int cost) {
    addBehaviour(new OneShotBehaviour() {
        /**
         *
         */
private static final long serialVersionUID = 1L;

public void action() {
    catalogue.put(resName, new Integer(cost));
    System.out.println(resName+" inserted into catalogue. Price = "+cost);
    }
    } );
}

private class OfferRequestsServer extends CyclicBehaviour {
    /**
     *
     */
private static final long serialVersionUID = 1L;

public void action() {

```

```

MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
    // Msg received
    String resName = msg.getContent();
    ACLMessage reply = msg.createReply();

    Integer cost = (Integer) catalogue.get(resName);
    if (cost != null) {
        // The requested resource is available. Reply with the price
        reply.setPerformative(ACLMessage.PROPOSE);
        reply.setContent(String.valueOf(cost.intValue()));
    }
    else {
        // The requested resource is NOT available.
        reply.setPerformative(ACLMessage.REFUSE);
        reply.setContent("not-available");
    }
    myAgent.send(reply);
}
else {
    block();
}
}

private class PurchaseOrdersServer extends CyclicBehaviour {
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public void action() {
        MessageTemplate mt =
MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
        ACLMessage msg = myAgent.receive(mt);
        if (msg != null) {
            // ACCEPT_PROPOSAL Msg received
            String resName = msg.getContent();
            ACLMessage reply = msg.createReply();

            Integer cost = (Integer) catalogue.remove(resName);
            if (cost != null) {
                reply.setPerformative(ACLMessage.INFORM);
                System.out.println(resName+" sold to agent "+msg.getSender().getName());
            }
            else {

                // The requested resource is not available anymore for some reasons.
                reply.setPerformative(ACLMessage.FAILURE);
                reply.setContent("not-available");
            }
        }
        myAgent.send(reply);
    }
    else {
        block();
    }
}
}
}

```

Agent: Offer simulator agent (interface)

```

package multiAgentSystems;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

class OfferSimulatorGui extends JFrame {
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    private OfferSimulator myAgent;

    private JTextField resField, costField;

    OfferSimulatorGui(OfferSimulator a) {
        super(a.getLocalName());

        myAgent = a;

        JPanel p = new JPanel();
        p.setLayout(new GridLayout(2, 2));
        p.add(new JLabel("Resource Name:"));
        resField = new JTextField(15);
        p.add(resField);
        p.add(new JLabel("Cost:"));
        costField = new JTextField(15);
        p.add(costField);
        getContentPane().add(p, BorderLayout.CENTER);

        JButton addButton = new JButton("Add");
        addButton.addActionListener( new ActionListener() {
            public void actionPerformed(ActionEvent ev) {
                try {
                    String title = resField.getText().trim();
                    String price = costField.getText().trim();
                    myAgent.updateCatalogue(title, Integer.parseInt(price));
                    resField.setText("");
                    costField.setText("");
                }
                catch (Exception e) {

```

```
JOptionPane.showMessageDialog(OfferSimulatorGui.this, "Invalid values. "+e.getMessage(),
"Error", JOptionPane.ERROR_MESSAGE);
    }
    }
    });
    p = new JPanel();
    p.add(addButton);
    getContentPane().add(p, BorderLayout.SOUTH);

    // Make the agent terminate when the user closes
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            myAgent.doDelete();
        }
    });

    setResizable(false);
}

public void showGui() {
    pack();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int centerX = (int)screenSize.getWidth() / 2;
    int centerY = (int)screenSize.getHeight() / 2;
    setLocation(centerX - getWidth() / 2, centerY - getHeight() / 2);
    super.setVisible(true);
}
}
```