DIPLOMARBEIT

Automation Agents with a Reflective World Model for Batch Process Automation

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs

unter der Leitung von

Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Markus Vincze
Univ. Ass. Dipl.-Ing. Wilfried Lepuschitz

E376
Institut für Automatisierungs- und Regelungstechnik

eingereicht an der Technischen Universität Wien
Fakultät für Elektro- und Informationstechnik

von

Bakir Šahović
Matr.-Nr.: 0325081
Radetzkystraße 4/16
A-1030 Wien
Österreich

Wien, im September 2010

Abstract

Need for innovation on the global markets pushes the boundaries of manufacturing and production and requires them to become more adoptable and flexible in order to keep up with the ever-changing global conditions. This thesis presents a new approach in batch process automation that could potentially improve the current situation in this field. Batch processes are widely spread in industries, especially in chemical and pharmaceutical industry. They enable flexible manufacturing of products by employing different equipment of a plant for various activities. This quality makes them especially suitable for smaller plants.

In order to optimize the currently applied approaches to batch processes, control systems used in realization of the process can be distributed to make plants more flexible and failure tolerant. The IEC 61499 standard can be used for the realization of distributed systems, for it defines function blocks (graphical representations of algorithms) that support distribution of components.

Research in the field of distributed systems is developing in direction of intelligent systems capable of autonomous performance. With this in mind, this thesis offers a look into agent technology and its application in the domain of batch processes. This technology can be used for programming almost all kinds of software (network applications, mobile applications, database applications, etc.). It is also applicable in other fields of technology like e.g. robotics, manufacturing, transportation systems and so on.

Agents generally coexist with other agents in so called Multi-Agent Systems (MAS). This thesis presents a special approach that involves automation agents acting as parts of MAS. Automation agents represent hardware components of plants. They are autonomous software components that use a reflective world model as their knowledge data source. Automation agents are composed of layered architecture and their low level control is developed according to IEC 61499, whereas high level control is implemented using JAVA programming language.

This thesis will show that employment of automation agents with reflective world model in batch process automation increases flexibility and failure tolerance of systems, which are two very important aspects of production.

Kurzfassung

Die Nachfrage nach Innovationen auf den globalen Märkten erweitert die Grenzen der Fertigung und Produktion und verlangt von ihnen mehr Anpassungsfähigkeit und Flexibilität, um Schritt mit den sich ständig verändernden globalen Bedingungen zu halten. Die vorliegende Arbeit stellt einen neuen Ansatz in der Automatisierung von Batchprozessen vor, der den aktuellen Zustand in diesem Bereich verbessern könnte. Batchprozesse werden häufig in der Industrie eingesetzt, vor allem in der chemischen und Pharmaindustrie. Durch den Einsatz der verschiedenen Geräte einer Anlage für unterschiedliche Aktivitäten, machen Batchprozesse die Herstellung von Produkten flexibel. Diese Eigenschaft macht sie besonders geeignet für kleinere Anlagen.

Zur Optimierung derzeit angewandter Ansätze im Bereich der Batchprozesse, werden die eingesetzten Steuerungssysteme verteilt, um Anlagen flexibler und fehlertoleranter zu machen. Der IEC 61499 Standard wird für die Realisierung verteilter Systeme verwendet, denn er definiert jene Funktionsbausteine (grafische Darstellung von Algorithmen), die die Verteilung der Komponenten unterstützen.

Im Bereich verteilter Systeme wird immer mehr in Richtung intelligenter Systeme geforscht, die eine autonome Abwicklung von Prozessen ermöglichen. In diesem Sinne stellt diese Arbeit Agententechnologie und deren Anwendung in der Domäne von Batchprozessen vor. Diese Technologie kann für die Programmierung nahezu aller Arten von Software verwendet werden, z.B. Netzwerkapplikationen, mobile Applikationen, Datenbankanwendungen usw. Darüber hinaus ist Agententechnologie auch für andere Gebiete der Technik, wie z.B. Robotik-, Fertigungs-, Transport-Systeme usw., gut geeignet.

Agenten werden in der Regel mit anderen Agenten in sogenannten Multi-Agenten Systemen (MAS) eingesetzt. Diese Arbeit präsentiert einen besonderen Ansatz zur Anwendung von Automationsagenten. Automationsagenten sind Teil des MAS und repräsentieren Hardwarekomponenten einer Anlage. Sie sind autonome Softwarekomponenten, die ein reflektives Weltmodell (engl. reflective world model) als ihre Datenquelle verwenden. Automationsagenten bestehen aus einer Schichtenarchitektur, wobei ihre untere Steuerungsebene (engl. low level control) auf IEC 61499 basiert und die obere Steuerungsebene (engl. high level control) mit der höheren Programmiersprache JAVA entwickelt wurde.

Diese Arbeit zeigt den Einsatz von Automationsagenten mit reflektivem Weltmodell in der Automatisierung von Batchprozessen, um die Flexibilität und Ausfallsicherheit eines Systems zu erhöhen, was zwei sehr wichtige Aspekte der Produktion sind.

Acknowledgement

Let me start by expressing my deepest gratitude to all those who supported and helped me during my studies at the Vienna Technical University. This thesis would not have been possible without the patience and support I was given by my family in the times I needed them the most. I would also like to thank my fiancée who helped me prepare for my exams and read this thesis quite a number of times trying to find even the smallest flaws of argumentation.

I would like to thank the people at ACIN for all the technical and professional support that they so kindly provided me with whenever I needed it. I feel very grateful to Professor Markus Vincze for being my supervisor. I also owe my deepest gratitude to DI Wilfried Lepuschitz, who has helped me with his constructive comments on the thesis and has made available his support in a number of other ways. I am especially thankful to him for his cooperativeness, even on weekends, and for the willingness to help me finish the thesis on schedule.

I also cannot forget the help and support from Dr. Munir Merdan and Dr. Mathieu Vallée. Their inputs surely improved my knowledge in the domain of multi-agent systems and helped me understand the subject-matter.

Bakir Šahović

Contents

List of Figures

1 Introduction

Globalization of markets seems to be one of the hallmarks of the 21st century. Among its many consequences are growing competition and need for new high-quality products. These circumstances push companies to try to shorten their time-to-market and increase production in order to stay competitive.

It is well known that mass production is cheaper in Asia than in Europe or USA. That is why especially the European industry needs to focus on increasing the quality of its products. Furthermore, one should not forget that the quality is only one side of the coin. In addition to that, products of today and tomorrow have to be innovative in their nature. Manufacturing of such products requires ability for swift changes, adaptations and flexibility of production plants. In order to achieve all of these goals, new processes need to be employed. An example of improvements in this field is the increased presence of computer-based production in industry. Computer-based systems gather the information about the environment via sensors, process it and consequently influence and change its state through involvement of actuators.

Currently employed process techniques, from types of processes and programming techniques to infrastructure of plants, do not always satisfy customers' needs. Continuous, discrete and batch processes are all widely spread throughout industries and are used for production. Nevertheless, batch processes are mostly used in chemical industries, food manufacturing and some other. Focus of this thesis is on the domain of batch process automation and their optimization proposal.

Batch processes, much like anything else, are easier to employ when they are standardized. One of the standards applied in this field was IEC 61131. It defined and standardized many programming techniques that, even though widely used, do not support distributed control architecture. Consequently, a new standard, IEC 61499 function blocks, was introduced in order to employ more flexible and failure tolerant production of goods. Function blocks are graphical representations of algorithms and they support distributed architecture. This technology makes it possible to employ batch processes on distributed systems.

Employing of IEC 61499 in batch process automation can be additionally enhanced by involving agent technology. Agents, operating in coordination with each other in so called Multi-Agent Systems (MAS), are autonomous and independent units that bring aspect of artificial intelligence to process automation. As a result, each mechatronic device can be provided with its own controller, assigned only to the functions provided by it. Even though implementation of this new technology may cause additional costs for companies at first, the flexible and failure tolerant production that it enables has the potential of actually decreasing expenditures in the long run. Plants can then be described as intelligent systems, because they are able to make decisions on their own, handle failures and minimize the shutdown time needed for switching production from one product to another. It also allows for defective parts to be mended or replaced without having to stop the whole system. By and large, developing MAS for application in batch process automation poses an intriguing task that will require implementation of so called automation agents. Utilization of that special agent type turns each mechatronic device into an autonomous and independent entity. These agents use world models as their knowledge data source.

Motivation for this thesis, derived from the aforementioned facts, is developing a MAS that consists of automation agents and additional functional agents and supports distributed control architecture. This MAS will finally be implemented and tested on a batch process plant.

1.1 General Objectives

This thesis discusses development and implementation of agent technology in batch process automation. Agent technology can be applied in the domain of process automation in many different ways, but in this thesis the focus is laid on a special approach that employs automation agents with reflective world models. Beside automation agents, MAS additionally contains functional agents that are responsible for the communication with users and management of recipe execution.

In order to comprehend the significance of this idea, one needs to be familiar with nature of automation agents. It can generally be said that automation agents consist of high level and low level controls. High level control is implemented in JADE (Java Agent DEvelopment) framework, whereas IEC 61499 function blocks was used for the low level. Automation agents are primarily responsible for controlling the hardware.

World model is part of high level control and serves as knowledge data source of individual automation agents. Automation agents make decisions based on their world models. World model is called reflective as it contains information about single automation agent.

The discussed idea needs to be analyzed and implemented on the plant in Odo-Struger laboratory at the Automation and Control Institute (ACIN)[1], Vienna University of Technology. Engineering of this specific MAS implies development of the whole process, including control of mechatronic devices, communication between agents, failure detection and failure recovery.

1.2 Task Definitions

There are several steps that need to be carried out for the purpose of this thesis. Tasks can be summarized as following:

- Architecture of automation agents (incl. reflective world models) needs to be designed and implemented for each device on the laboratory plant.
- Development of low level control, in accordance with IEC 61499 function blocks, of each automation agent that represents hardware on the plant.
- Development and implementation of the workflow of high level control (in JADE) of every automation and functional agent.
- Automation agents with reflective world model and functional agents need to be tested on the laboratory plant. Testing includes monitoring of system behavior in an abnormal situation (in this case a failure) and measuring duration of message transmission from high level to low level control.
- Discussion of results, advantages and disadvantages of this approach.

1.3 Structure of the Thesis

All of the aforementioned points will be presented and discussed in more detail, for this thesis consists of seven distinctive chapters. Chapter 2 introduces the reader to the state of the art in distributed batch process automation. It provides an introduction to batch processes, basics of IEC 61499 and discusses current research in the field of MAS. Chapter 3 explains a special MAS approach in batch processes that requires usage of automation agents and world models. This topic includes a discussion about different types of automation

[1]www.acin.tuwien.ac.at, accessed in September 2010.

agents, analysis of communication between agents and application of reflective world model.

Implementation of the approach explained in Chapter 3 is the topic of Chapter 4. It also gives a description of the plant and utilized hardware and software. Automation agents and their world models are explained and communication between them as well as recipe execution is analyzed. At the end of Chapter 4 the message transmission duration is measured in failure free scenario and results are presented.

Chapter 5 gives an overview of the obtained results of testing on the plant. Measuring duration of system recovery after failure detection is also a part of the performed tests.

Last two chapters include a discussion of the obtained results and advantages and disadvantages of this approach. They also give an overview of possible future research on this topic.

2 State of the Art

As already mentioned in Chapter 1, continuous need for innovation requires flexible automation plants. After a brief presentation of automation systems, this chapter gives introduction to a widely spread process type, batch process. Batch processes have become popular due to their adaptiveness and prospects of improvement.

Batch processes have recipes according to which output is produced. Every recipe consists of defined flow of actions that are required to be done in order to produce that output. Those actions are programmed and *"installed"* on Programmable Logic Controller (PLC) or Distributed Control System (DCS). Among many programming techniques used in industry for programming controllers, this chapter provides a brief description of a specific one, function block diagram according to the norm IEC 61499 published by International Electrical Commission (IEC).

The IEC 61499 standard supports distributed architecture and specifies flow of events and algorithms, as in distributed systems one or more processes can be executed simultaneously. Application of distributed systems can be improved by employing agent technology. Current development in the field of agent technology will be discussed in more detail at the end of this chapter. This will provide a starting point for later chapters, as this thesis presents a new approach in programming of flexible, failure tolerant batch processes based on agent technology and IEC 61499.

This chapter is outlined in the following manner. First it gives an introduction to batch process automation. That is followed by brief explanation of a programming technique based on IEC 61499 function blocks. Following section discusses agent technology in more detail and presents several current developments and trends of their application in batch processes. Final section deals with definition and characteristics of ontology that is commonly used in agent systems.

2.1 Automation Systems

In order to understand the topic that is about to be discussed, one first needs to become familiar with few basic definitions from the field of *automation*. The root of the word *"automation"* comes from Greek *automatos* which means *"to move by itself"*. German institute for norms (Deutsches Institut für Normung, DIN) defines *"to automate"* in DIN IEC 60050-351 [17] as following:

- to automate: *"to employ means to enable self-acting functions in a system"*;
- system: *"set of interrelated elements considered in a defined context as a whole and separated from their environment"*;
- automaton: *"self-acting artificial system, whose behavior is governed either in a stepwise manner by given decision rules or continuously by defined relations and whose output variables are created from its input and state variables"*;

Automation is a noun of the verb to automate, which means to apply automation to something [40]. Specifically, automation is applied to systems in order to produce goods. Focus of production is technical process, whereas automation of this process is referred to as process automation [7].

According to DIN 66201, *process* and *technical process* are defined as follows [13]:

- process: *"the totality of activities in a system which are influencing each other and by which material, energy or information is transformed, transported or stored"*;
- technical process: *"a process whose physical quantities can be acquired and influenced by technical means"*;

In his book Barker et al. [3] identified three distinct technical process types in process automation:

- Continuous process can be described as one in which flow of material or product is continuous. Product of this type of process is manufactured in different equipments as a result of material processing. Continuous processes can be described with differential equations and transition functions [7]. Some examples of this process type are generation and distribution of electricity, production of steel, etc.

- Discrete process is the one where output appears in discrete quantities. A specified quantity of products moves in lots between workstations. Production of cars or those of mother-boards are just some of the examples of this process type.
- Output of batch process appears in quantities of materials or lots. Furthermore, it has a defined beginning and an end. It also possesses characteristics of both continuous and discrete processes. Some examples are soap manufacturing, production of pharmaceuticals or beverages.

All of the above mentioned technical process types are more or less equally important and have their place of implementation throughout industry. Nevertheless, the focus of this thesis will remain on the batch processes.

2.2 Batch Processes

The word batch has several meanings. According to the Oxford English Dictionary [40], *"batch"* describes a quantity produced at one operation. It can also mean a quantity of anything coming at a time or a number of something put together.

In process automation batch has a slightly different meaning. It is defined as material produced during batch process and as entity that represents production of material. Batch process can be described as a process which, during a production over a finite time, uses one or more pieces of equipment in a defined order and on a defined schedule. During the production a finite quantity of products (batch) is produced as a result [3].

During batch process, equipment of a plant can be used one or more times to perform different tasks, which is not the case with continuous processes. For instance, during a continuous process all operations are performed at the same time [46].

Nowadays, there are still many products manufactured using continuous processes even though initially batch processes used to be utilized for the production [3]. Main reason for shifting to continuous processes is a lack of required skills and operators to employ in batch production. Continuous processes have brought considerable profit to industries and switching of production to batch processes is therefore developing rather slowly. Still, usage of continuous processes is slowly declining in the developed world. At the same time, utilization of batch processes is rapidly spreading to production of pharmaceuticals, agrochemicals, food additives, photosensitive material, vitamins, beverages and many more [46].

During batch process, plant is in a state of constant change [22]. Employing equipment in batch processes makes it adaptable to change, which makes these processes interesting for small companies with low capacities. Level of complexity increases, but so does the flexibility of the plant. This has for a consequence that the equipment of a plant employing a batch process is generally operating in a non steady state. In other words, the equipment is under constant change. One part of a plant can perform various operations at different times and all that within one recipe [21].

It is worth stating at this point, that batch processes are quite economical for small companies, as they require less process equipment [3]. Manufacturing with flexible equipment makes it possible to output large amounts of different products. This art of production shortens time-to-market of newly innovated products.

2.2.1 Advantages of Batch Processing

It is undeniable that there is a number of positive arguments for batch processing. Certainly one of the biggest advantages is flexibility and failure tolerance. A plant that is used for fabrication of one product can easily be transformed for production of another one. Two or more activities can be carried out at the same time. Waiting for one activity to be finished is rendered unnecessary. This characteristic combined with the fact that during batch process output usually appears as number of products and lots shortens time-to-market of new and innovative products.

In addition, each part of equipment in a plant that employs batch processes can be used for more than one activity. To illustrate, one pipe can be used for transporting of material to the reactor and from the reactor. Additionally, if a plant contains multiple reactors with same properties (heating, mixing, etc.), any of those reactors can be used to perform the given task. In other words, generally there is no need to define which reactor has to be used. Consequently, smaller plants can be used for production of various products.

Furthermore, every component of a plant can act as an entity for itself. Batch processes allow *modular distributed control architecture* that is based on *multidisciplinary devices*. These devices contain all relevant hardware and software components required for execution of a task. This concept is similar to the concept of object-oriented programming: reusability of components and encapsulation of functions [28].

In case that one part of equipment stops working it can easily be substituted by another one. In some cases, if the defective part is currently not needed for the process, it can be replaced without stopping the plant or production.

2.2.2 Challenges in Batch Processing

As much as flexibility of distributive systems is an advantage of batch processes, one cannot ignore the fact that it can also be a disadvantage. Imagine a simplified system for production of PVC like illustrated in Fig. 2.1 and imagine the scenario where different materials from one or more different reactors are moving simultaneously to one or more other reactors. In such a situation some pipes might be used for bidirectional transporting of material. Additionally, it might be necessary to clean the pipes between transportations [21].

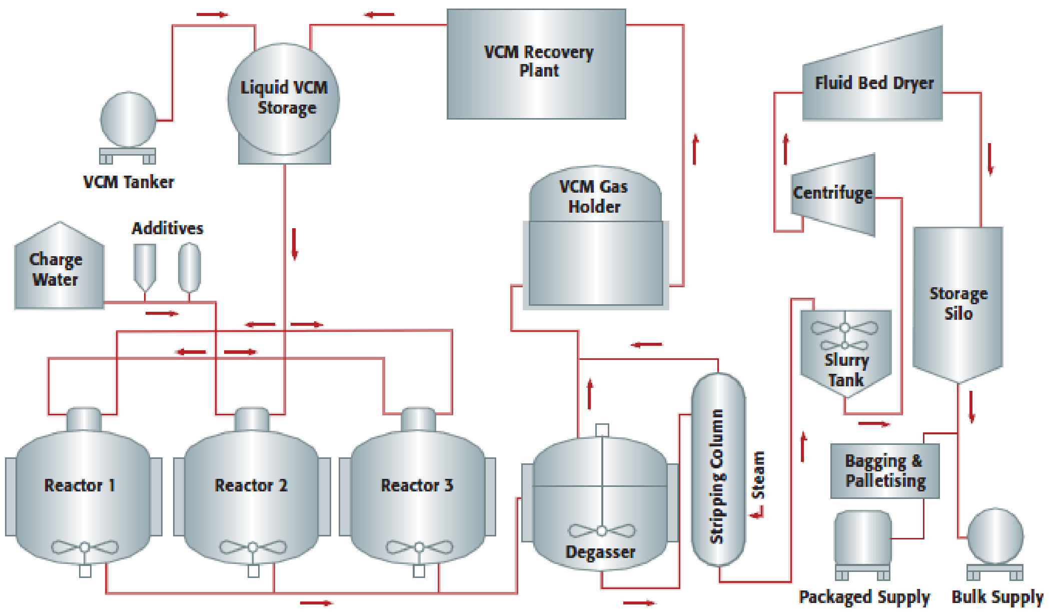


Figure 2.1: Simplified Model of a PVC Manufacturing Plant [21]

Such a scenario might pose a quite complex engineering problem that requires certain know-how in the area of batch processing.

One more matter that is characteristic to batch processes is detection of failures. In case of continuous processes, when equipment works in a steady state, it is easy to track failures. In batch processes equipment and flow of material are under constant change, so tracking and finding of failures can be very time consuming. To illustrate this point, let it be said that 60% of system development effort with regard to batch processes is absorbed by abnormal events [21].

2.2.3 IEC 61512 Standard for Batch Processes

The first standard for batch processes approved in 1995 by the Instrument Society of America was S88. It was later adopted by IEC as the 61512-1 standard. This was indeed a big step in process automation, because it was the first time that understanding of terms was harmonized for everyone working in this field [21].

It is important to emphasize how important the adoption of the IEC 61512-1 standard was. Its significance is threefold. First, adoption of this standard enables easier communication between all stakeholders in one project. Second, it allows different vendors to produce independent, yet compatible parts that can be used when building a plant. And third, it eases the development of algorithms and software for batch processes. In the light of these facts, one may expect that this will lowers the costs of batch process development.

Beside general terminology used within batch processes, the standard IEC 61512-1 includes definitions for structural model and recipe concept [20]. When speaking about structural models that are defined in IEC 61512-1 one must distinguish carefully between *physical model*, *process model* and *procedural model*.

Physical model describes the physical aspect of a company and hierarchical relationships between various entities in batch manufacturing. This model has seven layers starting from the top to bottom with *enterprise*, *site*, *area*, *process cell*, *unit*, *equipment module* and *control module*.

Procedural model consists of four layers: *procedure*, *unit procedure*, *operation* and *phase*. This model defines transition between physical and process model. These types of models are used to describe controlling of actions within parts of equipment in order to perform tasks for the process.

The process model contains four hierarchical layers. Those are *process*, *process stages*, *process operations* and *process actions*.

Besides general strategy and structural model, IEC 61512 defines recipe as list of instructions (*tasks*) in order to execute a process. All batch plants have recipes that give information about output being manufactured.

Each recipe contains a *recipe head* where general information about it is specified. The recipe head contains product identification, version number, the date of issue and the like.

Beside these specifications, recipe also contains information about *material* and *production* data. For instance, it can specify input and output of a process.

Furthermore, recipe contains different *equipment requirements* to produce an output. This part of recipe is connected to the physical model.

Recipe also contains a *recipe procedure*. The procedure defines strategy of a process and enables production of batches. This part of the recipe is connected to the procedural model.

Within one recipe additional information can also be specified, like security, safety or regulatory information.

2.2.4 Programming Standards in Batch Processing – Current and Future Trends

The main focus of this thesis lies on process management and control. Upon finishing planning and scheduling (two additional phases in batch processing), implementation of the process, also called controlling of the process, takes place. This procedure belongs to the domain of lower layers of the aforementioned structured models. Even though IEC 61512 defines the concept of batch processes, it lacks a definition of controlling process. However, standard IEC 61131 [18] provides this definition. It defines the implementation of control processes on Programmable Logic Controller (PLC) using one of the various programming techniques. This manner of implementing batch processes is nowadays primarily used in the industry. IEC 61131 consists of following parts [39]: general information and technical environment that is defined by the standard, general hardware model of PLC with equipment and test requirements, programming languages, user communication, fuzzy control programming and implementation guidelines.

There are many techniques of programming PLC. In particular, there are Ladder Diagram (LD), Structured Text (ST), Instruction List (IL), Function Block Diagram (FBD) or Sequential Function Chart (SFC). This thesis does not tackle all of the programming techniques mentioned above as they are not within its scope. The only programming technique that will be discussed in further detail is the FBD technique. For further information on other programming techniques interested reader is referred to technical literature, e.g. [24].

FBs were first defined in IEC 61131-3 and did not support distributed architecture. They are represented as graphical algorithms, containing only data inputs, with workflow from left to right. Within every FB is an algorithm code that can, for example, be written in structured text.

The definition of FB was later extended by the norm IEC 61499. This norm defines usage of FBs within distributed system domain. As it will be shown in following sections, FBs also have event inputs and outputs according to IEC 61499. In this event driven execution of algorithm it is possible not only to program standard procedural applications, but also to implement distributed systems on an Object-Oriented Programming (OOP) basis.

The IEC 61499 standard is a new promising standard, still mainly present in the research circles. The industry is hesitant to change the winning team – the IEC 61131 standard. However, the focus of this thesis is not the programming technique per se, but application of batch processes in distributed systems, as was mentioned in Chapter 1. The next section discusses IEC 61499 FBs, which are the basis for distributed systems.

2.3 IEC 61499 – Function Blocks

Application of batch process in distributed systems could not be accomplished with existing programming techniques based on IEC 61131. Therefore, the new standard, IEC 61499, had to be introduced. Function Blocks (FB) that were initially defined in IEC 61131-3, were amended for usage in distributed architecture by this new norm.

Simply put, FB as a programming technique are nothing more than graphical representation of algorithms [3]. This new concept enables FBs to become part of a reusable, modular and complex Distributed Control Systems (DCSs). Nowadays there is an evident trend toward distributed systems among all programming techniques. Distributed system architecture allows, in a similar manner like object-oriented programming, encapsulation, polymorphism and reusability of its components.

When software is programmed to control a mechatronic component, it is deployed on one of control systems. Mainly used control systems in industries nowadays are based either on DCS or PLC. DCS are usually used in petrochemical industries, and they utilize few large central processors. Those processors communicate with sensors, actuators and controllers located on a plant through a local network. In DCS there is a central supervisory and control station that is responsible for supervision of the whole system. Distributed components that are positioned out in the plant provide local control such as closed loop control.

PLCs are generally utilized to operate discrete processes like those in automotive industries. In a large PLC system there is a number of PLCs and a HMI unit communicating via one or more high speed networks. PLCs are constructed for field-level operation and can be exposed to extreme conditions, like for instance high temperatures. PLCs contain numerous inputs and outputs (I/Os) for sensor and actuators signals.

Both DCS and PLC, even though widely spread in industries today, were developed for monolithic software packages. They are often difficult to reuse, change and integrate with new components. Software packages that are written for one application are not easily included in another, even if they run on the same machine and use the same programming language.

With these requirements a new trend of advanced, more complex control systems is developed. Every device within this system, such as pump, valve or sensor can have a built-in controller. This controller would then be linked to other more intelligent controllers on other devices, like temperature controllers, HMI panels, etc.

In order to accomplish this level of distributed systems, a completely new approach to software development is required. There is an evident potential of implementing IEC 61499 FBs as a possible approach to develop such systems.

2.3.1 IEC 61499 Reference Models

IEC 61499 defines a reference model for distributed systems. This reference model is organized hierarchically and consists of five sub-models: *system model*, *device model*, *resource model*, *application model* and *function block model* (Fig. 2.2).

This standard also defines three more types of models: *distribution model*, *management model* and *operational state model.* In contrary to the five models mentioned above that represent the physical and/or application part, the three latter models are more functional. Since management and operational state models are beyond the scope of this thesis, they will not be discussed any further. The distribution model is the subject of the next section.

System model System model is shown in Fig. 2.2a. It is a composition of devices that are connected to each other via a network. They can also be connected to different networks. Applications in a system are either spread over more devices or only part of one device. One device can provide its functions to more applications simultaneously.

Controlled process is not part of system model.

Device Model Device is composed of zero or more resources (Fig. 2.2b). Device usually has an interface to a network and an interface to a process. Device that has no resources embodies a resource itself.

Process interface is used for communication with I/Os on physical devices (sensors, actuators). Data exchange flows through this interface. Communication interface is used to exchange data with other resources on the same or remote devices.

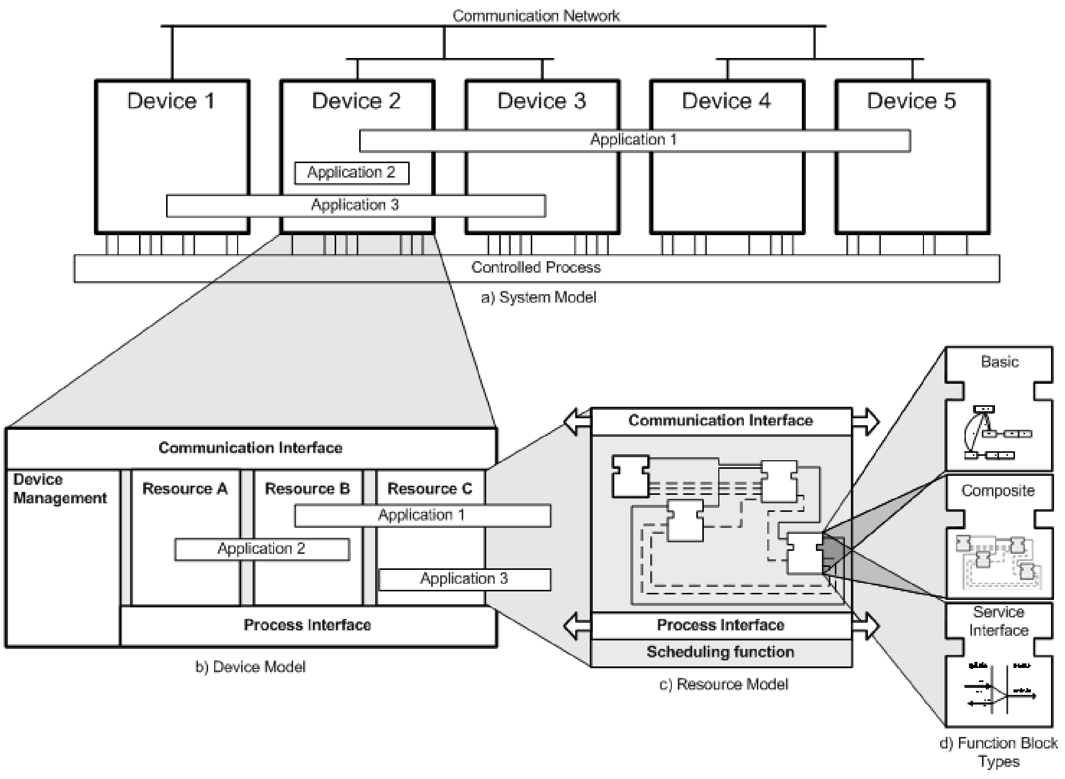


Figure 2.2: Reference Models according to IEC 61499 [52]

Resource Model Resource is part of a device (Fig. 2.2c). It is a network of function blocks and is considered to be the functional unit of an application. A resource and its functions must not affect other resources within the same device.

Resource is responsible for reading data from process interface and sending data to communication interface and vice versa.

Application Model Applications are composed of function blocks and they can be distributed over one or more resources within one or more devices. FBs are connected with event and data connections. If an application is spread over different resources, FBs are connected via Service Interface (SI) FBs. FBs of one application can also be grouped to form sub-applications.

Function Block Model Function block concept was first defined in IEC 61131-3 as a programming technique of PLCs. In this norm FB was defined as graphical language. FBs can be connected to other FBs by connecting the data flow between output and input. They contain single algorithm that was executed after receiving data from the previous FB. With this type of execution of FBs, it is complicated to determine the exact order of algorithms that are executed within them, as it was not easy to determine when one FB was going to be invoked.

Later on the new standard IEC 61499, which defines FBs that can be used in distributed systems, was developed. According to IEC 61499, FBs contain event inputs, data inputs, event outputs and data outputs. FBs also contain algorithms that are triggered by input events (Fig. 2.3).

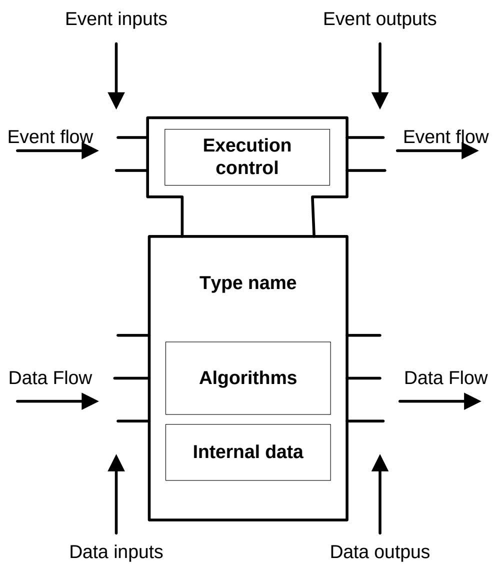


Figure 2.3: Function Block according to IEC 61499 [32]

Algorithms, internal data and execution model are hidden within FBs. Execution control defines which event is mapped to which algorithm. What a FB actually does is specified by internal algorithms. Depending on a type of controller, an algorithm is written in the ST according to IEC 61131-3 or in one of the high level programming languages like C++ or JAVA [6]. Internal data is needed for execution of algorithms, like internal constants or variables that are used in more than one algorithm.

IEC 61499 defines three types of function blocks: *basic function block*, *composite function block* and *service interface (SI) function block*.

Basic function block contains algorithms that are invoked by input events.

After algorithm execution, data output is set and an event output signal is triggered.

Composite function block consists of a network of function blocks. Events and data I/Os of inside function blocks are connected in such a manner to form a more complex capacity as a result. Composite function blocks can contain basic function blocks or another composite function blocks.

SI function block provides interface required for establishing communication with a function blocks network that is deployed on other resources (on the same or different devices). SI function blocks are also used for communication with hardware like sensors, actuators and other devices.

IEC 61499 also specifies an execution model for function blocks. For a basic function block first a value comes to a data input. After receiving an event input, the corresponding algorithm uses input data to produce output data. When the algorithm is done, it sends signal to the corresponding output event. With this process one FB is executed and the next one can begin. For more details about FB and IEC 61499 see [32], [6] and/or [48].

Distribution Model As it is shown in Fig. 2.2, applications can be distributed over different resources in one or more devices. Distribution is done when function blocks of one application are assigned to different resources and communication between them is performed via a network and SI function blocks.

Distribution of an application or a sub-application should not influence the functionality of a given application. Applications or sub-applications that are distributed over several resources can only be influenced by communication networks and by the speed of data transfer over that network.

Another important feature of applications is that they cannot be replicated, but it is possible to make a new copy of a sub-application and then use it again.

Function blocks on the other hand cannot be split into parts and distributed over devices or resources. All elements of one function block must stay within one resource. Timing of a function block depends only on one device and it is not affected by the network. Function block can only be replicated by making a new instance of the same FB type.

2.3.2 Why IEC 61499?

With the IEC 61499 FB standard, development of software for batch processes is moving in the direction of distributed systems. Common algorithms, functions and methods as parts of software are grouped to FBs. FBs enable software functions to become reusable without being strictly bound to a particular program. Concept of FB resembles to a widely spread concept of objects in object-programming. They represent an encapsulated application and reduce complexity of the program. Additionally, IEC 61499 also enables more flexible production, shortening time-to-market of the new and innovative products.

On the one hand, manufacturers will have the opportunity to choose components from different vendors without worrying about compatibility problems. Vendors, on the other hand, will be able to offer their clients a greater range of products. They will also be able to adapt their products according to their clients' needs in a shorter period of time.

2.3.3 Migration to IEC 61499

Even though the IEC 61499 standard brings many benefits to the field of batch processes, such as reusability, distribution or encapsulation of functionality, this standard has yet to be adopted by the industry. Every company tends to use familiar methods, tools and ways of developing as long as they bring them profit and they are often reluctant when adopting new technologies [41]. This behavioral habit might slow down the transition from old standards to IEC 61499.

One more reason why companies avoid shifting to new technologies is the risk that this new technology would not be accepted by other vendors and clients. There is also a risk of business partners transferring to other, completely new technologies. Costs of switching to IEC 61499 are influenced by existing investments within companies. Company that invests more into education of its employees, development tools and infrastructure (controllers that do not support IEC 61499) will accordingly suffer higher costs of changing to the new technologies. Consequently, adoption of new technologies is more difficult to implement.

Since migration to IEC 61499 is not within the scope of this thesis, the discussion will remain limited to the argument above. For additional information see [41] and/or [16].

2.4 Multi-Agent Systems (MAS)

Concept of distributed system is a new trend in process automation, but the idea of having distributed systems is not completely new. In fact, OOP as a programming technique has shown the advantages of not having everything on the same place, as is the case in monolithic software.

Using the idea of OOP as basis, a new programming technique has been developed. Agent-Oriented Programming (AOP) as a new software pattern has spread not only over the development of classical software applications, but also over mobile technologies, process automation and so on [4].

This new concept is a meeting point of theories about artificial intelligence and distributed systems. Central object of AOP is agent. Autonomy and ability to communicate are only two of numerous characteristics of agents. In this chapter, the idea of agent systems is described in more details.

At the beginning of this section, the basic idea of agents, as well as their general characteristics, is described. That is followed by a description of communication between agents and the FIPA standard. The second part of this section focuses on recent developments of agent systems used in process automation.

2.4.1 Agent Technology

Technology of today is slowly but definitely progressing towards distributed systems. Terms agent or agent technology are mentioned more frequently in the recent written research in this field. In almost all spheres of computer technology, starting with computer networks, operating systems, databases, artificial intelligence and going further to automation technology, agents have found the way to spread their influence.

One should note here that there is no strict and official definition of agent. However, there are some explanations what agents represent and define [4]. It can generally be said that agent is a software component that performs some kind of service for other agents or users. According to the Oxford English Dictionary [40] term *computer agent* is defined as follows:

> *"A program that performs a task such as information retrieval or processing on behalf of a client or user, esp. autonomously."*

This definition is relevant because when speaking about agents, one generally assumes a software agent. This thesis also discusses a different type of agent, so called automation agent (see Chapter 3). Automation agent consists of software that is split into high level part, responsible for decision making and communication to other agents, and low level part that is mainly responsible for communication with hardware. Additionally, hardware components are also considered to be a part of automation agents. It can be said that automation agents are software representation of hardware devices.

Agents typically co-exist with other agents in multi-agent systems. However, one of the most important characteristic of agents is their autonomy. It means that agents are independent and can work or exist without depending on other agents or systems.

Within a MAS, agents have opportunity to interact with each other. This interaction is generally based on communication by exchanging messages and reacting upon receipt of a message. Throughout this exchanging of messages agents try to perform given tasks.

2.4.2 Communication between Agents

The aforementioned agent communication is one of the most important aspects of MAS based distributed systems. Agents communicate with users or each other in order to solve given tasks. Every communication requires a language and this is no exception to that rule. Agents communicate using one of so called agent communication languages.

Nowadays, the most spread agent communication language is the FIPA-ACL (Foundation for Intelligent, Physical Agents – Agent Communication Language). The main advantage of FIPA-ACL is that it allows usage of different content languages.

FIPA is an international non-profit foundation which promotes usage of agents throughout industry. This organization performs a task of developing specifications for communication between agents, but its members are not obligated to use or implement any standards brought to them by FIPA [4] [10] [9].

FIPA-ACL message, standardized message used for communication between agents, can contain different parameters that can be sent (see Table 2.1).

The only mandatory requirement for message sending is a valid *performative* parameter, whereas all the others are optional. If a message contains some additional parameters after all, its content can, according to FIPA, be any string or bit combination. This allows us to transfer more information by using for instance XML as content language [4] [5].

2.4.3 Concept of Multi-Agent Systems

New developments throughout different areas of industry suggest that there is a potential for improvement that could be brought about by MAS. Flexible, fault tolerant batch processes are almost impossible to imagine without this agent technology. MAS can be represented as a complex society, whose members have responsibilities over different domains, where they communicate with each other in order to solve problems or do their tasks. Fig. 2.4 shows a MAS

Parameter	**Description**
performative	Type of communicative act of an ACL Message
sender	Name of the agent that sends the message
receiver	Recipients of the message
reply-to	Name of the agent that should receive a reply to the messages in this conversation thread
content	Content of the message
language	Language of the content
encoding	Specific encoding of the language
ontology	Meaning of the symbols in the content
protocol	Interaction protocol used by the sending agent within the ACL message
conversation-id	Conversation identifier, used to represent the ongoing sequence of communicative acts
reply-with	Expression that the responding agent uses to identify the message
in-reply-to	Reference to an earlier action to which this message is a reply
reply-by	Time and/or date by which a reply should follow

Table 2.1: FIPA-ACL Message Parameters [10]

with several agents working and communicating on different hierarchical levels. It can be seen that each agent has different field of responsibility or interest.

As already mentioned, agents usually react upon receiving a message. This is actually the main difference between agents in AOP and objects in OOP. On the one hand, in order to use a method or functionality of one class in OOP, a new object of that class needs to be instantiated. It then needs to call the desired method from that class. On the other hand, in order to perform an analogue operation within AOP, agents start by exchanging messages. After a receiver agent receives the message, the corresponding behavior will start. There are defined constraints on when a behavior can be started. Those constraints are defined within each behavior. When a behavior is started, an activity is performed and/or the communication is continued as the agent that received the message replies to the sender.

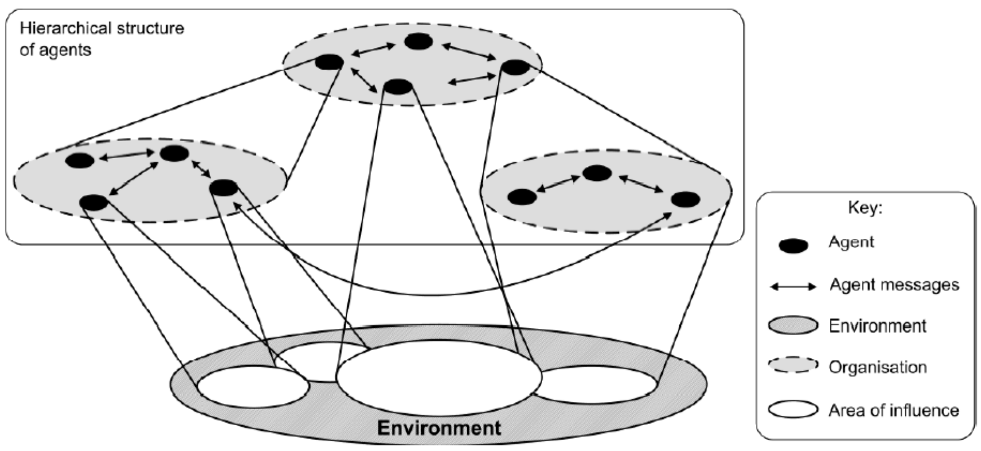


Figure 2.4: Example of MAS Society [42]

2.4.4 Usage of Multi-Agent Systems in Batch Process Automation

MAS are increasingly being applied in almost all areas, starting with industrial process control, telecommunication, robotics and also in transportation, network management, mobile technology and so on [4].

Advantages of MAS were first recognized by industrial applications and its techniques were initially tested there. Examples of industrial applications where MAS were tested are process control, transport logistics, manufacturing, etc.

In addition to the above mentioned applications of MAS, this technology is also used in other domains, like software development, air traffic control, database, artificial intelligence, etc.

This section focuses on research currently being done throughout the field of process automation, where MAS techniques are applied. One of the proposed approaches to apply MAS in batch processes is explained in [15]. This research deals with the problem of scheduling in MAS based processes. Hong et al. considered introducing one control agent and four execution agents as a possible solution to this problem. In the role of the control agent he proposed a so called manage agent, which would be responsible for administering, coordinating and monitoring of other agents. Execution agents would be order agent, resource agent, model agent and scheduling agent.

Collecting customers' orders and defining manufacturing requirements are functions of order agents. Resource agent coordinates activities of manufacturing resources such as people, materials, facilities, etc. during production. Model agent models the scheduling problem. For better understanding of the problem graphical representation, such as Petri-net [23], is usually used. Finally, there is scheduling agent that offers solutions for optimizing scheduling problems.

These agents perform given tasks in the following manner. A user inputs all the necessary requirements, as well as specifications about the desired product. Since the order agent possesses knowledge through its libraries on what tasks can be performed, it defines the order. This is followed by a graphical representation of the whole problem, done by the model agent. The manage agent then forwards the task to the resource agent. As a consequence of this development, the scheduling agent produces the most suitable scheduling plan.

Another research which dealt with MAS in batch process control systems was done by Samakoko et al. in [44] and Hamaguchi et al. in [14]. They proposed a way to improve scheduling and monitoring in batch processes by introducing an autonomous decentralized control system with four agents. Those would include recipe agents, transfer agents, equipment agents and utility agents. Their aim is to develop such batch process control system, which supports plug and play of equipments without modifying initial recipes in the process.

Recipe agent is responsible for giving instructions to equipment agent about equipment operations, while moving with material through a plant. Upon receiving instructions, equipment agent performs operations (e.g. mixing, heating, etc.) by controlling physical components. Transfer agents also have the responsibility to control physical components, but their domain is restricted to transport equipment such as valve, pump and so on. Supply and inventory of the utilities (e.g. steam, cooling water, etc.) needed within the plant are performed by utility agents.

Yet another interesting approach to employing MAS in this field was proposed by Pirttioja et al. [43]. It differs from other formerly mentioned applications in that it analyzes application of MAS for monitoring tasks in process automation. As already mentioned in Sec. 2.2 equipments in batch processes run in a non-steady state and observation of critical changes during the operation is of crucial importance. This proposed solution lists five applicable agents: client agent, information agent, process agent, wrapper agent and directory facilitator.

Each of these agents plays a certain role in this agent society. Client agents provide an interface to users and offer them supervising services. However, they first have to receive data from a source via information agents. Process agents monitor physical components (e.g. sensors) for changes and send data to information agents. It might be significant to say that these process agents are at the lowest hierarchical level. Data from physical components are translated to common format by wrapper agents. Finally, a list of available agents and their services are administrated by FIPA standardized directory facilitator [11].

Possibilities of application of MAS in batch processes were also studied by Seilonen et al. [45]. He recognized that MAS could potentially be applied to increase failure tolerance of batch processes. He argues that this could be achieved by introducing two modules for fault handling: fault detection and diagnostics and fault recovery. Every agent in the system monitors status of physical components within its domain. Values collected from the physical components are periodically scanned by the aforementioned module fault detection and diagnostics. Detection of failure inevitably leads to initialization of fault recovery. The procedure of fault recovery involves actuators which help the system achieve a failure-free state.

Different approaches that were described in this chapter show some of developments in this field. However, they represent only a portion of various applications of MAS in batch process automation. This thesis will focus on one approach with special diligence. That is the approach which utilizes the automation agents with reflective world model as described by Vallée et al. for usage in transport domain [49]. These automation agents control mechatronic components and use representations of their surrounding. This surrounding is represented as world model, which is called reflective as it contains information about the specific agent itself. Chapter 3 deals with this topic in more detail.

2.4.5 Ontologies

It is worth stating at this point that world models of automation agents use ontologies to describe themselves. In order to understand the concept of world model one should first get familiar with this type of representation.

Studer et al. in [47] defines ontology as “a formal, explicit specification of a shared conceptualization”. In order to get a grip of this definition, first there is a need to understand what are conceptualization, formal and explicit specification and sharing.

Firstly, conceptualization is a relatively complex term that can be defined by mathematical definition for extensional relational structure [12].

> *“Extensional relational structure is a tuple (D, **R**) where D is a set called the universe of discourse and **R** is a set of relations on D.”*

It is obvious that **R** in the above definition represents a set of ordinary mathematical relations on D. These relations make a representation of a specific world. World in this context is in [12] defined as follows:

> *"With respect to a specific system S we want to model, a world state for S is a maximal observable state of affairs, i.e., a unique assignment of values to all the observable variables that characterize the system. A world is a totally ordered set of world states, corresponding to the system's evolution in time. If we abstract from time for the sake of simplicity, a world state coincides with a world."*

Admittedly, this definition of conceptualization involving extensional relational structure is rather simple and incomplete. However, it should be sufficient for the understanding of further sections.

Secondly, for expressing world states it is necessary to choose a suitable language capable of this task. A language is considered suitable if it can formally and explicitly specify relations between entities in a world. In this context, formal language specifies facts that can be interpreted by a machine and explicit means that all participants of the world are explicitly defined. For further particulars about language characteristics may the reader be referred to technical literature e.g. [2] and/or [1].

Finally, there is the third term in the definition of ontology that needs to be explained, namely sharing. Ontologies are used to define knowledge of world. This knowledge is exchanged and shared between and within a system [51]. In order to share the knowledge all entities need to be familiar with the ontology language.

This brief introduction to ontologies will hopefully serve as basis which will help the reader understand the contents of Chapter 3 and reflective world model of automation agents.

2.5 Conclusion

Previous sections of this chapter delivered an insight into a special aspect of process automation, namely MAS based distributed systems for batch processes. It started with introduction to automation systems and technical processes and continued with a special process type, the batch process. Further text mentioned various advantages of batch processes like flexibility and failure tolerance as well as disadvantages like complexity due to constant work in non-steady state. It has been shown that batch processes were standardized with IEC 61512 standard. In view of these facts, it is understandable that the batch processes are recognized as a convenient and promising paradigm in process automation.

Among wide range of programming techniques used in batch processes, this chapter described the technique based on IEC 61499 FBs. With characteristics like encapsulation, modularity and reusability FBs provide basis for engineering of distributed systems. As it is shown, it is possible to group functions into one FB and deploy them either on one or more devices. Communication between devices is performed via network infrastructure.

This concept of distributed systems can be expanded with MAS. As a result, each equipment part can be represented as an autonomous agent.

With special approach to using automation agents in batch process, process automation control systems are becoming distributed, more flexible and failure tolerant. Doubtlessly, such systems will play an important role in the future of manufacturing.

3 Multi-Agent Systems for Batch Processing Automation

This chapter explains a special approach applied in batch processes that is based on Multi-Agent Systems (MAS) illustrated in [49] and [29]. *Automation agents* stand in the center of this approach. Their main responsibility is controlling the equipment. Automation agents can also be represented as abstract groups of other automation agents. In that case, abstract automation agents could control equipment directly or alternatively over sub-agents.

Aside from automation agents this approach considers three more types of functional agents, specifically *order agents*, *task agents* and *work agents*. Their tasks are communication with users, finding available hardware for task execution and execution of given tasks by managing automation agents, respectively. Furthermore, there are also *Directory Facilitator agents* (DF agents), responsible for managing lists of services managed by each individual automation agent. It will be shown that, if necessary, additional agents can be implemented. In case of need for failure detection and recovery, *failure handling agents* can also be added to the system [37].

The remainder of this chapter is outlined as follows. In the beginning sections, all agents are individually explained in more detail. This is followed by a description of workflow and communication between agents. Finally, applications of world model and ontology as well as their relation to automation agents is illustrated. This section serves as a theoretical introduction to Chapter 4, where the implementation of this approach on a real plant is presented.

3.1 Architecture of Multi-Agent Systems

As we have already seen, MAS are agent societies composed of several agents, those being functional agents (order agents, task agents, work agents, failure handling agents, DF agents) and automation agents. Each of these agents has different responsibilities. This section describes functional and automation agents in more details.

3.1.1 Functional Agents

Order Agent

MAS usually contain one order agent within their structure. Order agents are functional agents and as such consist entirely of software. Their main task is to receive messages from users or other systems (in case that an end product depends on more systems), stating what output needs to be produced. In the next step, this order agent searches for the corresponding recipe and creates a job that contains information on product type, amount and job ID.

If a recipe for the desired product is found, it is sent to a task agent, because these agents are responsible for execution of duties specified in the recipe. After all tasks have successfully been performed, the task agent informs the order agent that the recipe is finished and a batch is produced. Thereupon the order agent gives information about successful completion of the task to the user or the other system and manufacturing of the next product can start. In case that the recipe could not be completed successfully, the user or the other system has to be correspondingly informed.

Task Agent

Task agents belong to the group of functional agents and play an important role in MAS. They receive recipes from order agents. In order to produce a batch according to a given recipe, task agents first need to find automation agents necessary for the task. For this purpose, a list of all automation agents and services they perform has to be obtained from a DF agent. If all required agents are available, task agent creates tasks according to the recipe. Those tasks can either be production or transport. These tasks are later sent to a work agent, because they manage automation agents.

In a case of a transport task, the task agent passes information about the source and the destination of the material that is to be transported to a work agent. In most cases it is also necessary to provide some additional information, like for example required amount of the material. Production tasks are mostly performed in one place e.g. inside of a reactor. A production task could for instance be a chemical reaction that would require some sub-tasks like mixing different materials, heating, cooling and so on.

Upon successful completion of one task the next one starts by creation of a new task and transmission of a message to the work agent. In case that the work agent reports a failure, the current task is stopped and a failure description is sent to a failure handling agent with a request for troubleshooting. If a solution to the failure can be found, the task agent informs the work agent and the execution of the recipe can continue.

After all of the tasks have been performed, the task agent informs the order agent that the recipe is completed. In case that the task agent was not able to finish all of the tasks successfully, the order agent is accordingly notified.

Work Agent

Work agents manage automation agents and communicate with them. They receive tasks from task agents. These can either be transport or production tasks. After receiving a transport task with associated source and destination information, work agent generates a path using one of the path finding algorithms (e.g. Dijkstra algorithm [26]). Ultimately, the work agent sends messages to automation agents in order to have them make changes on hardware components. These changes can be for instance opening a valve, starting a conveyor belt, etc.

If the work agent receives a production task, it sends messages to the corresponding automation agents and gives them instruction about activities that they need to perform.

When all activities involving one particular task are finished, the work agent passes the information on to the task agent. In case of a failure, a corresponding error message is forwarded from an automation agent to the task agent. In the adverse case a confirmation about successful completion of the task is sent.

Failure Handling Agent

Failure handling agents' main function is bringing system back to normal if such necessity presents itself. They deal with solving anomalies that can appear in a system. As was already mentioned, if an automation agent cannot perform a given activity, it is marked as having a failure and a work agent gets informed. It then notifies a task agent, which as a result stops the ongoing task and sends an error message to a failure handling agent. The failure handling agent researches its knowledge for a solution of the error. If the solution is found, the task agent is informed and the workflow continues.

Directory Facilitator Agent

DF agents manage lists of automation agents and their services. This type of agents and its functions is defined by FIPA in [11].

Every automation agent can register itself with this agent at the startup. The DF agent is also familiar with the services provided by each of the automation agents. Similarly, if an automation agent determines that if has a failure, it can be deregistered from DF agent.

3.1.2 Automation Agents

As already mentioned in previous sections, automation agents control physical components in production systems. Whereas it is common for functional agents to be represented in a system with only one agent of each above mentioned kind, the situation with automation agents is such, that usually each part of the equipment is represented by one individual agent.

Automation agent architecture usually consists of two layers. These layers are called High Level Control (HLC) and Low Level Control (LLC). One should note here that there are automation agents that do not incorporate LLC, as it will be explained in further text.

HLC contains world model repository and is on the one hand responsible for communication with other agents and on the other hand for decision making. LLC contains an interface to the physical component. Through that interface, a LLC that is deployed on embedded controller is in charge of direct hardware control.

Classification of automation agents is illustrated in Fig. 3.1. One can see that for the purpose of this approach they have been divided into three groups. First group of automation agents (Fig. 3.1a) are PHL automation agents and they consist of physical components, high level control and low level control (PHL). Second group is called PH automation agent (Fig. 3.1b), because it does not contain a low level control. The last group are HL automation agents and they consist solely of a high level control and a low level control (Fig. 3.1c). Each of these automation agent types is explained in more detail in further text.

PHL Automation Agent

This type of automation agents is the most common one. It consists of a physical component, a high level control and a low level control (Fig. 3.2). These automation agents usually control only one part of equipment like for example a sensor or a valve. In that case PHL automation agent is responsible for collecting data from the sensor or for opening or closing the valve. Further, it can generally perform all activities related to interaction with mechatronic devices. In this approach, each mechatronic device can have its own PHL automation agent.

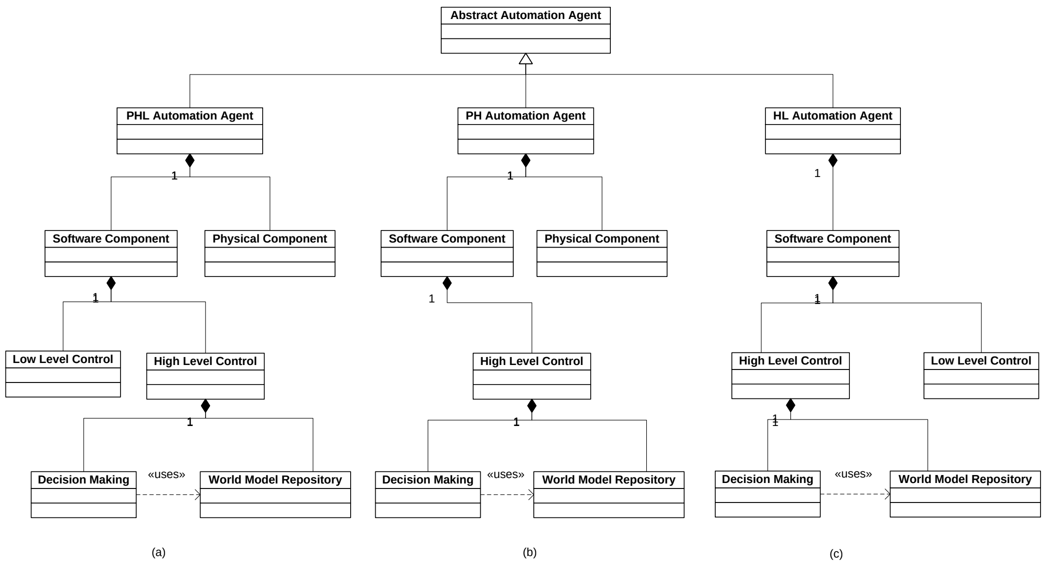


Figure 3.1: Types of Automation Agents

PH Automation Agent

PH Automation agents consist of a physical component and a high level control. The physical component in this case does not have to be a mechatronic one. An example of this automation agent type would be an agent representing a reactor (reactor PH automation agent). Reactor is a physical component, but it has no other purpose or ability to do something except holding the material in itself. Nevertheless, such agents play an important role in this specific agent based approach.

This type of agent aggregates a group of other automation agents. These can be sensors, mixers, heaters, etc. For example, the reactor PH automation agent groups all devices located inside of the reactor, like sensors and tools (heater, mixer, etc). PH automation agent controls activities performed within reactor using other agents. In addition, one of the functions of the reactor PH automation agent is to provide a communication interface for functional agents and other automation agents. This goes to show that communication of devices within the reactor with each other and with other devices outside of the reactor flows through the reactor PH automation agent (Fig. 3.3).

Admittedly, functional agents generally communicate directly to PHL and/or HL automation agents as long as they are not grouped by the PH automation agent. However, if PHL and/or HL automation agents are grouped by a PH

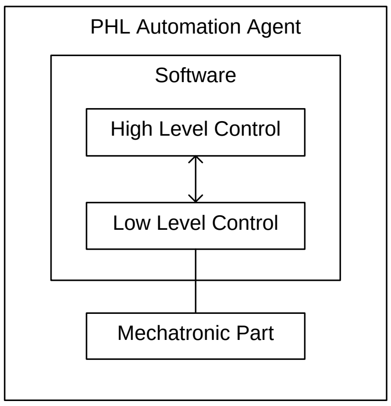


Figure 3.2: PHL Automation Agent

automation agent, the communication flows through the PH automation agent (see Fig. 3.3).

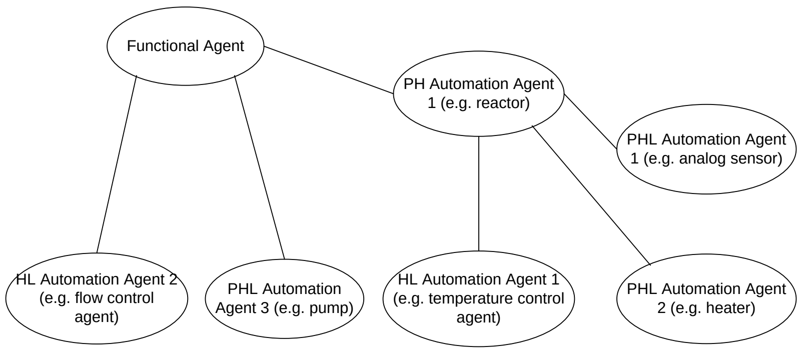

Functional Agent
PH Automation Agent 1 (e.g. reactor)
PHL Automation Agent 1 (e.g. analog sensor)
HL Automation Agent 2 (e.g. flow control agent)
PHL Automation Agent 3 (e.g. pump)
HL Automation Agent 1 (e.g. temperature control agent)
PHL Automation Agent 2 (e.g. heater)

Figure 3.3: Communication between PH Automation Agent, Functional Agent and other Automation Agents

PH automation agent is a rather complex type of automation agent. It usually has a whole world model implemented in its knowledge, in contrary to simple automation agents, like sensors or valve automation agents. Its duty is to manage various activities that run simultaneously. One of the characteristics that helps it perform its tasks is its ability to detect anomalies in a system. For instance, it can detect if a reactor is full, so it prohibits other material from entering it. This execution of activities falls in the domain of world models and will be discussed in further sections.

HL Automation Agent

The last type of automation agents that is discussed here is HL automation agent, as shown in Fig. 3.1c. This agent consists only of software components HLC and LLC. It does not have any physical representation and it is normally used to provide a real-time capability to the system. In the contrary to PHL automation agent, that also provides real-time capability, HL automation agent does not communicate with mechatronic devices directly but through LLC of PHL automation agents.

An example of HL automation agent is temperature control agent (Fig. 3.4). This hysteresis controller is used to keep temperature between two values.

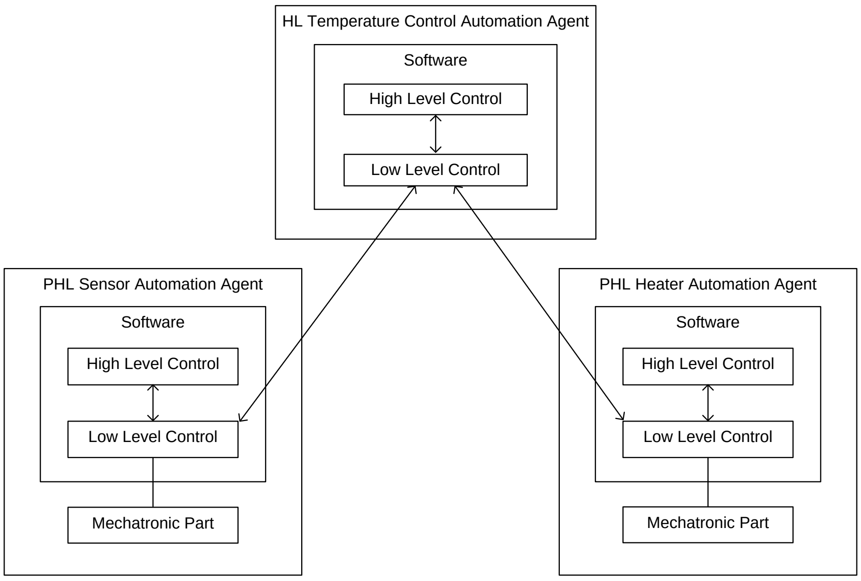


Figure 3.4: Application of HL Automation Agent with PHL Automation Agents

LLC of all three agents illustrated in Fig. 3.4 are deployed on one or more controllers. It can be seen that each of these agents also contains HLC, but the communication between them is performed through the LLC.

3.1.3 Production Workflow Using Multi-Agent Systems

It has been mentioned in the previous sections that agents exchange messages among themselves in order to perform given tasks. These messages contain all relevant information for task execution. Admittedly, they can be written in any content language, but one of the most popular languages is XML, which enables easy transmission of different information using only one string in *message content*.

Tasks and responsibilities of individual agent types were already explained in Sec. 3.1.1 and 3.1.2. Additionally, Fig. 3.5 illustrates a workflow of batch production using MAS in form of UML sequence diagram.

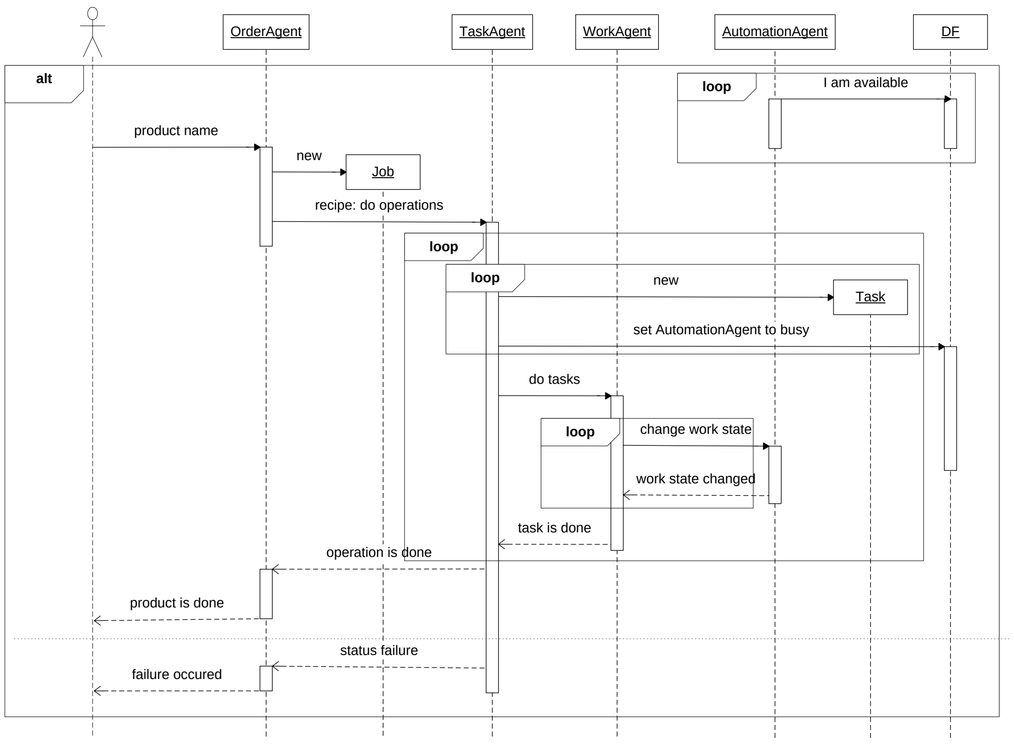


Figure 3.5: Production Workflow [29]

3.2 Architecture of Automation Agents

As already mentioned in previous sections, automation agents control hardware components in production systems. That control can be executed directly, given that automation agents contain physical components and LLC. Alternatively, it can be performed indirectly over HLC or LLC of other automation agents. HLC and LLC have different realizations and responsibilities. They are explained in the following.

3.2.1 High Level Control (HLC)

HLC is present in every automation agent's architecture. It is responsible for general behavior of agents and their communication with other agents. It also receives and sends messages as a part of information exchange between agents.

HLC mainly consists of following modules, as shown in Fig. 3.6:

- World model repository containing a reflective world model.
- Low level interface, responsible for establishing communication with LLC and providing its functions. This interface is also used for data transfer from and to LLC.
- Communication manager that is responsible for communication with other agents.
- Decision making is responsible for reactions to changes on plant or within individual agents. This part represents business logic of automation agents. Decisions are made considering requests from other agents, world model and state of LLC.

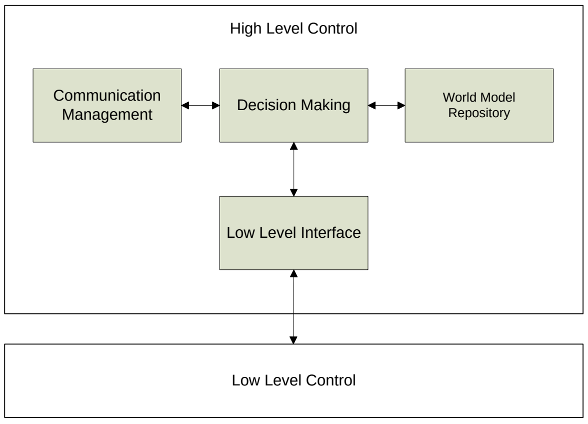


Figure 3.6: High Level Control [50]

HLC is programmed in a high level programming language. Even though this will be explained in more details in Chapter 4, it is worth stating at this point that JADE-framework has been used to program HLC for the purpose of this thesis. JADE stands for Java Agent DEvelopment framework, middleware used for development of agent based applications [4] and it is fully in accordance with FIPA standards [9]. All agent types explained in the sections above run on JADE platform. Agents run autonomously on JADE platform and they are able to communicate with each other by exchanging messages using Agent Communication Language (ACL).

3.2.2 Low Level Control (LLC)

Not all automation agents contain LLC. Only agents that are used to directly control one or more mechatronic devices have LLC in their architecture. These are particularly PHL and HL automation agents.

LLC has two responsibilities. For one, like in case of simple devices (valve, mixer, etc.), it forwards data from the HLC to the device. By doing this, they provide an interface between hardware and the HLC.

The other responsibility is providing the real-time capability. LLC is usually deployed directly on a controller and it is in direct contact with the device. The signal from and to the device does not have to travel over a network. That is why all functions that need to work in real time are implemented in LLC. These functions are for instance a PI-Controller or a hysteresis controller. There are also some cases where a reaction to an event is needed promptly. Case in point would be an abnormal activity from some device, for instance a hysteresis controller. In case that *failure detection and handling unit* (which is in this case also implemented in LLC) detects a failure of a sensor that controls temperature, the heater needs to be turned off immediately in order to prevent more damage [37].

LLC control can be programmed in any programming language mentioned in Sec. 2.2.4. The only requirement is that it can be deployed onto a controller. But, according to the approaches based on [29] and the one explained in this thesis, LLC is realized with IEC 61499 FB. The main reason for that is the fact that IEC 61499 supports distribution over several controllers, as MAS also work on distributed systems basis. LLC applications can be distributed over several resources over one or more devices. There are also cases where LLC employ SI function blocks in order to establish communication between two different resources. Additionally, an interface to hardware is provided through analog and digital I/Os. They can collect data from sensors and provide data to actuators.

3.2.3 Communication between HLC and LLC

For the purpose of this thesis, HLC is implemented in JADE and LLC was realized with software based on the IEC 61499 FB standard. HLC is started on a PC and FBs were deployed on a controller (see Chapter 4 for more details). Communication between them is done over a local network.

Since these two realizations are different, a special interface is needed in order to establish the right communication. The interface was designed in accordance with the principle *"separation of concerns"* [36]. First, both entities need to be in the know of the communication channel, since they have to use the same one in order to be able to establish the connection. Secondly, one should note here that the communication between HLC and LLC is asynchronous. IEC 61499 supports event-triggered communication and upon receiving a request the execution in LLC is started. As such it is suitable for this type of communication. The execution in HLC is done in a similar manner.

Finally, it is important for both HLC and LLC to understand the data that they transmit, for the data needs to be interpreted in the correct way. LLC sends the data to HLC in IEC 61499 data types. Casting to a data format recognized by HLC is done directly in HLC, as such changes are easily done with high-level programming languages, like JAVA.

As was already mentioned above, the implementation in LLC is deployed on a controller. Additionally, a FB network is connected with the channel to HLC over SI function blocks. Listeners on HLC are constantly active. Therefore, if an event occurs, the corresponding behaviors are started. As HLC runs on PC, a network based connection needs to be established. In order to establish a connection HLC and LLC both use standard protocols like UDP or TCP/IP. There are also some other ways to establish communication between HLC and LLC, like using other protocols (for instance industrial Ethernet protocol) or deploying HLC on a controller too. These implementations are out of scope of this thesis and for more details see [36] and/or [34].

3.2.4 Communication between Two Automation Agents Realized in LLC

As already mentioned, sometimes real-time capability of agents is required. This is for example the case when some parts of HLC functions need to be migrated to LLC. The communication between automation agents can be done either within the same resource and the same device, or it can be distributed over a system. In case of distribution, automation agents use SI function blocks (*publisher* and *subscriber*) to communicate with each other. As a rule, SI function blocks are initialized with exact IP and port information in order to be able to establish communication with the desired automation agent [32]

[29].

The realization of communication between agents in LLC is required as HLC does not have a real-time capability. According to the tests performed by Lopez et al. it takes approximately 2 seconds for the LLC-HLC-HLC-LLC communication to be performed [35]. Even though the tests performed during the implementation of this thesis show different results, the implementation of functions that require a real-time capability are still implemented in LLC. For more details see Sec. 4.6, 5.3 and Chapter 6.

3.2.5 Reflective World Model

With creation of automation agents and distribution of these agents over systems, these have become more flexible and fault tolerant. With the distribution, automation agents become autonomous entities and require and contain knowledge only about their environment because they are not part of centralized and monolithic systems.

It was mentioned that HLC of each automation agent consists of four components, namely communication management, decision making, low level interface and world model repository. Repository denotes a place where information about production environment is stored. Due to this knowledge, automation agents are familiar with static situation around them and their environment. They are also familiar with the activities performed by them and other agents in the environment [49].

An important aspect of this approach to storing information in a world model repository is that, as agents are distributed and can run individually on different controllers, they hold information about themselves within those repositories. Each automation agent is aware of the position it holds within the system and of its relations to other agents. It is also aware of the activities that it can perform. Due to these factors, this model of knowledge is called reflective world model.

World model has many advantages. For one thing, each agent only contains knowledge relevant for its performance. Furthermore, the whole system is not described in one place, but decentralized over several agents. Above all, one of the most important advantages of world model is its assistance in detection of anomalies [37].

3.2.6 Contents of World Model

In order to obtain information about its environment, automation agent needs to know its position in relation to other agents. It also requires information regarding activities that affect other agents as well as the activities that affect itself. In order to provide these information world model contains two submodels [50] (Fig. 3.7):

- Situation model describes current agent situation. Agent situation represents its physical (i.e. height, length, etc.) and functional characteristics (i.e. ability to heat, mix, etc). Situation model consists of ontology and facts.

  Ontology model is similar to UML class model. It describes entities that are relevant to automation agents and relations between these entities. These entities are usually other agents that have certain relations to the automation agent.

  Facts model describes current situation specified by an ontology model using a language defined by the ontology. As an example, facts state that an automation agent *T101* represents a reactor and that the reactor *T101* contains an analog sensor *B101*. In contrary to ontology model that describes only general knowledge of that system (actual part of the system where the automation agent is involved), facts model describes the current situation of the system (at the beginning). Due to constant change in batch processes facts model might change during production.

- Activity model describes potential activities performed by the automation agent (itself) and also expected activities performed by other agents. Potential activities are represented abstractly and they are categorized into sets of activities. Actions are organized hierarchically and can also be categorized into subclasses. Every level of subclass specifies additional information about an activity.

  For instance, it can be said that import of material is a subclass of activities defined in a reactor world model. Additionally, this activity can be refined by being divided into two sub-activities. Those two sub-activities are import from a valve *V101* and import from a valve *V104* (in the case when reactor has two entries).

  The second part of activity model is reserved for expectations and observations. Agents have an overview of ongoing and scheduled activities and are aware of them. These expectations and observations are classified by type of activity and timing when they are supposed to start or finish. Observations define rather precise timing of occurrences, whereas expectations are conditioned by the timing given by the observations. This can be illustrated if for instance two reactors (*T101* and *T102*) are given and only one of them has an analog sensor for measurement of material level. A pump, an analog flow sensor and a valve could also be added between them. An expected result of transporting material from the reactor *T101* to the reactor *T102* is specified in their world models. At the time *(t0, t0)* beginning of material export from *T101* is observed and consequently at the time *(t1-1, t1)* finishing of material import into *T102* is expected[2]. These observations are performed using change of value returned by analog sensors. It is expected that no failure of valve or pump between these two reactors occurs until both of these activities are done (time *t1*).

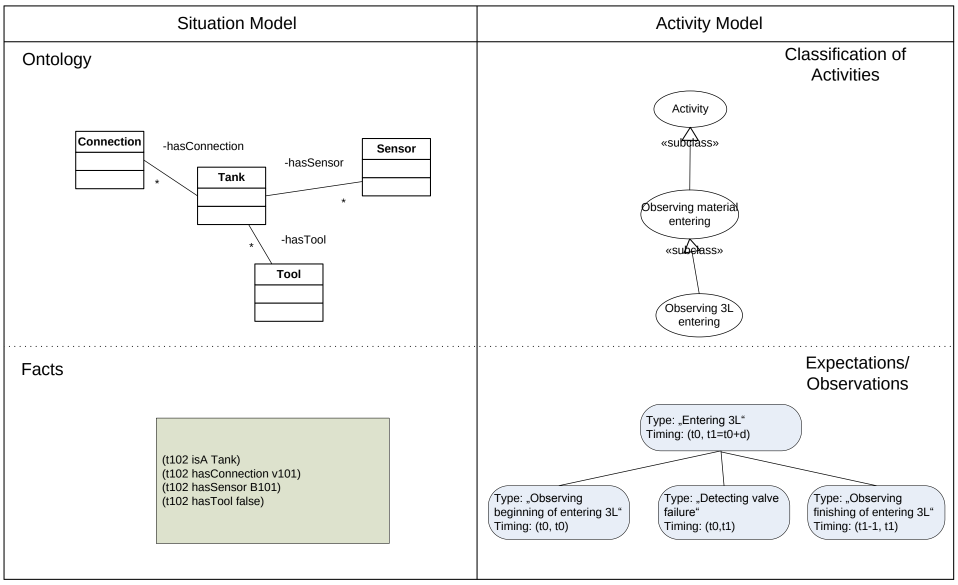

Situation Model
Activity Model
Ontology
Connection
-hasConnection
Tank
-hasSensor
Sensor
-hasTool
Tool
Classification of Activities
Activity
«subclass»
Observing material entering
«subclass»
Observing 3L entering
Facts
(t102 isA Tank)
(t102 hasConnection v101)
(t102 hasSensor B101)
(t102 hasTool false)
Expectations/ Observations
Type: „Entering 3L" Timing: (t0, t1=t0+d)
Type: „Observing beginning of entering 3L" Timing: (t0, t0)
Type: „Detecting valve failure" Timing: (t0,t1)
Type: „Observing finishing of entering 3L" Timing: (t1-1, t1)

Figure 3.7: Simplified World Model of a Reactor

These two above mentioned models represent reflective world models defined in agents' knowledge. They are different, as situation model represents a rather static view of the system and activity model describes a dynamic model that includes timing and activities of agents. Through this different approach to information from the environment, agents are able operate autonomously in MAS.

[2]Time (t1, t2) describes the time interval between start time t1 and end time t2.

3.3 Failure Detection and Recovery in MAS Based Batch Processes

With help of its four aforementioned entities, HLC monitors changes in a system and reacts upon them. This observation of every change helps HLC to detect anomalies. Every automation agent using its world model detects failures that are found in its domain, which depends on whether they are PHL, PH or HL automation agents. Besides detection of failures, agents can also isolate them. If an automation agent detects that is has a failure, all activities that are affected by it stop. The corresponding activities are also isolated and cannot be performed anymore, unless the isolation is eliminated. If one automation agent performs only one activity, than the whole automation agent has to be isolated and cannot be used. The isolation can be eliminated if there is no failure with the hardware. This is done by setting the *failure* parameter to false. The failure is set to false when a message containing this information is received from a user or from other agents.

Detection of failures can be performed in two different ways. One of them is detection of anomalies in LLC. This type of detection is usually present in HL automation agents. It has already been mentioned that HL automation agents are mostly used for real-time requirements. A good example would be a scenario with a reactor that contains a heater and an analog temperature sensor. A hysteresis controller is implemented in LLC in order to avoid a long signal travelling time needed for the transmission to HLC and then back to LLC. A better solution is to set a watchdog in LLC responsible for monitoring the temperature. In that case the temperature set point is sent from HLC and depending on the current temperature the heater is turned on or off. Lack of change in current temperature after receiving the set point shows either a failure with the heater of with the sensor.

In some cases detection of failures in LLC is not sufficient. In such a situation failure detection can be performed in HLC. As it was mentioned, HLC uses a world model repository as its knowledge basis for decision and communication to other agents. Expectations and observations from the activity model can be used for detection of failures. Observations and expectations can be used in the case explained above, with the heater and the sensor in the reactor. If the current and the desired temperatures are known, then the heating process can be observed. If after a defined amount of time, the set temperature is not reached then there is a failure either with the sensor or with the heater, as possible failure of communication infrastructure is not considered.

While detection and isolation of failures is done locally within each agent, the third step, recovery from the anomaly, does not belong to the domain of automation agents. Recovery from the failure can mean two things: bringing the agent with failure to a failure-free state or trying to bypass this failure by avoiding using of the faulty agent. An example for the first one is, if a reactor agent figures out that it is full and cannot accept more material, an agent responsible for failure recovery solves that problem by giving a task to a work agent to transport some of the material to other tanks. An example of the second one is if an agent detects that a heater in a reactor is not working, then the solution is to transport all material to another reactor with a fully functional heater and heat it up there.

In this way, failures within every agent and within the whole MAS can be detected and recovered. This approach is partially based on [37]. In the next chapter, some of the failure scenarios including detection and recovery will be discussed and it will be explained how it is dealt with on a real system.

3.4 Conclusion

This chapter discussed an agent based approach to batch process control. This approach is based on automation agents that contain world model repository according to [49] and [29]. Additionally, some new aspects of this approach have been considered.

It has been shown how different agents communicate with each other. In a nutshell, a user gives product information to an order agent, which controls automation agents via task and work agents. It has also been shown that agents are autonomous, flexible and independent from other agents. Generally, having one automation agent for each part of hardware makes a system failure tolerant, which is one of the most desired and needed characteristics in batch processes.

Automation agents consist of two layers, HLC and LLC. Each of them has a different domain and responsibility in a system. By dividing control of the hardware and decision making to LLC and HLC respectively, the system becomes more modular and easier to maintain, extend and also adapt for changes. With this concept, agents are able to detect, isolate and, with help of other agents, also recover failures in the system.

In the following chapter, application of this approach on a real system will be discussed. Each of the above mentioned components are implemented and tested on a laboratory plant in Odo-Struger Laboratory at ACIN, Vienna University of Technology. For more details see Chapter 4.

4 Implementation of Automation Agent Architecture with Reflective World Model on a Laboratory Process Plant

Previous chapters delivered analysis of theoretical background for implementation of the discussed approach. Also relevant for this issue is testing of its fitness for application on an actual plant. This phase was carried out in the Odo-Struger laboratory at the Automation and Control Institute (ACIN), Vienna University of Technology. Before the obtained results are presented, it is necessary to illustrate what kind of software, hardware and plant was operated for the purpose of the experiment. Furthermore, it is necessary to describe automation agents with their world models, since their characteristics can vary depending on many factors. It will also be shown how each automation agent communicates with other agents in order to perform a given task. Final sections will be dedicated to the execution of a defined recipe.

During the recipe execution the local network was monitored and the message exchange between HLC and LLC is analyzed. It will be shown how long it takes to send a message from a LLC to a HLC, to process that message and to send a reply message back to the LLC.

4.1 Equipment and Hardware

4.1.1 Target Plant

The plant used for implementation and testing is, as was already mentioned, stationed in the Odo-Struger laboratory at the ACIN. It is a plant that is primarily used for study purposes. Its construction is shown in Fig. 4.1.

As it can be seen, the plant consists of two reactors, in future text denoted as

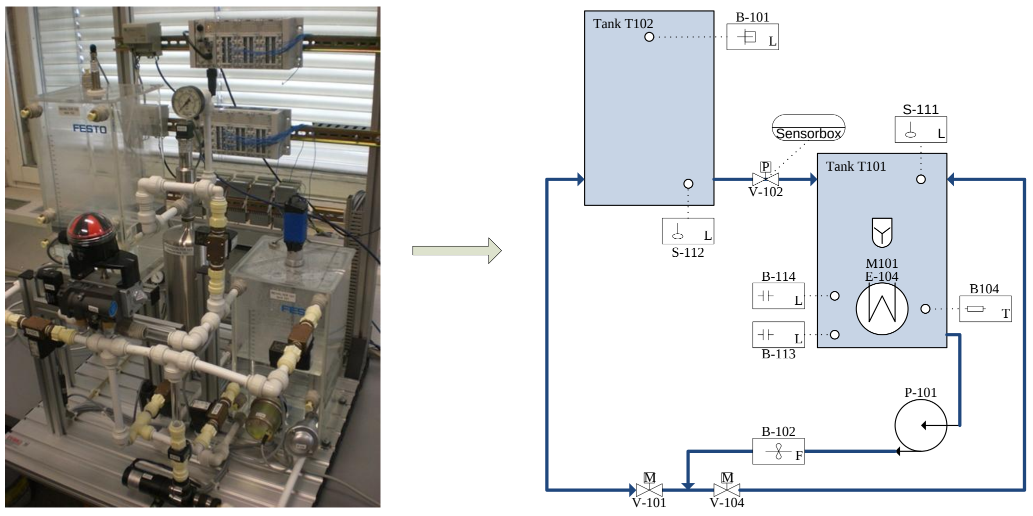


Figure 4.1: Laboratory Plant in the Odo-Struger Laboratory

tanks, *T102* and *T101*. They are connected via several different paths. First connection goes via the valve *V102*, which is a pneumatic valve containing a *sensorbox* that indicates if it is closed or open. The tank *T102* is positioned higher in relation to the tank *T101* and this path is only unidirectional. Other paths that allow material to be transported from the tank *T102* to the tank *T101* is via valves *V101, V104*. This path is only partially unidirectional and the rest is bidirectional.

The bidirectional part of the pipeline goes through the valve *V101*. This valve enables the material to be transported from the tank *T101* to the tank *T102* using the pump *P101*. There is a path that enables transport from the tank *T101* back to the tank *T101* using the pump *P101* and the valve *V104*, even though it was not used during the implementation. Flow through the pump can be measured by the analog flow sensor *B102* that is positioned after the pump. The plant also contains two more analog sensors. The first one, the distance sensor *B101*, is positioned in the tank *T102* and performs the task of showing the level of material in that tank. The other one is the temperature sensor *B104* in the tank *T101*.

Perhaps it should also be mentioned that the plant also contains a mixer (*M101*) and a heater (*E104*) positioned in the tank *T101* as well as four digital sensors that indicate if the material in the tanks reached the defined level or not.

Even though the observed plant is rather simple and mainly suitable for studying purposes, it was sufficient for the purpose of this experiment. It was possible to implement all of the automation agents on it, as it will be shown in the further text. For a complete hardware overview see Fig. 4.2. Further details about different components and tasks they perform will only be mentioned if found necessary for the understanding of the experiment.

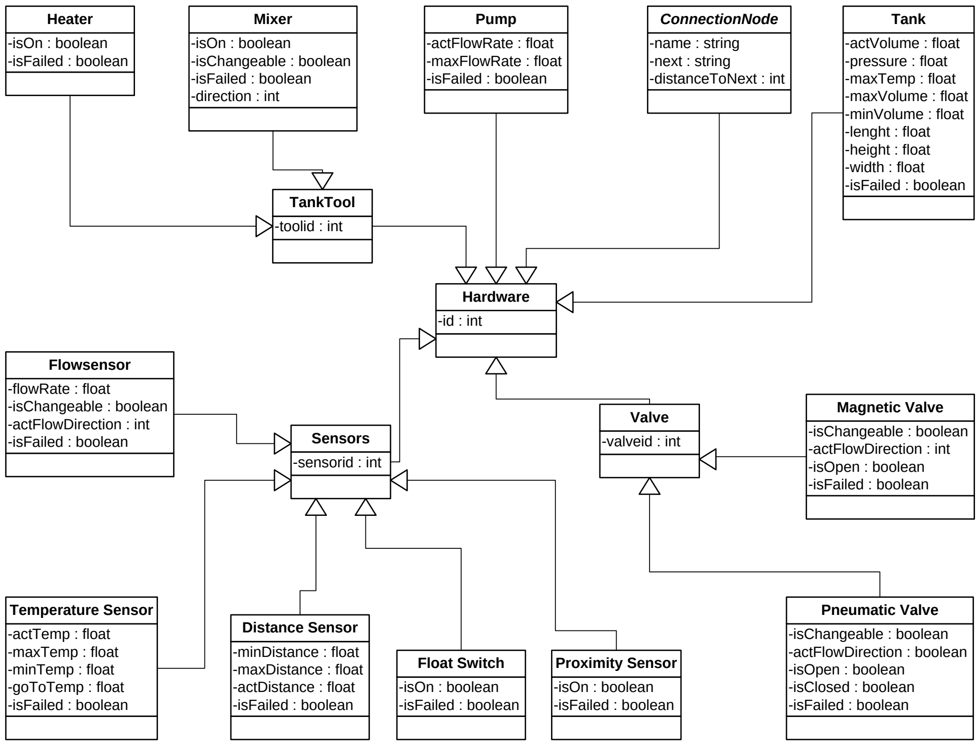


Figure 4.2: Hardware Overview

4.1.2 Control System

The whole system is implemented and run on a PC and two controllers, connected to the same local network. The PC is used for the deployment of a HLC including a user interface. A LLC is, even though it was developed on a PC, deployed and run on the controllers.

Controllers that are used, are CPX-CEC-C1 controllers by FESTO[3] (see Fig. 4.3). These controllers use Xscale-PXA255 400 MHz agile Intel micropro-

[3]http://www.festo.com/ext/en/10485.htm, accessed in September 2010.

cessors and operate with 28 MB Flash and 24 MB RAM. Two such controllers are used, each of them controlling a different part of the equipment. The plant equipment is connected to the controllers via analog and digital I/Os.

Figure 4.3: Controller CPX-CEC-C1 [8]

An important aspect of this system is that the controllers are connected to each other and to the PC over a local network. They are connected via digital and analog I/Os to the sensors and actuators. Each mechatronic device uses a unique *IP* and *port* for communication with other parts of the system.

4.2 Software

4.2.1 Implementation of HLC

As previously mentioned, the HLC is developed in JADE [4] and subsequently deployed and run on the PC. JADE stands for Java Agent DEvelopment framework, which is a middleware completely written in JAVA. JADE enables development of agent based applications. Programmers use JAVA and JAVA specific programming technique to develop agents. The only difference between ordinary JAVA based programming and JADE is that in JADE two distinct agents (represented as special JAVA classes) exchange data by sending messages, instead of creating new instance of the agent and calling its methods. JADE is used for this thesis in its version 3.7. However, the version 4.0.1 has been released since then and is currently available for download[4].

JADE is equipped with *runtime environment, library* and *graphical tools.* Runtime environment enables implementation of agents. In other words, agents are activated and executed in a runtime environment, that is also called *container.* Every container can have one or more agents. Active containers can be grouped to form a *platform.*

Developing agent based applications requires additional library classes as a supplement to standard JAVA classes. Those libraries contain agent specific methods used for agent initialization, communication, behaviors, etc.

Another equally important characteristic of JADE is that it enables monitoring of active agents and their activities once they start. It is possible to observe the communication between them as well as perform administrative actions.

4.2.2 Implementation of LLC

The LLC is developed in accordance with IEC 61499 FBs, as this programming technique supports development of distributed systems. For development of LLC *4DIAC* is used [53]. 4DIAC stands for *"framework for distributed automation and control systems".* It provides open-source, IEC 61499 supported framework and it secures establishment of automation and control environment.

4DIAC project is still being developed and there is a certain room for improvement. It consist of two sub-projects, *FORTE* and *4DIAC-IDE.* FORTE is implemented in C++ and it is a runtime environment that enables IEC 61499 FBs to run on controllers. 4DIAC-IDE is a development environment based on the Eclipse[5] plug-in. It is used for modelling networks of IEC 61499 FBs.

After modelling FB network and implementing required algorithms within each FB, the whole FB network is deployed on a controller. In the plant shown in Fig. 4.1 mechatronic devices are connected to two different controllers and LLC of each device is mapped to the corresponding controller. HLC and LLC communicate in the manner that was explained in Sec. 3.2.3.

[4]http://jade.tilab.com/, accessed in September 2010.
[5]www.eclipse.org, accessed in Semptember 2010.

4.3 Agent Overview and Communication between Agents

It was mentioned in previous sections that in this MAS approach a unique agent is employed for every mechatronic device. Fig. 4.4 displays all the implemented automation agents and the work agent. It also shows the communication network between automation agents. As it can be seen, not all automation agents can communicate directly with each other. Communication between agents is usually performed by employing functions of a HLC (by exchanging messages using methods included in JADE), with two exceptions to that rule. *Temperature control agent* and *flow control agent* require a real-time capability and can therefore not be performed in HLC.

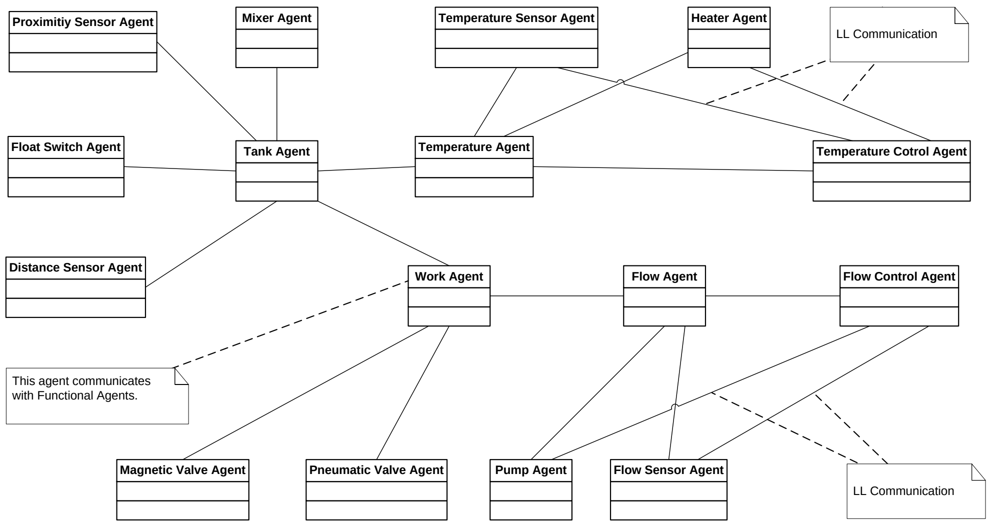


Figure 4.4: Agent Communication Network

It is evident that every mechatronic device is represented by a PHL automation agent (compare Fig. 4.4 and 4.1). Those agents are either grouped by a PH or a HL automation agent (see Sec. 3.1.2).

PHL automation agent can be a sensor or an actuator. These agents have two responsibilities: to send data to and to receive data from mechatronic devices. For instance, LLC of the analog distance sensor B101 is programmed in such a way, that it can either return the actual value from the sensor or notify if the desired predefined value is attained. Sensors can additionally have the responsibility to detect failures in LLC (see Sec. 5.1.1). Agents representing actuators can only send data to the actuator (with exception of the valve V102 that contains a sensorbox). They can perform no intelligent or any other sophisticated actions. Furthermore, PHL automation agents can only communicate to either PH or HL automation agents as well as work agent.

PH automation agent can come in form of tank, flow or temperature automation agent. It is self-explanatory that tank agent includes a physical component and a high level control, but flow and temperature agents do not have any physical representation. They belong in this group because of the similarity of their functions with those of tank automation agents. Their physical representation can be a heater and a temperature sensor for temperature automation agent, or for flow automation agent, a pump and a flow sensor. They have no direct control over these components, much like a tank agent has no direct control over a tank.

The third and the last type of automation agents is HL automation agent. There are two agents of this sort present in the system. One is a temperature control agent and the other one is a flow control agent. The temperature control agent is realized as a hysteresis controller and the flow control agent is realized as a PI-Controller. Both of these controllers are implemented in LLC, as HLC has no real-time capability. The temperature control agent communicates with LLC of the heater agent and the temperature sensor agent (see Fig. 4.4 and 3.4 for more details). In a similar way the flow control agent communicates with LLC of the pump agent and the flow sensor agent.

Besides automation agents, the system also contains functional agents. Several types of functional agents were implemented, namely order, task, work and failure handling agent. Additionally, there is also a DF agent, which is included in JADE by default and was therefore not implemented separately. The functional agents communicate with each other in the manner shown in Fig. 3.5. It should perhaps be mentioned that work agent is the only functional agent that can communicate with automation agents, as is shown in Fig. 4.4.

4.4 Reflective World Model of Automation Agents

Reflective world model as knowledge data source of automation agents was introduced in Sec. 3.2.5. It was mentioned that world model consists of a situation model and an activity model. World model of an automation agent is called reflective, as it contains knowledge specific to that particular automation agent.

A complete world model, consisting of a situation and an activity model could in this experiment only be defined for the tank automation agent. This agent is the only one (in the plant in Fig. 4.1) that was complex enough to include all sub-models of a world model. Other automation agents, like sensor or valve agent, do not contain complete world models as they are too simple.

Since the tank is the only automation agent that was suitable for the world model analysis, this example will be presented in the following. It can subsequently analogously be used for designs of other world models for other automation agents. Even though world models of some agents might not be complete, they can still be useful when for instance designing big plants that have cross-linked dependencies and requirements.

4.4.1 Reflective World Model of Tank Automation Agents

A simplified world model of a tank was already introduced in Sec. 3.2.6 and Fig. 3.7. This section gives a more detailed overview and explains complete world models of the two aforementioned tanks in the plant, T102 and T101. Even though they belong to the same group of agents, their world models are not identical. This is because they employ different sensors and actuators to perform activities within them.

Situation Model

Even though the two tanks in the plant do not have the same mechatronic devices inside of them, it is safe to say that they are rather similar. Generally, a tank includes its physical representation and all kinds of actuators and sensors inside of it within its definition. When they are individually instantiated, adequate attributes stating which sensor and actuator they have are passed to a corresponding agent.

Both of the observed tanks have the same ontology as presented in Fig. 4.5. It is observable that each tank can theoretically have infinite number of tools, sensors and connection nodes. Fig. 4.5 also shows that each tank has characteristic parameters like dimensions, minimum and maximum temperature, etc. One of the most important parameters is *isFailed*. This parameter indicates if a tank has a failure or not. When there is a failure, the corresponding tank cannot be used until the failure is eliminated. It is also possible to make more specific failure parameters, so that they can indicate the sensor or the action that is affected by this failure.

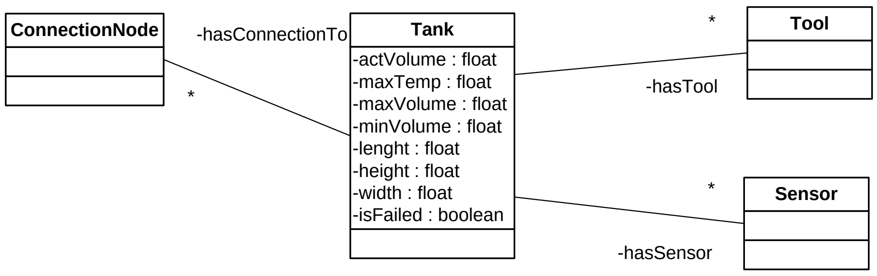


Figure 4.5: Ontology of the Tank Agents *T101* and *T102*

However, the situation model is different for the tanks T102 and T101 in its other part, called *facts*. It generally describes the initial situation of a system by giving specification of ontology (see Sec. 3.2.6). The tank T102 has following belonging facts:

```
(T102 isA Tank)
(T102 hasConnectionTo V102)
(T102 hasConnectionTo V101)
(T102 hasSensor B101)
(T102 hasSensor S112)
(T102 hasTool false)
```

The above facts state which sensors and actuators the tank T102 has at the beginning of the process. It is observable that it has two sensors, B101 and S112, two connections to the valves V102 and V101 and no tools. Facts for the tank T101 are different, as this tank contains other sensors and actuators. They are stated in the following:

```
(T101 isA Tank)
(T101 hasConnectionTo V102)
(T101 hasConnectionTo V104)
(T101 hasConnectionTo P101)
(T101 hasSensor B104)
(T101 hasSensor S111)
(T101 hasSensor B113)
(T101 hasSensor B114)
(T101 hasTool E104)
(T101 hasTool M101)
```

The facts for the tank T101 show that it has three connections (V102, V104 and P101), four sensors (B104, S111, B113 and B114) and two tools (E104 and M101).

Activity Model

The fact of the matter is surely that, since the tanks T102 and T101 do not have the same equipment, they cannot perform the same activities. The tank T101 contains a heater and a mixer, which makes it able to perform the *heating* and *mixing* activities. Activity model of the tank T102 is rather simple, as this tank contains less equipment (see Fig. 4.6). It is shown that the PH automation agent of the tank T102 cannot itself perform any activities. It can only observe changes of state and react upon them. In view of these facts, it is clear that these observations can be used for detection of failures of equipment and abnormal situations in the tank.

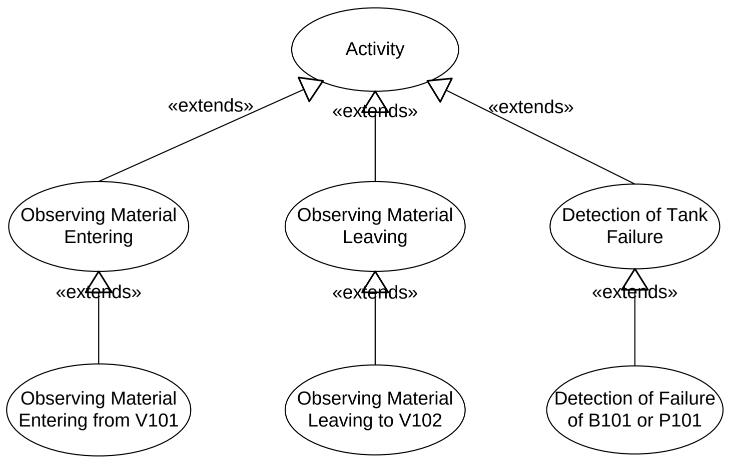

Activity
«extends»
«extends»
«extends»
Observing Material Entering
Observing Material Leaving
Detection of Tank Failure
«extends»
«extends»
«extends»
Observing Material Entering from V101
Observing Material Leaving to V102
Detection of Failure of B101 or P101

Figure 4.6: Classification of Activities for the Tank Agent *T102*

The tank T101 cannot perform any activities by itself either, since it is not an actuator. However, state of the material can be changed by utilizing tools inside of it. Change of state means that it can be heated and mixed. Fig. 4.7 shows the first part of the activity mode, classification of activities. There are two types of activities that can be performed. First one is observing activities that are used to detect failures. How detection of failures functions will be a subject of Chapter 5. The second activity type is execution of the actual activity, like heating, mixing, etc. The tanks T101 and T102 are able to detect abnormal states and other abnormalities within their domain.

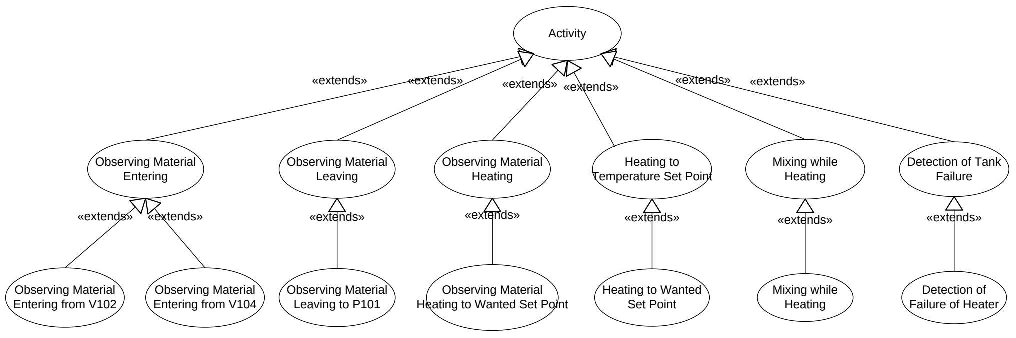

Activity
«extends»
Observing Material Entering
Observing Material Leaving
Observing Material Heating
Heating to Temperature Set Point
Mixing while Heating
Detection of Tank Failure
Observing Material Entering from V102
Observing Material Entering from V104
Observing Material Leaving to P101
Observing Material Heating to Wanted Set Point
Heating to Wanted Set Point
Mixing while Heating
Detection of Failure of Heater

Figure 4.7: Classification of Activities for the Tank Agent *T101*

Second part of the activity model describes expectations and observations of activities performed by an automation agent. Observation of activities is defined in a similar way for all activities. To illustrate this point, one only needs to refer to the activity *observation of entering from V101* (see Fig. 4.8). This scenario was tested on the plant and the results will be presented in Chapter 5. In a nutshell, if after a certain amount of time, the expected amount of the material is not transferred from T101 to T102 using P101 and V101 (part of the recipe – see sections below), then the importing activity will be stopped, as soon as the failure is discovered.

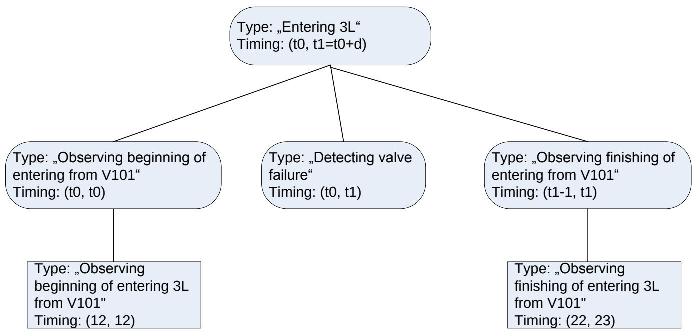

Type: „Entering 3L“
Timing: (t0, t1=t0+d)
Type: „Observing beginning of entering from V101“
Timing: (t0, t0)
Type: „Detecting valve failure“
Timing: (t0, t1)
Type: „Observing finishing of entering from V101“
Timing: (t1-1, t1)
Type: „Observing beginning of entering 3L from V101"
Timing: (12, 12)
Type: „Observing finishing of entering 3L from V101"
Timing: (22, 23)

Figure 4.8: Observation of Activities for the Tank Agent *T102*

It can be seen from the Fig. 4.8 that the activity consists of two sub-activities, *observation of activity*, which is started at the beginning and observation if *failure* will be returned as the return value from the sensors. If a failure is determined, the activity is stopped. In a failure free situation, the activity can be finished when the tank is notified that the desired material level was reached, by receiving message from the analog sensor B101.

4.5 Performing the Batch Process according to the Specified Recipe

MAS based approach that was explained in Chapter 3 as well as system design from the above sections of this chapter is implemented on the plant shown in Fig. 4.1. Every process is performed according to a defined recipe in order to obtain the desired product. The recipe that is implemented for the purpose of this research contains three tasks.

The first task is to transport 5 liters of material from the tank T102 to T101 using the shortest path, assuming there are no failures on that path. After that in the second step the material in the tank T101 is heated to the set temperature of 35 ° Celsius and mixed simultaneously. When the set temperature is reached, it needs to be sustained for 15 seconds. The last task was to transport 5 liters of material from T101 back to T102 also using the shortest path. In this section it will be assumed that the recipe is performed without failures. Failures will be tested in Chapter 5.

In order to manufacture a product by using the above mentioned recipe, the product name has to be given to the order agent. The order agent then has to find the recipe for that product in its knowledge base and sends the recipe to the task agent. Upon this, the task agent divides the recipe into different tasks and sends them one after another to the work agent (see Fig. 3.5). For the purpose of better understanding of the workflow, there will be no analysis of the message exchange between functional agents in the following sections. A message containing the task was manually input and sent from the task agent to the work agent. For this reason the first message sent from the task agent to the work agent is not shown in the figures. A graphical tool, *Sniffer*, included in JADE was used for monitoring the message exchange[4]. Several snapshots of the communication are presented in the next sections.

Figures that show the message exchange between agents contain shortcuts for agent names. *WA* represents the work agent and *TA* the task agent. Additional shortcuts that are used are *TCA* for the temperature control agent, *FCA* for the flow control agent and *Temp* and *Flow* for the temperature and the flow PH automation agent, respectively.

4.5.1 Task 1: Transport 5 Liters of Material from the Tank T102 to the Tank T101

The first task of the recipe is, as already mentioned, transporting 5 liters of material from T102 to T101. This is done through the valve V102, as this is the shortest path. For the purpose of finding the shortest path, the *Dijkstra* path finding algorithm [26] was used. In general, work agent receives a message and additional parameters from task agent specifying which action should be performed. In this particular case, the task name is *transport*, the source is the tank T102 and the destination is T101. The task is sent using XML as the content language and it reads as follows:

```
<task>
<taskname>transport</taskname>
<param1>T102</param1>
<param2>T101</param2>
<param3>5L</param3>
<param4></param4>
<param5></param5>
</task>
```

The work agent has to find the shortest path between these two tanks. When the path is found, the work agent interrogates the tank T102 if it has enough material inside to transport 5 liters to another location. It also checks with the tank T101 if it has enough place to store that amount of material. The tank T102 can assess the current amount of material inside of it by using the sensor B101, but the tank T101 can only say if it is or is not full by employing the digital sensor S111. After fulfilling all the requirements for the material transport, meaning that there is enough material in T102 and T101 is not full, the valve V102 is opened. The valve V102 contains a sensorbox that can determine if the valve is opened or not. Once the valve is successfully opened (the valve agent informs the work agent) the message is sent to the tanks, requesting them to inform the work agent when the material is transported. This is determined by the tank T102 and its sensor B101. The action of transporting material can also be stopped if the tank T101 is full which is determined by the sensor S111. For more detailed overview of the message exchange in the HLC during this task see Fig. 4.9. Once the task is performed, the work agent sends a notification to the task agent.

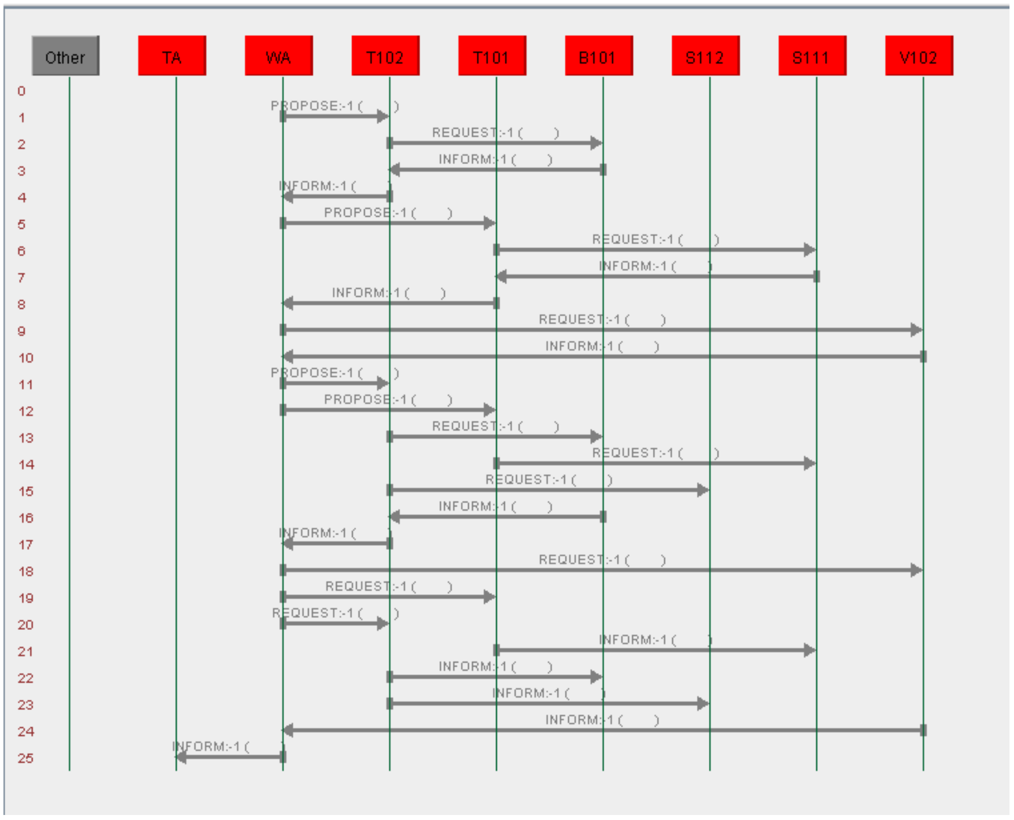


Figure 4.9: Transport from Tank T102 to Tank T101 – Snapshot of JADE Sniffer

4.5.2 Task 2: Heat the Material in the Tank T101 up to 35 ° Celsius

The second part of the recipe is entirely performed in the tank T101. The work agent receives the task from the task agent. The task contains the primary task name, *heating*, and the desired temperature. Additionally, it contains the name of the tank in which the task should be performed. The task agent knows and can pass on the tank name because this task is part of the recipe and the material was transferred to the tank T101 in the first part of the recipe (Sec. 4.5.1). In order to heat the material up equally, the mixer is used during the whole heating process. Once the set temperature is reached, it has to be sustained for a certain amount of time (15 seconds in this case). This task is also sent using XML and the message content reads as follows:

```
<task>
<taskname>heating</taskname>
<param1>T101</param1>
<param2>35</param2>
<param3>mixing</param3>
<param4></param4>
<param5></param5>
</task>
```

The above described task was tested on the aforementioned plant. The communication between agents during this recipe is shown in Fig. 4.10. It is evident that there is no communication to the heater agent in HLC. Instead, the temperature control agent is used and it uses a LLC specific communication for controlling the heater (see Fig. 3.4).

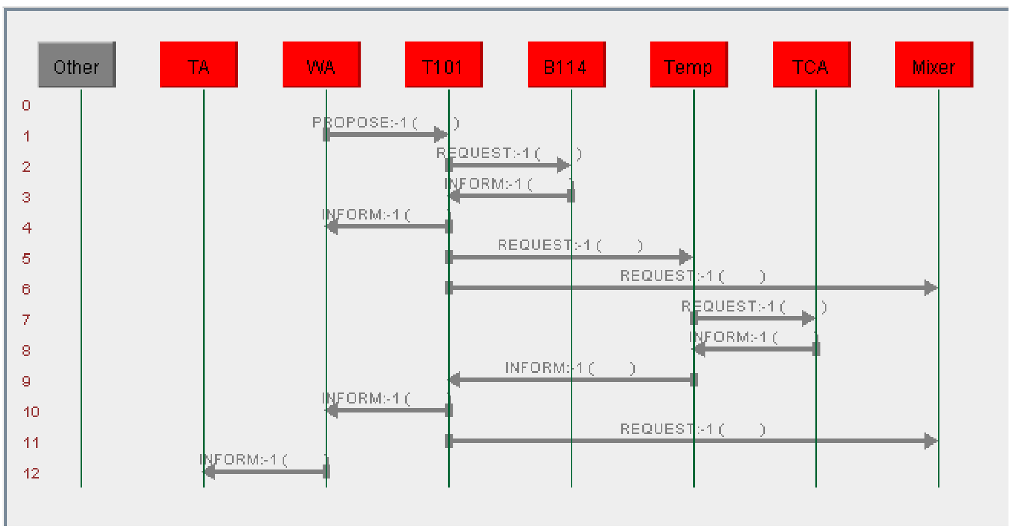


Figure 4.10: Heat the Material in Tank T101 up to 35 °C – Snapshot of JADE Sniffer

4.5.3 Task 3: Transport 5 Liters of Material from the Tank T101 to the Tank T102

The last task from the recipe is to transport the material back to the tank T102. This is another transport task and is therefore performed similarly to the first task (Sec. 4.5.1). Thus the work agent uses the Dijkstra path finding algorithm to find the shortest path. That path is found through the pump P101 and the valve V101. The task is sent using XML content language, specifying the source and the destination tank as well as the amount of material that should be transported:

```
<task>
<taskname>transport</taskname>
<param1>T101</param1>
<param2>T102</param2>
<param3>5L</param3>
<param4>fca</param4>
<param5>2L/min</param5>
</task>
```

This task has two additional parameters. One of them states that flow control should be utilized by employing the flow control agent (FCA). The other one specifies that the flow is set to 2 L/min. This control is performed in the LLC, since it requires a real-time capability. As it was previously mentioned, control of the flow can be achieved by using a PI controller.

Analogous to the first task, the work agent interrogates if the tank T102 has the capability to import the desired amount of material and if there is enough material in tank T101 before the transport is started. The tank T102 uses the analog sensor B101 to determine the current volume of the tank. The tank T101 on the other hand can only use the digital sensor B113 that gives information if the tank is empty or not. If there are no failures and the task can be performed, the agents exchange messages as shown in Fig. 4.11 in order to perform the task.

4.5.4 Usage of World Model when Performing a Task

From the above description of the recipe execution it is not obvious why the world model of each agent and especially of the tank agent is so important when implementing MAS. This section addresses this issue and analyses it more closely.

First of all, it is important to remember that a work agent can only be aware of tank agents and agents that are outside of a tank (valve and flow agent). When the work agent sends a message to the tank T102 informing it that 5 liters of its material will be transformed to another tank, it has no information about sensors that reside inside of the tank T102. In case of entering of material to the tank T101, the same message structure that is sent

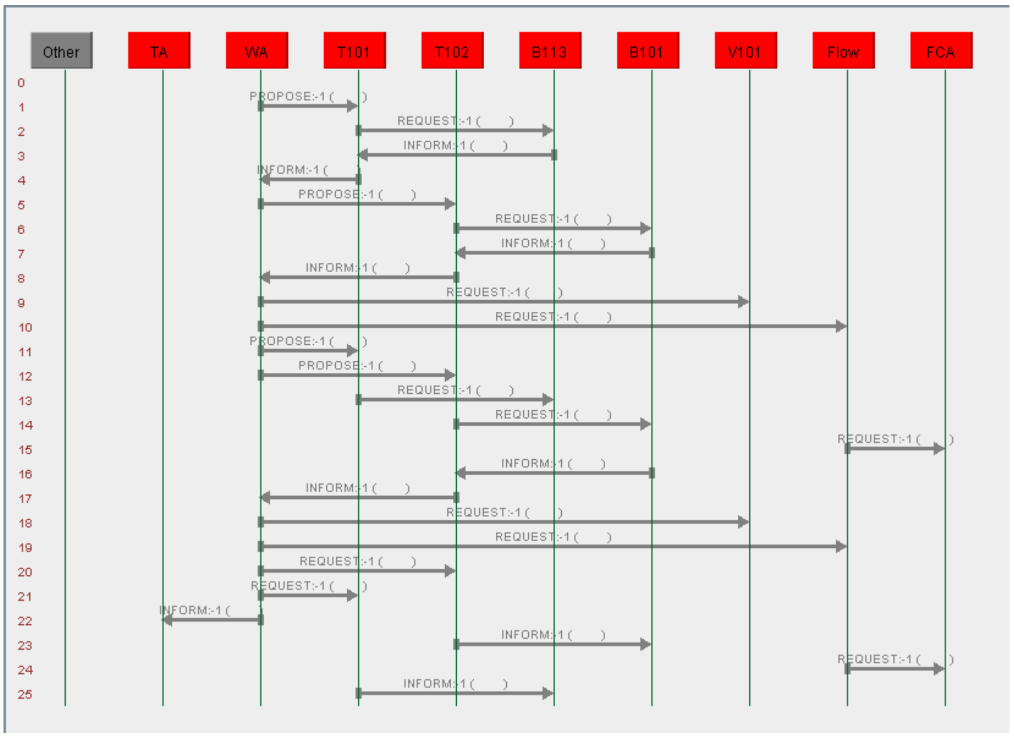


Figure 4.11: Transport from the Tank T101 to the Tank T102 – Snapshot of JADE Sniffer

to the tank T102 is also sent to the T101. After receiving the message, tank agents use corresponding sensors (and actuators if needed) for task execution by employing their world models. The message was written in XML content language and reads as follows:

```
<msg>
<action>startLeaving</action>
<value>5L</value>
</msg>
```

If there were no analog sensors in any of the tanks, the material would continue to be transported unless one of the tanks was empty or the other one was full. Tanks themselves are generally aware of the sensors that they carry and can stop an activity upon a change of a sensor state.

Tanks are also generally aware of the activities that can be performed within them. For instance, in the second above mentioned task, the work agent sent a message to the tank T101 informing it that the material should be heated up and mixed. The verification if that tank contains a heater and a mixer is performed by the tank itself after the message is received. If this request had been sent to another tank, the reply message would state that the tank cannot perform the desired activities.

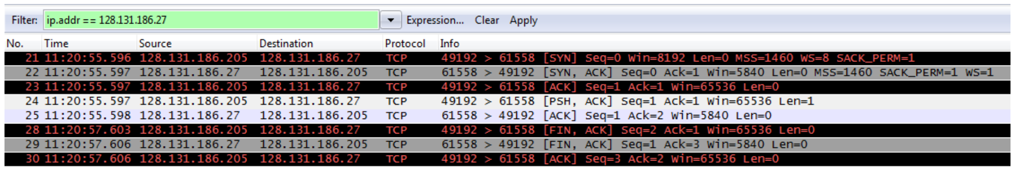
4.6 Measuring Duration of Message Transmission in a Failure Free Situation

The fact that communication between HLC and LLC is done via a local network was already mentioned in previous sections. HLC is run on a PC and LLC on controllers. Standard TCP protocol is used as a communication protocol [25] and measuring of message transmission duration was performed using a network protocol analyzer *Wireshark*[6]. Communication from HLC to LLC was tested in the following points: opening the valve V104, starting the heater and starting the mixer. The communication HLC-LLC-HLC is tested by sending the message to the sensor B101 asking for the actual value. Exchange of messages during execution of the task 1 from the recipe is measured (Sec. 4.5.1).

Fig. 4.12 shows the process of communication between HLC and LLC when a message to open the valve V104 is sent. The HLC is running on a PC that has the IP address 128.131.186.205 and the controllers have the IP 128.131.286.27 and 128.131.286.28. The valve V104 has the port 61558. Fig. 4.12 shows the whole communication including a TCP specific handshake displaying flags like SYN, ACK and FIN [25]. Flags with PSH and ACK (push and acknowledge) are relevant for the data transmission (see message number 24 and 25 in Fig. 4.12). It can be seen that it lasts approximately 1 millisecond from the moment when the HLC sends the data until the LLC sends the acknowledgement that it has received the data. Similar results are also obtained when sending data to the heater and the mixer.

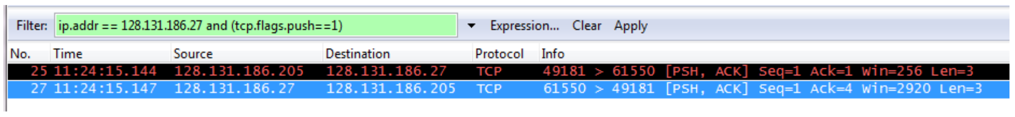
Another test that is performed is requesting an actual value from the sensor B101. The sensor B101 with the port 61550 is an analog sensor running on the controller with the IP address 128.131.186.27. Fig. 4.13 only shows messages with the flag PSH, as they are relevant for the data transmission. It is obvious from the Fig. 4.13 that it takes approximately 3 milliseconds until the LLC sends back the value requested from the sensor B101. The message number 25 displays sending of a message from HLC to LLC and the message number 27

[6]www.wireshark.org, accessed in September 2010.



```
Filter: ip.addr == 128.131.186.27        Expression...  Clear  Apply
No. Time         Source          Destination     Protocol Info
 21 11:20:55.596 128.131.186.205 128.131.186.27  TCP      49192 > 61558 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_PERM=1
 22 11:20:55.597 128.131.186.27  128.131.186.205 TCP      61558 > 49192 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1 WS=1
 23 11:20:55.597 128.131.186.205 128.131.186.27  TCP      49192 > 61558 [ACK] Seq=1 Ack=1 Win=65536 Len=0
 24 11:20:55.597 128.131.186.205 128.131.186.27  TCP      49192 > 61558 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1
 25 11:20:55.598 128.131.186.27  128.131.186.205 TCP      61558 > 49192 [ACK] Seq=1 Ack=2 Win=5840 Len=0
 28 11:20:57.603 128.131.186.205 128.131.186.27  TCP      49192 > 61558 [FIN, ACK] Seq=2 Ack=1 Win=65536 Len=0
 29 11:20:57.606 128.131.186.27  128.131.186.205 TCP      61558 > 49192 [FIN, ACK] Seq=1 Ack=3 Win=5840 Len=0
 30 11:20:57.606 128.131.186.205 128.131.186.27  TCP      49192 > 61558 [ACK] Seq=3 Ack=2 Win=65536 Len=0
```

Figure 4.12: Opening the Valve V104 – Snapshot of Wireshark



```
Filter: ip.addr == 128.131.186.27 and (tcp.flags.push==1)        Expression...  Clear  Apply
No. Time         Source          Destination     Protocol Info
 25 11:24:15.144 128.131.186.205 128.131.186.27  TCP      49181 > 61550 [PSH, ACK] Seq=1 Ack=1 Win=256 Len=3
 27 11:24:15.147 128.131.186.27  128.131.186.205 TCP      61550 > 49181 [PSH, ACK] Seq=1 Ack=4 Win=2920 Len=3
```

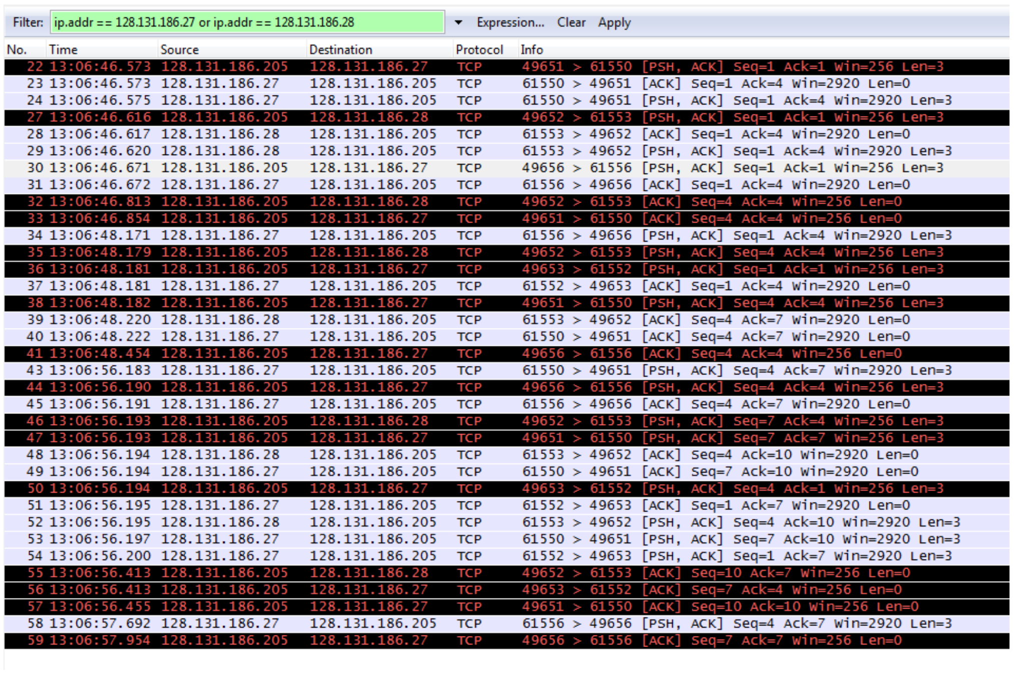
Figure 4.13: Request the Actual Value from the Sensor B101 – Snapshot of Wireshark

displays a reply to the HLC.

In order to establish the duration of data transmission between HLC and LLC more precisely the following test is performed. Network traffic during the first task of the recipe is captured (see Sec. 4.5.1). The workflow shown in this figure can be compared to the workflow presented in Fig. 4.9. For better understanding and easier following of the workflow it should also be mentioned that the sensor S111 has the IP and the port 128.131.186.28:61553 and the valve V102 has 128.131.186.27:61556. It should also be mentioned that Fig. 4.14 displays only messages that contain flags PSH and ACK, in order to maintain clarity and better understanding. When the flag PSH is set to true, then the message is sent and when the flag ACK is set, the message is received.

Message number 22 in Fig. 4.14 is triggered by the message 2 in Fig. 4.9. A message requesting data from the sensor B101 is sent from its HLC. The answer from the LLC to the HLC is sent approximately 2 milliseconds later. The message number 27 in Fig. 4.14 is triggered by the message 6 in Fig. 4.9. This message is sent in order to obtain the value of the digital sensor S111. An answer is sent back to its HLC after approximately 4 ms.

When the current values of the sensors B101 and S112 are received in respective HLC, HLC of tank automation agents calculate if material transport can start. After that a message is sent to the HLC of the V102 valve agent, indicating to the LLC that the valve should be opened. The message from the HLC of the valve V102 to its LLC is the message number 30 in Fig. 4.14 (triggered by the message number 9 in Fig. 4.9). It can be seen that the



Filter: ip.addr == 128.131.186.27 or ip.addr == 128.131.186.28
Expression...
Clear
Apply

```
No. Time         Source           Destination      Protocol Info
 22 13:06:46.573 128.131.186.205  128.131.186.27   TCP      49651 > 61550 [PSH, ACK] Seq=1 Ack=1 Win=256 Len=3
 23 13:06:46.573 128.131.186.27   128.131.186.205  TCP      61550 > 49651 [ACK] Seq=1 Ack=4 Win=2920 Len=0
 24 13:06:46.575 128.131.186.27   128.131.186.205  TCP      61550 > 49651 [PSH, ACK] Seq=1 Ack=4 Win=2920 Len=3
 27 13:06:46.616 128.131.186.205  128.131.186.28   TCP      49652 > 61553 [PSH, ACK] Seq=1 Ack=1 Win=256 Len=3
 28 13:06:46.617 128.131.186.28   128.131.186.205  TCP      61553 > 49652 [ACK] Seq=1 Ack=4 Win=2920 Len=0
 29 13:06:46.620 128.131.186.28   128.131.186.205  TCP      61553 > 49652 [PSH, ACK] Seq=1 Ack=4 Win=2920 Len=3
 30 13:06:46.671 128.131.186.205  128.131.186.27   TCP      49656 > 61556 [PSH, ACK] Seq=1 Ack=1 Win=256 Len=3
 31 13:06:46.672 128.131.186.27   128.131.186.205  TCP      61556 > 49656 [ACK] Seq=1 Ack=4 Win=2920 Len=0
 32 13:06:46.813 128.131.186.205  128.131.186.28   TCP      49652 > 61553 [ACK] Seq=4 Ack=4 Win=256 Len=0
 33 13:06:46.854 128.131.186.205  128.131.186.27   TCP      49651 > 61550 [ACK] Seq=4 Ack=4 Win=256 Len=0
 34 13:06:48.171 128.131.186.27   128.131.186.205  TCP      61556 > 49656 [PSH, ACK] Seq=1 Ack=4 Win=2920 Len=3
 35 13:06:48.179 128.131.186.205  128.131.186.28   TCP      49652 > 61553 [PSH, ACK] Seq=4 Ack=4 Win=256 Len=3
 36 13:06:48.181 128.131.186.205  128.131.186.27   TCP      49653 > 61552 [PSH, ACK] Seq=1 Ack=1 Win=256 Len=3
 37 13:06:48.181 128.131.186.27   128.131.186.205  TCP      61552 > 49653 [ACK] Seq=1 Ack=4 Win=2920 Len=0
 38 13:06:48.182 128.131.186.205  128.131.186.27   TCP      49651 > 61550 [PSH, ACK] Seq=4 Ack=4 Win=256 Len=3
 39 13:06:48.220 128.131.186.28   128.131.186.205  TCP      61553 > 49652 [ACK] Seq=4 Ack=7 Win=2920 Len=0
 40 13:06:48.222 128.131.186.27   128.131.186.205  TCP      61550 > 49651 [ACK] Seq=4 Ack=7 Win=2920 Len=0
 41 13:06:48.454 128.131.186.205  128.131.186.27   TCP      49656 > 61556 [ACK] Seq=4 Ack=4 Win=256 Len=0
 43 13:06:56.183 128.131.186.27   128.131.186.205  TCP      61550 > 49651 [PSH, ACK] Seq=4 Ack=7 Win=2920 Len=3
 44 13:06:56.190 128.131.186.205  128.131.186.27   TCP      49656 > 61556 [PSH, ACK] Seq=4 Ack=4 Win=256 Len=3
 45 13:06:56.191 128.131.186.27   128.131.186.205  TCP      61556 > 49656 [ACK] Seq=4 Ack=7 Win=2920 Len=0
 46 13:06:56.193 128.131.186.205  128.131.186.28   TCP      49652 > 61553 [PSH, ACK] Seq=7 Ack=4 Win=256 Len=3
 47 13:06:56.193 128.131.186.205  128.131.186.27   TCP      49651 > 61550 [PSH, ACK] Seq=7 Ack=7 Win=256 Len=3
 48 13:06:56.194 128.131.186.28   128.131.186.205  TCP      61553 > 49652 [ACK] Seq=4 Ack=10 Win=2920 Len=0
 49 13:06:56.194 128.131.186.27   128.131.186.205  TCP      61550 > 49651 [ACK] Seq=7 Ack=10 Win=2920 Len=0
 50 13:06:56.194 128.131.186.205  128.131.186.27   TCP      49653 > 61552 [PSH, ACK] Seq=4 Ack=1 Win=256 Len=3
 51 13:06:56.195 128.131.186.27   128.131.186.205  TCP      61552 > 49653 [ACK] Seq=1 Ack=7 Win=2920 Len=0
 52 13:06:56.195 128.131.186.28   128.131.186.205  TCP      61553 > 49652 [PSH, ACK] Seq=4 Ack=10 Win=2920 Len=3
 53 13:06:56.197 128.131.186.27   128.131.186.205  TCP      61550 > 49651 [PSH, ACK] Seq=7 Ack=10 Win=2920 Len=3
 54 13:06:56.200 128.131.186.27   128.131.186.205  TCP      61552 > 49653 [PSH, ACK] Seq=1 Ack=7 Win=2920 Len=3
 55 13:06:56.413 128.131.186.205  128.131.186.28   TCP      49652 > 61553 [ACK] Seq=10 Ack=7 Win=256 Len=0
 56 13:06:56.413 128.131.186.205  128.131.186.27   TCP      49653 > 61552 [ACK] Seq=7 Ack=4 Win=256 Len=0
 57 13:06:56.455 128.131.186.205  128.131.186.27   TCP      49651 > 61550 [ACK] Seq=10 Ack=10 Win=256 Len=0
 58 13:06:57.692 128.131.186.27   128.131.186.205  TCP      61556 > 49656 [PSH, ACK] Seq=4 Ack=7 Win=2920 Len=3
 59 13:06:57.954 128.131.186.205  128.131.186.27   TCP      49656 > 61556 [ACK] Seq=7 Ack=7 Win=256 Len=0
```

Figure 4.14: Performing Task 1 from the Recipe – Snapshot of Wireshark

HLC needed approximately 51 milliseconds to process the data received from the sensors. Time that lapsed between the messages number 29 and 31 was approximately 52 milliseconds (communication LLC-HLC-HLC-LLC).

The results presented above are only few samples of the performed tests. Other results of the performed tests indicate a slightly different duration of transmission but a similar order of magnitude (see Table 4.1). Several examples of the obtained results from the measuring of message transmission duration for the communication LLC-HLC-HLC-LLC are presented in the table. These results are obtained from Fig. 4.14 and from Fig. 5.4 described later in Sec. 5.3. Additionally the experiment of measuring the message transmission duration during the task 1 was performed twice and the sample results of the second round are also shown in the Table 4.1. From the results presented in this table it can be concluded that message transmission duration from LLC to HLC, HLC to HLC and HLC to LLC adds up to approximately 30 ms.

Sample Nr.	**Duration [ms]**
1.	42
2.	52
3.	10
4.	49
5.	8
6.	36
7.	15
8.	29
9.	9
10.	9
11.	47
12.	48

Table 4.1: Duration of Communication LLC-HLC-HLC-LLC

4.7 Conclusion

The arguments and results that were presented suggest that the MAS approach using automation agents and reflective world model can be implemented on a plant and used in batch process automation. It was shown that tank automation agents use their world models as knowledge base to communicate with other agents and also to perform diverse activities. For the purpose of this research a tank agent with a complete world model was implemented, as it is the only agent that could perform more complicated activities and observations.

A recipe was tested on the above described plant and execution of several tasks was presented through message exchange. Agents communicate and share information among each other by means of message exchange in order to perform given tasks. Communication between HLC and LLC is shown on the example of execution of the first task from the recipe. Message transmission duration between HLC and LLC in failure free case was measured.

Task execution presented in this chapter was described only for a failure free situation. The truth of the matter is surely that failures and abnormal situations can happen in batch process automation. Therefore the next chapter demonstrates behavior of the system assuming a presence of a failure.

5 Failure Detection and Recovery

Introduction of the new approach to batch process automation offers many advantages to the industries employing it. It was shown in the previous chapter that MAS with automation agents and reflective world models can be applied to batch process automation. A defined recipe for production of a specific batch was tested on a plant in a failure free situation.

This chapter describes behavior of a system in a setting where a failure occurs. Three phases of failure handling process, namely *failure detection, failure isolation* and *failure recovery*, were already mentioned in Sec. 3.3. Failures can be detected by using functions of LLC as well as by using world models of automation agents. In the first case, a failure can be detected in LLC and recovered by HLC. For the purpose of this research, two examples of this failure handling type will be presented. One of them employs the analog sensor B101, but, due to the simplicity of the plant, the recovery part could not be implemented. The second scenario involves the valve V102 not being able to get opened.

It was already mentioned that failures can be detected, isolated and recovered by using only HLC. In order for the recovery to work, it has to be assumed that there are no failures with the communication channel between HLC and LLC, as HLC needs values from LLC to perform its functions.

It should also be mentioned that failure recovery is very often not even possible due to deficient functions of plants. Recovery of a failure in future text refers to failure isolation and stopping of an activity that caused the failure.

5.1 Failure Detection and Recovery Using Both HLC and LLC

5.1.1 Failure Detection in LLC by Using Analog Sensors

Failure detection in LLC was performed by employing analog sensors in the system. This function is implemented for the distance sensor B101. In this scenario, when the failure is detected in the LLC, the HLC gets informed.

It was mentioned in previous sections that, when actions *entering* and *leaving* are performed, the tank agent sends a message to the analog sensor agent B101 requesting from it a notification when the end value is reached. This message is sent in the following format:

```
<msg>
<action>informWhen</action>
<value>5L</value>
</msg>
```

With this message the sensor agent receives instruction to inform the tank agent when the tank is filled with 5 liters of material. If the sensor agent receives this message it means that the level of material is about to change. Upon this the timer in LLC is started and 10 seconds later the initial and the current value are compared (see Fig. 5.1 – INOLD represents the initial value and INNEW the value after 10 seconds). If the difference between them is less than a defined value (*epsilon* in Fig. 5.1) it indicates that there is a failure. The difference value is taken to be 30, as the controller returns integer values representing the values of the sensor. Accuracy of a controller depends on its architecture. In this case, the delta value of 30 was sufficient to determine if the level of material in the tank has changed[7].

If a failure is detected, error code is sent to HLC. This error code could represent a failure of a sensor, or a failure with actions (entering or leaving). In any case, the action being performed at that moment has to be stopped and that sensor as well as actions are isolated and marked as *failed*.

The failure recovery could not be implemented due to lack of devices. If there had been another analog sensor in that tank, it could have been used to establish if there is a failure with sensor or the problem lies with execution

[7]For a 12-bit controller architecture, used in this case, maximum binary value of 1111 1111 1111 corresponds to decimal value of 4095. Decimal value of 30, used for failure detection, corresponds to binary value of 11110 and this delta value is required for analog sensors during failure detection, as their value fluctuates constantly due to various reasons.

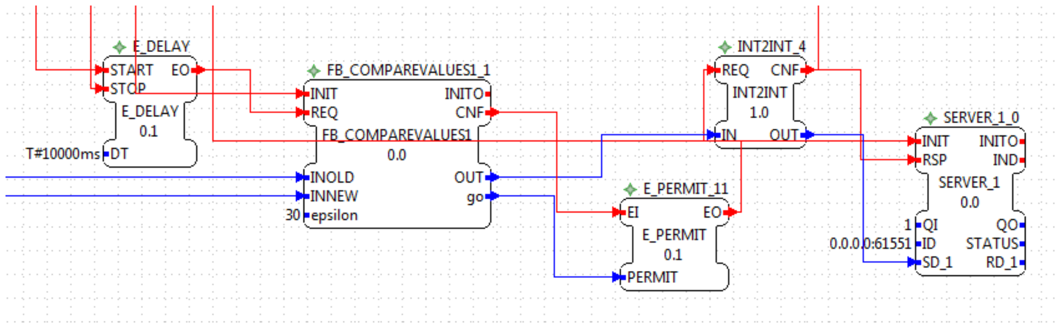


Figure 5.1: Failure Detection in LLC of Analog Sensor B101 – Snapshot of 4DIAC-IDE

of the activity. For more precise failure detection the flow analog sensor B102 could be used, but one cannot be certain that there is no failure with the pipeline between the tank T101 and the tank T102. If the sensor B102 shows no failure, whereas the sensor B101 does, this can mean either that the sensors are malfunctioning or that the pipeline between the tanks is broken.

5.1.2 Failure Recovery Using Failure Handling Agent

Another case of failure detection that involves complete recovery combined the valve V102 (see plant in Sec. 4.1) and the path finding algorithm. The valve V102 contains a sensorbox with two digital sensors. These sensors indicate if the valve is opened or closed. When the instruction to open the valve is sent, the LLC of the valve agent V102 controls if it actually gets opened and the corresponding value is sent to the HLC.

The first task of the recipe (see Sec. 4.5.1) required the material to be transported from the tank T102 to T101 through the valve V102, because this is the shortest path. The valve V102 is a pneumatic valve, which means that it needs compressed air to change its state. For the purpose of this experiment the air supply was cut off. Thereafter command to open was sent, but, as expected, the valve could not be opened. As a result a failure message was sent to the HLC. In this example the failure was detected in the LLC.

In the second step of the failure handling, isolation, this valve was marked as having a *failure* and could not be used until the failure was removed. This also implies that this valve cannot be considered for the purpose of finding a path from one tank to another.

These two phases, detection and isolation, do not involve other agents. However, the third one, recovery, is performed by employing additional agents, especially the failure handling (FH) agent (see Sec. 3.1.1).

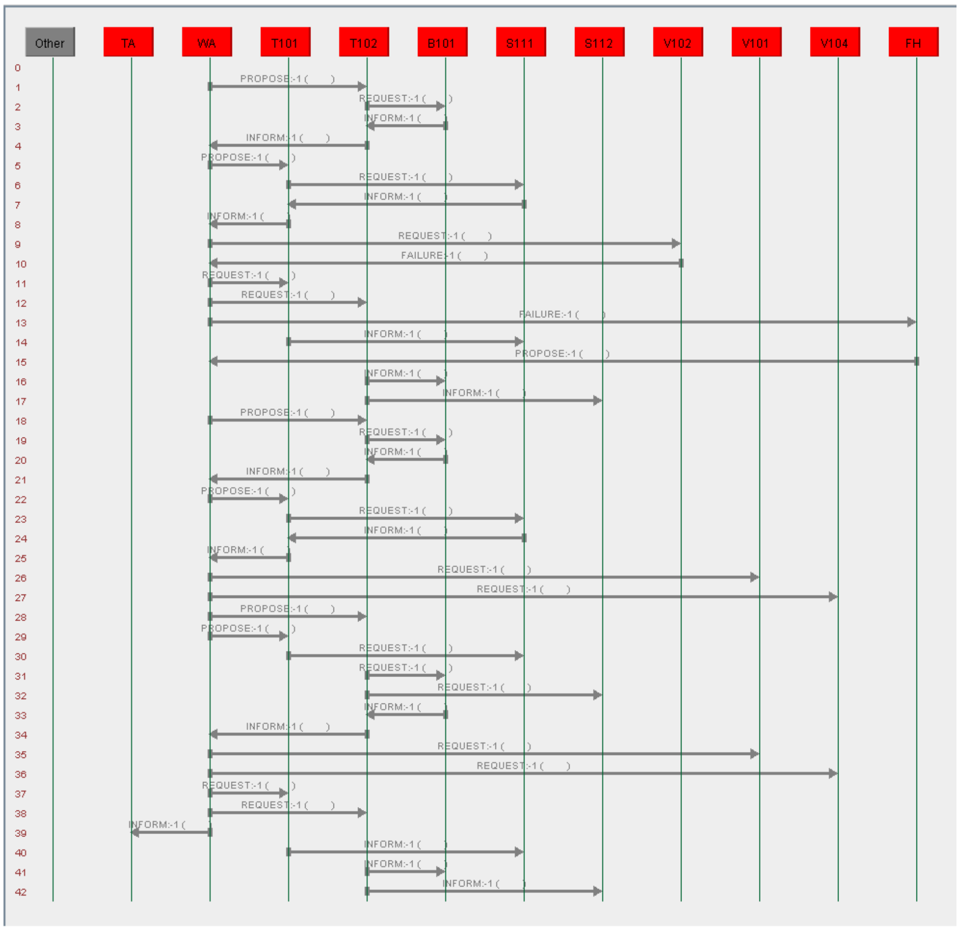


Figure 5.2: Failure Recovery by Finding Another Path

Fig. 5.2 shows message exchange when a failure occurs during the first task of the recipe (Sec. 4.5.1). The message number 10 shows the valve V102 informing the work agent that there is a failure. After the work agent receives the message, it stops all activities connected with that task and asks the FH agent for a solution of the failure. The FH agent suggests finding another path for the transport of the material from the tank T102 to T101 to the work agent (the message number 15 in Fig. 5.2). Starting with the message number 18, the task gets performed from the beginning, only in this case the path finding algorithm will not consider the route through the valve V102. The path used for the transport is via V101 and V104. After that the system continues to work as usual.

5.2 Failure Detection and Recovery Using the World Model

The failure detection scenario explained in Sec. 5.1.1 describes a situation where an analog sensor detects a failure at the beginning of an action. This shows that an action that starts without a failure does not have to finish in the same way. For detection of a failure at the beginning of an action only one agent needs to be employed (in case of failure described in Sec. 5.2 it is analog sensor agent).

During the action *entering* in the tank T102, the material is transported through the pump P101 (see Fig. 4.1). The pump with the flow sensor agent and the flow control agent is grouped around the flow PH automation agent. Failure detection was further improved by employing a flow control agent in that process. During the third task of the recipe (Sec. 4.5.3) the message that is sent to the work agent is:

```
<task>
<taskname>transport</taskname>
<param1>T101</param1>
<param2>T102</param2>
<param3>5L</param3>
<param4>fca</param4>
<param5>2L/min</param5>
</task>
```

From the above task it is obvious that transporting 5 liters of material takes 2.5 minutes if the pump is transporting 2 liters per minute. This fact updates the world model of the tank T102 and the observation of the action *entering* is used to detect the failure (Fig. 4.8). The tank T102 observes the activity of *entering* from the valve V101. Since it knows when the action is scheduled be finished, it will detect a failure if this does not happen. A similar case was explained by Merdan et al. in [37].

The failure used as a basis for failure handling scenario described in this section was also caused manually by opening the valve V104. In this case part of material could flow back to the T101 and the *expectation* from the world model could not be satisfied.

Fig. 5.3 shows the message exchange for this failure. One might notice that until the message number 15 there is no difference to Sec. 4.5.3. Starting with the messages number 16 and 17, where the tank agent informs the sensor and the work agent that there is a failure. Consequently, all activities in the tank have to be stopped. The work agent closes the valves and stops the pump as means of prevention. Finally, the work agent informs the task agent that the transport was not successfully done.

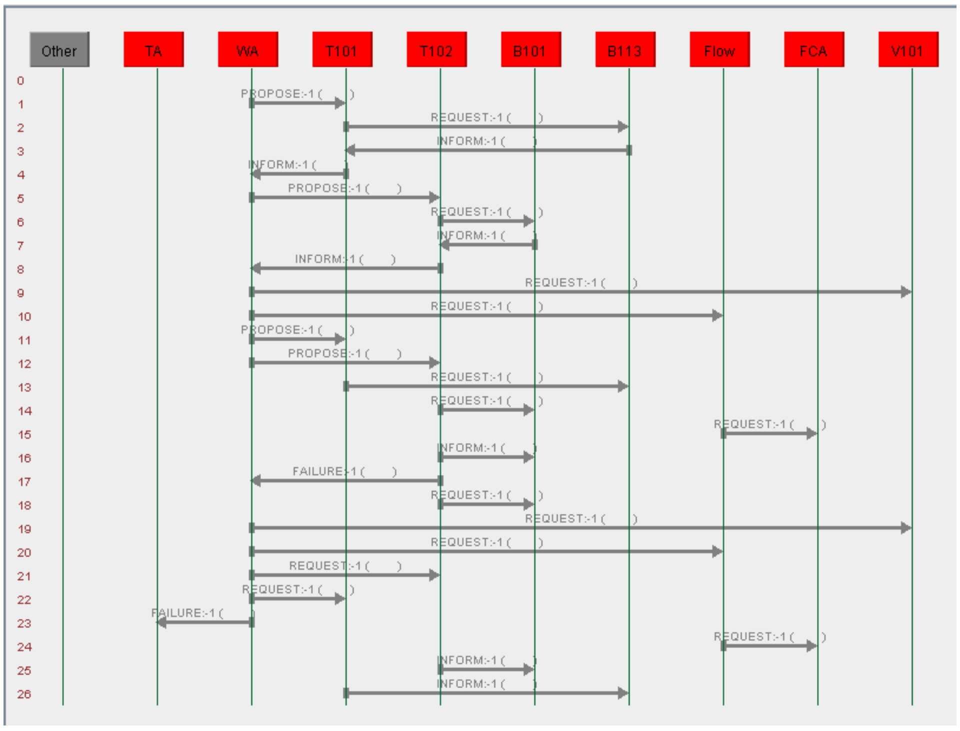


Figure 5.3: Failure Detection using World Model in HLC

In order to improve failure detection and recovery, additional observations can be added to the above model. To rely merely on a final verification at the end of the process, like in the above case, can hardly be considered sufficient. Alternatively, the process could be supplemented with an additional verification after one minute, stating if 2 liters of material were transported. If the failure that occurred in the first minute is timely detected, this could prevent possible damage caused by executing the whole transport.

5.3 Measurement of Duration of Failure Handling

Behavior of the system when the valve V102 cannot be opened was described in Sec. 5.1.2. It was mentioned that immediately after the failure was detected, the whole transport task is stopped. After that the work agent, according to the instruction from the FH agent, finds another path in order to transport material from the tank T102 to T101.

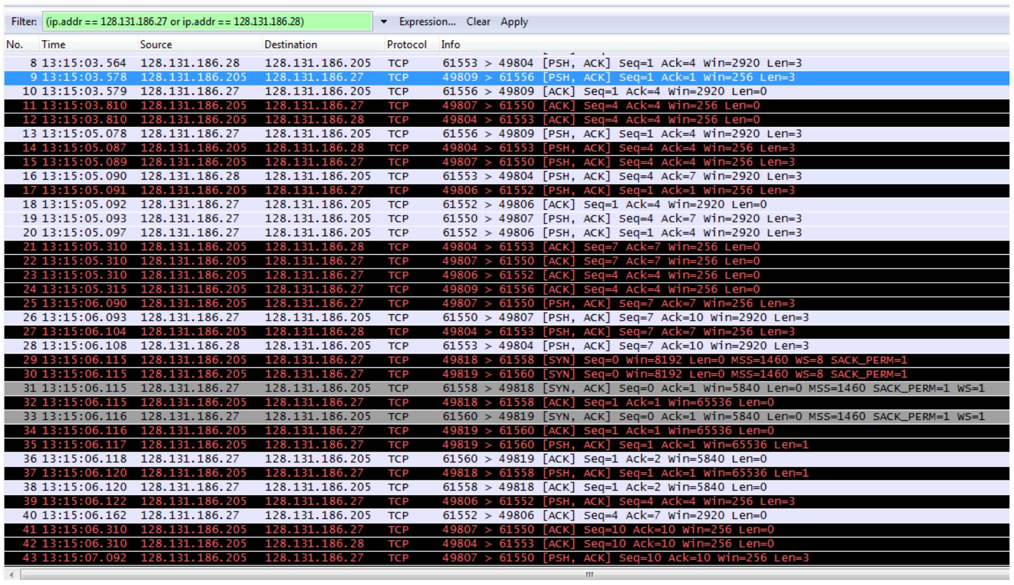
During this scenario, exchange of messages over the network was monitored and a snapshot from Wireshark is shown in Fig. 5.4. This figure shows only a portion of the communication, from the point where the V102 valve agent detects the failure, until the point where the work agent opens the valves V101 and V104, because these two are used as the alternative path. The valve V101 has the IP and port 128.131.186.27:61560 and the valve V104 has 128.131.186.27:61558. The message number 13 in Fig. 5.4 triggers the message number 10 in Fig. 5.2. With this message, LLC of the valve V102 reports the failure to its HLC. After isolating V102, the work agent opens the valves V101 and V104. This action is represented by the messages number 35 and 37 in Fig. 5.4 which are triggered by messages number 26 and 27 in Fig. 5.2. The part of the task before failure detection and after failure recovery is same as in the failure free case.

In Fig. 5.4 it is shown that the systems needs between 1.039 and 1.042 seconds from the point where the failure is detected until the point where the system recovered the failure.

5.4 Conclusion

Chapter 4 presented behavior of the system in a failure-free situation, when execution of the recipe can be performed without interruption. However, abnormal situations and behaviors are common in batch processes. This chapter presented two ways of failure handling. The first one is by employing LLC for failure detection and the second one is by employing HLC for failure detection.

Failure isolation can be done in LLC as well as HLC. In the last step, failure recovery is performed in HLC. It was shown that when a valve failure (failure of the valve V102) occurs, the system needs approximately 1.04 second to isolate the component with failure and to recover. This is a very good result considering that it would surely take longer to reroute manually. This failure handling could also have been done by employing centralized monolithic systems. However, if this was not a manually induced failure, but a failure on



Filter: (ip.addr == 128.131.186.27 or ip.addr == 128.131.186.28) Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
8	13:15:03.564	128.131.186.28	128.131.186.205	TCP	61553 > 49804 [PSH, ACK] Seq=1 Ack=4 Win=2920 Len=3
9	13:15:03.578	128.131.186.205	128.131.186.27	TCP	49809 > 61556 [PSH, ACK] Seq=1 Ack=1 Win=256 Len=3
10	13:15:03.579	128.131.186.27	128.131.186.205	TCP	61556 > 49809 [ACK] Seq=1 Ack=4 Win=2920 Len=0
11	13:15:03.810	128.131.186.205	128.131.186.27	TCP	49807 > 61550 [ACK] Seq=4 Ack=4 Win=256 Len=0
12	13:15:03.810	128.131.186.205	128.131.186.28	TCP	49804 > 61553 [ACK] Seq=4 Ack=4 Win=256 Len=0
13	13:15:05.078	128.131.186.27	128.131.186.205	TCP	61556 > 49809 [PSH, ACK] Seq=1 Ack=4 Win=2920 Len=3
14	13:15:05.087	128.131.186.205	128.131.186.28	TCP	49804 > 61553 [PSH, ACK] Seq=4 Ack=4 Win=256 Len=3
15	13:15:05.089	128.131.186.205	128.131.186.27	TCP	49807 > 61550 [PSH, ACK] Seq=4 Ack=4 Win=256 Len=3
16	13:15:05.090	128.131.186.28	128.131.186.205	TCP	61553 > 49804 [PSH, ACK] Seq=4 Ack=7 Win=2920 Len=3
17	13:15:05.091	128.131.186.205	128.131.186.27	TCP	49806 > 61552 [PSH, ACK] Seq=1 Ack=1 Win=256 Len=3
18	13:15:05.092	128.131.186.27	128.131.186.205	TCP	61552 > 49806 [ACK] Seq=1 Ack=4 Win=2920 Len=0
19	13:15:05.093	128.131.186.27	128.131.186.205	TCP	61550 > 49807 [PSH, ACK] Seq=4 Ack=7 Win=2920 Len=3
20	13:15:05.097	128.131.186.27	128.131.186.205	TCP	61552 > 49806 [PSH, ACK] Seq=1 Ack=4 Win=2920 Len=3
21	13:15:05.310	128.131.186.205	128.131.186.28	TCP	49804 > 61553 [ACK] Seq=7 Ack=7 Win=256 Len=0
22	13:15:05.310	128.131.186.205	128.131.186.27	TCP	49807 > 61550 [ACK] Seq=7 Ack=7 Win=256 Len=0
23	13:15:05.310	128.131.186.205	128.131.186.27	TCP	49806 > 61552 [ACK] Seq=4 Ack=4 Win=256 Len=0
24	13:15:05.315	128.131.186.205	128.131.186.27	TCP	49809 > 61556 [ACK] Seq=4 Ack=4 Win=256 Len=0
25	13:15:06.090	128.131.186.205	128.131.186.27	TCP	49807 > 61550 [PSH, ACK] Seq=7 Ack=7 Win=256 Len=3
26	13:15:06.093	128.131.186.27	128.131.186.205	TCP	61550 > 49807 [PSH, ACK] Seq=7 Ack=10 Win=2920 Len=3
27	13:15:06.104	128.131.186.205	128.131.186.28	TCP	49804 > 61553 [PSH, ACK] Seq=7 Ack=7 Win=256 Len=3
28	13:15:06.108	128.131.186.28	128.131.186.205	TCP	61553 > 49804 [PSH, ACK] Seq=7 Ack=10 Win=2920 Len=3
29	13:15:06.115	128.131.186.205	128.131.186.27	TCP	49818 > 61558 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_PERM=1
30	13:15:06.115	128.131.186.205	128.131.186.27	TCP	49819 > 61560 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=8 SACK_PERM=1
31	13:15:06.115	128.131.186.27	128.131.186.205	TCP	61558 > 49818 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1 WS=1
32	13:15:06.115	128.131.186.205	128.131.186.27	TCP	49818 > 61558 [ACK] Seq=1 Ack=1 Win=65536 Len=0
33	13:15:06.116	128.131.186.27	128.131.186.205	TCP	61560 > 49819 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM=1 WS=1
34	13:15:06.116	128.131.186.205	128.131.186.27	TCP	49819 > 61560 [ACK] Seq=1 Ack=1 Win=65536 Len=0
35	13:15:06.117	128.131.186.205	128.131.186.27	TCP	49819 > 61560 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1
36	13:15:06.118	128.131.186.27	128.131.186.205	TCP	61560 > 49819 [ACK] Seq=1 Ack=2 Win=5840 Len=0
37	13:15:06.120	128.131.186.205	128.131.186.27	TCP	49818 > 61558 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=1
38	13:15:06.120	128.131.186.27	128.131.186.205	TCP	61558 > 49818 [ACK] Seq=1 Ack=2 Win=5840 Len=0
39	13:15:06.122	128.131.186.205	128.131.186.27	TCP	49806 > 61552 [PSH, ACK] Seq=4 Ack=4 Win=256 Len=3
40	13:15:06.162	128.131.186.27	128.131.186.205	TCP	61552 > 49806 [ACK] Seq=4 Ack=7 Win=2920 Len=0
41	13:15:06.310	128.131.186.205	128.131.186.27	TCP	49807 > 61550 [ACK] Seq=10 Ack=10 Win=256 Len=0
42	13:15:06.310	128.131.186.205	128.131.186.28	TCP	49804 > 61553 [ACK] Seq=10 Ack=10 Win=256 Len=0
43	13:15:07.092	128.131.186.205	128.131.186.27	TCP	49807 > 61550 [PSH, ACK] Seq=10 Ack=10 Win=256 Len=3

Figure 5.4: Part of Communication during Failure Recovery – Snapshot of Wireshark

the controller which hosts the valve V102 and all the other components, the system would have stopped and human interaction would have been necessary. This would undoubtedly take longer than 1.04 second.

Furthermore, if there was a failure on the whole controller, the failure could not be detected by employing only LLC. Instead, detection can be done by combining HLC and involving the world model of the valve agent V102.

6 Discussion of Results

In previous chapters it was shown that distribution of systems is possible by employing automation agents with reflective world models. LLC of automation agents is developed by using standard IEC 61499 function blocks that support distribution of every mechatronic component in a system through their event based execution. HLC is developed with JADE and it uses FIPA standardized messages for communication between agents.

The multi-agent system presented in this thesis offers a new approach to batch process automation. By implementing agent technology, systems are getting new dimension of artificial intelligence and thereby evident improvement to the currently employed techniques are achieved. This new agent based approach raises some difficulties as well. This section discusses the results of implementation of agent technology in batch process automation.

6.1 Gains of the MAS Based Approach

Every automation agent contains a world model that serves as its knowledge source. Automation agents are aware of what happens in their immediate environment, but not necessarily of that what happens in other parts of the system. Being independent from each other, gives them advantage in situation where one part of the system malfunctions or stops working.

Furthermore it is shown that this approach makes it possible to solve abnormal situations without human interaction. It was demonstrated that the system can react if one part of the equipment stops working properly. In the case explained in Sec. 5.1.2, the valve V102 could not be opened. This failure was caused on purpose by cutting off the air supply for this pneumatic valve, in order to observe the corresponding behavior of the system. Failures on real systems could be caused by actual disruption on controllers, or if the air supply was cut off for real. In that case the manual reconfiguration would surely take longer than the obtained result of 1.04 second needed in the case presented in Chapter 5.

Another advantage of this type of application is that automation agents do not depend on other components and could easily be added and removed from a system with minor adaptations. HLC and LLC are autonomous and perform operations according to specified events and behaviors. If the plant (Fig. 4.1) was being equipped with an additional tank (for instance a tank that has the same world model as the tank T101), only the IP address and ports of the newly deployed mechatronic components and the route to and from the new tank for the path finding algorithm would need to be updated. After that the system would be ready to start performing recipes that involve the new tank.

One more important gain of the system and the implementation presented in previous chapters is that the communication duration between HLC and LLC was noticeably reduced compared to the approach presented by Lopez et al. in [35]. In that approach HLC and LLC were both deployed on a controller and used shared memory to exchange the data. They used an embedded controller with the aJile100 microprocessor that works at 100 MHz. Their results indicate that it takes approximately 2 seconds for the message to cover the way LLC-HLC-HLC-LLC. In the approach presented in this thesis, the whole HLC was deployed and run on a PC with 4 GB of RAM and an Intel Core 2 Duo CPU, P8600 working at 2.4 GHz. HLC and LLC control were connected through a local network and the message transmission duration over the path LLC-HLC-HLC-LLC lasted approximately 30 milliseconds. Even though a duration of 30 milliseconds for the path LLC-HLC-HLC-LLC could be considered sufficient for the actions that require a real-time capability, implementation of the PI and hysteresis controller was still done in LLC in order to avoid consequences of a possible network breakdown that could cause considerable damage. This could for instance happen if the heater could not be stopped from working after it had been turned on. Even if the LLC of the PI-controller and the sensor were deployed on two different controllers and connected over a network, the possibility of network breakdown would still exist. However, the probability of thereby caused problems would be reduced due to fewer network connections as compared to a situation where a communication to the HLC is also carried out.

6.2 Outcome of the Implementation

Implementation of automation agents with reflective world models in batch process automation enables systems to become distributed, flexible and failure tolerant. Autonomous agents have potential to support the *plug and play* requirements. This new approach is tested on a small plant for study purposes and it shows remarkable performance and results worthy of further research, because it seems to have a great potential.

However, there are also few possible difficulties with the implementation. One of them is occasional inability to forecast behavior of agents and the system. The agents communicate with each other in order to execute their tasks. This communication needs to include some defined constraints for the agent performance in order to prevent possible undesirable behavior.

For the implementation explained in Chapter 4 a PC and two controllers were used. This controller architecture fulfills the usual requirements of the plant that was used for testing. It might be necessary to use additional controllers and PCs in bigger-scale plants, which could consequently lead to additional costs. Additional expenses might also be caused by switching to this technology from the one currently employed, as well as by the need for education of personal developing and interacting with these new systems.

This approach focuses on the controlling part of batch process and does not tackle its *planning* and *scheduling*. As it was shown in Chapter 2, research in this field is being carried out. Admittedly, these aspects were not implemented for the purpose of this thesis. However, due to generic and modular architecture of the system it is possible to extend it with those additional aspects of batch process automation.

6.3 Summary of the Results

Results of the implementation can be summarized in the following:

- Distributed autonomous systems are able to work and perform activities independently.
- MAS are able to detect, isolate and recover failures without human interaction.
- Within this implementation, message requires approximately 30 milliseconds for the path LLC-HLC-HLC-LLC. HLC can be implemented for systems, where 30 milliseconds of latency time for some actions does not affect the performance of the system.
- LLC and HLC can be reused for implementation of other similar components.
- Automation agents representing mechatronic devices can easily be added or removed from the system as they are independent.

7 Conclusion and Outlook

The presented MAS approach is relatively new in process automation. It enables shortening of time-to-market and it enables flexible production.

In this thesis implementation of automation agents with reflective world model was researched and discussed. Every mechatronic component was represented by a single automation agent. Additionally other automation agents (like those representing a tank or those controlling temperature) as well as functional agents were implemented in the system.

Automation agents consist of HLC and LLC. LLC developed according to IEC 61499 was deployed and run directly on a controller and HLC was developed according to the FIPA standards and run on a PC. It was shown that even though these parts of automation agent communicate over a network, this does not cause long latency time.

It was also shown that a system such as the one implemented can execute a recipe on its own and that in case of a failure it is possible to recover in 1.04 second. It is possible to detect failures in LLC as well as in HLC using world models. These are contained in HLC of every automation agent and they are used as knowledge base for decision making.

The focus of this thesis lies on the controlling part of batch processes. Automation agents perform their activities according to specifications of their behaviors, which are activated upon message reception. This implementation could in the future work be extended to scheduling and planning of batch processes.

Further possible future work includes testing of this implementation on a more sophisticated and equipped plant. Behavior of the system could be tested for a situation where the plant (Fig. 4.1) is for instance equipped with additional tanks or additional pipelines and pumps that provide alternative paths for material transport.

It is also possible to install additional analog and digital sensors that can improve failure detection and also apply additional actuators for failure recovery. It was already mentioned before that all of this supplementary equipment can be added with minimal adaptations. Among the required adaptations is preparation of new recipes as well as configuration of failure handling. By adding new equipment it would be possible to implement and test various tasks and also to combine it with scheduling and planning.

This approach and the implemented MAS serves as basis for future developments in this field. It was shown that this approach can be implemented and used in batch processes. However, there are many aspects that still need to be researched and tested on more sophisticated and complex plants.

Bibliography

[1] Grigoris Antoniou, Frank van Harmelen. *Web Ontology Language: OWL*, pp 91-110. In *Handbook on Ontologies*. Steffen Staab, Rudi Studer. Springer-Verlag Berlin Heidelberg, Germany. 2009.

[2] Franz Baader, Ian Horrocks, Ulrike Sattler. *Description Logics*, pp 21-43. In *Handbook on Ontologies*. Steffen Staab, Rudi Studer. Springer-Verlag Berlin Heidelberg, Germany. 2009.

[3] Mike Barker, Jawahar Rawtani. *Batch Process Management*. Newnes, an imprint of Elsevier, Oxford, UK, 2005.

[4] Fabio Bellifemine, Giovanni Caire, Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley&Sons Ltd., West Sussex, England, 2007.

[5] Frank Dignum, Mark Greaves. *Issues in agent communication*. Springer-Verlag Berlin Heidelberg, Germany, 2000.

[6] D. Dimitrova, S. Panjaitan, I. Batchkova, G. Frey. "IEC 61499 Component Based Approach for Batch Control Systems". In *Proceedings of 17th World Congress of the International Federation of Automatic Control (IFAC 2008)*, Seoul, Korea, July 2008.

[7] Bernard Favre-Bulle. *Automatisierung komplexer Industrieprozesse, Systeme, Verfahren und Informationsmanagement*. SpringerWienNewYork, Vienna, Austria, 2004.

[8] FESTO Controller CPX-CEC-C1. http://www.festo.com/ext/en/10485.htm, September 2010.

[9] FIPA Standards. FIPA – Foundation for Intelligent Physical Agents. http://www.fipa.org, 2010.

[10] FIPA Standard. *FIPA ACL Message Structure Specification*. Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2002. Public Available Specification. http://www.fipa.org.

[11] FIPA Standard. *FIPA Agent Management Specification.* Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2004. Public Available Specification. http://www.fipa.org.

[12] Nicola Guarino, Daniel Oberle, Steffen Staab. *What is Ontology?*, pp 1-17. In *Handbook on Ontologies.* Steffen Staab, Rudi Studer. Springer-Verlag Berlin Heidelberg, Germany. 2009.

[13] Wolfgang A. Halang, Krzysztof M. Sacha. *Real-time systems: implementation of industrial computerised process automation.* World Scientific Publishing, Singapore, Singapore, 2001.

[14] T. Hamaguchi, T. Hattori, M. Sakamoto, H. Eguchi, Y. Hashimoto and T. Itoh. "Multi-agent Structure for Batch Process Control". In *Proceedings of the 2004 IEEE International Conference on Control Applications*, Taipei, Taiwan, September 2004.

[15] Xia Hong, Song Jiancheng. "Multi-Agent Based Scheduling for Batch Process". In *Proceedings of the 8th International Conference on Electronic Measurement and Instruments (ICEMI 2007)*, pp 2-464-2-267, 2007.

[16] Tanvir Hussain and Georg Frey. "Migration of a PLC Controller to an IEC 61499 Compliant Distributed Control System: Hands-on Experiences". In *Proceedings of the 2005 IEEE of the International Conference on Robotics and Automation*, Barcelona, Spain, April 2005.

[17] IEC 60050-351. *International Electrotechnical Vocabulary – Part 351: Control technology (IEC 60050-351:2006).* DIN Deutsches Institut für Normung e.V. Beuth Verlag GmbH, Berlin, Germany, 2009.

[18] IEC 61131. *Batch control – Part 1: Models and terminology.* International Electrical Commission, Geneva, 1997. Public Available Specification.

[19] IEC 61499-1. *Function blocks – Part 1: Architecture.* International Electrical Commission, Geneva, Switzerland, 2005. Public Available Specification.

[20] IEC 61512-1. *Batch control – Part 1: Models and terminology.* International Electrical Commission, Geneva, 1997. Public Available Specification.

[21] David James. “Batch of the day”. In *IET computing and control engineering*, pages 30–35, August/September 2006. http://www.theiet.org/control

[22] David James. “Batch Process Automation”. In *IET computing and control engineering*, pages 32–37, December/January 2006/07. http://www.theiet.org/control

[23] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods, and practical use.* Springer-Verlag Berlin Heidelberg, Germany, 1997.

[24] Karl-Heinz John,Michael Tiegelkamp. *IEC 61131-3: Programming Industrial Automation Systems.* Springer-Verlag, Berlin, Germany, 2010.

[25] Charles M. Kozierok. *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference.* No Starch Press Inc., San Francisco, USA, 2005.

[26] Sven Oliver Krumke, Hartmut Noltemeier. *Graphentheoretische Konzepte und Algorithmen.* Vieweg Teubner, Fachverlage GmbH, Wiesbaden, Germany, 2009.

[27] Wilfried Lepuschitz, Franz Königseder and Alois Zoitl. “Conjunction of a Distributed Control System based on IEC 61499 with a Commercial Batch Management System”. In *Proceedings of the 2009 IEEE Conference on Emerging Technologies & Factory Automation*, pp 1-8, 2009.

[28] Wilfried Lepuschitz. *Distributed Control in Process Automation – with the focus on possible applications of IEC 61499.* Master’s thesis, Technical University of Vienna, Vienna, 2008.

[29] Wilfried Lepuschitz, Gottfried Koppensteiner, Manuel Barta, Thanh Tung Nguyen and Constantin Reinprecht. “Implementation of Automation Agents for Batch Process Automation”. In *Proceedings of the 2010 IEEE International Conference on Industrial Technology (ICIT2010)*, pp 524-529, 2010.

[30] Wilfried Lepuschitz, Mathieu Vallée, Alois Zoitl, and Munir Merdan. “Online Reconfiguration of the Low Level Control for Automation Agents”. In *Proceedings of the 36th Annual Conference of IEEE Industrial Electronics (IECON2010)*, Glendale, AZ, USA, November 2010, IN PRESS.

[31] Wilfried Lepuschitz, Alois Zoitl, Mathieu Vallée, and Munir Merdan. "Toward Self-Reconfiguration of Manufacturing Systems Using Automation Agents". In *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 2010, IN PRESS.

[32] Robert Lewis. *Modelling control systems using IEC 61499.* Technical report, Technical University of Vienna, 2005.

[33] Dennis Lock. *Project management.* Gower Publishing Limited, Hampshire, UK, 2007.

[34] O. J. Lopez and J. M. Lastra. "A Real-Time Interface for Agent-Based Control". In *Proceedings of the 2nd IEEE International Symposium on Industrial Embedded Systems (SIES2007)*, pp. 49–54, 2007.

[35] Omar J. Lopez Orozco and Jose L. M. Lastra. *Agent-Based Control Model for Reconfigurable Manufacturing Systems.* In *Proceedings of the 12th IEEE Conference on Emerging Technologies and Factory Automation (ETFA2007).* pp 1233-1238, 2007.

[36] Munir Merdan, Wilfried Lepuschitz, Ingo Hegny and Gottfried Koppensteiner. "Application of a Communication Interface between Agents and the Low Level Control". In *Proceedings of the 4th International Conference on Autonomous Robots and Agents (ICARA2009)*, pp 628-633, 2009.

[37] Munir Merdan, Wilfried Lepuschitz, Bakir Šahović, Mathieu Vallée. "Failure Detection and Recovery in the Batch Process Automation Domain using Automation Agents". In *Proceedings of the 2nd International Conference on Advances in Computing, Control, and Telecommunication Technologies*, Jakarte, Indonesia, December 2010, IN PRESS.

[38] Munir Merdan, Mathieu Vallée, Wilfried Lepuschitz, and Alois Zoitl. "Monitoring and diagnostics of industrial systems using automation agents". In *International Journal of Production Research*, Vol. 49, No. 5, 2011, IN PRESS.

[39] Peter Neumann. *SPS-Standard: IEC 61131: Programmierung in verteilten Automatisierungssystemen.* Oldenbourg Industrieverlag GmbH, München, Germany, 2000.

[40] Oxford Englisch Dictionary Online.
http://dictionary.oed.com/, 2010.

[41] J. Peltola, J. Christensen, S. Sierla, K. Koskinen. “A Migration Path to IEC 61499 for the Batch Process Industry”. In *Proceedings of the 5th IEEE International Conference on Industrial Informatics*, pp 811-816, 2007.

[42] Teppo Pirttioja. *Applying Agent Technology to Contructing Flexible Monitoring Systems in Process Automation.* PhD thesis, Helsinki University of Technology, Helsink, Finland, 2008.

[43] Teppo Pirttioja, Aarne Halme, Antti Pakonen, Ilkka Seilonen, Kari Koskinen. “Multi-agent System Enhanced Supervidion of Peocess Automation”. In *Proceedings of the IEEE Workshop on Distributed Intelligence and Its Applications (DIS2006)*, pp 151-156, 2006.

[44] Masaru Sakamoto, Hajime Eguchi, Takashi Hamaguchi, Yutaka Ota, Yoshihiro Hashimoto and Toshiaki Itoh. “Agent-Based Batch Process Control Systems”. In *M.Gh. Negoita et al. (Eds.): KES 2004, LNAI 3214*, pp. 398-404, Springer-Verlag Berlin Heidelberg, Germany, 2004.

[45] Ilkka Seilonen, Kari Koskinen, Teppo Pirttioja, Pekka Appelqvist, Aarne Halme. “Reactive and Deliberative Control and Cooperation in Multi-Agent System Based Process Automation”. In *Proceedings of the 2005 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Espoo, Finland, 2005.

[46] Paul N. Sharratt. *Handbook of batch process design.* Blackie Academic and Professional, an imprint of Chapman&Hall, London, UK, 1997.

[47] R. Studer, R. Benjamins, and D. Fensel. “Knowledge engineering: Principle and methods”. In *Transactions on Data and Knowledge Engineering, 25(1-2*, pp. 161-198, 1998.

[48] K. Thramboulidis, S. Sierla, N. Papakonstantinou, K. Koskinen. “An IEC 61499 Based Approach for Distributed Batch Process Control”. In *Proceedings of 5th IEEE International Conference on Industrial Informatics (INDIN2007)*, Vienna, Austria, July 2007.

[49] Mathieu Vallée, Hermann Kaindl, Munir Merdan, Wilfried Lepuschitz, Edin Arnautović and Pavel Vrba. “An Automation Agent Architecture with A Reflective World Model in Manufacturing Systems”. In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics (SMC2009)*, San Antonio, Texas, USA, pp 305-310, 2009.

[50] Mathiheu Vallée, Munir Merdan, Pavel Vrba. "Detection of Anomalies in a Transport System using Automation Agents with a Reflective World Model". In *Proceedings of the 2010 IEEE International Conference on Industrial Technology (ICIT2010)*, pp 489-494, 2010.

[51] Denny Vrandečić. *Ontology Evaluation*, pp 293–313. In *Handbook on Ontologies.* Steffen Staab, Rudi Studer. Springer-Verlag Berlin Heidelberg, Germany. 2009.

[52] Alois Zoitl. *Basic Real-Time Reconfiguration Services for Zero Down-Time.* PhD thesis, Technical University of Vienna, Vienna, Austria, 2007.

[53] 4DIAC-Consortium. "4DIAC – framework for distributed industrial automation and control, open source initiative" 2010. Access date Septeber 2010. [Online]. Available: www.fordiac.org