TECHNISCHE UNIVERSITÄT WIEN

DISSERTATION

Self-Reconfigurable Manufacturing Control based on Ontology-Driven Automation Agents

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften (Dr.techn.)

unter der Leitung von
Ao.Univ.–Prof. Dipl.–Ing. Dr.techn. Markus Vincze
E376
Institut für Automatisierungs– und Regelungstechnik

eingereicht an der
Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik

von
WILFRIED LEPUSCHITZ
Matrikelnummer: 9840198
Gersthofer Straße 93/18, 1180 Wien, Österreich

Wien, im April 2018

Tag des Rigorosums:	10. April 2018
Vorsitzender:	Univ.–Prof. Mag.rer.nat. Dr.rer.nat. Gottfried Strasser
Prüfer:	Ao.Univ.–Prof. Dipl.–Ing. Dr.techn. Markus Vincze
Begutachter und Prüfer:	Prof. Dr. Paulo Leitão
Begutachter:	Prof. Ing. Vladimír Mařík, DrSc., dr.h.c.

Acknowledgements

Composing a thesis and carrying out the work that leads to it is never something that can be achieved alone. A multitude of people have contributed to this work in one or the other way and with these words I would like to thank them. I truly hope that I did not forget anybody and I am already really sorry if that actually happened...

First of all I am grateful to Prof. Bernard Favre-Bulle for receiving the chance to start my academic career as university assistant at the Automation and Control Institute (ACIN), Vienna University of Technology. It was sad to see him leave this world much too early at the early beginnings of my work at the institute and I wish I would have had the chance to learn more from this dedicated “service provider for the students” (words by Prof. Favre-Bulle himself), which he truly was.

I am thankful for sharing the interest in modern automation technology with Prof. Gerfried Zeichen, who envisioned such systems already at a time, when the available technologies were by far not ready for realizing his ideas.

Thanks to Prof. Georg Schitter, who provided detailed comments on countless aspects of this thesis due to his extensive scientific expertise. Taking these comments into account represented an intensive learning experience and clearly improved my thesis and scientific writing style.

My sincere gratitude I would like to express to Prof. Markus Vincze for taking over the role of supervisor in times when others cannot. I appreciate his faith in my work as well as his valuable comments for further improving my thesis and I am grateful for continuing to work with him in joint projects.

A thesis like this requires additional co-supervisors providing expert feedback and in this context I am grateful for having Prof. Vladimír Mařík and Prof. Paulo Leitão in fulfilling this task. Both are well respected experts with impressive achievements regarding modern manufacturing systems technologies.

Regarding my time at ACIN, I would like to thank Prof. Alois Zoitl for his technical insights and for heading the Agile Control group that time – it was a pleasure to be part of it! Furthermore, my gratitude goes to Dr. Ingo Hegny, Dr. Reinhard Hametner, Dr. Monika Wenger and Martin Melik-Merkumians for multiple discussions and suggestions as well as support regarding various technical and scientific matters. Despite being employed at a different institute but working on the same project, I would also like to thank Dr. Matthieu Vallée for the interesting joint work, for being able to use my rusty French a bit, and for the statement: "Life is a matter of priorities." I also want to express my gratitude to Dr. Pavel Vrba for the joint project work and the introduction to the beautiful city of Prague – this truly had consequences as can be read below...

An office room mate is a good motivator for having work-related content on the computer screen as well as for discussing leisure time experiences. Therefore, I am happy that I had the following pleasurable office room mates across the years at ACIN: Andreas Schimmel, Michael Steinegger and Alexander Prostejovsky.

Valuable contributions were provided by my students Benjamin Grössing and Christoph Krofitsch. Both are extremely talented persons and their skills greatly supported the technical implementation of this work. It is just a matter of time until we hear one or both of their names in the context of a startup company exiting with exceptionable success.

I am thankful to ACIN's secretary Gabriele Grabensteiner and her always friendly way in providing reliable support in any administrative matters by combining efficiency with a nice smile.

I would like to express my sincere thanks to Dr. Gottfried Koppensteiner, who involved me in his Sparkling Science projects, encouraged me towards outreach work, and finally offered me the perfectly fitting job position upon leaving ACIN, at which I am still very happily engaged.

This thesis would not have been possible if it weren't for my colleague and dear friend Dr. Munir Merdan, who was the major factor on how this whole work could be achieved at all. He was the driving force behind the grants that financed this work and the most valuable discussion partner in composing and achieving this thesis. Working with him has always been a pleasure and therefore I am happy that we still write grants, papers and reports together in the frame of our joint projects. Let's see where this takes us...

I dedicate this thesis to my father Wilfried Lepuschitz senior, who has always been proud of the steps of my career and who has been a continuous supporter of my deeds. My warm thanks go to my mother Margit Lepuschitz, who has always been there to help and support, whatever is needed – "taking" seems to be a foreign word for you. Words can hardly express my gratitude to you both for providing the cards to my hands, which I can use in this game of life!

Furthermore, I want to thank my aunt Sieghild Lepuschitz, my sister Isabella Stiasny, and my brother Ehrenfried Lepuschitz for being family – this will always count.

Finally, I have to say that I am unbelievably lucky for asking "And what are you waiting for?" to a pretty girl in a bar in Prague a couple of years ago – truly every action in life is a game changer and this question has ultimately led to my own growing family. Thank you my love, Dóra Treznyák, for being my wife, supporting my way of life, and being the mother of our wonderful children Erik, Kira, and number 3, who is to be born around the time of this thesis' defense and for whom we should finally decide on a name. You all truly show me that life is indeed a matter of priorities and you are on top of the list!

This research was performed in the frame of the projects OntoReA (FIT-IT: Semantic Systems program, FFG project no. 815132, 2008–2011) and PrOnto (BRIDGE program, FFG project no. 829576, 2011–2014), both funded by the Austrian Federal Ministry for Transport, Innovation and Technology.

Vienna, April 2018 Wilfried Lepuschitz

Abstract

The dynamics of the 21^{st} century market with continuously decreasing life cycle spans of products require manufacturing systems, which are able to support mass customization. However, current manufacturing systems are not able to cope with this requirement due to their rigid and therefore inflexible structure. On the contrary, reconfigurable manufacturing systems that are composed of basic process modules allow the addition, removal or modification of these modules for providing specific functionality when needed. According software concepts are necessary for controlling such a manufacturing system based on modular components.

Agent technology represents a suitable approach for realizing the control software of a manufacturing system composed of modular components as each agent constitutes an autonomous software component, which can be mapped onto an according physical component. Applying this technology enables intelligent control components and is expected to increase a manufacturing system's flexibility and reconfigurability as well as its robustness in the case of failures.

This thesis introduces a hybrid architecture for the agents controlling the physical components of a manufacturing system. According to hybrid agent architectures, each agent is composed of two control layers. The higher level control layer is responsible for an agent's global behavior and makes use of a flexible knowledge base in the form of an ontology for representing information. The lower level control layer is real-time capable and has access to the sensors and actuators of the controlled component. Each agent possesses diagnostic capabilities for detecting failures as well as a reconfiguration infrastructure for modifying the provided functionality.

The approach is applied for automatically configuring the control software of a manufacturing system by making use of a resource ontology that describes the concepts of the target system. Furthermore, the ontology comprises the software concepts for the control application to enable the agents to perform this automatic software composition at system startup. Using the reconfiguration infrastructure, the agents are also able to adapt their functionality as a reaction on a global reconfiguration of the control system, which can be required for instance in the case of a detected failure.

An implementation on a laboratory pallet transport system shows the feasibility and benefits of the presented approach. The evaluation shows that the system is able to keep its performance despite the failure of a component at nearly the same level as in the failure free case.

Kurzfassung

Die Dynamik des heutigen Markts erfordert Produktionssysteme, die in der Lage sind eine große Produktvielfalt mit geringem Aufwand zu erzeugen, um rasch auf individuelle Kundenwünsche reagieren zu können. Jedoch bieten die derzeit eingesetzten Produktionssysteme aufgrund ihres starren Aufbaus nicht genügend Flexibilität. Im Gegensatz dazu sind rekonfigurierbare Produktionssysteme aus modularen Komponenten aufgebaut, welche flexibel modifiziert beziehungsweise je nach benötigter Funktionalität dem System hinzugefügt oder aus dem System entfernt werden können. Entsprechende Software-Konzepte sind daher für die Beherrschbarkeit und Steuerbarkeit solch eines verteilten Systems notwendig.

Der Einsatz von Agententechnologie erweist sich als ein geeignetes Konzept für die Steuerungssoftware von verteilten Produktionssystemen, da jeder Agent eine autonome Softwarekomponente repräsentiert, welche für die Steuerung einer entsprechenden physikalischen Komponente einsetzbar ist. Dieser Ansatz ermöglicht intelligente Steuerungskomponenten und es ist zu erwarten, dass damit die Flexibilität und Rekonfigurierbarkeit eines Produktionssystems sowie dessen Robustheit im Falle von auftretenden Fehlern erhöht werden kann.

Diese Dissertation beschreibt eine aus zwei Ebenen bestehende Architektur für jene Agenten, welche die phyischen Komponenten eines Produktionssystems steuern. Bezogen auf einen Agenten ist die obere Steuerungsebene ist für dessen globales Verhalten verantwortlich und verwendet dabei eine flexible Wissensbasis in der Form einer Ontologie für die Repräsentation von Information. Die untere Steuerungsebene ist echtzeitfähig und kann direkt auf die Sensoren und Aktoren der zugehörigen physischen Komponente zugreifen. Jeder Agent verfügt weiters über Diagnosealgorithmen zur Fehlererkennung sowie über eine Rekonfigurationsinfrastruktur für die Modifikation der eigenen Funktionalität.

Der präsentierte Ansatz wird für die automatische Konfiguration der Steuerungssoftware eines Produktionssystems eingesetzt und verwendet eine Ressourcenontologie für die Beschreibung der Komponenten und Konzepte des Systems. Zudem enthält die Ontologie weiters eine Beschreibung der Softwarekonzepte, um den Agenten die automatische Zusammenstellung der Steuerungsapplikation bei Systemstart zu ermöglichen. Mit Hilfe der Rekonfigurationsinfrastruktur sind die Agenten außerdem in der Lage, ihre Funkionalität zu adaptieren um adequat auf eine globale Rekonfiguration des Systems zu reagieren, welche beispielsweise aufgrund der Feststellung eines Fehlers notwendig werden kann.

Eine Implementierung auf einem Laboraufbau eines Palettentransportsystems zeigt die Machbarkeit sowie den Nutzen des vorgestellten Ansatzes. Die Evaluierung zeigt, dass das System in der Lage ist, trotz eines Fehlers den Durchsatz auf nahezu der gleichen Höhe zu halten wie im fehlerfreien Fall.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

4DIAC	*Framework for Distributed Industrial Automation and Control*
ACL	*Agent Communication Language*
ACS	*Autonomous Cooperative System*
AI	*Artificial Intelligence*
CNC	*Computer Numerical Control*
CNP	*Contract Net Protocol*
FB	*Function Block*
FBOS	*Function Block Operating System*
HLC	*High Level Control*
HMS	*Holonic Manufacturing Systems*
I/O	*Input/Output*
ID	*Identifier*
IEC	*International Electrotechnical Commission*
JADE	*Java Agent Development Framework*
KASA	*Knowledge-based Multi-Agent System Architecture*
KQML	*Knowledge Query and Manipulation Language*
LLC	*Low Level Control*
MAS	*Multi-Agent System*

MASON	*Manufacturing's Semantics Ontology*
MAST	*Manufacturing Agent Simulation Tool*
MDE	*Model-Driven Engineering*
MES	*Manufacturing Exuection System*
O3neida	*Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation*
OWL	*Web Ontology Language*
PC	*Personal Computer*
PLC	*Programmable Logic Controller*
RCA	*Reconfiguration Application*
RFID	*Radio-Frequency Identification*
SCADA	*Supervisory Control and Data Acquisition*
SOA	*Service-Oriented Architecture*
TCP	*Transmission Control Protocol*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*

CHAPTER 1

Introduction

The world of industrial manufacturing faces serious challenges due to the dynamics of the 21st century market. Even though the demand for mass commodities remains constant or even rises as a result of growing demand from emerging economies, the continuously decreasing life cycle span of products such as mobile phones combined with an increasing individualization significantly boosts the product variety [1]. In this context, manufacturing systems are required to support mass customization with the capability of "made-to-order" instead of "made-to-stock" [2]. The concept incorporates "the ability to provide individually designed products and services to every customer through high process agility, flexibility and integration" [3]. The principle of mass customization was anticipated by Alvin Toffler already in 1970 and detailed for the first time by Stan Davis in 1987 [4]. The automotive industry, once prototypical for mass production, was among the earlier adopters of mass customization around 1990 with Toyota offering a five-day delivery for customers, who could decide from a range of modular options concerning their desired car [5]. Even though the possibilities and conceptual aspects were already described before the turn of the millenium, the importance of mass customization as a major manufacturing strategy in various domains ranging from the food industry to mobile phones emerged only during the last decade, thereby introducing new technological demands and challenges [6]. Put in a nutshell, production systems are forced to "produce a number of different, high-quality products via short production runs, short changeover times, and low work-in-process" [5].

1.1 Evolution of Manufacturing Paradigms

Throughout history, mankind applied various technologies for automating different production processes thereby reducing the manual workload. The term automation itself descends from the Greek word *automatos* meaning "self moving, self thinking". In the context of manufacturing, automation implies the use of artificial instruments in order to automatically run a process. According to Polke, a process is by definition a set of consecutive interacting activities within a system for transforming, storing or transporting material, energy or information [7]. Concerning a technical installation, automation refers to arranging certain machines and instruments on the installation for ensuring an automatic job execution [8]. The development and application of new technologies is often also accompanied by a shift in the prevalent manufacturing paradigm, which also includes organizational aspects. Both technical as well as organizational innovations aim at keeping or creating competitive market advantages. Figure 1.1 depicts the evolution of manufacturing during the modern industrial era after the second industrial revolution around the mid-19*th* century [9], which encompasses the following manufacturing paradigms [10]:

Mass production and dedicated manufacturing systems: The basis of mass production was set during the mid-19*th* century in American armories with the introduction of interchangeable parts for produced guns. Therefore, this principle became known also as "armory practice" but spread into civil manufacturing sectors later during that century. The full potential of this paradigm was firstly exploited within the Ford Motor Company for the production of the Ford Model T in the beginning of the 20*th* century [11]. Dedicated manufacturing systems with fixed tooling and automation later paved the way for producing specific products and parts at high volumes [10, 12].

Lean manufacturing: Requirements towards a higher product quality and a further reduction of costs led to the paradigm of lean manufacturing, which combines a wide range of practices. It includes the integration of principles such as just-in-time delivery of raw goods and supplier management, as well as quality assurance cellular manufacturing. The aim is to achieve an efficient manufacturing system for producing goods at high quality with little or no waste [13].

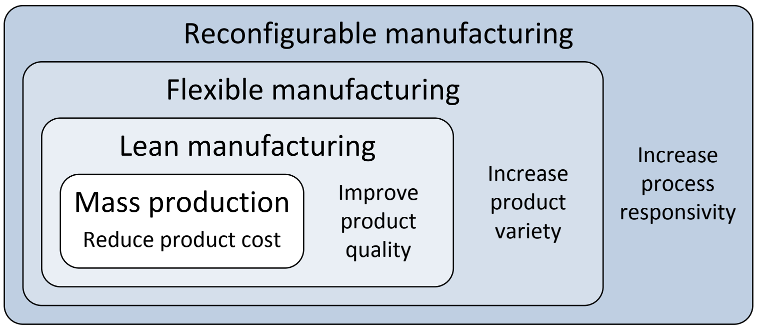

Figure 1.1: Evolution of manufacturing paradigms, based on [10].

Flexible manufacturing: Due to the occurrence of more dynamic market conditions with a demand of a larger product variety, flexible manufacturing systems were introduced. Such a system encompasses a fixed hardware configuration with tooling for several product part types of the same part family and programmable software for handling changes in the production schedule. Thus, several types of parts can be manufactured using one system with rather short changeover times [10, 12]. As flexible manufacturing systems are characterized by single-tool operations, their throughput is lower than that of dedicated manufacturing systems. In combination with higher equipment costs, the costs per produced part are significantly higher [14].

Reconfigurable manufacturing: To further increase the flexibility of a production system and thereby also its responsivity to changing market demands or technologies, reconfigurable manufacturing involves the usage of basic process modules for both hardware and software. Reconfiguration in this context encompasses the addition, removal or modification of these basic process modules for providing specific functionality when it is needed [10]. As can be seen in Table 1.1, this paradigm partially possesses or combines the properties of both dedicated as well as flexible manufacturing [14]. The main goal is to achieve flexibility in the production while maintaining a relatively high throughput. Thus, reconfigurable manufacturing is regarded as the key paradigm for realizing mass customization [12].

Property	**Dedicated**	**Flexible**	**Reconfigurable**
System structure	Fixed	Adjustable	Adjustable
Machine structure	Fixed	Fixed	Adjustable
Scalability	No	Yes	Yes
Flexibility	No	Yes	Yes
Productivity	Very high	Low	High
Costs per part	Low	High	Medium

Table 1.1: Properties of dedicated, flexible as well as reconfigurable manufacturing systems, based on [12, 14].

1.2 Background of Reconfigurable Manufacturing

Reconfigurability is defined as “the ability to add, remove and/or rearrange in a timely and cost-effective manner the components and functions of a system which can result in a desired set of alternate configurations” [15]. Changing the system’s functionality, both with or without modifying its physical structure, is commonly accompanied by a modification of the according control software [16].

1.2.1 Rationales for Reconfigurable Manufacturing

Reconfiguration for manufacturing systems is concerned with the following challenges [14, 17]:

- Producing heterogeneous products in small lot sizes;
- Auto-configuration management; and
- Response to failures.

Producing heterogeneous products in small lot sizes: Reconfigurable manufacturing plants can provide the aptitude for very small production lot sizes but come along with the problem of an exploding number of possible states due to the large variability of machines and products [18]. Depending on the individual product attributes, specific product-related handling procedures (e.g. grasping the product) have to be performed. By providing a production plan describing the product’s attributes and required procedures, the control system can determine how to handle the product. Required information could even be gathered at run-time by inspecting the product with a vision system, which is especially vital for disassembly processes [19]. Evidently, designing control policies for all possible manufacturing trajectories will at least be very difficult or even impossible in the case of new product specifications [20].

Mechanically reconfigurable machines with replaceable components and tools are regarded as a promising approach for coping with heterogeneous products [16]. This concept requires the definition and deployment of a control architecture and framework, which supports organizing the assembly of different modules in a suitable configuration. Considering that the number of machine parts can exponentially increase the number of possible system configurations [14], the tight and accurate coordination between the hardware-near control layer, which will handle the configuration, and a higher functional layer, which will synchronize it with other parts of the manufacturing system, is required [17]. Besides, changing such a system's functionality often requires a modification of the according control software [21]. Bi et al. identified the lack of effective technologies that can be used for reconfiguration as a significant obstacle that limits the development of reconfigurable machines, which are regarded as a backbone of future manufacturing systems [22].

Auto-configuration management: In the context of reconfigurable manufacturing plants and reconfigurable machines, reliable auto-configuration management is an important challenge that has to be solved for enabling plug-and-play components, which autonomously embed themselves into the system without any special initialization procedure or elaborate human involvement. Extensive manual effort results in high costs and an associated loss of production time decreasing thereby the benefits of reconfiguration [23]. This is especially of significance concerning the deployment and maintenance of nodes in large networks of small devices, e.g. sensor networks, as their manual configuration is rather impractical [24, 25]. As a large manufacturing system involves the cooperation of a high number of components, programming the relationships between the system components while considering the quantity of interrelationships related to failures poses significant challenges [26]. In this context, automatic configuration is proposed as a reasonable means for making a system scalable in the presence of changes and for supporting dynamic adaptation [17].

Response to failures: Reconfiguration mechanisms can be applied for dynamically modifying a manufacturing system's behavior, in order to effectively deal with unusual conditions like equipment failures without stopping the production [27]. Dynamically responding to such failures is critical especially for large-scale systems due to the number of possible combinations of relevant exceptions, which grows exponentially with the system size [28]. While the emphasis during normal operation can for instance be put purely on the system performance, the presence of a failure can shift the emphasis towards keeping the system at least in operation with an adequate, but probably degraded, performance. This can be denoted as graceful degradation, which represents a trade-off between the reachable performance and the available system capabilities [29].

In the case of a transportation system, a component failure may require the system to change its topological configuration in order to provide the reachability of all possible destinations, such as storages or work stations. In certain cases this can be done by changing the direction of specific conveyors [30]. Such a reconfiguration on the system-level requires intersections that can tackle changes of the conveyor configuration involving thereby local reconfiguration tasks. Evidently, not only the hardware but also the control software of the intersections has to cope with such a change of functionality. An internal reconfiguration of this control software has to result in providing the appropriate functionality depending on the intersection's role (diverter or junction).

1.2.2 Requirements for Reconfigurable Manufacturing Systems

Christensen identified a set of requirements concerning the development of future industry systems such as reconfigurable manufacturing systems [31]. Such a system needs to provide means for failure handling to react accordingly in the case of machine malfunctions and unpredictable process yields, but it also needs to provide a fast reaction on issued rush orders. In this context, critical factors are the system's availability despite its size and complexity as well as the system's robustness concerning its operability in the case of component failures. Extensive flexibility with self-reconfiguration capabilities is required to respond adequately to continuously changing product designs and small lot sizes. Even though identified in the last century, these requirements remain to be up to date [32], as future production systems are required to provide a high degree of reliability and fault tolerance as well as capabilities for reconfiguration [2].

According to Bi et al., the design requirements for achieving a reconfigurable manufacturing system, which is capable of answering to the challenges of mass customization, are as follows [22]:

- The control system should be scalable and upgradeable for providing the possibility of changing the manufacturing system's functionality or capability.
- For achieving scalability, especially in the case of geographical distribution of the physical system components, the control system should consist of modular components for both hardware and software.
- In order to have the manufacturing system achieve global system objectives, the control system should decompose the global objectives into local objectives that can be achieved by the modular components.
- For reacting on changed task specifications, the control system should have the capability of self-reconfiguration.

The design requirements mentioned above are in line with the key characteristics identified by Mehrabi et al. [10] as well as Koren and Shpitalni [14]:

- Modularity: The functionalilites, realized with either hardware, software or both, need to be designed in the form of modular components.
- Integrability: The components have to be designed to facilitate an easy integration in the system.
- Customization: The system needs to be flexible in order to match the application, i.e. the production of heterogeneous products.
- Convertibility: Capabilities are required, which allow a quick transformation of functionality to suit new production requirements.
- Diagnosibility: Diagnostic mechanisms for an automatic detection and identification of failures and problems need to be provided.

These key characteristics are of vital importance for keeping the efforts concerning reconfiguration in acceptable boundaries while keeping the system in operation [14].

1.3 Control System Architectures

Likewise to the manufacturing paradigms, also the control system architectures evolved gradually, and literature provides different views on this evolution and differs also concerning the terminology used for the various architectures and their elements. In the presented context, control and the term "controller" refers not to closed loop control theory but to the electronic devices for controlling an industrial process. Such a device is connected to the process via sensors as well as actuators and contains algorithms for affecting the process, which can and likely does involve also closed loop control algorithms [33]. These algorithms were manifested in logic relay circuits until the introduction of the Programmable Logic Controller (PLC) for replacing the relay-based control algorithms by programmable software [34].

1.3.1 Classification of Control System Architectures

One possibility is to classify the control architectures based on the relation between the controllers and the manufacturing equipment.

Centralized architecture: Control systems with a central entity were introduced during the 60s of the 20^{th} century (see Figure 1.2a). The pure centralized control architecture was common at the introduction of computer technology in automation replacing thereby analog controllers and panel-board displays. Due to the limitations and high costs of computer hardware at that time, all sensors and actuators were connected to a single central process controller using a star topology [35]. However, the central entity represents a single point of failure, which can result in a complete loss of control, making this architecture obsolete in current industrial practice for the automation of an entire production facility [1]. Thus, this type of architecture is found nowadays only on a small scale with single production cells controlled by one PLC [36].

Hierarchical architecture: By decomposing a complex problem into partial problems and distributing it over several layers, an arborescent hierarchical structure is achieved (see Figure 1.2b). Within this hierarchy, the relations between the control levels are based on the principles of a master/slave concept [37]. This control structure integrates the centralized cell level structures based on PLCs with functions, such as order priority or resource allocation, as well as even business functions, such as cost accounting on higher levels. It is commonly applied over the different layers of a manufacturing system in the industry [36]. As pointed out, the cell controllers, which

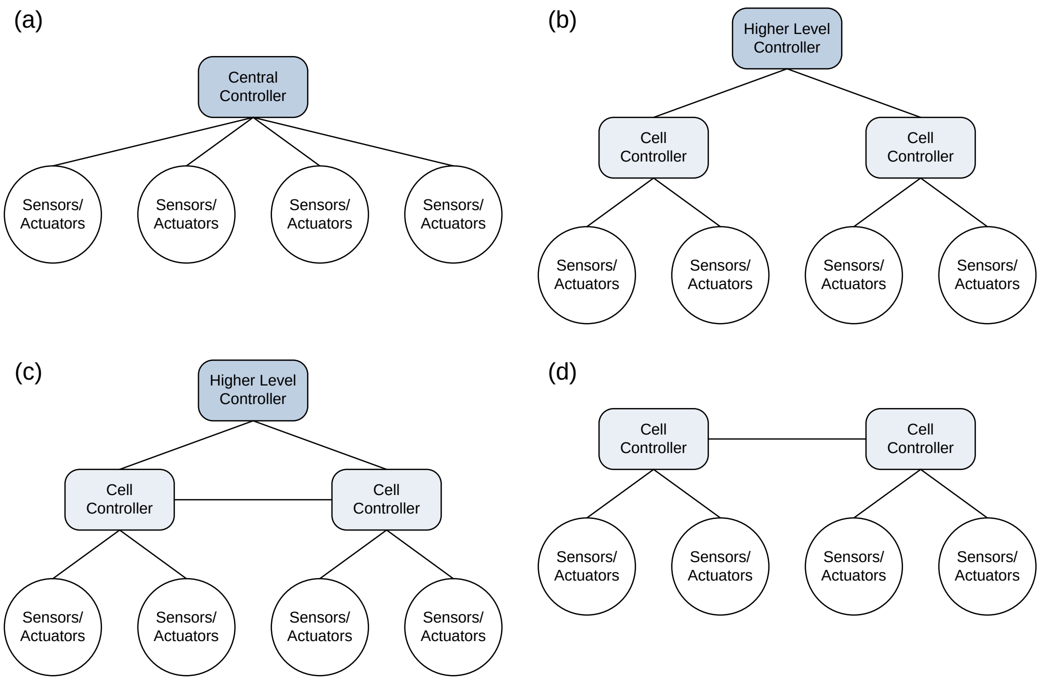

Figure 1.2: Classification of control architectures based on the relation between controllers and the manufacturing equipment: centralized architecture (a), hierarchical architecture (b), modified hierarchical architecture (c), and heterarchical architecture (d), based on [36].

are commonly PLCs in manufacturing systems, act as slaves for the higher control layers [36]. The resulting rigid hierarchical structure and communication brings the advantage of clarity and efficient scheduling of production operations as long as the conditions are projectable and stable, so for this environment the factory workload can be optimized [38]. These attributes facilitate the success of hierarchical control architectures in mass production domains [1]. While the implemented control mechanisms of hierarchical architectures are developed for optimizing fixed processes, they are not designed for coping with reconfigurable components and functionalities [39]. In this context, reconfiguration processes performed on a specific level have most likely a significant impact on other control layers, requiring appropriate adaptions also there due to the rigidity and strong linkages between the layers of such a control architecture [40].

Modified hierarchical architecture: By loosening the pure and strict vertical master-slave relationships between the control layers and adding horizontal interaction between the modules of a distinct layer (see Figure 1.2c),

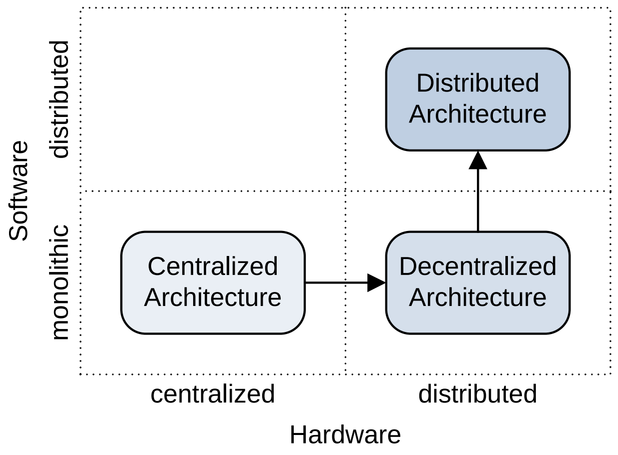

Figure 1.3: Classification of control architectures based on the configuration of hardware and software [8].

the modified hierarchical architecture improves the responsivity of a manufacturing system concerning disturbances [37]. Moreover, functions such as job dispatching and coordination may be added to the controllers on the cell level [36]. While this architecture is superior to the strict hierarchical architecture regarding the process responsivity, it still incorporates its drawbacks regarding reconfigurability [40].

Heterarchical architecture: Based on the significantly increased performance of industrial controllers, the heterarchical architecture was introduced, which allocates more complex tasks to the controllers on the cell level without a central supervising entity (see Figure 1.2d). Using extensive communication, the cell controllers themselves are responsible for planning and dispatching the production activities. A disadvantage hereby is the difficulty of global production planning optimization with the planning and scheduling algorithms carried out locally by the cell controllers [38]. However, as additional capabilities such as diagnostics and fault tolerance are also provided by the cell controllers [36], a fast local response of the control system to occurring events in the factory is possible, which encompasses the planning and scheduling activities [38]. A clear advantage of such an architecture is its modularity due to its inherent structure when composed of self-contained components [41]. Regarding the engineering process the software complexity is not increased when the number of components increase [36], but the analysis of the global system behavior as well as deadlock detection is more elaborate [41].

Figure 1.3 presents an alternative classification, which distinguishes control system architectures based on the configuration of hardware and software into the following control structures [8]:

- Centralized architecture: This architecture is compliant with the centralized architecture mentioned in the previous classification. Both hardware and software are centralized, respectively monolithic regarding their structure, which means that all sensors and actuators are hardwired to a central process controller and all control software tasks are merged in a single application executed within this single controller. As mentioned before, this architecture is obsolete in industrial practice regarding entire production facilities.

- Decentralized architecture: Based most likely on fieldbusses, the decentralized architecture employs still a single process controller but with decentralized Input/Output (I/O) modules. Hence, these hardware modules are distributed in the field and the usage of fieldbusses brings advantages such as more flexibility and a robust data transfer over long distances between the I/O modules and the central entity. On the contrary, the software is still executed as a single monolithic application within the central controller. Currently applied control systems based on PLCs for the cell level with decentralized I/O modules correspond to this type of architecture.

- Distributed architecture: The principles of the distributed architecture correspond to the heterarchical architecture as mentioned above. Instead of a central controlling entity, multiple controllers in the field including intelligent actuators incorporating embedded controllers are responsible for governing the tasks of the manufacturing system. Thus, the control software is no longer monolithic but instead distributed among the hardware devices. Composing production system components that incorporate both the hardware and their control software allows the formation of self-contained entities. Evidently, this approach implies challenges concerning the very different engineering and programming processes compared with systems based on PLCs as well as the communication efforts between the entities. Nevertheless, it significantly increases the modularity of the system and thereby its flexibility by employing exchangeable system components.

In the context of reconfigurable manufacturing systems, rigid hierarchical architectures are not suited for such a dynamic environment, as it is nearly impossible to preprogram adequate top-down responses for any abrupt changes [42]. Moreover, the complexity of the control system significantly increases in the case of large manufacturing systems [43]. On the contrary, a heterarchical/distributed architecture composed of modular and self-contained components satisfies the requirements of modularity and customization. Besides, this approach comes along with the benefit of code compartmentalization, easing the decision on where to put certain functionalities and providing therefore clear ownership boundaries among the software developers [44]. However, the development and application of a distributed architecture is connected with certain challenges to be met [45]:

- Communication: The distributed entities must be able to interact and synchronize while meeting timing requirements, which requires according (real-time) communication mechanisms.
- Modularity: In order to support adaptability for reconfigurable manufacturing, the system components need to be developed in a modular and self-contained way. Self-contained in this context means that the component contains all parts necessary for being complete, such as internal algorithms and data structures as well as interfaces to the component's environment.
- Openness: For enabling upgrades or modifications of the system, which can involve the integration of new components, the component interfaces should be implemented using non-proprietary specifications.
- Dependability: To be applied in industrial practice, the system must provide a sufficient degree of reliance due to economic and safety reasons for avoiding downtimes as well as danger to the environment respectively.

1.3.2 Emerging Control Software Paradigms for Distributed Control Systems

As a distributed control architecture incorporates a complex software system, structures and techniques are required for an easier handling of the inherent complexity [46]. Some general software engineering principles have been introduced for managing a control system's complexity:

- Decomposition: Dividing a large problem or system into smaller parts that can be addressed in relative isolation and managed easier supports its handling due to the limit of each part's complexity [47]. For applying decomposition it is necessary to define the roles of the subsystems or components respectively.

- Abstraction: Emphasizing some details while suppressing others delivers simplified models of a software component. This eases the design process of a complex system as not all the details of an artifact's implementation need to be considered for using it [48].

- Organization: Determining organizational relationships between components allows the formation of component groups, which can act as units providing more sophisticated functionalities [46]. Organization also incorporates principles such as inheritance taken from object-oriented systems as well as subroutines, as is common in procedural programming languages.

In the context of distributed control systems, a lot of recently performed as well as ongoing research concerning the employed control software paradigms is focused on the topics Holonic Manufacturing Systems, Service Oriented Architectures and Agent Technology [49]. Regarding manufacturing systems, research in all three topics heavily influences each other [2, 50]. Autonomous components are the basic building blocks for all three approaches. Even though "autonomy is a somewhat tricky concept to tie down precisely" [51], it actually means "that the system should be able to act without the direct intervention of humans" [52]. Consequently, even a simple thermostat represents an autonomous component.

Holonic Manufacturing Systems: The distributed architecture served as the basis for the developments of the Holonic Manufacturing Systems (HMS) Project [53], which was "an international industrially driven project addressing systematization and standardization, research, pre-competitive development, deployment and support of HMS architectures and technologies for open, distributed, intelligent, autonomous and co-operating systems through a global partnership". Based on the Greek word *hólos* meaning "whole" and the suffix *on* for denoting a particle, Arthur Koestler defined the word *holon* for entities, which behave "partly as wholes" and "wholly as parts" depending on the point of view [54]. This paradigm was adopted for the development of HMS, which are constituted by a highly distributed component-based architecture [55]. In this context, an entire manufacturing system is regarded as a holon incorporating other holons such as machines and components as well as non-physical entities like customer orders or production plans [45]. The research in the field of HMS has been coupled with research in the field of agent technology for manufacturing (see below as well as Section 2.1), as this technology is regarded as a suitable platform for developing holonic systems [43].

Service Oriented Architectures: Emerged from information technology, the Service-Oriented Architecture (SOA) is a design framework for composing an information system by incorporating services, which are program units that independently execute their function upon invocation [56]. For realizing an SOA, certain properties have to be taken into account in the engineering process [57]. Services encapsulate logic for a distinct context or task but multiple services can also be combined to a collective service. In order to minimize dependencies between services, they are designed as autonomous components with only loose coupling to other entities. Encapsulation, autonomy and loose coupling require according interfaces for ensuring interoperability between the software components [50]. An SOA is a suitable framework for a distributed control system but the automatic composition of services for reconfigurable manufacturing represents a current research challenge [58]. While start-up procedures for an automatic orchestration of services for production machines have been reported recently [59], the life-cycle support of SOA-based systems, as it is required for reconfiguration, is yet an open point [60]. As SOA systems and agent technology share substantial similarities, the ongoing research in these fields is closely coupled [50]. Some even regard agents as the key realization technology for creating SOA systems [61].

Agent Technology: The idea and notion of agents stems from the domain of Artificial Intelligence (AI), but it only makes up about one percent of an agent [62]. Generally, an intelligent agent can be viewed as an autonomous entity, which senses, decides and operates within its environment [63]. Thereby, the agent acts as an individual problem solver which considers its own course of action [42]. Agent technology complies well to the software engineering principles mentioned above for managing a complex control system:

- Decomposition: Complex systems are composed of multiple subsystems, which operate in conjunction for achieving the objectives of the complete system. This viewpoint can be adopted likewise for components realizing subsystems. Broken down, each component is responsible for achieving its own local objectives. Taking this objective-centric decomposition as basis means that the individual components should encapsulate their own control for achieving these local objectives [52]. This is a significant difference to a passive object and delivers truly self-contained entities. Besides, a distributed architecture has no single location of control as the control software is distributed among the participating components meaning that each component encapsulates its own control software. Accordingly, applying multiple agents allows representing the multiple locations of control as well as multiple perspectives or competing interests [64].

- Abstraction: The abstraction level of the agent-based approach is generally higher than the one of purely object oriented paradigms, as the behavior of agents more closely resembles human behavior in a simplified form [65], meaning that an agent represents a goal-driven, autonomous and proactive entity.

- Organization: Agent-based systems incorporate explicit representations of the organizational relationships and structures, which allows the development of a distinct agent (type) in relative isolation [46]. By employing mechanisms for flexibly forming or disbanding these relationships and structures, agents can be added to or removed from the system incrementally, which represents a significant asset for a reconfigurable manufacturing system.

For implementing the control software of a reconfigurable manufacturing system, agent technology offers modularity, decentralization, autonomy, scalability as well as re-usability [42, 46]. In regard to mass customization, Da Silveira et al. recommend the implementation of a multi-agent system composed of autonomous components for increasing the manufacturing system's flexibility and fault tolerance [3].

In order to ensure the correct interaction of agents, they have to share a common understanding of the syntax and semantics of the exchanged information [66]. The agents' environment representation needs to incorporate the environments' structure and the characteristics of objects, such as other agents [67]. Consequently, a knowledge base is required, which allows the explicit specification of an application domain while incorporating semantics into the data. This provides an explicitly understandable data format and promotes the data exchange between system entities [68].

A further challenge to be met is the consideration of time constraints. Controlling a manufacturing system in an industrial environment encompasses execution decisions involving real-time constraints. Hence, to be applicable in industrial environments, agent technology needs to be integrated with a real-time capable control layer [69, 70]. As the behaviors based on complex models require longer computation times, they have to be combined with a real-time control subsystem offering simple reactive behaviors [71]. Consequently, modular control components are required that comprise both a low level software entity with real-time capabilities and a high level software entity in conjunction with an extensive environment representation [49].

In regard to reconfigurable manufacturing, the development of according control software with reconfiguration capabilities represents an area of significant concerns [72]. Semantic models provide knowledge in machine-readable form but need to be combined with suitable reconfiguration mechanisms [49]. In this context, the reconfiguration of the hardware-near control layer is regarded as an important task for agent technology in manufacturing [71].

1.4 Contributions

The contribution of this thesis is the design and development of an agent architecture, which can be employed for controlling the physical components of a reconfigurable manufacturing system. An according knowledge base for the agents is defined to capture the relevant details of the manufacturing environment. Furthermore, a reconfiguration infrastructure is presented, which allows an agent to configure and modify its hardware-near control layer according to the required functionality of the controlled component. For showing the feasibility and advantages of the presented approach, it is implemented accordingly on a real system. Hence, the thesis encompasses the following contributions:

1. Design of an agent architecture for controlling the physical components of manufacturing systems: An agent has to encompass a high level software entity being able to make use of declarative knowledge as well as a hardware-near control entity with real-time capabilities. Consequently, the agents should be designed in a manner for achieving self-contained components. Thus, it is easier to modify the layout of a manufacturing system as the components are not strongly coupled regarding their control software. An agent also needs to involve mechanisms for detecting failures. Besides, the agent architecture should be designed to be usable for different kinds of manufacturing system components, e.g. intersections, grippers, or valves. The following papers describe the agent architecture in detail:

    - **W. Lepuschitz**, A. Zoitl, M. Vallée, and M. Merdan. "Towards Self-Reconfiguration of Manufacturing Systems using Automation Agents". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 41.1 (2011), pp. 52-69. [73]
    - **W. Lepuschitz**, M. Vallée, M. Merdan, P. Vrba, and J. Resch. "Integration of a Heterogeneous Low Level Control in a Multi-Agent System for the Manufacturing Domain". In: *Proceedings of*

  the 14th IEEE International Conference on Emerging Technologies and Factory Automation. 2009, pp. 1-8. [74]

- **W. Lepuschitz**, V. Jirkovsky, P. Kadera, and P. Vrba. "A Multi-Layer Approach for Failure Detection in a Manufacturing System based on Automation Agents". In: *Proceedings of the 9th International Conference on Information Technology: New Generations*. 2012, pp. 1-6. [75]

2. Definition of a knowledge base that is suitable for the automated configuration and reconfiguration of the control software: Knowledge about the system and its components should be described in a declarative manner. Thus, an approach for automated configuration and reconfiguration is easier to adopt for different kinds of target systems. The following paper introduces the usage of an ontology as knowledge base for the configuration process:

   - **W. Lepuschitz**, A. Zoitl, and M. Merdan. "Ontology-Driven Automated Software Configuration for Manufacturing System Components". In: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. 2011, pp. 427-433. [76]

3. Development of reconfiguration mechanisms integrated in the agent architecture: The agents should be able to reconfigure themselves for adapting their functionality. Thus, they can react to changed requirements by their environment, e.g. in the case of a modified system layout. The following papers present the reconfiguration mechanisms incorporated in the agent architecture:

   - **W. Lepuschitz**, A. Zoitl, M. Vallée, and M. Merdan. "Towards Self-Reconfiguration of Manufacturing Systems using Automation Agents". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 41.1 (2011), pp. 52-69. [73]
   - **W. Lepuschitz**, M. Vallée, A. Zoitl, and M. Merdan. "Online Reconfiguration of the Low Level Control for Automation Agents". In: *Proceedings of the 36th Annual Conference of the IEEE Industrial Electronics Society*. 2010, pp. 1359-1364. [77]

4. Implementation and validation of the developed approach: The approach should be implemented on the "Test-bed for Distributed Holonic Control", which is introduced in Section 3.2. Test cases are to be defined and evaluated regarding the benefits of the approach in the con-

text of reconfiguration. The following papers describe the validation and evaluation of the approach on the testbed:

- **W. Lepuschitz**, B. Groessing, M. Merdan, and G. Schitter. "Evaluation of a Multi-Agent Approach for a Real Transportation System". In: *Proceedings of the IEEE International Conference on Industrial Technology.* 2013, pp. 1273-1278. [78]
- **W. Lepuschitz**, B. Groessing, and M. Merdan. "Automation Agents for Controlling the Physical Components of a Transportation System". In: *Industrial Agents: Emerging Applications of Software Agents in Industry.* Ed. by S. Karnouskos, P. Leitão. Elsevier, 2015, pp. 323-339. ISBN: 978-0-12-800341-1. [79]

1.5 Thesis Outline

Following this introduction, Chapter 2 gives an overview about agent technology and ontologies. Furthermore, possible technologies for the hardware-near control layer are presented and compared regarding their reconfiguration aptitude. The chapter is concluded with the research questions concerning the work presented in this thesis.

Chapter 3 introduces the architecture, which is applied for the agents controlling the physical components of a manufacturing system. The two control layers of this architecture are described including the required interface for the intra-agent communication. Moreover, the diagnostic capabilities of the two layers as well as the reconfiguration infrastructure are presented.

An approach for automatically configuring a control system composed of automation agents is presented in Chapter 4. It is based on a generative programming approach and uses a declarative knowledge base. This knowledge base comprises information about the physical components in conjunction with the software concepts of the control application. Thereby, the agents are able to automatically compose their hardware-near control layer.

Chapter 5 is concerned with the local self-reconfiguration of the agents for providing alternative functionality as a reaction on a detected failure. The concepts of Chapter 4 are used and extended for enabling an agent to reconfigure its hardware-near control layer according to changed conditions of the agent's environment. Moreover, experimental results are presented which show the benefits of this approach.

Finally, Chapter 6 concludes this thesis by discussing the achievements accomplished in this work and by giving answers to the research questions. Moreover, further research possibilities are pointed out.

CHAPTER 2

State of the Art

In the following an overview of agent technology is given, which covers agent architectures and the topic of knowledge representation. Examples of agent-based applications in the industry are presented as well as approaches for failure detection and reconfiguration. In regard to the hardware-near control layer, two industrial standards are introduced and analyzed concerning their aptitude towards reconfiguration. Moreover, existing reconfiguration approaches for this control layer are presented and analyzed.

2.1 Agent Technology

According to Wooldridge, there is no universally accepted definition of the term *agent* [51]. He states that autonomy is essential for an agent but for instance the capability of learning might be important for some applications but undesired for others. Nevertheless, the idea of agents is used in various domains ranging from economic theories [80] and sociology [81] to computer science and artificial intelligence [82].

In regard to the usage of the term agent for denoting a software-based computer system, the following general properties can be attributed to an agent [82]:

- Autonomy: Agents operate autonomously without the direct intervention of other entities and perform decisions according to their state.
- Reactivity: Agents are part of an environment such as the physical

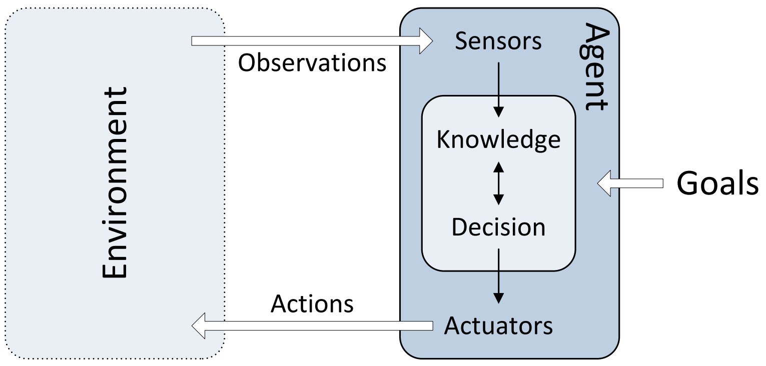

Figure 2.1: An agent interacting with its environment by making observations with its sensors and by executing actions with its actuators according to decisions based on the agent's knowledge and goals, based on [42, 83].

world including other agents, which they are able to perceive and react upon in the case of occurring changes.

- Proactiveness: Apart from reacting in response to the environment, agents are able to perform goal-directed behaviors in a proactive manner.
- Social ability: Even though agents can act autonomously, they do interact or negotiate with other agents (including also humans) for achieving goals in a cooperative or competitive manner using some kind of Agent Communication Language (ACL).

Figure 2.1 depicts the interaction of an agent with its surrounding environment. After making observations using its sensors, the agent combines this information with its knowledge about the environment. Depending on the agent's goals and preferences concerning the states of the environment, it then initiates and executes actions to change those states accordingly using its actuators [42]. Also agents without any physical embodiment obtain sensory inputs by means of file contents or received messages and act likewise upon their environment [83].

2.1.1 Agent Architectures

Depending on the agent's design and tasks in a system, it can incorporate various modules [84]:

- Communication interface: The communication interface is used for the agent's communication with other agents. Commonly used ACLs are the Knowledge Query and Manipulation Language (KQML) [85] and the FIPA-ACL [86]. Both ACLs share a lot of commonalities [87] and define message types and semantics for describing content.

- Perception and execution: The agent obtains data from its environment by using sensors. This module can also include mechanisms for filtering relevant information out of raw sensor input. Using actuators, the agent can also act on its environment.

- Knowledge representation: The knowledge base of an agent incorporates a representation of the domain of application, which describes details about products, tools and machines, as well as processes. Furthermore, concepts about the agent itself regarding its state, location and skills are contained in the knowledge base. Besides, it can encompass social knowledge on how to interact with other agents, e.g. the used ACL or the cooperation strategy.

- Reasoning: Performing reasoning represents a core ability for agents as it enables an agent to make decisions according to its knowledge. Reasoning can be implemented using simple rule heuristics, deductive logic, inductive logic or self-organization [88], as well as mechanisms such as constraint-, case- or schema-based reasoning, neural networks and fuzzy logic.

- Learning: By employing learning mechanisms, the agent is able to adapt its behavior to a changing environment due to modified system functionalities or occurred failures. Various mechanisms used for the agent's reasoning are usable for its learning, such as the case-based approach, neural networks or fuzzy logic [69].

- Coordination: This module defines how actions are performed when other agents are involved, which can encompass cooperative but also competitive strategies.

- Control, planning and scheduling: This module is responsible for mapping the agent's goals into operational actions, thereby manifesting its proactive behavior. This can be performed using either precompiled plans or the reasoning mechanisms for dynamically creating such plans [63].

An agent does not necessarily need to incorporate all modules. Besides, multiple modules can also be combined to one module, or a module can be split into several sub-modules. The architecture of an agent defines how it actually combines the information from its environment with its internal state for determining the actions to perform [89].

Reactive architecture: A reactive agent can be viewed as an intentional system, which means that its goals are implicitly implemented in the agent's behavior but not explicitly represented in a model of the surrounding world. Perceptions directly trigger specific actions and in the case that several actions are appropriate, they have to be performed in parallel or the most suitable one is chosen based on priorities [90]. In this sense, the mechanisms of a reactive agent resemble the execution of classic PLC control code.

Deliberative architecture: A deliberative agent relies on a far more extensive knowledge base incorporating an explicit representation of the surrounding environment, i.e. a symbolic model of the world. Decisions are performed by logical reasoning about the states within this world model. The challenge for this type of agent lies in representing the real world in an according symbolic description and enabling the agents to reason about it [91]. Compared to a reactive agent with predefined strategies, the computational costs are higher, but this is justified by the generative potential of the deliberative approach [90]. Regarding manufacturing systems, pure deliberative approaches tackle operations management or production planning and scheduling [71] and are thereby used in the context of a Supervisory Control and Data Acquisition (SCADA) system or a Manufacturing Execution System (MES).

Hybrid architecture: Even though the deliberative approach is significantly more powerful than the reactive approach due to the possibility of reasoning and learning to cope with new situations, its computation time may impede a rapid response to important environmental events [90]. Hence, combining the deliberative with the reactive approach, and thereby the best of both worlds, delivers the architecture of a hybrid agent. Generally, the two subsystems of a hybrid agent are structured according to a layered architecture with the subjacent reactive layer having a precedence over the deliberative layer, e.g. for ensuring the safety of the environment by guaranteeing a fast reaction on events [82]. Furthermore, the agent is capable of combining information at a higher level of abstraction in its deliberative layer for achieving its goals. In this context, the reactive layer provides its functionality as services to the deliberative layer [55]. Using such a layered architecture supports the modularization of an agent by clearly separating the functionalities, which then requires only a restricted knowledge each layer

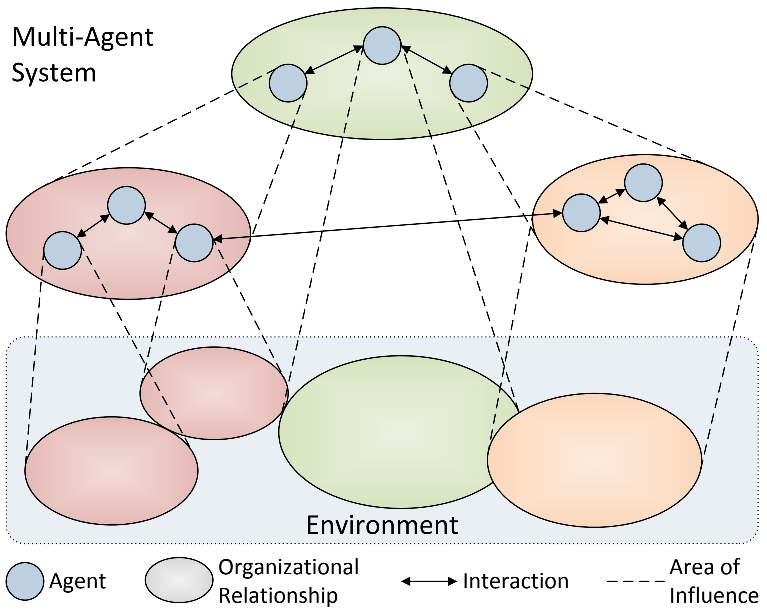

Figure 2.2: General structure of a multi-agent system [42].

needs to possess and consider [92].

2.1.2 Multi-Agent Systems

A Multi-Agent System (MAS) is constituted by a network of agents, which typically interact with each other (see Figure 2.2). Compared to a system relying on a single entity or agent, which may be a bottleneck due to limited resources or fail at a critical time, a MAS represents a society of autonomous interacting components dedicated to provide solutions for larger-scale problems [93]. Even the interconnection of multiple existing legacy systems is thereby possible by applying wrapper agents around each of these systems for enabling their interoperation [94].

Within a MAS, data are decentralized and each agent possesses most likely only a partial model of the surrounding environment, following individual goals, which may differ from the other agents' goals. Nevertheless, due to the cooperation of the agents, the MAS can provide solutions that are beyond the individual capabilities of the participating agents [42]. Cooperation is realized using different methods:

- Communication: Agents exchange information by communicating with each other. This can be done either using direct communication via messages or indirect communication via the environment using actuators and sensors [42].

- Specialization: Likewise to the physical components of a manufacturing system with each providing one or a set of functionalities, also agents are designed to provide specific services [84]. Two approaches can be distinguished [95]: functional decomposition and physical decomposition. Functional decomposition means the assignment of functions such as order planning or transportation management to specific agents. On the contrary, physical decomposition uses agents for the representation of physical entities such as machines, tools or products.

- Coordination: For ensuring a successful operation, coordination is crucial as otherwise "a group of agents can quickly degenerate into a chaotic collection of individuals" [84]. Coordination mechanisms are either based on information exchange among coequal agents, which can also involve agents acting as mediators, or on supervision with agents having a degree of control over others.

- Collaboration: For achieving common goals, collaboration encompasses techniques for assigning tasks to the agents [42]. For example, the Contract Net Protocol (CNP) is an auction-based bidding mechanism for allocating tasks to distributed nodes in a competitive manner by awarding a task to the best provider [96].

- Organization: Agents act on a defined influence sphere, which can for instance be a specific machine they control. Besides, the organization of a MAS also determines the potential interaction partners of an agent [42].

The collective behavior of the agents is denoted as emergent behavior as it emerges from their interactions with each other [97]. While intended emergent behavior is one of the main fundaments of agent technology, the possibility of unintended emergent behavior represents a significant obstacle regarding the acceptance of this technology in the industry [71]. In this context, simulation plays a valuable role for analyzing a system before its real deployment, which can also be adopted for agent-based systems [70]. However, Pěchouček and Mařík point out that more elaborate methods and tools are yet required regarding the verficiation and testing of agent-based systems [98].

Nevertheless, for designing and implementing the next generation of distributed and intelligent manufacturing systems, the application of agent technology has been widely recognized as an enabling approach [46, 99]. In the context of manufacturing systems, agent technology is expected to provide the following attributes:

- Robustness and reliability [93, 100, 46, 98]: The system is able to tolerate uncertainties and can cope with failures of components, as agents with redundant functionalities or appropriate interagent coordination are found dynamically.
- Reconfigurability [98, 101]: Reconfiguration provides fault tolerance and enables an agent-based manufacturing system to react on component failures or changed manufacturing demands.
- Flexibility [93, 46]: Agents with different abilities are able to adaptively organize themselves for problem solving.
- Re-planning and re-scheduling [46, 98]: The system is capable of re-planning its activities in the case of component failures or changed manufacturing demands thereby re-scheduling its job orders.
- Development and reusability [93, 98, 101]: The modularity of the agent approach can also ease the design process, as subtasks can be identified and assigned to specific agents instead of programming the complete task of the system within a monolithic control software structure. The functionality of specific agents can also be reused for solving different problems.
- Maintainability and scalability [93, 100, 98, 102]: A system composed of modular components, i.e. agents, is easier to maintain than a centralized system. Separating a system into its modules facilitates maintenance processes as each module is simpler to maintain than a complete system. Moreover, the number of agents can be modified due to this modular structure.
- Plug-and-play [98]: A further development of scalability is a system's ability concerning "plug-and-play", which is also referred to as "plug-and-operate" or "plug-and-produce". Given that this ability is supported, the modular components of a manufacturing system can be added to or removed during operation for quickly changing the system's provided functionalities.

However, although regarded as a promising approach and deployed in some applications throughout the last few years, the widespread adoption of agent-based approaches by industry is still missing. The following weaknesses are identified as reasons for that:

- Emergent behavior [103]: Emergence is integral in a distributed system for achieving the global behavior of such a system. However, due to a lack of formal procedures or algorithms for verifying a large-scale agent-based system, there is no guarantee that the emergent behavior will lead to the desired function of the complete system.

- Missing trust and concerns regarding the stability [93, 104]: Missing trust in the idea of delegating tasks to autonomous agents is existent due to concerns regarding the stability and survivability especially in unpredictable occurrences of system failures.

- Lack of performance evaluation [71]: Besides the missing guarantee on the desired function of an agent-based system, there is also a lack of algorithms and metrics for evaluating the operational performance.

- Absence of techniques and tools for deployment [49, 98]: Despite having also been investigated by industrial companies, a missing lack of design and development approaches mature enough for industrial deployment is existent.

- Custom-developed applications and lack of standardization [49, 105]: The applications reported in literature were developed in a custom-based approach, which means that the development efforts can be spread neither over multiple customers nor over time. This is due to a lack of agreed standards regarding agent technology.

- Real-time capabilities [49, 69]: Pure agent-based infrastructures do not meet real-time capabilities as required for directly controlling industrial equipment. The linkage with a real-time capable control layer based on other technology has therefore been always necessary.

2.1.3 Application of Agent Technology in the Industry

Agent technology has been applied in industry in various domains [49] ranging from automotive industry to logistics for handling high level tasks such as production planning and scheduling down to field level tasks in manufacturing control.

One of the first applications of agent technology in industry was realized by Rockwell Automation in cooperation with BHP Billiton for increasing the productivity of a steel milling process [106]. In this approach, autonomous cooperative units, respectively agents, represent either a specific product or some equipment for performing process operations. The main task of this agent-based control system is to determine a suitable set of equipment for producing a particular type of steel rod. Due to safety concerns, the agents did not control the process directly but provided recommendations to the operator [107].

Another agent-based solution for the steel industry was developed by the German Research Center for Artificial Intelligence in cooperation with the Saarstahl AG [108]. The so-called AgentSteel system's task encompass the planning and scheduling of the production process in the steelwork factory Völklingen. According to the orders of customers, the system calculates solutions centrally based on graph theory and route finding for the daily target schedules. Each of the aggregates producing the steel is represented by a distinct agent, which monitors the data from its aggregate in regard to the calculated schedule. In the case the agent detects that its schedule leaves the determined time window, it tries to adapt its schedule, which can affect also other aggregates. The system employs mechanisms such as the CNP [96] for reassigning tasks to other aggregates.

Bussmann and Schild report the development of a flexible transportation system with an associated agent-based control within the frame of the Production 2000+ project [39]. The overall goal of the system is to optimize the throughput and routing of cylinder heads to designated work stations. The system was evaluated in a series of simulations based on real manufacturing data (product types, processing times, failure characteristics, etc.), which showed the agent-based control's flexibility and tolerance towards failures of machines as well as of control units. Thus, a performance of 99.7% of the theoretical optimum of the throughput was achieved, which is significantly higher than the performance of the conventional manufacturing lines of that time with their maximum of about 75% of the theoretical throughput [109]. Moreover, the control system was installed as a bypass to an existing manufacturing line for cylinder heads in the Daimler Chrysler plant in Stuttgart-Untertürkheim for validating the results of the simulations under real manufacturing conditions. Even though the prototype showed a throughput increase of 20% during day-to-day operation for five years until the end of the cylinder heads life cycle [49], the system was not installed another time as the total costs for the system were considered to be too high. Schild and Bussmann conclude that for the automotive industry the costs for the higher flexibility are too high compared to the gained performance increase, but it might provide a different economic impact for industries requiring more flexibility [109].

The presented industrial applications mainly acted as demonstrators for showing the deployment feasibility of agent technology in industry. Without specifying concrete performances measurements, the authors of the first and second application mentioned above claim that the agent-based system performed well regarding process scheduling. The third application of an agent-based system mentioned above showed a throughput increase during day-to-day operation and measurements are presented by the authors. Demonstrators with more detailed performance analyses are required for investigating possible benefits of the application of agent technology in industry [98].

These solutions are usable for production planning and scheduling in their designated target system but they are hard to modify or adopt to other systems as the agents' knowledge structures are "too static, cumbersome and poor" [49]. Requirements such as flexibility and reconfiguration have led to the introduction of semantics and ontologies for the agents' knowledge representation in manufacturing systems [110].

2.1.4 Knowledge Representation

In order to perform reactive or proactive actions autonomously, an agent has to possess knowledge about its environment [42], but also about its own state and skills [84]. Derived from the field of psychology, knowledge can be separated into procedural knowledge and declarative knowledge [111].

Procedural and Implicit Knowledge: Procedural knowledge refers to a skill that can be performed based on subconceptual processing [112]. Thus, the knowledge is implicitly embedded in the performing entity and this information can therefore not be easily extracted [113] or modified [114]. Using the procedural knowledge does not necessarily involve declarative knowledge [112]. In regard to agent technology, procedural knowledge is implicitly manifested in an agent's reactive behaviors. Besides, also a deliberate agent needs to possess procedural knowledge to a certain extent for executing rules even if they are described in a declarative form as it needs "knowledge of how to do things" [111].

Declarative and Explicit Knowledge: Declarative knowledge involves the description of facts and things [113]. It can be articulated using formal language or mathematical expressions and therefore be constituted in an explicit form. Declarative knowledge is required for conceptual processing [112] and can be altered according to new information about the environment [111]. A declarative knowledge base can be described using different representation schemes:

- Logical representation: A knowledge base in this form is composed of logical formulas involving concepts such as variables and functions.

The formulas can be described using a specific logic as for instance first-order logic [115] or fuzzy logic [116]. Logical formulas can be added or removed for modifying the knowledge base, and inference rules can be used for tasks such as constraint checking or information retrieval. As pure logical representation schemes do not incorporate organizational principles, large knowledge bases tend to be unmanageable, which is a major drawback [114].

- Network representation: Such a form of representation scheme comprises objects for describing individuals of the world to be modelled and associations for representing the relationships between those individuals. Such a knowledge base, denoted as a semantic network, allows modifications by inserting or removing objects as well as by altering the associations between them. By using association types like `instance-of` for classification or `part-of` for aggregation, a network representation allows the organization and efficient management of even large knowledge bases. Deductions are for instance possible by using path-based inference along the associations, node-based inference with additional pattern nodes, or the combination of these inference approaches [117]. Also neural networks fall in this representation category, which are commonly used for machine learning [118] and pattern recognition [119]. Compared to logical representation, network representation incorporates organizational aspects and is easier to understand for humans. However, a drawback of a pure network representation is the lack of standard terminology and formal semantics [114], which means that there is no standard interpretation for the expressed knowledge [120].

- Frame-based representation: Derived from the semantic networks approach, the frame-based representation incorporates the concept of a frame, which represents a data structure for capturing a stereotypical situation [114], e.g. a type of machine. Thus, a frame contains general knowledge about object classes, thereby meaning knowledge that is true for most subclasses or individuals of this class. In a frame-based representation scheme, knowledge is therefore expressed in an object-oriented way [121]. The attributes of a frame are described in so-called slots of which each can either contain a distinct value or an association to another frame. The values are inherited of the superclass but they can be overridden by the distinct subclass or individual [120]. Frame-based representation enhances the network representation by allowing slots to contain procedural knowledge for automatically reacting on knowledge updates [122], respectively for describing the usage of the frame [121]. Besides, this type of knowledge representation shares the drawback of missing standards.

Regarding agent technology, declarative knowledge empowers a deliberate agent to apply logical reasoning in the form of inference rules [123]. Moreover, the declarative knowledge can even encompass the methods and procedures themselves in a declarative form [113].

Consequently, a hybrid agent architecture incorporates procedural as well as declarative knowledge. The development of both knowledge types can be performed by applying principles from Model-Driven Engineering (MDE), which involves the creation of abstractions for reducing the complexity of the real world [124]. Thus, a model is created, which represents information of distinct content for an entity to be used in a specific context [125]. Regarding the reactive layer of an agent executed for instance in a PLC, MDE can be used for developing control code based on a model defined in a modeling language such as Unified Modeling Language (UML) [126]. In regard to the deliberate layer of an agent, the declarative knowledge representation needs to provide the following properties:

- Provision of a symbolic environment representation [91, 127];
- Interpretability for the agents [123, 128];
- Comprehensibility for the human operator in regard of engineering processes [49, 128];
- Object orientation [123, 127, 128];
- Provision of common information among multiple entities to allow the exchange of information between them [71, 68]; and
- Extensibility and adaptability of the comprised concepts [127, 128].

For enabling knowledge sharing and reuse, ontologies have become widely used in the fields of AI and agent technology [129]. This might be explained by the fact that the concept of ontologies combines characteristics of logical as well as network and frame-based representation schemes. According to Gruber, an ontology is defined as an explicit specification of a conceptualization [130]. It incorporates classes (which are sometimes referred to as concepts) with according properties (sometimes referred to as slots) for describing their attributes, such as the size of an object, as well as the relationships between the classes (see Figure 2.3). Hence, an ontology is structured according to a network in an object-oriented way likewise a frame-based representation scheme, which makes it comprehensible for humans. A knowledge base is then constituted by the ontology in conjunction with a set of individual instances of the classes [131]. Ontologies can be described by using standardized formal languages like Extensible Markup Language (XML) for ensuring shareability and interoperability [132]. The most common ontology language currently in use is Web Ontology Language (OWL) [133], which incorporates description logics and thereby formal semantics, a main characteristic of logic representation schemes.

Using an ontology, it is possible to provide a semantic description of an agent's domain of application (e.g. the manufacturing domain) by specifying the meaning of entities and their relationships in the domain, as well as defining parameters, constraints, and possible consequences of actions [134]. Given appropriate procedural knowledge, autonomous entities like agents are able to access an ontology for retrieving information [135]. Moreover, based on information obtained from the environment they can manipulate the ontology's contents.

A significant advantage of such an approach is the possibility of modifying the knowledge base during runtime without the need for reprogramming. Product plans for new products, but also additional process knowledge can be added to be found and used by the entities of an agent-based manufacturing system. Hence, the usage of semantics and ontologies provides extensive capabilities of dynamic behavior for MAS [49].

Concerning the manufacturing domain, various ontologies have already been developed for particular purposes. A taxonomy of manufacturing components with physical resources (e.g. machines and tools) and concepts for production orders, operations and disturbances is described in [37]. The Manufacturing's Semantics Ontology (MASON) [132] focuses on likewise aspects and is based on three top-level classes: resource, operation and entity (see Figure 2.3). Machines and tools are regarded as resources, while raw material and other common helper concepts are considered to be entities. Operations are related to the process description and cover the performed procedures. An equipment module ontology with the focus on the functions and behaviors of equipment entities and their connections is presented in [136]. Even though not stated explicitly as an ontology, an initiative for establishing a framework for hardware as well as software interoperability at all enterprise levels was started by the Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation (O3neida) consortium [137]. The aim is to develop a searchable repository by collecting relevant product

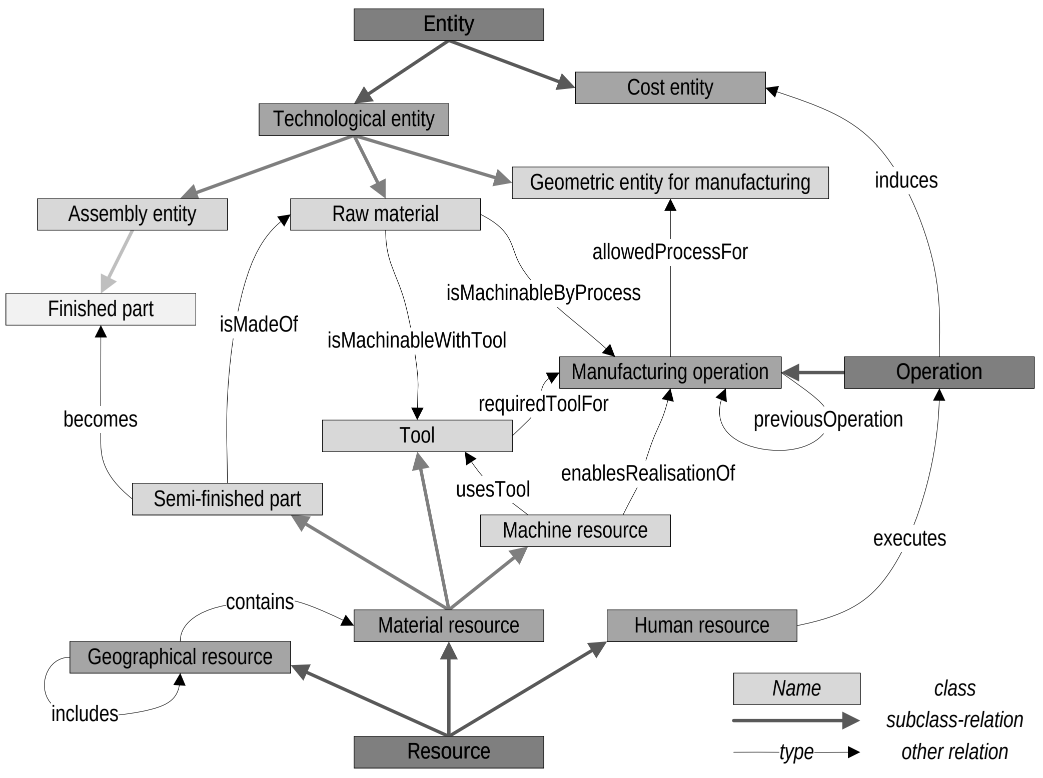

Figure 2.3: Overview of the MASON ontology's classes (boxes) and their relationships (arrows) [132].

data from manufacturers and integrators to encapsulate intellectual property with appropriate semantic information [138]. A complementary work is concerned with merging ontologies for mechatronic devices and automation reference models also covering both hardware and software features [139]. An ontology comprising also the concept of controllers is described in [140].

2.1.5 Failure Detection based on Agent Technology

Manufacturing systems with a modicum of human involvement have to possess the capability of autonomously recognizing occuring problems to impede production downtimes or even harm to their environment. If it is not possible to prevent them, the system has to react appropriately besides performing the scheduled regular operations. The early detection of a failure using automated monitoring can help to prevent unscheduled shutdowns, to reduce production losses, and to decrease the equipment maintenance frequency [141]. However, the weak modularisation of existing failure diagnostic systems causes difficulties related to upgrades or maintenance. This emphasizes the vital importance of the integration of monitoring and diagnostic systems directly in the manufacturing components for enabling them to detect and identify occuring failures [142]. A control architecture, which supports advanced diagnostics and simplifies the development of monitoring and diagnostic systems, will ease maintenance tasks and might increase such a system's acceptance by providing transparent insights concerning its functionality. This highlights the growing need for monitoring and diagnostic systems that are flexible, modular in their structure, cost effective and easy to maintain [143]. Besides, in a frequently reconfigured system, the usage of a static and pre-programmed diagnostic system seems rather impractical, due to the high number of configurations and critical situations to be considered. Hence, such a system requires components which are capable of self-diagnosis as well as mechanisms for monitoring emerging behaviors resulting from the cooperation of multiple components [144].

In this context, the multi-agent approach is regarded as a promising way for establishing intelligent system architectures for fault diagnosis in control systems [145]. Considerable research is reported on the development of diagnostic systems based on agent technology.

Bae et al. describe an intelligent real-time fault diagnostic system based on the application of a hierarchical artificial neural network [146]. Failures can be detected by single components of the system as well as by the cooperation of entities. Davidson et al. report the usage of MAS technology for automating the management and analysis of data from SCADA systems and digital fault recorders [147]. Tichy et al. introduce the Autonomous Cooperative System (ACS) framework, which is concerned with distributed industrial control applications [28]. It is applied for controlling a modelled scaled-down chilled water system of a U.S. Navy ship. Thereby, failures such as a water leakage are detected and alternative piping routes are determined.

Even though the presented approaches confirm the capability of agent technology for diagnostic tasks, they share the drawback of hard-coded information processing and reasoning behaviors [148].

Regarding the application of ontologies for diagnosis purposes, Albert et al. describe a concept for building a distributed architecture for the detection of anomalies in an industrial application [149]. Ontologies are employed for capturing details about the production plant and the agents' behaviors, as well as about the diagnostic algorithms. An implementation of the approach is not reported. Bunch et al. report on the application of a MAS for monitoring complex chemical processes [141]. They define notification ontologies for classifying event characteristics and notification directives. The aim of the approach is solely to provide extensive monitoring capabilities for the plant personnel but not to control the process itself.

2.1.6 Agent Technology for Reconfigurable Manufacturing

In regard to the paradigm of reconfigurable manufacturing, agent-based technologies meet the requirements needed to implement the corresponding control software [150]. Considerable research efforts have been done to improve the system-wide reconfigurability of a manufacturing system.

Bruccoleri et al. demonstrate that an agent-based model for a manufacturing system, containing ten machines of which one is reconfigurable, improves the performance in the case of a machine breakdown [151]. They propose an object-oriented high level control structure for error handling [152], which enables the determination of an alternative production schedule after the machine breakdown. Thus, product parts shall be routed to the reconfigurable machine that has to adapt itself accordingly. Based on the results of simulation, a performance increase of 17.4% is calculated in comparison to a control system without such mechanisms. However, the actual machine reconfiguration itself is not tackled in their approach.

Leitao and Restivo describe an adaptive holonic control architecture for distributed manufacturing systems denoted as ADACOR, which addresses an agile reaction to emergence and change, increasing the agility and flexibility of an enterprise when it has to operate in volatile environments [153]. While the production schedule is organized centrally during normal operation, the holons reorganize their production schedule by a high degree of interaction using a multiround CNP approach for task allocation in the case of an error. Regarding a production system with three manufacturing cells with several machines each, experimental results for two failure cases have shown that the loss of productivity in the ADACOR system is significantly lower (by 59% and 72%) than in a hierarchical control system. Reconfiguration in this approach is perceived as the reorganization of the services provided by the holons, which control the machines of the manufacturing cells.

Zhang et al. define an agent-based control architecture regarding agile manufacturing cells [154]. In their approach, each machine of a manufacturing cell is controlled by a distinct agent, which offers the machine's functionalities as services. The manufacturing cell is considered reconfigurable as machines and their agents can be added or removed. Thus, reconfiguration in this approach is considered like in the ADACOR approach as a reorganization of the provided services.

Further approaches such as the one by Lim and Zhang [155] as well as the one by Lima et al. [156] regard the reconfigurability of a production system in terms of dynamic process planning and scheduling by means of coordination mechanisms in a MAS.

The mentioned research work indicates the applicability of agent technology for reconfigurable manufacturing especially from a high level perspective. However, the integration of this technology with a suitable hardware-near control layer considering real-time information is considered as a further challenge and important issue for its wider application in the industry [69].

2.2 Technologies for the Hardware-near Control Layer[1]

Most multi-agent architectures mentioned in the literature are concerned with the high level structure, giving the hardware-near low level control layer only marginal attention. However, an easy, fast and transparent integration of physical automation components with agent technology is recommended for applying the multi-agent concept in industrial applications [70]. This can ensure real-time responsiveness of the agent-based control system and provide self-contained components for achieving a higher modularity [49].

To leverage the advantage of reconfigurable systems, the hardware-near control layer should be as flexible as the multi-agent driven higher level infrastructure. Currently the mainly used real-time control architecture in industrial automation is based on the standard IEC 61131 (see Section 2.2.1). A new emerging technology is the standard IEC 61499 (see Section 2.2.2), which extends IEC 61131 with the goal of increased flexibility and improved control software quality due to the incorporation of software engineering principles such as portability, configurability and interoperability [157, 158]. Both are candidate technologies for providing the hardware-near control layer of a control system based on a hybrid agent architecture. Regarding their aptitude towards reconfiguration, both standards are analyzed based on the findings of Zoitl [159].

2.2.1 IEC 61131—Programmable Controllers

The standard IEC 61131 was created by a workgroup of the International Electrotechnical Commission (IEC) consisting of representatives of PLC software developers, manufacturers and users [160]. It is divided into eight parts covering various topics concerning PLC systems such as equipment requirements or communication via fieldbus.

Part IEC 61131-3 [161], specifying languages for PLC programming, is widely accepted and spread in the manufacturing domain. According to

[1]Parts of the contents of this section were previously published in [73]

Zoitl et al. [158], the development of this standard was necessary due to the large number of different programming approaches and languages since the invention of the PLC. It comprises the definition of two textual programming languages (instruction list and structured text) and three graphical programming languages (ladder diagram, function block diagram and sequential function chart) for the development of control applications.

2.2.1.1 Applicability of IEC 61131 Control Software for Agent Technology

Several MAS applying IEC 61131 for the hardware-near control layer are reported in the literature.

Colombo et al. describe an agent-based control system called Factory Broker for the realization of an HMS [162] based on the Production 2000+ project [39] mentioned in Section 2.1.3. Computer Numerical Control (CNC) components as well as PLCs are employed for hosting the hardware-near control software of the machine agents controlling the production tools. Auction-based interactions are performed for dispatching production tasks to the machine agents [163]. Likewise to already presented applications of agent technology, the Factory Broker system performs reconfiguration on a high level scale concerning job scheduling. The agent-based system is able to reach an increased availability and a 35% more balanced throughput compared to a conventional system [162]. According to Schoop et al., this permits more precision regarding the pre-planning of the production output [164].

A MAS for shop floor assembly with a small transport system controlled by a single Beckhoff PLC is introduced by Cândido and Barata [165]. Software wrappers are employed for encapsulating the PLC-based hardware-near control layer into a set of agents. As the implementation details of this control layer are abstracted into services on the higher level agent layer, also the integration of legacy systems into the MAS is possible with this approach. Even though only emulated in the presented approach, the authors state that a system with every component having its own controller would improve the plug-and-play capabilities.

Developed by Rockwell Automation, the Manufacturing Agent Simulation Tool (MAST) [166] is presented by Vrba et al. for controlling a conveyor system in the manufacturing domain on top of a control layer programmed in ladder logic. Communication and message parsing between the agents and the subjacent control layer is realized by using the PLC data tables. Cooperatively, the agents of MAST search for routing paths and try to find alternative solutions in the occurence of a failure. MAST incorporates a “demo” mode with the lower level control logic and data table provided in a Java-based emulation for the simulation and evaluation of MAS for large scale transport systems.

Another multi-agent architecture reported by Rockwell Automation is concerned with the automation of a shipboard system [167], which was already briefly mentioned in Section 2.1.5. The focus of this system is to provide cooling water to the different systems of a United States Navy vessel depending on their momentary requirements. Moreover, diagnostic algorithms are incorporated to detect failures or complete losses of system components. In such cases, the system tries to reconfigure the distribution of cooling water accordingly to maintain the functionality of the remaining components.

To simplify the proliferation of MAS into industrial practice it might appear reasonable to base the hardware-near control layer on this standard. However, unfortunately IEC 61131 possesses certain drawbacks concerning its suitability for reconfiguration and none of the approaches with agent technology mentioned above tackles control code reconfiguration.

2.2.1.2 Reconfiguration Aptitude of IEC 61131

The applications presented in the previous section employ IEC 61131 for the hardware-near control layer and introduce system reconfiguration means purely on a higher level. Reconfigurability on the hardware-near control layer and its modularity are generally limited due to the centralized nature of PLCs based on the standard IEC 61131 [32, 168]. This section provides a closer look on the aptitude of this standard concerning reconfiguration.

As mentioned before, the standard IEC 61131-3 was developed with the main goal to unify the way of programming PLCs [158]. However, it was also attempted to introduce at that time current concepts from the domain of software engineering into the world of industrial control, with the aim of improving the quality, modularity, and reusability of control applications. The main concept towards this approach is the Function Block (FB), which encapsulates specific functionality and provides it to different applications in the same way. For developing applications, FBs are connected with each other in a data driven approach. By applying this concept, a key requirement for a reconfigurable architecture is fulfilled as the FBs represent modular pieces of control software [169].

However, additional means are existent in the standard that break the modularity and encapsulation of FBs, such as global variables. Global variables are often used in control applications as they simplify the data exchange between FBs [159]. This comes with the drawback of linking FBs together via a hidden interface even though they seem to be decoupled and completely independent from each other. Hence, changing one FB can result in major consequences in other application parts, although the FB is apparently not connected to these parts. At least the header of FBs comprises a declaration of the used global variables. However, a second global variable mechanism exists, which is called *var access*. By applying this mechanism, internal data of an FB can be provided as publicly accessible for the whole application as global data. For supporting dynamic reconfiguration, especially these hidden interfaces have to be considered and treated, which complicates dynamic reconfiguration in IEC 61131-3 significantly [159].

Furthermore, another key point for dynamic reconfiguration is the control of the execution order of the application's elements. IEC 61131-3 applies a cyclic execution model with typical cycle times in the order of 10 to 100 ms [170]. The execution order is not defined explicitly but by the data connections between the FBs. This impedes a direct control of the execution order by the user or reconfiguration coordinator [171]. Hence, as the execution of an FB is not explicitly triggered, it can be difficult to determine a suitable time frame for a reconfiguration process in order to avoid an ongoing algorithm execution of that FB [169]. Therefore, the practice for reconfiguring IEC 61131 systems is to change an application at once on the whole [172] and not gradually as it is required for modifying only functionality parts of a distributed control system with modular components [159].

Finally, no unified (re-)configuration interface exists, which allows reconfiguration coordinators (such as a software agent) to change the application. In current practice, these interfaces are vendor-specific and not publicly available due to intellectual property protection [159]. However, an open interface would be required in the case of a heterogeneous system with devices from different vendors, as it can be existent in industrial automation [168].

2.2.2 IEC 61499—Function Blocks

The standard IEC 61499 [173] has been developed in order to incorporate new demands of flexible and adaptive production systems into industrial controls. It introduces a generic architecture and guidelines for the usage of FBs in distributed industrial control systems. An IEC 61499 FB represents a functional unit of software containing data and algorithms as well as internal variables to constitute an internal state (distinguishing it therefore from a pure function) [157, 174]. No global data access is available and the FBs can be developed and tested independently from the control application. Engineering a system with IEC 61499 FBs conforms to the paradigm of object- and component-oriented design, as it is common in general software engineering [175]. Consequently, the functionality provided by an FB behaves the same, independently from the context it is used in and independently from other components connected to it [176]. Therefore, FBs require a well defined interface, but they can likely have an interface comprising various different event and data ports (inputs as well as outputs). This may result in large interfaces (i.e. many ports) with numerous connections for connecting different FBs, which is a drawback concerning the automated (re-)wiring of FB networks in a (re-)configuration process for instantiating or changing an application's functionality. The adapter concept of IEC 61499 is dedicated to reduce these issues by allowing the combination of a set of events and data to form a functional interface [171]. As they can incorporate input and output ports, adapter interfaces themselves are neither denoted as input or output ports but as plugs and sockets.

2.2.2.1 Applicability of IEC 61499 Control Software for Agent Technology

The design of IEC 61499 is strongly coupled with the HMS project [177] as many developers have been active in both projects. The main idea is the achievement of an agile and adaptive lower level control architecture as basis for an HMS [31]. Therefore, adaptability and reconfigurability were main requirements for developing the IEC 61499 architecture. In this context, the linkage of agent technology to real-time information and its integration with the FB-technology is seen as a promising solution for the integration of manufacturing process planning, scheduling, and execution control [69].

In the domain of distributed control, Lopez and Lastra use the IEC 61499 standard as lower part of an agent architecture for controlling physical components of a manufacturing system. The hardware-near control layer based on IEC 61499 serves as a proxy for the superjacent agent layer concerning its interaction with the process. Both layers are run on a Java-based embedded controller with an interface for the intra-communication between the two control layers using shared memory for accessing data [178]. The work does not involve an extensive application, but shows that the integration of both control layers on one embedded controller is in principle possible. Tests regarding the inter-agent-communication on the agent layer reveal a message time of approximately 2 seconds on the given controller infrastructure, which is not sufficient for real-time collaboration between multiple agents. Thus, such communication needs to be performed on the hardware-near control layer with its real-time capabilities, making in this case use of the likewise message-based communication protocols of IEC 61499 [179]. Reconfiguration of the control code is not reported in this approach.

Vyatkin and Peniche intend the utilization of IEC 61499 for more than realizing the actual process functionality (e.g. drilling a hole into a workpiece) as they suggest encapsulating higher level agent functions directly into FBs as well [180]. This includes for instance the registration of an agent to a separately implemented agent broker, which maintains a list of the provided component functionalities. Applying such an approach renders the integration of both control layers into one controller significantly easier. However, reconfiguration mechanisms are not presented and the approach does not involve any declarative knowledge representation for the agents as a PLC-like standard as IEC 61499 is used for implementing the higher level functions. As previously mentioned, a modification of a knowledge base constituted in procedural knowledge is elaborate and not easily adaptable.

Likewise to IEC 61131, also IEC 61499 is in principle suitable as a basis for the development of the hardware-near control layer. However, in the context of reconfiguration IEC 61499 offers advantages as compared to IEC 61131.

2.2.2.2 Reconfiguration Aptitude of IEC 61499

The architecture of IEC 61449 is based on IEC 61131-3 and extends it in several points. Likewise, the FB represents the main element of function encapsulation. However, in IEC 61499 no global data and indirect data access is available [159]. Therefore, FBs can be developed and tested independently from the control devices and from the application they are used in. This greatly increases the reusability and furthermore eases the reconfiguration as the impact of changing or replacing an FB can directly be derived from the elements it is connected to.

As a second major change, the data-driven approach of IEC 61131-3 is replaced by an event-driven approach. That means that additional event connections control the execution order in an IEC 61499 application, meaning that the order of execution in an application is explicitly specified. This ability can also be utilized during the reconfiguration process. Commonly a reconfiguration process is triggered when for instance a process variable reaches a specific value or a specific point in time is met. This makes an event-driven approach an appropriate choice for the control software, as a reconfiguration could be triggered directly by control application events [181]. Furthermore, the event-driven approach allows an increased flexibility in controlling the execution behavior of the application during the reconfiguration process [159].

For supporting the requirements derived from the HMS project, a basic (re-)configuration interface is defined in IEC 61499, which encompasses a standardized command syntax for downloading, modifying or uploading applications. However, this basic interface allows only conducting simple reconfiguration tasks, as a full support for dynamic reconfiguration is beyond the scope of the standard [182]. For dynamic and real-time constrained reconfiguration this interface is not sufficient and an improved infrastructure is needed.

2.2.3 Reconfiguration Aptitude Comparison

A control paradigm that supports logical as well as spatial software distribution and reconfiguration is recommended for highly flexible systems in the manufacturing domain [157, 183].

This perception goes along with two key issues concerning the development of the hardware-near control layer: an increased element reuse as well as the dynamic control software configuration and reconfiguration, in order to adapt quickly to changed production requirements. In this context, a modular and strongly component-oriented architecture with control software components, interacting via defined interfaces, is considered to be the enabling technology [159].

Based on the analyses presented in Section 2.2.1.2 and 2.2.2.2, Table 2.1 shows a comparison between the standards IEC 61131 and IEC 61499 regarding their general suitability for reconfiguration processes. Fulfilled criteria are represented by cells in green color while those not met are marked in red. Yellow cells indicate partially fulfilled criteria.

Criterion	**IEC 61131**	**IEC 61499**
Control system architecture	🟥 centralized/ decentralized	🟩 distributed
Execution model	🟥 cycle-based	🟩 event-driven
Functionality encapsulation	🟨 in FBs, but hidden interfaces possible	🟩 in FBs, no hidden interfaces
Global variables	🟥 yes	🟩 no
Unified reconfiguration interface	🟥 not provided	🟨 limited basic interface provided

Table 2.1: Comparison between IEC 61131 and IEC 61499 concerning their aptitude towards reconfiguration.

Evidently, compared to IEC 61131, the standard IEC 61499 fulfills most of the criteria and represents a suitable base architecture for a dynamically reconfigurable control layer. While IEC 61499 encompasses a syntax for reconfiguration commands that have to be implemented in a runtime environment for being compliant with the standard, IEC 61131 does not incorporate such a definition. Reconfiguration in the latter standard is thus rather a matter of provided distinct tools than standardization [158]. In this context, control software is committed as a whole onto the controller [32]. The lack of general suitability and available open tools might be the reasons why the approaches of online reconfiguration reported in the literature are based on IEC 61499.

2.2.4 Existing Reconfiguration Approaches

Zhang et al. present a system architecture for dynamic reconfiguration with the introduction of an orthogonal adaptation framework [184]. Their framework consists of additional data and event flows arranged orthogonally to the FBs, monitoring and controlling the configuration of the control program. The additional control flow is used by configuration control elements in each of the FBs [182]. A working implementation is presented, which shows that the FB network of a running application can be modified using a manual FB manager interface. As each FB has to bring its configuration control elements preprogrammed with it, the usage of existing standard FBs is impeded. Not complying to the standard represents a significant drawback of this approach regarding the portability of the developed FBs counteracting therefore the genuine openness of IEC 61499 [158]. However, without this configuration control element no reconfiguration is possible in this approach. Higher level entities such as software agents are intended to utilize this approach for automatically reconfiguring control applications, but details about such mechanisms are not presented.

Brennan et al. introduce the contingencies approach, which is based on preprogrammed configurations suitable for reacting on anticipated system states [185, 186]. It requires the determination of possible failure states as well as suitable recovery means before taking the system into operation. Predefined reconfiguration tables comprise the alternate configurations that can be chosen by a higher level entity such as an agent, for instance in the case of a device failure. However, the reconfiguration tables have to be maintained in an actual condition, requiring therefore an update each time even a minor change is made to the system. Olsen et al. reveal more details about an actual implementation of the contingencies approach [187]. An IEC 61499 based reconfiguration manager provides an interface for agents to reconfigure the subjacent control application. The implementation employs configuration agents that utilize this interface for loading or unloading control applications on the controller hardware. The reconfiguration process consists of three main steps: killing the running application on the controller, transmitting the alternate configuration to the controller, and finally starting the new control application on the controller. Due to limited resources of the available controller hardware, the entire demonstration implementation runs on a Personal Computer (PC). The contingencies approach is merely an adaption of state-of-the-art technology as current industrial control systems based on IEC 61131 provide likewise functionality [159]. Reconfiguration cannot be performed while the system is in actual operation and the need to effectively stop the running application is a drawback that results in downtime.

Brennan et al. also describe the soft-wiring approach, which shall ensure more flexibility for the hardware-near control layer as it is not based on preprogrammed configurations [185, 186]. By using services of the Function Block Operating System (FBOS), a higher level composed of configuration agents is able to replace existing FBs and to modify event and data connections between FBs. These agents possess information of the FB interfaces, which describes how an FB is to be connected with an existing FB in an application. These mechanisms shall allow the realization of dynamic reconfiguration. However, details about this interface information and how it is used are not provided. Concerning an actual implementation, only a rough version of the FBOS has been developed and no further research has been reported [159]. Thus, it cannot be concluded if the approach would be indeed applicable for dynamic reconfiguration.

Gouyon et al. present a reconfiguration framework for the development of a product-driven shop floor control system and its integration with a control system based on IEC 61499 [18]. The approach incorporates models for describing the manufacturing system capabilities as well as the production plans in the form of states and transitions. Product routes are determined using synthesis of the models of the production cell and the required assembly operations. A global configuration management agent is employed, which transforms the resulting desired system states and transitions into structured text or ladder diagrams within IEC 61499 FBs. This means that in principle a monolithic control application is generated. Thus, the approach seems to be promising in regard of dynamically instantiating a centralized architecture but it is not usable for being applied in a system composed of self-contained and autonomous entities with reconfiguration capabilities. Besides, an implementation on industrial controllers is not reported and only mentioned as future work by the authors, which hinders further analysis efforts about the applicability of the approach.

A further reconfiguration support based on IEC 61499 and implemented in Real-Time Java for the Archimedes Execution Environment is reported by Thramboulidis et al. [188, 189]. In their approach the reconfiguration process is split into two phases. During the first phase the preparation is performed with low priority. In this phase the definitions of FBs are down-loaded to the device, FB instances are created, and data connections are established. This phase may be interrupted at any time by higher priority control elements. During the second phase the time-critical reconfiguration tasks are performed, such as rewiring event connections or stopping FBs. As it is shown by Zoitl, this is not valid as the creation of a data connection can disturb the existing application and change its execution behavior [159]. The consistency in the reconfiguration process is not considered, so this approach provides only a basic support for structural reconfiguration. However, first measurements on the timing behavior of the basic structural reconfiguration are provided. A reconfiguration process encompassing the addition of two FBs to an existing application including their wiring takes about 32 ms. This means that such a reconfiguration process requires only roughly the time of one cycle in a typical PLC application, which may seem sufficiently fast for a range of applications. However, the approach does not tackle distribution and the issue of determining which FBs or connections should be modified.

Khalgui et al. report a reconfiguration agent for modifying control software based on the standard IEC 61499 for embedded control devices [190]. The agents for controlling the production process sequences are at first modeled using state machines [191] based on the formalism of *Net Condition/Event Systems* [192], which represent an extension of *Petri nets*. A reconfiguration engine receives notifications from an interpreter realized by a small network of IEC 61499 FBs in the case of failures detected by sensors. The reconfiguration agent employs a converter, which incorporates a conversion table that defines an IEC 61499 configuration for each supported scenario, and sends according XML code blocks for performing reconfigurations [193] such as the creation, wiring or deletion of FBs. As the conversion table is defined before the system is taken into operation, this approach is comparable to the contingencies approach mentioned before. Hence, it does not support the tailoring of reconfiguration processes during runtime and is therefore not suitable for dynamic reconfiguration.

Alsafi and Vyatkin introduce an ontology-based reconfiguration agent, which creates a new system configuration due to changes in the manufacturing requirements or environment. In their approach, an ontology is used for modeling the manufacturing system components as well as the process operations [194]. Based on the given requirements, the reconfiguration agent infers the required process operations from the ontology model [195] and issues the creation of IEC 61499 applications within the distributed controllers of the system [140]. Likewise to a previously mentioned approach, also this approach is based on a single reconfiguration agent, which makes it therefore a suitable candidate for dynamically setting up a centralized architecture.

However, as an implementation on industrial controllers is not reported and only mentioned as future work by the authors, a further conclusion about the applicability of the approach cannot be given.

Zoitl presents a reconfiguration infrastructure with a unified reconfiguration interface for modifying IEC 61499 control software in distributed controllers. Reconfiguration services are identified for changing the structure as well as the execution state of the application and the comprised FBs. By composing a reconfiguration application with the according reconfiguration services encapsulated within additionally instantiated FBs, control applications can be reconfigured even regarding hard real-time constraints [159]. An implementation is reported encompassing the successful reconfiguration of a closed-loop control strategy while keeping an inverted pendulum in balance. In the given case, a specific FB is substituted by three other FBs, which constitute a different control strategy. However, planning and implementing such a reconfiguration application by hand is considered to be an elaborate process, so this infrastructure needs to be incorporated into an architecture for conducting reconfiguration processes autonomously [196].

Table 2.2 gives an overview of the mentioned approaches concerning the following attributes:

- Support of dynamic reconfiguration during runtime (A): Reconfiguring an application during operation allows the avoidance of downtimes.
- Support of distributed reconfiguration of the components (B): Distributed reconfiguration represents a prerequisite for the composition of self-contained components with self-reconfiguration capabilities.
- Conjunction with agent technology (C): A reconfiguration process needs to be planned and executed. Agent technology can be used for automating such a process.
- Existence of an implementation (D): An implementation shows that a conceptual approach is in principle feasible.

Fulfilling all mentioned criteria would deliver an implemented approach, which incorporates dynamic reconfiguration mechanisms that can be planned and executed by distributed self-contained components. Thus, the key requirements as defined in Section 1.2.2 could be met.

As can be seen in Table 2.2, none of the reported approaches fulfill all mentioned criteria. Zhang et al. do not report but intend the usage of agent technology for planning the reconfiguration processes. However, this approach can only be used with particular FBs incorporating specific configu-

Authors	**A**	**B**	**C**	**D**	**Comment**
Zhang et al.	√	√		√	Preprogrammed configuration control elements are required for each FB, which impedes the usage of standard FBs for the control applications. Hence, this approach requires a customized FB library.
Brennan et al. (contingencies approach)		√	√		Complete applications are deleted and created based on predefined configurations. Thus, dynamic reconfiguration is not supported by this approach.
Brennan et al. (softwiring approach)	√	√	√		Details on how a reconfiguration is planned and executed are not presented. No working implementation of the approch is reported in the literature.
Gouyon et al.	√		√		A single reconfiguration agent is employed as the distributed control entities possess no self-reconfiguration abilities. An implementation is not reported.
Thrambolidis et al.	√			√	A central entity reconfigures the control applications in the distributed controllers. A consistency problem is existent within this approach.
Khalgui et al.		√	√	√	Complete applications are deleted and created based on predefined conversion tables. Hence, dynamic reconfiguration is not supported by this approach.
Alsafi and Vyatkin	√		√		A single reconfiguration agent is employed as the distributed control entities possess no self-reconfiguration abilities. An implementation on controllers is not reported.
Zoitl	√	√		√	The reconfiguration application is generated manually, which is a rather elaborate process.

Table 2.2: Comparison of existing reconfiguration approaches.

ration control elements. The soft-wiring approach by Brennan et al. is suitable for the usage of standard FBs, but details about the planning process and a working implementation are not reported. The approach presented by Zoitl is also suitable for using standard FBs and a working implementation is shown. However, the according reconfiguration application has to be generated manually. Most other discussed approaches either do not support dynamic reconfiguration or are unsuitable for self-reconfigurable distributed entities.

2.2.5 Recent Developments for the Reconfiguration of Control Code

Within a concept of bridging SOAs with IEC 61499, Dai et al. describe services that shall allow the modification of FB types (e.g. inserting a state into a basic FB) during runtime [197]. While these services for modifying the internal architecture of FB types are presented only theoretically, an emulation on the basis of an airport baggage handling system is shown that involves the insertion of an additional FB type into an operating system. An IEC 61499 runtime denoted as function block service runtime is used for hosting the FB application. The presented use-case shows the commands required for inserting a new FB type. Afterwards an existing FB instance is replaced with an instance of this new FB type. However, it is not clarified how the decision on adding the new functionality is performed.

Sorouri et al. introduce an agent architecture denoted as MIRAs (Modular, Intelligent and Real-time Agent), which involves a hardware-near control layer based on IEC 61499 [198]. In this architecture, each agent represents either a mechatronic component or a product. Using this architecture, production steps are created dynamically based on customer orders and the agents shall be able to generate their own control code [199]. However, details on how an agent actually generates the IEC 61499 FBs are not presented. A previous approach by the authors incorporates the definition of rules for describing mechatronic components and their properties on the basis of one robot as use-case [200]. Solutions for the robot configuration can then be found by using these rules in conjunction with given goal positions and conditions to consider. As further work the authors envisage the extension of this approach for generating software components based on IEC 61499.

Priego et al. present an architecture that allows the automatic generation and update of IEC 61131 control code [201]. It comprises several models for describing the components and operations of a plant as well as the work pieces and hardware components of PLCs. Plant and product model are parsed into a so-called hierarchical interconnected state space model with the implementation of ontological semantics. An algorithm is employed that determines executable operations based on preconditions of the system state and takes operation durations into account [202]. This delivers an operation strategy, which is then transformed into IEC 61131 control code. Finally a complete automation project is generated, which is loaded into the PLC when its execution has stopped in a safe state. Consequently, the main drawback is the necessity to stop the PLC during the code generation process, which is typical for approaches that are based on the standard IEC 61131.

Besides of modifying the functionality of a system, automatic reconfiguration can also be used for relocating control code from one controller to another. Yan et al. report on an approach for relocating IEC 61499 FBs between devices in order to provide dynamic resource allocation mechanisms for distributed industrial automation systems [203]. Consequently, this approach does not focus on choosing FBs to compose an application but on choosing devices as targets to download FBs. Load balancing between devices is done by analyzing workload information offered by the employed NxtStudio IEC 61499 platform [204] in regard to a device's computational power. Streit et al. describe a further concept, which is concerned with the relocation of control software in the case of a component failure [205]. Several conceptual scenarios are presented as rationales for such an approach, but implementation details are yet missing. The authors state that in future work a deployment algorithm will be chosen, which ensures that a controller possesses the required computing resources to run allocated control software.

Most approaches mentioned in this section are concerned with the topic of control software reconfiguration in regard of functionality likewise to this work and were reported during its composition. Those presented by Dai et al. and Sorouri et al. sound promising, but certain details on how to generate IEC 61499 software are missing or envisaged as future work. On the contrary, Priego et al. provide details about their approach but it is designated for control code based on IEC 61131. Finally, the approaches described by Yan et al. and Streit et al. focus on software reconfiguration with a different purpose, which is the relocation of control code from one controller to another.

2.3 Summary and Research Questions

As pointed out in Chapter 1, the paradigm of reconfigurable manufacturing is essential for realizing mass customization by providing manufacturing systems with changeable components and thereby with modifiable functionality.

As rigid hierarchical control architectures are not suitable for such a dynamic environment, a distributed control architecture composed of modular components is required.

For implementing such an architecture, agent technology is regarded as an enabling approach. Even though its application in the industry is yet limited, agent technology has been successfully used concerning diagnostics and reconfiguration as shown in Section 2.1. In this context, a knowledge representation based on ontologies allows the explicit specification of the agents' domain of application.

In order to be applicable in an industrial environment, MAS have to incorporate an appropriate hardware-near control layer for accessing the field level with its sensors and actuators. Even though the standard IEC 61131 is widely accepted in the industry, its reconfiguration aptitude is limited as shown in Section 2.2.1.2. Besides, Zoitl et al. report a lack of tools for dynamic reconfiguration processes in IEC 61131 [158]. On the contrary, IEC 61499 has been designed in regard of adaptability and reconfigurability. Moreover, several reconfiguration approaches are already reported in the literature as presented in Sections 2.2.4 and 2.2.5.

However, according to the analyzed state-of-the-art, a reconfiguration approach and implementation, which enables an agent controlling a component of a manufacturing system to reconfigure its hardware-near control software based on standard FBs, is yet missing. Nevertheless, concepts covering various aspects of such an approach are described in the literature that can be taken as a basis for its design and development. An overview of the aspects identified for achieving the objectives defined in Section 1.4 is given below. Moreover, the research questions for this thesis are presented on the basis of these identified aspects.

A suitable agent architecture for controlling the manufacturing system components needs to be developed. On the one hand an agent has to encompass a high level software entity for making use of declarative knowledge. On the other hand it has to incorporate a control layer with real-time capabilities for accessing the hardware. If the agents are designed in a manner for achieving self-contained components, it is easier to modify the layout of a manufacturing system. Achieving a *hybrid agent architecture* is therefore *identified aspect 1*.

Diagnostic mechanisms have to be incorporated in the agent architecture, which enable the monitoring of the controlled component. Failures should be detected using a component's sensors or by combining local information with data gathered from other entities of the system. The integration of *diagnostic mechanisms* represents *identified aspect 2*.

Taking both aspects 1 and 2 into account leads to research question 1.

Research Question 1: Is it possible to realize self-contained components of a manufacturing system and incorporate diagnostic mechanisms by using agent technology?

A knowledge base is required, which is suitable for the automated configuration and reconfiguration of the control software. If the knowledge about the system and its components is described in a declarative manner, an approach for automated configuration and reconfiguration is easier to adopt for different kinds of target systems. An according ontology needs to be defined, which provides the knowledge for autonomously configuring and reconfiguring the hardware-near control layer. Defining an *ontology as agent knowledge base* is therefore *identified aspect 3.* Hence, the second research question arises.

Research Question 2: Is an ontology apt for representing the concepts about the physical system and the control software, so that an agent is able to automatically configure its hardware-near control layer?

The development of reconfiguration mechanisms that can be integrated in the agent architecture is required, so that agents are able to reconfigure themselves for adapting their functionality. Consequently, they can react to changed requirements by their environment, e.g. in the case of a modified system layout. Using the dynamic reconfiguration mechanisms, an agent should be able to reconfigure its hardware-near control layer. The integration of *dynamic reconfiguration mechanisms* represents *identified aspect 4.* The third research question is derived from this aspect.

Research Question 3: Is it possible to integrate reconfiguration mechanisms in both layers of a hybrid agent architecture for achieving self-reconfiguration?

In order to validate the developed approach, an implementation is required. The approach should be implemented on a real system, i.e. the "Test-bed for Distributed Holonic Control". Test cases have to be defined and evaluated regarding the benefits of the approach in the context of reconfiguration. The *implementation and validation* of the developed approach is therefore *identified aspect 5.* This leads to the fourth research question.

Research Question 4: Is the developed approach feasible and beneficial on a real (laboratory) system?

CHAPTER 3

Automation Agent Architecture[1]

This chapter covers the core entity of the work presented in this thesis: the *Automation Agent*. The term itself is derived from the term automation component, which is used for instance by Sünder et al. for denoting an autonomous device in a distributed control system [206, 207]. Automation agents shall serve as the basic building blocks of the control system and are responsible for directly controlling the physical components. Embedded within a MAS, they represent the lowest layer of agents but thereby provide access to the actual functionality of the manufacturing system.

3.1 Agent Layers

Generally, agents are used either for representing physical entities or for encapsulating functions [208]. In this context, a distributed intelligent control systems based on agent technology can be divided into two layers: the physical layer (i.e. represented by the automation agents) and the functional layer.

3.1.1 Functional Layer

Agents of the functional layer are those, which do not incorporate a physical representation. They are employed for carrying out higher level functions such as the scheduling of jobs or the distribution of tasks. For example, the

[1]Most contents of this chapter were previously published in [73, 74, 75, 77, 148].

Knowledge-based Multi-Agent System Architecture (KASA) introduced by Merdan encompasses the following agents on the functional layer [101]:

- Contact Agent: Created at the initialization of the system, this agent type is employed for managing the MAS. It creates other agents of the MAS and performs organizational and supervisory functions.

- Order Agent: When a product order is submitted to the MAS, an agent of this type is created, which is responsible for determining the suitable manufacturing equipment and for localizing the required materials for the production process. Furthermore, it decomposes the product order into work orders, which it delegates to a supply agent.

- Supply Agent: Agents of this type receive work orders, from which they extract specific manufacturing tasks. Based on these tasks, they coordinate the execution of the production process by negotiating with the agents controlling the manufacturing equipment.

Depending on the system architecture, the functional agents may exist in redundant forms for avoiding bottlenecks. The well-disposed reader may find more information concerning the provided functionality of these higher level agents in [101, 209].

3.1.2 Physical Layer

The physical layer encompasses the mechatronic components consisting of actuators and sensors. Each automation agent incorporates the controlled physical parts, which can be regarded as the agent's embodiment. The software part of the automation agent serves as its intelligence and controls the physical parts for achieving the agent's goals. This includes the operation of the mechatronic component at its own level as well as the collaboration with other entities of the system. In order to do so, the automation agent is equipped with rules and behaviors about collaboration as well as the controlled mechatronic component's usage, which involves also diagnostic tasks.

When connected to the functional layer, the automation agents register themselves at a functional agent (such as the directory facilitator in the Java Agent Development Framework (JADE) [210]), which maintains a list of active agents, in order to make their services available to the other agents. Figure 3.1 depicts a possible conjunction of the automation agents with the functional layer of KASA. Communication between the physical and the functional layer is based on messages using a communication infrastructure (e.g. ethernet) and a protocol for ensuring the common understanding of content between the communication partners (e.g. FIPA ACL [86]).

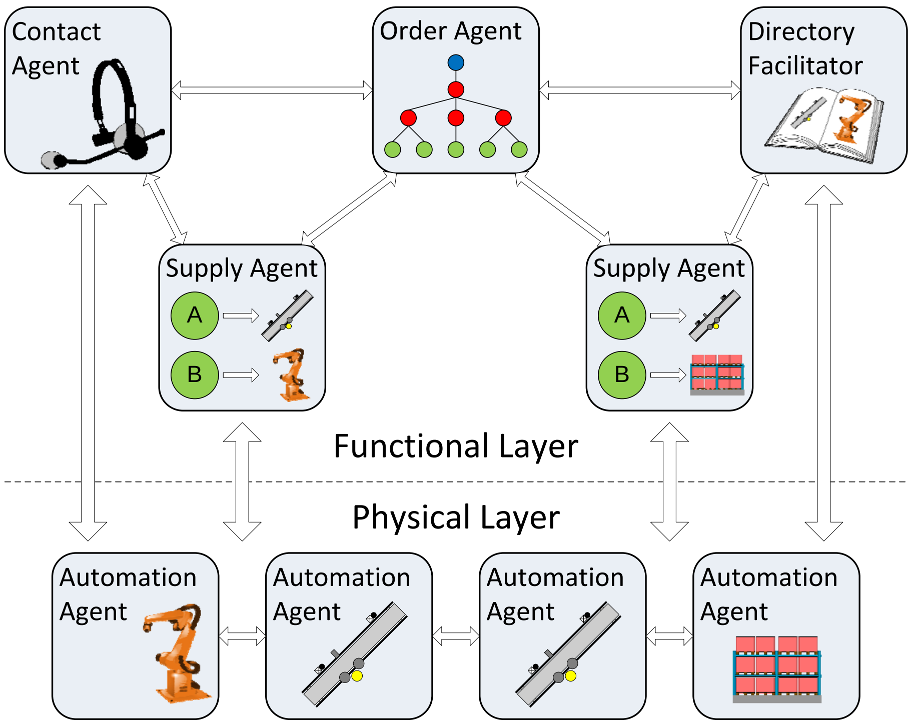

Figure 3.1: Possible conjunction of the automation agents with the functional layer of KASA, based on [78].

3.2 Target System for Implementation

In order to facilitate an easier understanding, the presented concepts are explained on the basis of the system used for the exemplary implementation (Objective 5 in Section 1.4): a pallet transport system with components mainly by Festo Didactic [211] that consists of 45 conveyor belts with 32 intersections as well as 6 indexstations with grippers for holding pallets (see Figure 3.2). Each component of this transport system (i.e. intersection, indexstation and conveyor) is incorporated within one automation agent. This constitutes a distributed control system with interconnected components.

A set of 38 embedded controllers of the type CPX-CEC-C1 by Festo (described in Section 3.8.3) is employed to control the indexstations and intersections as well as the conveyors. The indexstations and intersections incorporate Radio-Frequency Identification (RFID) readers as subcomponents for identifying pallets. Each of the RFID readers is connected via a serial interface with a small embedded controller of the type Digi Connect ME [212]. In total, 118 controllers are employed in the pallet transport system.

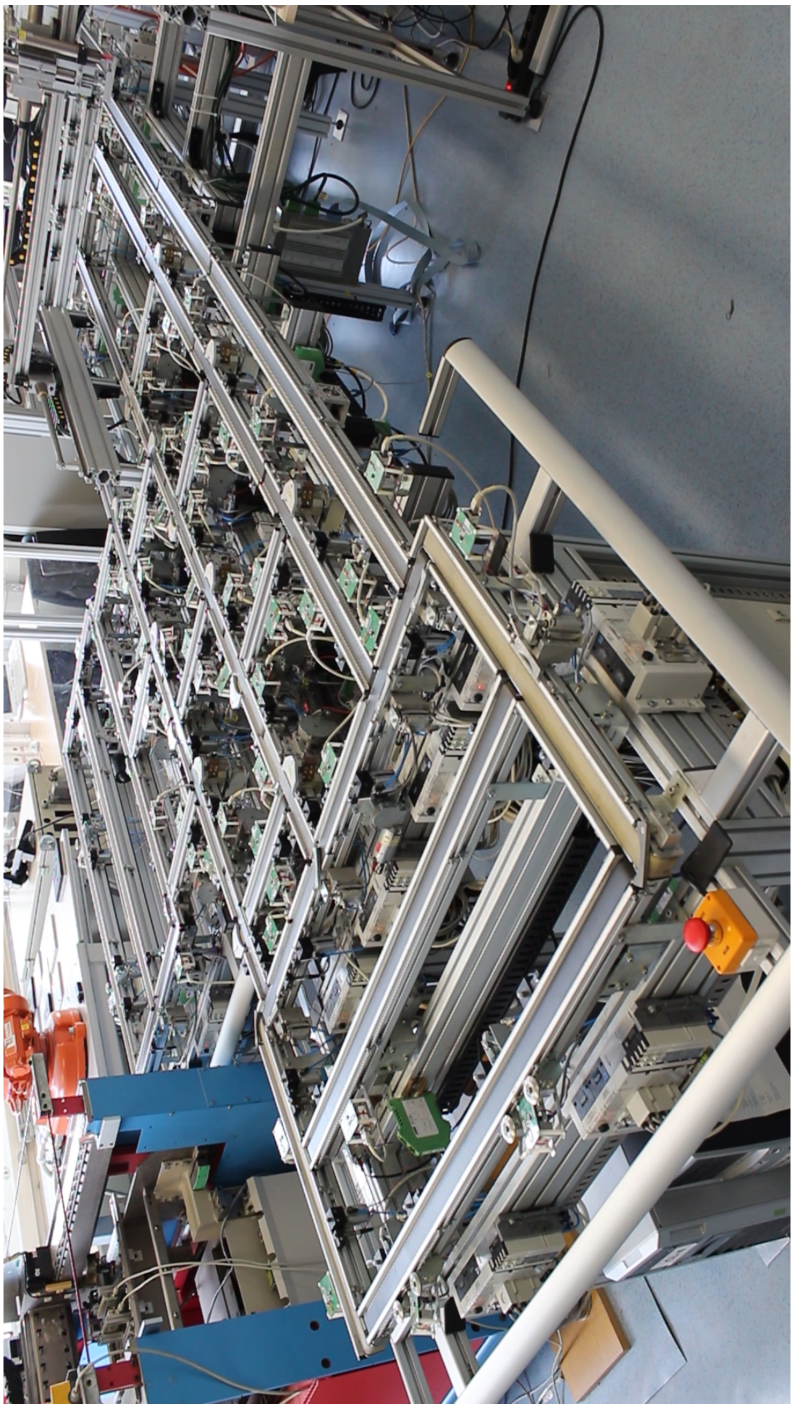

Figure 3.2: Picture of the pallet transport system used for the exemplary implementation of the agent-based control system. It consists of 45 conveyor belts with 32 intersections as well as 6 indexstations with grippers for holding pallets. 80 RFID readers are employed for identifying pallets. The complete system is controlled by 118 embedded control devices.

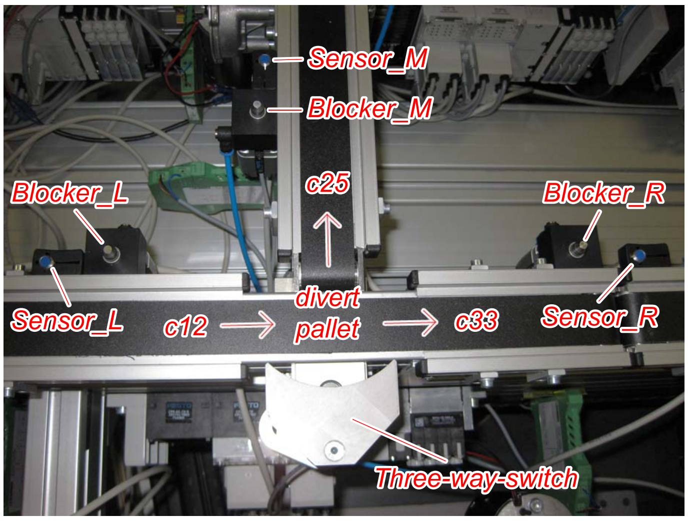

Figure 3.3: Example intersection `i1`, which acts as a diverter in the pallet transport system [73].

3.2.1 Intersection as Running Example

The details of the automation agent architecture are described using the example of an automation agent controlling an intersection in the pallet transport system. The example intersection is named `i1` and is located at the crossing point of two conveyors (see Figure 3.3). Within the agent, this crossing of the two physical conveyors is handled as a crossing of three logical conveyors denoted as `c12`, `c25` and `c33`. While the conveyors are controlled by their own agents, the automation agent responsible for the intersection has control over a set of subcomponents: three sensors for detecting pallets, three blockers for stopping them, and one switch for routing them. Moreover, RFID readers are employed on each adjacent conveyor for identifying the pallets' Identifier (ID) and designated destination. Depending on the direction of conveyor `c25`, the intersection either works as a junction with two incoming and one outgoing logical conveyor or as a diverter with one incoming and two outgoing logical conveyors. As the physical conveyor composed of `c12` and `c33` can be operated in two directions, the intersection incorporates in total four operational modes. The required routing flexibility is provided by the switch, which can be positioned in three positions and is therefore referred to as a three-way-switch. For the running example in this chapter, the intersection operates as a diverter with `c12` as its incoming conveyor. In the remainder of this work, the term "conveyor" refers to a logical conveyor if not otherwise noted, as only the logical conveyors are of actual importance for the operational principles of the automation agents.

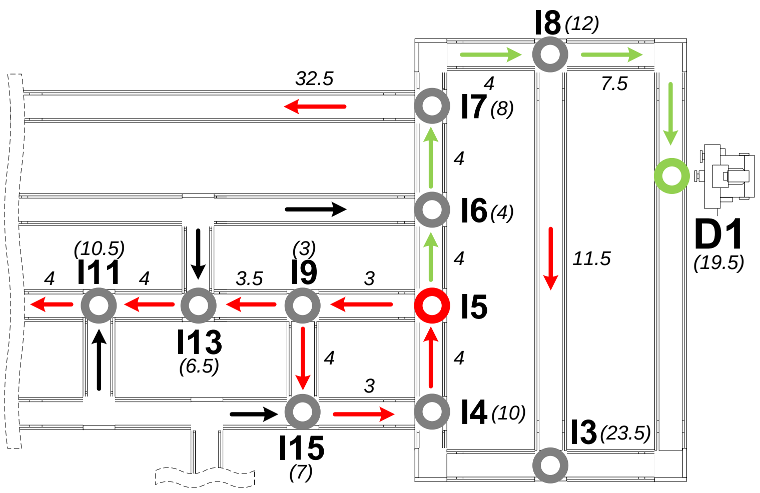

Figure 3.4: Dijkstra's algorithm is used to determine the shortest path from intersection `i5` to destination `D1`. The small number next to each conveyor represents that conveyor's cost (i.e. time required for passing this conveyor). The small number in brackets next to each intersection's name indicates the cumulative costs to reach that intersection from intersection `i5`. The calculated shortest path in this example is composed of the conveyors marked with green arrows.

3.2.2 Routing in the Transport System

Routing represents a core functionality of a transport system. A routing algorithm is used by a contact agent for calculating routing tables during system startup. It is based on Dijkstra's algorithm [213] and creates a tree of paths from each distinct location (i.e. intersections) to all designated destinations (i.e. indexstations) in the system. Dijkstra's algorithm is a simple but effective algorithm for finding shortest paths. This is exemplified by finding the shortest path from intersection `i5` to destination `D1` (see Figure 3.4). Starting from intersection `i5`, the algorithm determines at first the costs to the nearest locations, which are intersection `i9` at the cost of 3 and intersection `i6` at the cost of 4. The algorithm then continues its search along the "cheapest" location, which is currently `i9`. After finding `i13` at a total cost of 6.5 and `i15` at a cost of 7, the next "cheapest" location for continuing the search is `i6`. While the some paths will end up too expensive or lead back to the origin (marked with red arrows in Figure 3.4), eventually one path will end at the designated destination `D1` (marked with green arrows) if such a solution exists. This path is the shortest path according to the calculated costs. Consequently, `i5` routes pallets straight to `i6` if they need to go to `D1`, which is saved in the local routing table of `i5`. Such calculations are carried out for every combination of intersection and destination to determine the routing tables of all intersections.

If all conveyors and intersections are operational, this system layout allows the algorithm to always find solutions from every intersection to all destinations, but of course a different layout or broken components can prevent that.

More details regarding the implementation of the routing algorithm, which was developed in the frame of a previous research project, are presented in [214].

3.3 Automation Agent Architecture

An architecture for the automation agents needs to be chosen in regard to the requirements such an agent has to fulfill when controlling a physical component.

3.3.1 Architectural Requirements

The requirements regarding the architecture of the automation agents controlling physical components are identified as follows:

- Safe operation [82]: The agent needs to ensure a proper working state of its controlled component and should not execute commands that would endanger the proper operation of the component or the system.

- Response time [55]: When interacting with the process, i.e. acting on the hardware-near level, response times between 10 μs and 100 ms are required. On the contrary, interactions with human operators may be performed in response times between 100 ms and 10 s.

- Diagnosis [98, 215]: Sensor information can be used for directly detecting local failures in the mechatronic component. In regard of the previous two requirements, such failures should be detected in a short time frame to quickly react appropriately. “Bigger scale” failures or undesired conditions might only be detectable by combining information from different sources, i.e. other components of the system.

- Coordination [55, 99, 179]: For achieving aims, which are beyond the individual capabilities of an individual agent, and for ensuring the proper functionality of the component in the manufacturing environment, the agents need to coordinate their activities. On the one hand, communication means are required to ensure that automation agents are able to communicate with the agents on the superjacent functional layer. On the other hand, real-time constraints might need to be met regarding specific process-relevant information flows, as it is for instance in the case of a closed-loop control spanning over two or more automation agents.

3.3.2 Type of Architecture

Section 2.1.1 introduces the general types of agent architectures. The presented architectures are analyzed in regard to the requirements mentioned in the previous section:

- Reactive architecture: The reactive architecture answers well to the requirement regarding response time. The lack of complex representations allows an entity based on a reactive architecture to respond quickly to changes in the environment [90]. Thus, this architecture is suitable for reacting in a short time frame on detected local failures and for ensuring a safe operation, if according mechanisms are implemented, e.g. using standard PLC control code. However, this architecture is not designed for merging information from multiple sources due to its implicitly implemented information structures. Despite the ability for real-time communication between entities, the common message sizes between 1 bit and 100 bytes, which result from the applied software technologies and their data types, are not sufficient for communication with agents on the functional layer [55].

- Deliberate architecture: A deliberate architecture encompasses an explicit representation of the surrounding environment. Such a knowledge base is suitable for merging information from multiple sources but can lead to higher computational costs due to its complex structure [90]. A longer response time on occurring events is therefore the consequence. Nevertheless, such a representation structure is easier comprehensible for human operators when interacting with the system. The common means and protocols for communication with message sizes between 100 bytes and 4 kilobytes [55] are sufficient for communicating with the other agents in the system.

- Hybrid architecture: Combining the reactive with the deliberative approach delivers a layered, hybrid architectures [216, 217]. This allows a

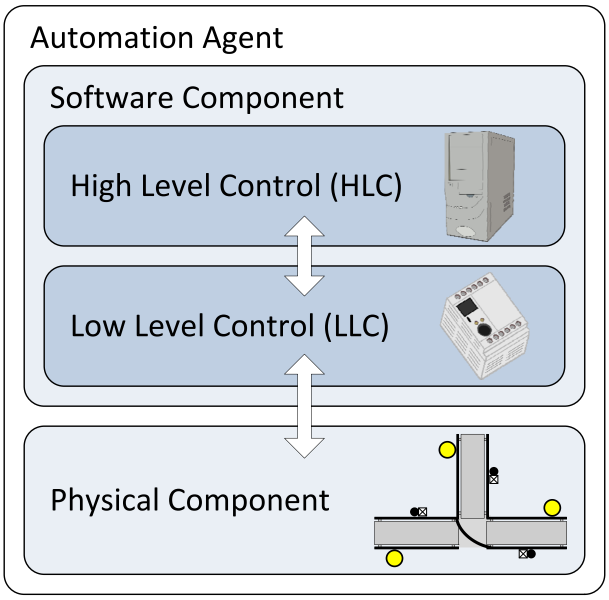

Figure 3.5: Architecture of an automation agent composed of the physical component as well as of a software component, which is further separated into HLC and LLC, based on [73].

separation of concerns as different requirements and scopes of consideration can be taken into account. Evidently, an interface is required between the two layers to ensure information exchange within the agent.

Consequently, a hybrid architectures is chosen for the automation agents to meet the identified requirements. In this context, the deliberate layer is represented by the High Level Control (HLC) while the Low Level Control (LLC) acts as the reactive layer. These two control levels are organized with the HLC being superjacent to the LLC (see Figure 3.5) [55]. The HLC layer is responsible for the control in regard to both the achievement of its own goals and the coordination with other entities of the system for achieving global goals. The LLC layer is responsible for directly controlling the mechatronic component using a set of reactive behaviors. Thereby it is linked to the sensors and actuators for supervising their actions and informs the HLC about the actual state of the component.

In the following sections, both control layers of the automation agent architecture are described in more detail. Besides, the interface for the communication between the two layers is introduced and the diagnostic capabilities of the automation agents are presented. Moreover, a reconfiguration infrastructure is introduced, which serves as the basis for the configuration and reconfiguration abilities of the automation agent.

3.4 High Level Control

The HLC is responsible for controlling a range of behaviors of the automation agent using information from its subjacent LLC as well as from other agents. Its decisions are influenced by the present and past observed states in the manufacturing environment. Figure 3.6 depicts the inner architecture of an automation agent's control software. The HLC is composed of four modules:

- The world model repository contains a *World Model*, i.e. a symbolic representation of the world of the agent, which includes its inner states as well as the surrounding environment. The world model constitutes the declarative knowledge of an automation agent and can be queried for reasoning about the actual states of the world before initiating actions. Generally, the world model is updated after recognizing changed world conditions or after performing actions. In regard of the running example, the world model of the agent controlling the intersection ranges just to its neighboring components apart from knowledge about its parts and inner states (see Section 3.4.1).

- The decision-making component is closely connected to the world model repository for reasoning about the states of the environment. Event notifications generated by the LLC, by communicating with other agents or by the world model trigger the decision-making procedures. These procedures then update the world model, request operations from the LLC, and communicate with other agents or issue a notification to a human operator (see Section 3.4.2).

- The communication manager provides facilities for managing the communication with other agents. In the presented work, it is provided by the agent framework JADE.

- The generic interface enables the HLC to communicate with the LLC. It provides facilities for receiving event notifications about the current operations of the LLC and for requesting the execution of particular operations from the LLC. An approach for the generic interface, which enables the HLC to communicate with various types of LLCs, is described in Section 3.6. However, this generic interface is yet on a conceptual level and consequently the final interface implementation used in the presented work only encompasses the message-based communication with LLCs of the standard IEC 61499.

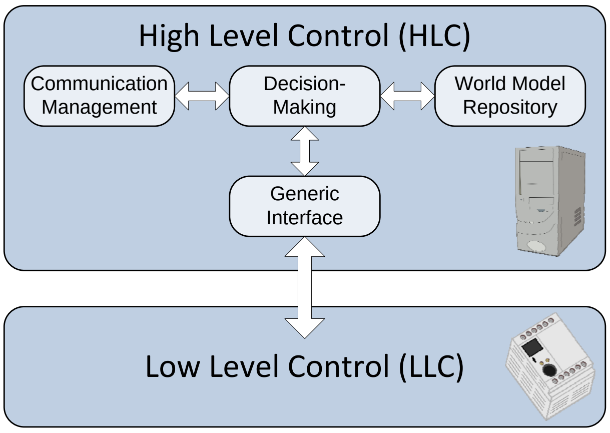

Figure 3.6: Inner architecture of an automation agent's control software with the HLC being separated into the modules: communication management, decision-making, world model repository, and generic interface [73].

3.4.1 World Model

The world model repository consists of two parts: the situation model and the activity model. The situation model holds knowledge about the automation agent's situation, which consists both of the agent's own characteristics and its relations to other entities in the system. The activity model holds knowledge about the activities of the agent, which provide an abstract representation for the actions, tasks and goals of the agents, either being observed in the system or expected to take place. Figure 3.7 illustrates the world model of the automation agent controlling intersection `i1`, which acts as a diverter. The situation model is composed of an ontology and a set of facts:

- The ontology contains knowledge about the agent's own characteristics and its relations to other entities in the system. This encompasses knowledge about the automation agent's environment and defines the relevant classes of the entities as well as the relations between them. It also serves as a vocabulary for referencing these classes and relations, thus ensuring the interoperability between different agents. In the case of an intersection, the ontology defines for example its relation to the adjacent components, which are the conveyors that can be the intersection's inputs or outputs. Such concepts and relations can be extracted from existing ontologies, such as [214].

- The facts express the knowledge about the current state of the world using the vocabulary defined by the ontology. In the case of the example intersection, the facts express that i1 is an intersection, which

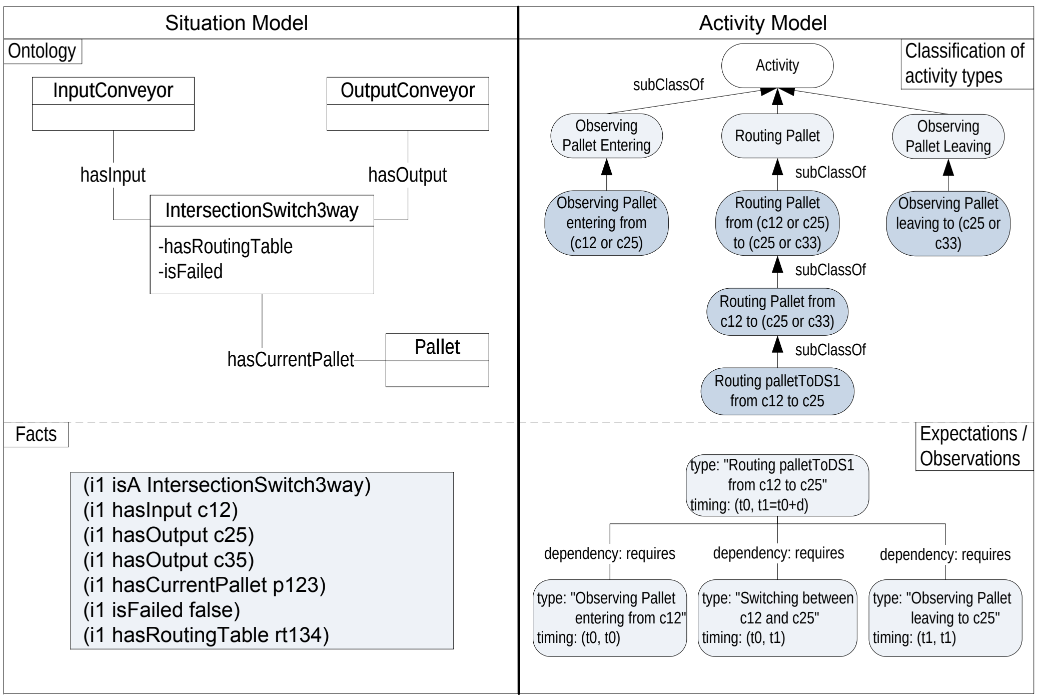
Situation Model
Ontology
InputConveyor
OutputConveyor
hasInput
hasOutput
IntersectionSwitch3way
-hasRoutingTable
-isFailed
hasCurrentPallet
Pallet
Facts
(i1 isA IntersectionSwitch3way)
(i1 hasInput c12)
(i1 hasOutput c25)
(i1 hasOutput c35)
(i1 hasCurrentPallet p123)
(i1 isFailed false)
(i1 hasRoutingTable rt134)
Activity Model
Classification of activity types
Activity
subClassOf
Observing Pallet Entering
Routing Pallet
Observing Pallet Leaving
Observing Pallet entering from (c12 or c25)
Routing Pallet from (c12 or c25) to (c25 or c33)
Observing Pallet leaving to (c25 or c33)
Routing Pallet from c12 to (c25 or c33)
Routing palletToDS1 from c12 to c25
Expectations / Observations
type: "Routing palletToDS1 from c12 to c25" timing: (t0, t1=t0+d)
dependency: requires
type: "Observing Pallet entering from c12" timing: (t0, t0)
type: "Switching between c12 and c25" timing: (t0, t1)
type: "Observing Pallet leaving to c25" timing: (t1, t1)

Figure 3.7: World model of an intersection agent acting as a diverter, which is separated into the situation model and the activity model [73].

has one input conveyor (`c12`) and two output conveyors (`c25` and `c33`). It is important to note that facts expressed in the situation model do not intend to represent the complete world of the agent. The facts are created during the instantiation process of an agent and represent the knowledge the agent uses during its high level control tasks. Updates of the facts can take place upon completion of control tasks. The knowledge represented in the facts can also be updated with information obtained from the agent's LLC or other entities of the system.

The activity model is composed of a classification of activity types and a model of expectations and observations:

- The classification of activity types incorporates the types of activities the agent can be involved. These types are organized hierarchically based on the subsumption relationship [218]. The primitive types are defined as direct subclasses of "`Activity`", while the derived types are defined by restrictions, which take the actual world of the agent into account. For instance, the generic type "`Routing Pallet`" is refined to the more specific type "`Routing Pallet from (c12 or c25) to (c25 or c33)`" and furthermore to the type "`Routing Pallet from` c12 to (c25 or c33)", corresponding to the considered intersection acting as a diverter. The further refined type "Routing palletToDS1 from c12 to c25" denotes the precise action of routing a particular pallet with a particular destination DS1 from a particular input conveyor c12 to a particular output conveyor c25.

- The expectations and observations form a model of the activities that are expected and observed by the agent. Expectations and observations are defined by the specification of a type (based on the classification of activity types) and timing, which is expressed using time intervals [219]. While observations can express a precise timing, the expectations rather express constraints on their timing. Expectations are linked by dependencies, indicating how observations on one expectation can have consequences on other expectations. Figure 3.7 depicts that for instance the expectation "Routing palletToDS1 from c12 to c25" should occur starting at time t0 and ending at time t1=t0+d, with d being the time for the pallet to pass through the intersection. This implies that the activity "Observing palletToDS1 from c12" should occur at t0, the activity "Switching between c12 and c25" should occur between t0 and t1, and the activity "Observing palletToDS1 to c25" should occur at t1.

The world model enables the HLC to maintain a representation of relevant aspects of the automation agent and its surrounding environment. It allows the automation agent to reason about the fulfillment of expectations or inconsistencies between the representation and the information it receives from its environment by sensors or other agents.

3.4.2 Decision-Making Component

The agent relies on procedural as well as declarative knowledge (see Section 2.1.4) for performing decisions. While the declarative knowledge is provided in the world model, the procedural knowledge is incorporated in the decision-making component. It consists of two parts:

- Agent behaviors: The behavior of an automation agent is constituted by a set of rules. Each rule comprises a condition and an action, which is executed if the condition is satisfied. In contrast to purely reactive behaviors, some rules are designed to also involve knowledge from the world model. For instance the generic rule for handling expectations is designed for the comparison of observations from the activity model with the according expectations. In case the comparison delivers a

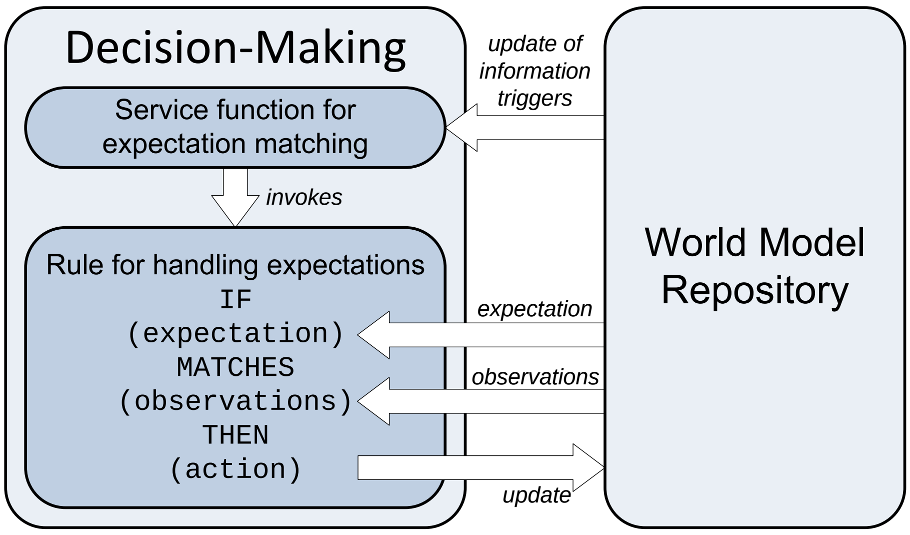

Figure 3.8: Example of decision making: An update of information in the world model triggers the service function for expectation matching. This function invokes the rule for handling expectations multiple times comparing each time an expectation with the observations made.

positive result, the action of the rule is executed, which can encompass an update of facts and observations as well as notifying the agent's LLC or another agent.

- Service functions: The service functions support the agent in being able to act. For instance one service function is responsible for adding the information of status updates from the LLC to the facts in the world model. Another service function for expectation matching is responsible for invoking the rule for handling expectations upon new or updated information in the world model. By invoking this rule for each expectation existent in the activity model, the fulfillment of an expectation based on the provided information can be recognized and the according action of updating the world model is triggered (see Figure 3.8).

Rules and service functions of the automation agents are realized as Java methods in the presented work.

3.5 Low Level Control

The LLC is reponsible for directly controlling the mechatronic component by employing a set of reactive behaviors, which are constituted using an industrial standard for being executed in an industrial controller. The investigation in Section 2.2 shows that the execution semantics of the widely applied standard IEC 61131 do not answer well to the requirements for distributed, reconfigurable manufacturing systems. The system's flexibility is limited due to the centralized nature of IEC 61131 and difficulty to manage changes dynamically [32]. Moreover, the cyclically scan-based execution nature of the PLC programs is sensitive to the order in which functional elements are placed in the program and specific complex synchronization problems in distributed environments could occur. Especially timing software switches as well as synchronizing internal states during the reconfiguration process is complicated [220]. Furthermore, tools are missing for dynamically reconfiguring individual parts of applications such as a single FB [158]. On the contrary, the standard IEC 61499 incorporates an event-based execution model thereby improving its suitability for dynamic reconfiguration. Besides, IEC 61499 runtime implementations exist that provide the basic mechanisms for dynamic reconfiguration.

The LLC, realized with a network of IEC 61499 FBs, provides the basic functionality of a component and possesses access to the component's sensors and actuators. Following a component-oriented design, a typical LLC application for a component contains only a small number of FBs. In the case of a diverter, the LLC encompasses eleven FBs. Figure 3.9 shows the FB network of the previously mentioned example diverter, which is connected to the incoming conveyor on its left side oriented from its three-way-switch. Its basic functionality is to route an incoming pallet onto one of the two outgoing conveyors and to block other pallets in the meanwhile.

To allow an easy reuse, the control software for each type of subcomponent is encapsulated in a specific composite FB type. All these FB types offer access to the corresponding controller I/Os that are connected with the distinct physical subcomponents. In the case of blockers and switches, this includes also the sensors for verifying the subcomponent's position after an induced actuator movement. Ten such FBs are responsible for controlling the subcomponents of an intersection acting as a diverter:

- Three FBs `FB_SensorDevice` handle the functionality of the three pallet sensors for detecting the arrival of pallets at the incoming conveyor as well as verifying that a pallet leaves the intersection on the designated outgoing conveyor.

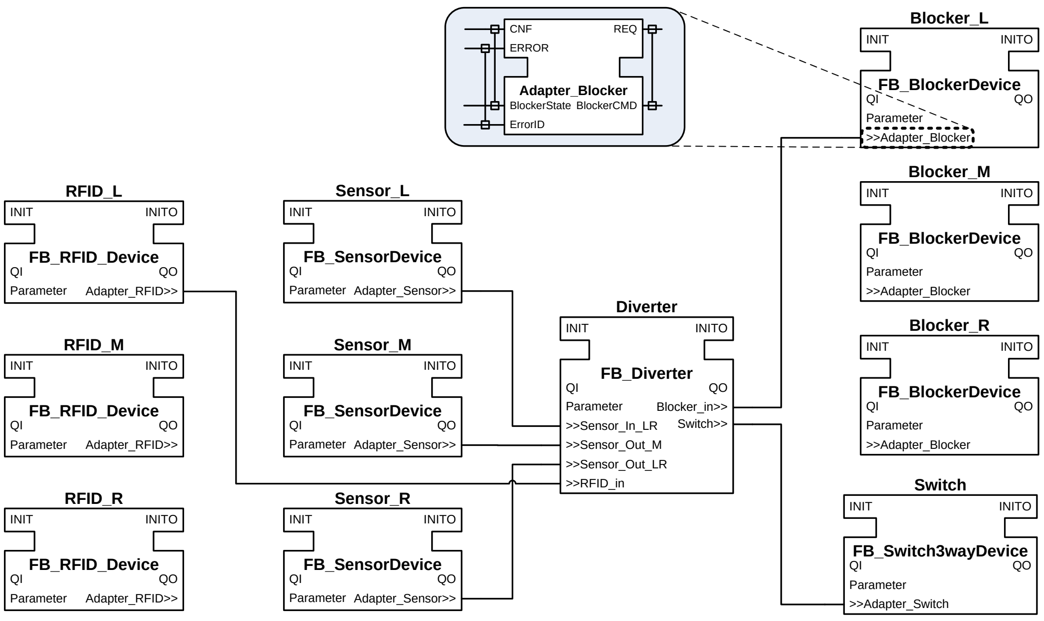

Figure 3.9: Low level control of a diverter realized with a network of IEC 61499 FBs, based on [73]. The central FB `FB_Diverter` is used for providing functionalities such as pallet routing while the other FBs control the subcomponents. Adapter connections encapsulate the event and data connections between the FBs. As can be seen, not all FBs are connected as this depends on the actual configuration of the intersection. For instance, the RFID readers at the outgoing conveyors are not used, therefore they are not connected with the central FB.

- Three FBs `FB_BlockerDevice` are reponsible for the three blockers at the adjacent conveyors for stopping pallets in the case a pallet is already located within the intersection. FBs of this type also access the sensors for verifying the blocker positions after induced actuator movements.
- `FB_Switch3wayDevice` controls the three-way-switch and is substituted by the FB `FB_SwitchStandardDevice` in the case of intersections incorporating another type of switch with only two positions. Likewise to `FB_BlockerDevice`, also this FB accesses position sensors for observing the state of the switch after an induced actuator movement.
- Three FBs `FB_RFID_Device` are used for receiving data from the three RFID readers at the adjacent conveyors. As each of the RFID readers is in fact controlled separately by one small embedded controller, `FB_RFID_Device` contains an FB for realizing an internal LLC-LLC

communication between the main controller of the intersection and the controller of the RFID reader.

In contrast to the FBs that control the subcomponents, functionalities such as the communication with the HLC and the routing of pallets are provided by the FB `FB_Diverter`. A local routing table, which is received from a contact agent via the automation agent's HLC, is stored within an FB inside `FB_Diverter` and used to determine the outgoing conveyor for a passing pallet. For each designated destination in the system (i.e. the indexstations), a number is stored that represents the outgoing conveyor to which the pallet has to be routed. A "0" represents the conveyor on the left when the intersection is viewed from above like shown in Figure 3.3. A "1" indicates the middle conveyor and a "2" represents the conveyor on the right side. The main operational sequence of a component's functionality is realized using the concept of finite state machines, as it represents a simple and elegant solution suitable for control software, which has to deal with timings and decisions [221]. Moreover, a state machine is straightforward to implement in the execution control chart of a basic FB, making this concept therefore easy to adopt for the LLC sequences. Two finite state machines within two distinct basic FBs are applied inside of `FB_Diverter` for handling on the one hand approaching pallets on the input conveyor and on the other hand leaving pallets on the output conveyors. In the case of a junction, two distinct state machines are responsible for handling the approaching pallets on the two input conveyors and a third state machine handles the leaving pallets on the output conveyor. State changes trigger an update message to the HLC, which can then use this status update for diagnostic tasks.

For carrying out its algorithms, `FB_Diverter` requires access to the pallet sensor, the blocker device and the RFID device that are located at the incoming conveyor, as well as to the pallet sensors that are located at the two outgoing conveyors. By using the adapter concept (see Section 2.2.2) for the connections between the FBs, only six connections between them are necessary. In the case of a junction, seven connections to the subcomponent FBs are required as a junction utilizes the RFID readers on two conveyors instead of only one. Connections to the FBs controlling the remaining subcomponents are not required. As adapters represent self-contained interfaces [222], the application of this concept leads to a very decoupled application design [223]. By clearly separating the two connected FBs, the employment of adapters facilitates the exchange of one or both of these FBs [224].

Apart from the reconfiguration aptitude of the LLC, its memory usage can be an important criterion regarding small embedded controllers that have limited memory capacities. An evaluation of different IEC 61499 implementation paradigms concerning their object code size is presented in [225]. A generated monolithic C program is identified as the most efficient paradigm concerning code size, while any flexible runtime environment shows a larger consumption of memory. However, compared to the pure compiled C code, only a flexible runtime environment can provide the capability of dynamic reconfiguration. Nevertheless, reducing the amount of consumed memory is regarded of value concerning embedded controllers. In this context, the component-oriented design of the LLC is evaluated in comparison with a less decoupled application design not based on adapter connections between the FBs, which has previously been developed by the author and used in an approach for controlling the physical components such as the example diverter. A control module for a CPX valve terminal of the type CPX-CEC-C1 by FESTO [226] is used as target system. It hosts an Xscale-PXA255 agile Intel microprocessor with 400 MHz, 28 MB Flash and 24 MB RAM. A Linux operating system is employed on this type of controller and the FORTE [227] is applied as the IEC 61499 runtime environment for executing the FB network. While the previously employed design consumes 1032 kB of memory, the component-oriented design for the LLC requires only 392 kB of memory usage, which represents a significant reduction of the memory usage in the controller by 62%. In the given case, with several MB of memory available on the FESTO controllers, the reduction of memory usage might not seem of importance. However, further software components could be deployed on such a controller. For instance, the integration of a framework for running the HLC directly on the controller poses a severe challenge in contrast to hosting such a framework on a regular PC due to the available memory [228]. Further challenges regarding memory and its usage exist in regard of wireless sensor networks [229] or applications in the domain of Internet of Things. Even though devices with only very basic computing capabilities are existent in these domains, they have to provide various functionalities such as communication services [230]. Consequently, the limited resources of these control devices needs to be taken into account regarding the employed software [231].

3.6 Communication Interface between the Control Layers

As an automation agent encompasses two control layers, which are furthermore based on different technologies, an interface is required to link these layers. IEC 61499 is chosen as technology for the hardware-near control layer of the approach presented in this work. Nevertheless, PLCs based on

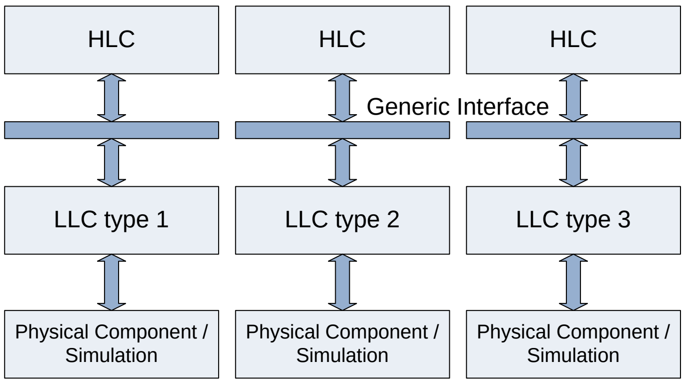
HLC
HLC
HLC
Generic Interface
LLC type 1
LLC type 2
LLC type 3
Physical Component / Simulation
Physical Component / Simulation
Physical Component / Simulation

Figure 3.10: Concept for integrating automation agents with different LLC types [74].

IEC 61131 remain common practice in the industry. To allow the application of automation agents also on control systems based on this standard, the concept of an interface between the control layers is presented, which represents a possible migration path for involving different types of LLCs. This concept has been developed in collaboration with industrial partners.

Figure 3.10 shows the concept of automation agents with different types of LLCs connectable to their HLC. In order to achieve this architecture, an interface is required that supports the communication between the HLC and different LLC types. In this context, the notion of a generic communication interface is used because vital parts of this interface shall follow the principle of generality in order to easily extend the range of supported LLC types. The lower level elements of such an interface that are in direct contact with the LLC need to be customized corresponding to the supported type likewise to a software driver. The interface is therefore required to connect the two layers, HLC and LLC, taking into account the different software paradigms that are applied on these layers. As this interface shall be usable for integrating various types of LLCs, corresponding extensions shall be achievable without extensive efforts.

Such an approach is not only applicable for real systems but also for simulation, as this is an effective way for testing scenarios and improving the quality of the solutions. Besides, the number of implemented multi-agent control systems is yet small and, therefore, simulation represents a valuable method for demonstrating their potential [232].

Commands and requests need to be sent from the HLC to the LLC, and reports and failure notifications from the LLC to the HLC. Hence, the purpose of the generic HLC-LLC interface is to make the functionalities from the LLC accessible to the HLC in a uniform way. This raises challenges as follows:

- The different types of LLC use different communication protocols (e.g. data table tags, channels), which have distinct properties (synchronous vs. asynchronous interaction, messages vs. tags, event-driven vs. cycle-based, etc.). Hence, a generic interface needs to support communication protocols with different properties.

- The different LLC types use distinct definitions of commands. Therefore, physical components of the same type are addressed differently depending on the way a command is defined in the applied LLC type.

To address these challenges, the approach for the generic interface incorporates:

- A high level language for expressing semantic descriptions of commands to the LLC, which enables the HLC to express commands using the essential concepts of the domain without having to know precisely how the invocation of the LLC is performed.

- An architecture for interpreting the commands expressed using the high level language and translating it into low level commands that are specific for each type of LLC, interacting with various kinds of LLC types in an asynchronous, event-driven way.

Figure 3.11 illustrates the principles of the generic interface. An example of commands for controlling a blocker is presented as it shows clearly the differences of LLCs based on IEC 61131 and on IEC 61499. In the following, a blocker named `blocker_01` operating at the place `P1` controlled by an IEC 61131 LLC, and a blocker named `blocker_02` operating at place `P2` controlled by an IEC 61499 LLC are considered for this example.

The upper part of Figure 3.11 illustrates a command sent by the HLC to the LLC, which is expressed using the high level description language. The lower part illustrates how the command is finally transmitted to the LLC, i.e. by writing a tag named “bl_01” with the value “0” in the case of an IEC 61131 LLC, or by sending a message with the content “0” to the channel “239.191.0.14:61100” in the case of an IEC 61499 LLC. The center of the figure illustrates the generic interface, which is composed of two parts: a generic interpreter and a set of specific adapters. The generic interpreter receives the command from the HLC and transfers it to a relevant adapter. This adapter then translates the command into a form specific for the applied type of LLC. The lower part of Figure 3.11 shows the involvement of

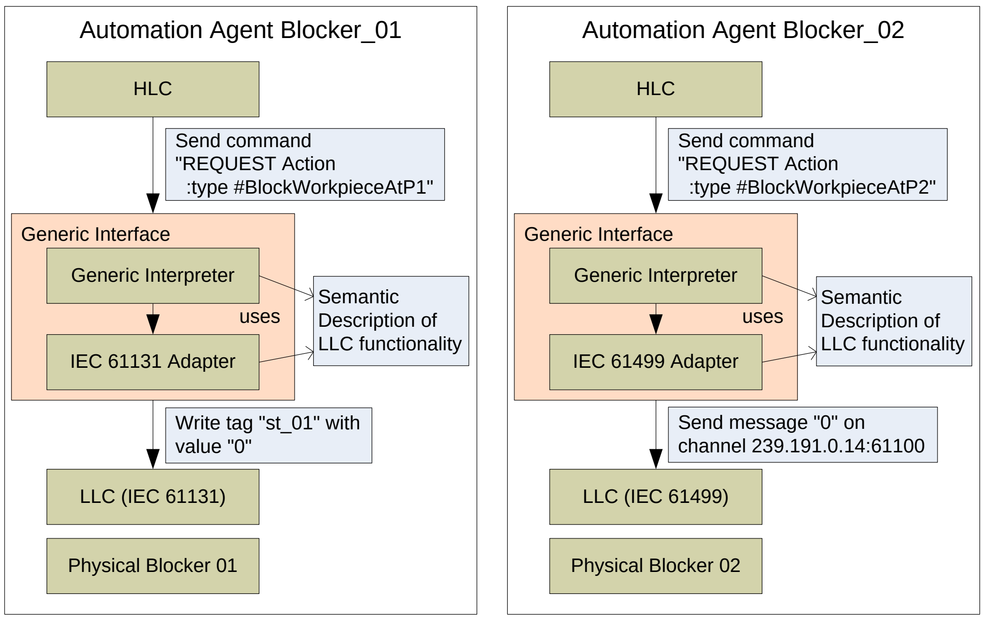
Automation Agent Blocker_01
HLC
Send command "REQUEST Action :type #BlockWorkpieceAtP1"
Generic Interface
Generic Interpreter
uses
IEC 61131 Adapter
Semantic Description of LLC functionality
Write tag "st_01" with value "0"
LLC (IEC 61131)
Physical Blocker 01
Automation Agent Blocker_02
HLC
Send command "REQUEST Action :type #BlockWorkpieceAtP2"
Generic Interface
Generic Interpreter
uses
IEC 61499 Adapter
Semantic Description of LLC functionality
Send message "0" on channel 239.191.0.14:61100
LLC (IEC 61499)
Physical Blocker 02

Figure 3.11: Principles of the generic HLC-LLC interface for supporting LLCs based on IEC 61131 and IEC 61499 [74].

a specific adapter for LLCs realized with IEC 61131 (tag-based) and a specific adapter for those realized with IEC 61499 (channel-based). Generally such adapters need to be developed specifically for each type of distinct LLC, respectively the LLCs host controller device (e.g. a specific PLC). Developing such adapters, often denoted also as drivers, represents a core activity of companies that link their software (e.g. SCADA software) with various controller devices (e.g. [233]).

The semantic description of the LLC functionality plays an essential role in the generic interface. It provides the necessary information for translating the high level command into a low level command. Listing 3.1 illustrates a description of the functionality of `blocker_01` for the standard IEC 61131. Listing 3.2 gives a similar illustration for the functionality of `blocker_02` for the standard IEC 61499. In this description, the functionality provided by a blocker is composed of two services. The first one, named "activate" enables the HLC to activate the blocker, thus blocking all incoming pallets. The second one, named "deactivate" enables the HLC to deactivate the blocker, thus releasing any incoming pallet. Each service is described using three elements: `id` gives an identifier for the service, `type` defines the type of service and `grounding` defines how to invoke the LLC for providing the service.

```
Service
  id = activate
  type = #BlockWorkpieceAtP1
  grounding = LogixTagGrounding(write, st_01, 0)

Service
  id = deactivate
  type = #ReleaseWorkpieceAtP1
  grounding = LogixTagGrounding(write, st_01, 1)
```

Listing 3.1: Description of `blocker_01` functionality for an LLC based on IEC 61131 [191].

```
Service
  id = activate
  type = #BlockWorkpieceAtP2
  grounding = ChannelTagGrounding(send,
     239.191.0.14:61100, 0)

Service
  id = deactivate
  type = #ReleaseWorkpieceAtP2
  grounding = ChannelTagGrounding(send,
     239.191.0.14:61100, 1)
```

Listing 3.2: Description of `blocker_02` functionality for an LLC based on IEC 61499 [191].

Using the type description, the generic interpreter matches the command requested by the HLC with the actual services provided by the LLC functionality. Based on the component description, it decides which adapter to use for sending the command to the LLC. In the case of a request to activate the `blocker_01`, it uses the "activate" service with a LogixTagGrounding in order to use the IEC 61131 adapter. In the case of `blocker_02`, the IEC 61499 adapter with ChannelGrounding has to be used. When an adapter receives the command expressed in the high level language, it performs two steps:

1. The adapter translates the command into a direct LLC command. To do so, it uses the information contained in the description of the functionality. In the given example, the translation is direct, as the command for activating the blocker does not require any parameter. In more complex cases, also parameters of a command might need to be handled, which requires according interpretation mechanisms to construct the correct LLC command.

2. The adapter sends the command to the LLC, using the relevant communication protocol. For blocking a pallet with `blocker_01`, this means that the adapter writes the value “0” into the tag named “bl_01”. In the case of `blocker_02`, the IEC 61499 adapter is used to send a message with the content “0” to the channel “239.191.0.14:61100”.

In addition to the simple case of commanding an action, illustrated by the example, the generic HLC-LLC interface also allows the HLC to subscribe and be notified of changes at the LLC level. In that case, the HLC sends a subscription command, which is translated by a relevant adapter in a similar manner. Whenever a relevant change happens at the LLC level, the adapter informs the HLC by sending a message expressed using the high level language.

Integrating a new type of LLC with the generic HLC-LLC interface is achieved in two steps:

1. Definition of the grounding language for the considered LLC.

2. Implementation of the adapter for accessing the LLC.

For an LLC based on IEC 61131, the grounding language enables the description of read and write operations of a particular tag value. Additionally, the notification of changes requires the adapter to monitor changes of tag values. This is done using existing subscribe-notify mechanisms provided by the tag-based interface as used for instance in MAST [166], which serves as a bridge between the adapter and an LLC based on IEC 61131.

For an LLC based on IEC 61499, the grounding language enables the expression of operations for sending and receiving messages with a given content on a given channel. For the notification of changes, the adapter only needs to listen for messages on the relevant channel, since IEC 61499 provides the appropriate notification mechanisms.

3.7 Using Automation Agents for Diagnostic Tasks

The occurrence of failures can reduce a system’s performance significantly. For avoiding such a decrease, failures have to be detected and properly treated if procurable. Different types of failures and anomalies can be distinguished [234]:

- Physical component failures involve the breakdown of a specific component as well as a temporary nonfunctional state (e.g. a pallet is stuck in an intersection).

- Software entity failures refer to a limited functionality of an agent or its complete inability for performing its tasks, which can be due to a breakdown of the controller or the PC hosting the software or due to a loss of communication.

- Anomalies encompass situations such as missing pallets or incorrect deliveries.

By employing diagnostic algorithms, the detection of failures and anomalies lies in the responsibility of the automation agents. Two forms of distributed diagnosis can be distinguished: semantically distributed diagnosis and spatially distributed diagnosis [235]. Semantically distributed diagnosis is performed by a heterogeneous group of agents with each having its own view of the system, modeling a different aspect. Spatially distributed diagnosis involves a group of agents, which jointly monitor and diagnose a distributed system by making use of each agent's detailed knowledge of a small part of the system in conjunction with the information exchanged between them.

A MAS can combine both types of diagnosis. Functional higher level agents model different functional aspects of the system and represent a heterogeneous group of agents. As such, they may rely on the semantically distributed diagnosis approach due to their heterogeneity [236]. On the contrary, the diagnosis concept of the automation agents is derived from the spatially distributed diagnosis approach. In such an approach each agent is responsible for a certain area of the system [237]. This is the case for automation agents, with each of them controlling a specific component of a spatially distributed system, such as an intersection in a pallet transport system. The automation agents are able to perform diagnostic tasks on both of their control layers by comparing their current state with the intended state. By recognizing differences between the current state and the intended one, failures can be detected. The diagnostic mechanisms of the LLC are purely related to failure detection within the mechatronic component using its sensor data. However, the HLC can obtain information also from other entities of the system to possibly determine both local failures and also those, which stretch across multiple components [238]. Moreover, by applying a so-called heartbeat mechanism also the liveliness of the controllers hosting the software parts can be monitored for detecting software entity failures.

3.7.1 Diagnostic Mechanisms at the Low Level Control

The FBs of the LLC for accessing the subcomponents (blockers, etc.) incorporate preprogrammed diagnostic algorithms for detecting invalid positions of subcomponents using information from the according sensors. Using timers is a common method in industrial practice for PLC programming for monitoring components in regard to error detection [239]. A timer is triggered as soon as a subcomponent is required to change its position. If the changed position is verified by the corresponding position sensor within the given time frame, the subcomponent's FB triggers a confirmation event to the central FB of the LLC (e.g. `FB_Diverter`). However, if the desired position is not confirmed within that time frame, an error event is issued to the central FB instead. Consequently, the LLC informs the HLC about this event which indicates a broken component. Depending on which subcomponent is apparently broken as well as on its position, the HLC can determine either a reduced or completely lost functionality of the component and informs its neighboring automation agents.

For example an error number, which is sent with the error event from `FB_Switch3wayDevice` to `FB_Diverter` and further on to the HLC, indicates the position of the switch based on the sensor data. Likewise to the numbers representing the conveyors in the routing tables (see Section 3.5), the "0" means that the three-way-switch is located on the left when the intersection is viewed from above like shown in Figure 3.3. A "1" indicates that the three-way-switch is located in its home position and a "2" represents the switch being on the right position. The number "3", which does not represent a position of the switch, is sent if the sensor data does not indicate a certain position. Based on the number, the HLC updates its world model, which is described in Section 3.7.2.

Apart from detecting an invalid position of a subcomponent, the LLC represents a necessary link for the diagnostic mechanisms of the HLC. In this context, it transmits status information to the HLC based on the gathered data from the sensors.

Moreover, predefined fault-recovery mechanisms can be incorporated in the LLC such as trying to free a stuck pallet by moving the switch. This task may be requested by the HLC in order to react on a detected failure.

3.7.2 Diagnostic Mechanisms at the High Level Control

In contrast to the LLC, which is able to detect only local failures, the HLC has the ability to also determine failures and anomalies, which encompass several components. Anomalies are detected by using the world model for finding differences between the expected and the observed activities. Status updates transmitted by the LLC are essential for carrying out the diagnostic

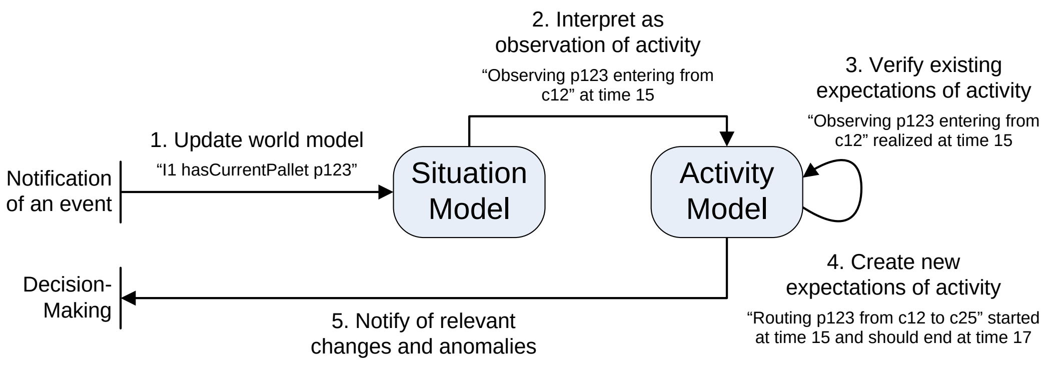
2. Interpret as observation of activity
"Observing p123 entering from c12" at time 15
3. Verify existing expectations of activity
"Observing p123 entering from c12" realized at time 15
1. Update world model
"I1 hasCurrentPallet p123"
Notification of an event
Situation Model
Activity Model
Decision-Making
4. Create new expectations of activity
"Routing p123 from c12 to c25" started at time 15 and should end at time 17
5. Notify of relevant changes and anomalies

Figure 3.12: Diagnostic sequence using the world model of the HLC for detecting changes and anomalies [148].

mechanisms at the HLC. The diagnostic sequence incorporates five steps (see Figure 3.12):

1. After the notification of an event, the world model is updated accordingly.

2. The current situation is analyzed regarding a potential observation of an activity.

3. In the case of a new observation, it is compared to existing expectations, in order to verify the fulfillment of one or several expectations.

4. Based on dependencies between the expectations, a new observation may also result in the creation of new expectations.

5. In the case of relevant changes or detected anomalies in the activity model, the decision-making component is notified for initiating appropriate actions.

The following example elaborates the usage of the diagnostic sequence on the basis of a pallet, which should leave an intersection.

A variety of causes can lead to the example case with the pallet not being detected by the output sensor. The situation is analyzed by using status updates from the LLC in conjunction with timing information in the world model. In the given example, the activity "`Routing p123 from c12 to c25`" is not completely observed as expected. Based on the available sensor information of the target system, one of the following conditions can be responsible for this anomaly (see Figure 3.13):

1. a switch failure occurs;

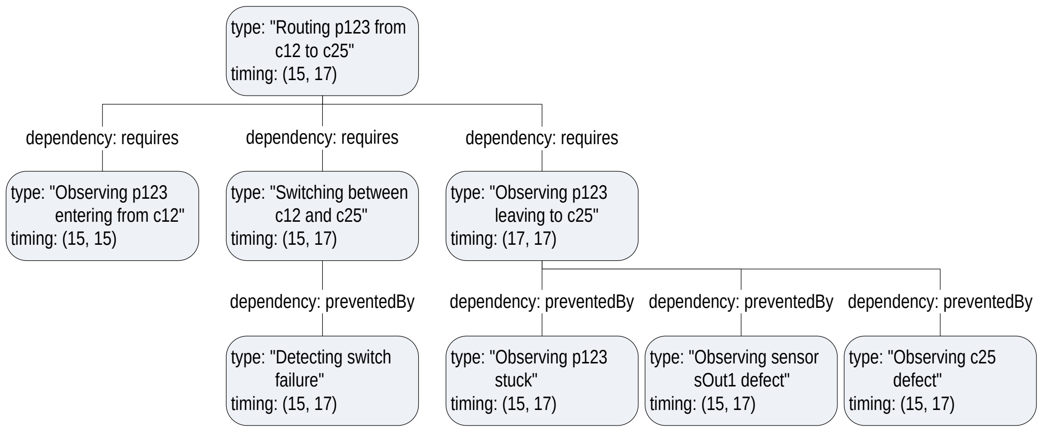

Figure 3.13: Expectations of possible activities responsible for the anomaly, based on [148].

2. the output sensor is defect;
3. the outgoing conveyor is defect; or
4. the pallet is stuck inside the intersection (without a switch failure).

A switch failure can be detected by the LLC using sensor information (see Section 3.7.1). In this case the HLC is notified by the LLC and updates its facts in the world model according to the error number as follows:

- Error number "0": The left conveyor `c12` is not accessible and the fact "`i1 hasInput c12`" is removed. As this is the only incoming conveyor, the intersection cannot be used in the current configuration.
- Error number "1": The middle conveyor `c25` is not accessible and the fact "`i1 hasOutput c25`" is removed. As a consequence, the according activity "`Routing palletToDS1 from c12 to c25`" is removed and the intersection can now only move pallets from `c12` straight on to `c33`.
- Error number "2": The right conveyor `c33` is not accessible and the fact "`i1 hasOutput c33`" is removed. Consequently, the according activity "`Routing palletToDS2 from c12 to c33`" is removed and the intersection can only act as a curve that can transfer pallets from `c12` to `c25`.
- Error number "3": The location of the switch is unclear and as a consequence, the fact "`i1 isFailed false`" is replaced with "`i1 isFailed true`".

In the given example, only errors “0”, “1” and “3” can occur because position “2” is the desired position. According to the dependencies, it can be concluded that the activity “`Routing p123 from c12 to c25`” cannot be performed completely if one of these errors occurs. Consequently, the HLC sends a message to the contact agent with the request to update the system representation by removing the connections, which are no longer passable. Error “3” means that this intersection has to be bypassed completely by its neighboring components.

In order to confirm a defect output sensor, the automation agent relies on information from another agent. Using the situation model, the corresponding following agent in the transport system can be identified and queried about the leaving pallet. In the case this agent detects the pallet after a given timeframe, it can be concluded that the output sensor is defect and that the pallet indeed has left the intersection.

A defect outgoing conveyor is indicated if the output sensor does not confirm the leaving pallet and the corresponding following agent does not confirm its arrival. According to the physical layout of the target system with the left and right conveyor of a switch being one physical component, this is only possible in the case of the middle conveyor. The consequences are likewise to error “1” of the switch failure. The fact “`i1 has output conveyor c25`” is removed from the world model and a message is sent to the contact agent requesting the removal of the according connection in the system representation. The switch is set to its home position and the pallet should leave the diverter on the outgoing conveyor `c33`.

If the first three expectations are not observed and there are no signals from the output sensors or the possible following that would indicate the location of the pallet, it is assumed stuck in the intersection. A stuck pallet renders the intersection impassable, which leads to an according update of facts in the world model and a notification of the contact agent.

On the basis of the defined expectations and the available information, the automation agent is able to deduce the cause of a detected anomaly. After gaining according information from either its LLC or from other entities of the system, observations may be added to the world model. As soon as an observation confirms one of the expectations, the cause of the anomaly is found and further actions can be initiated.

The presented failure cases should not be considered to be an exhaustive list of possible failures in such a target system. They only represent a sample of failure cases that can be identified based on the available sensor information for explaining the principles of the diagnostic mechanisms. More information regarding diagnostics using the HLC can be found in [148, 238].

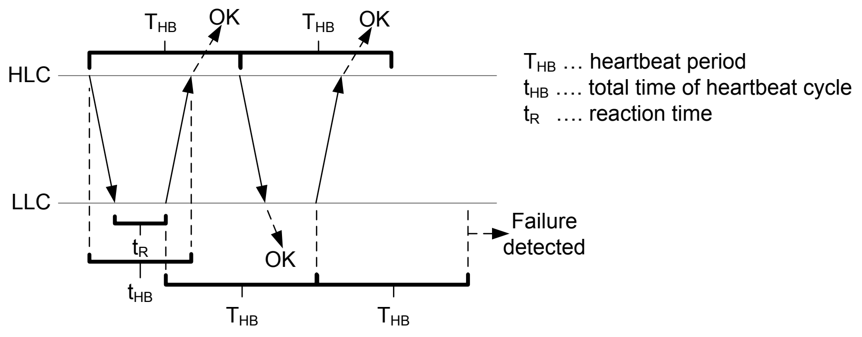

Figure 3.14: Heartbeat mechanism between the control layers of an automation agent [234].

3.7.3 Heartbeat Mechanism for Detecting Controller Breakdowns

Apart from failures and anomalies affecting the controlled components in the system and the moving pallets, also the components hosting the software parts can be affected by a breakdown, i.e. a PC hosting the HLC of one or several automation agents and the Festo CPX controllers each hosting one automation agent's LLC. This is a typical scenario for failure detection by exchanging "I am alive" messages periodically between the hosts [240].

Detecting a missing software part is possible by applying an approach based on a heartbeat mechanism, which means an exchange of request and answer messages between two entities [234]. Automation agents use an internal heartbeat between their HLC and LLC, which enables the two layers to monitor each other for verifying their operational status (see Figure 3.14). The heartbeat period has to be chosen sufficiently long as messages have to be transmitted over the network and then processed by the receiving entity. If an answer message is not received and processed within the specified time frame, the remaining layer notifies the system's agent management service, which has to react accordingly to this failure case. In case of a non-operational LLC, the controller can be rebooted, which is followed by the configuration of the LLC (Chapter 4 introduces the LLC configuration process). In case a restart of the non-operational layer is not successful, the component has to be bypassed by the system.

The heartbeat mechanism can also be used for monitoring functional agents with no physical representation. In this case, heartbeat messages are exchanged between different agents to verify their operational status.

3.8 Reconfiguration Infrastructure

While the internal structure of the HLC is flexible due to the modifiable knowledge structure in the form of the world model, the LLC relies on a reconfiguration infrastructure, which allows the modification of the IEC 61499 FB network. According to Wang and Shin, a reconfiguration process of the control software can encompass modifications as follows [16]:

- Modification of parameters: e.g. for adapting parameters of a closed-loop controller without changing its type;
- Modification of the execution sequence: The sequence of activities at a machine needs to be modified when the operation procedure changes (e.g. using the same machine to manufacture parts of another product).
- Modification of the information flow: In case the system is extended or reduced by physical components, their communication paths to the other components of the system need to be created or removed. Moreover, if relationships among components change (e.g. a subcomponent is allocated to a different higher-level component than before), the corresponding communication port linkages need to be adjusted.
- Modification of a component's provided functionality by changing or substituting parts of its control software elements: The addition, replacement or removal of software elements may be needed if new devices, control functions, and control algorithms are introduced (e.g. an intersection's role is changed from diverter to junction). The replacement of a software element can be viewed as a removal followed by an addition of such a part.

A reconfigurable control infrastructure has to ensure that the controlled process is not disturbed by the reconfiguration process. This encompasses issuing no wrong stimuli to the process and handling the resulting state changes correctly. Kramer and Magee have been one of the first who investigated reconfiguration processes for real-time systems and discussed the necessary infrastructure [241]. They suggest a configuration manager being responsible for conducting the reconfiguration process. The basis for the reconfiguration process is a so-called change specification. It is used for specifying desired changes to the system such as the introduction of new components or the modification of existing ones. For the execution of the transformation, the configuration manager needs knowledge about the existing system configuration, the system's state, and key properties of the software components involved. Based on this information it determines a sequence for applying the changes and also the prerequisites for each change.

In regard of the analysis of existing reconfiguration approaches based on IEC 61499 as presented in Section 2.2.4, the approach presented by Zoitl [159] is chosen as the basic infrastructure for enabling the automation agents to reconfigure their LLC. A working implementation is reported and the approach is suitable for the distributed dynamic reconfiguration of control components. Connecting this infrastructure with a suitable approach that automates the reconfiguration process would render the manual creation of reconfiguration applications (as proposed in Zoitl's approach) obsolete.

Zoitl proposes a programmable reconfiguration management, which provides the infrastructure for the reconfiguration of an application. Using the reconfiguration management, a so-called Reconfiguration Application (RCA) can be developed for conducting a reconfiguration process. While the RCA interacts with the target application during the reconfiguration, it has to gather the target application's current state and perform the modifications. The reconfiguration infrastructure provides dedicated interfaces for performing these tasks.

As the LLC of an automation agent is a real-time constrained control application, it has to react within certain time limits to state changes of the controlled process. These timing constraints need to be taken in to account regarding reconfiguration. In order to avoid stability problems during reconfiguration, Wang and Shin recommend that reconfiguration tasks are only performed at specific times during an application's execution at which it is not subject to timing constraints [16].

3.8.1 Basic Reconfiguration Services

Zoitl describes a set of so-called basic reconfiguration services that can be used for reconfiguring IEC 61499 applications on control devices [159]. These services are denoted as basic as they represent the minimal set of reconfiguration services that are required for performing control software reconfiguration tasks by executing an according sequence. The basic reconfiguration services are provided in the form of an internal as well as an external reconfiguration interface and can be grouped into the following classes:

- Structural Services are used for changing the structure of the control application. These services include the creation and deletion of FBs and connections as well as the modification of parameters.

- Library Services allow the addition or removal of type definitions (e.g. FBs or data types) in the control device.

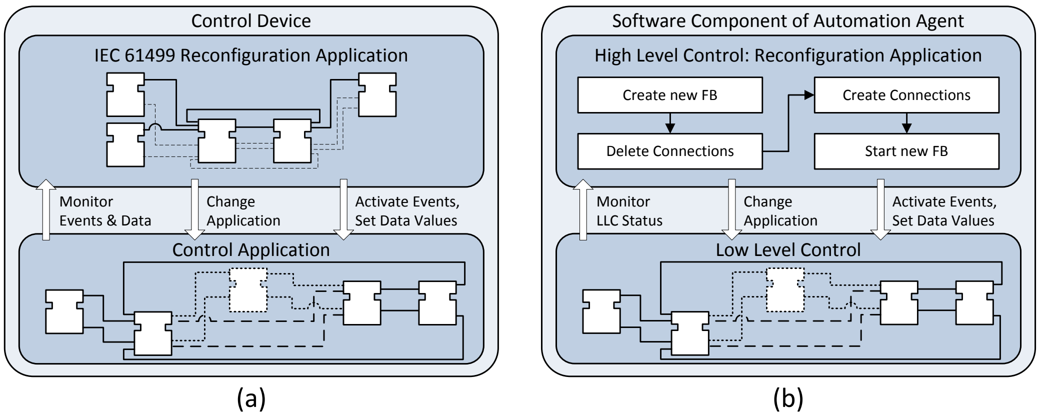

Figure 3.15: (a) An RCA realized with FBs of the internal reconfiguration interface interacts with the application that is reconfigured. All FBs run on the same control device. (b) The HLC of an automation agent acts as an RCA. It executes a sequence of commands that use the external reconfiguration interface for reconfiguring the LLC application. The figure is based on [73, 77].

- Execution Control Services are used for changing the execution state of FBs. This is needed for controlling if an FB shall respond to events or not.
- State Interaction Services enable the acquisition or modification of an FB's state, which is represented by the input and output data as well as by the FB's current internal data. Theses services are required for performing transition management algorithms.
- Query Services allow the retrieval of the target application's current structure. These services include the identification of the instantiated FBs and their connections as well as their types.

The internal reconfiguration interface is realized with a set of IEC 61499 service interface FBs. In this case, the RCA is tailored as an FB network, which results in the usage of one language for both applications. This has the advantage that the interaction interface between the RCA and the target application under reconfiguration is rather simple. Monitoring the event and data flow, setting data values, as well as issuing events are achieved directly by using specific FBs in the RCA (see Figure 3.15a). As mentioned in Section 2.2.4, the internal reconfiguration interface was used by Zoitl for reconfiguring the closed-loop control of an inverted pendulum [159].

On the contrary, the reconfiguration infrastructure allows performing a reconfiguration process remotely without an RCA composed of FBs. This approach is applicable for small control devices with not enough spare capacity for hosting an RCA or for reconfiguration processes, which do not need to fulfill tight real-time constraints. For such cases the basic reconfiguration services are provided by the infrastructure also as an external interface, which can be accessed via communication from external tools. Using Transmission Control Protocol (TCP) communication on Ethernet, a reconfiguration entity is able to access the external reconfiguration interface for requesting the execution of reconfiguration services. The reconfiguration request and its data is encapsulated in an XML string as defined in the standard IEC 61499. An example of a request for creating a connection between two FBs using the external reconfiguration interface is shown in Listing 3.3. Chapter 4 describes how the HLC is able to act as a reconfiguration application (see Figure 3.15b). Consequently, it uses the provided reconfiguration infrastructure for configuring the LLC.

```
<Request ID="#" Action="CREATE" >
  <Connection Source="FB1.Output" Destination="FB2.
     Input" />
</Request>
```

Listing 3.3: Example basic reconfiguration service "create connection" as an external reconfiguration command.

3.8.2 Suitability of the Reconfiguration Infrastructure

At the beginning of Section 3.8, several reconfiguration tasks are identified that have to be provided by a reconfiguration infrastructure. They can be summarized to changing parameters, changing the execution sequence, changing the information flow and changing a component's functionality. With the provided basic reconfiguration services described in the previous section, these tasks can be solved as follows:

- Changing parameters can be achieved with the structural services by removing and then setting the parameter (see Figure 3.16).
- Changing the execution sequence can be achieved with the structural services, i.e. with the services for creating and deleting connections (see Figure 3.17).
- Changing the information flow can be achieved with the the structural services, i.e. by changing communication IDs of FBs (likewise to Fig-

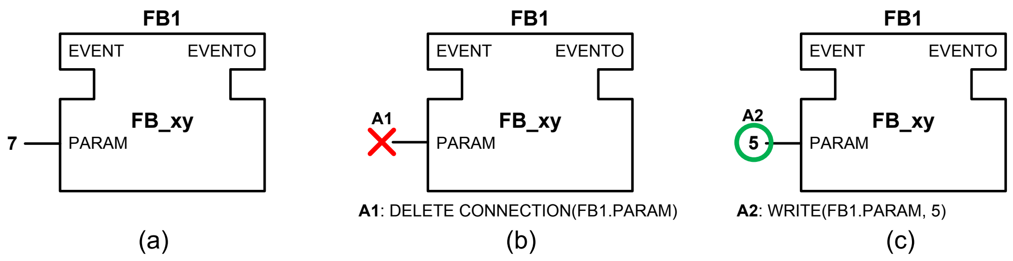

Figure 3.16: (a) FB with a parameter. (b) Removal of the parameter using the structural service “delete connection”. (c) Setting a new value of the parameter using the structural service “write”.

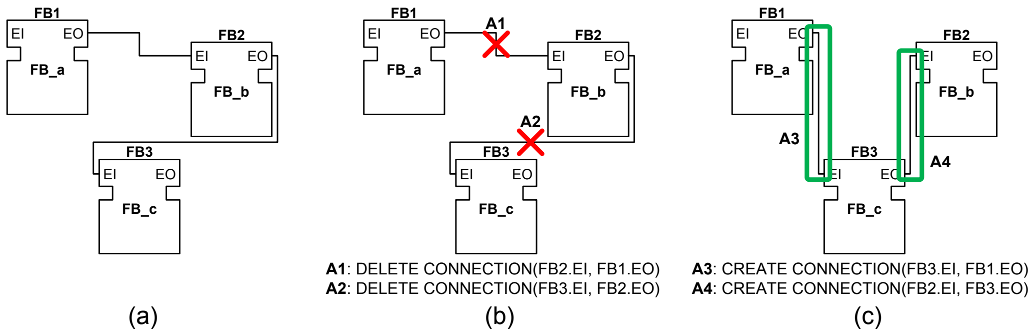
FB1
FB2
FB3
EI
EO
FB_a
FB_b
FB_c
A1
A2
A3
A4
A1: DELETE CONNECTION(FB2.EI, FB1.EO)
A2: DELETE CONNECTION(FB3.EI, FB2.EO)
A3: CREATE CONNECTION(FB3.EI, FB1.EO)
A4: CREATE CONNECTION(FB2.EI, FB3.EO)
(a)
(b)
(c)

Figure 3.17: (a) Initial FB network. (b) Removal of connections using the structural service “delete connection”. (c) Wiring of new connections using the structural service “create connection”.

ure 3.16) or by changing connections that provide those IDs (likewise to Figure 3.17).

- Changing a component’s functionality can be achieved mainly with the structural services, which allow to change the FBs and the connections between them, as well as with the state interaction services for adjusting new functionality (see Figure 3.18).

Apart from that, the execution control services can be used for activating or deactivating parts of the FB network.

By making use of this reconfiguration infrastructure, the automation agent is able to modify its LLC at runtime. Consequently, the FORTE is used as the runtime environment for the LLC and its FBs are developed using the corresponding engineering environment 4DIAC-IDE [242]. Thus, the HLC gains the ability of acting as an RCA using the external reconfiguration interface for transmitting according reconfiguration requests (see

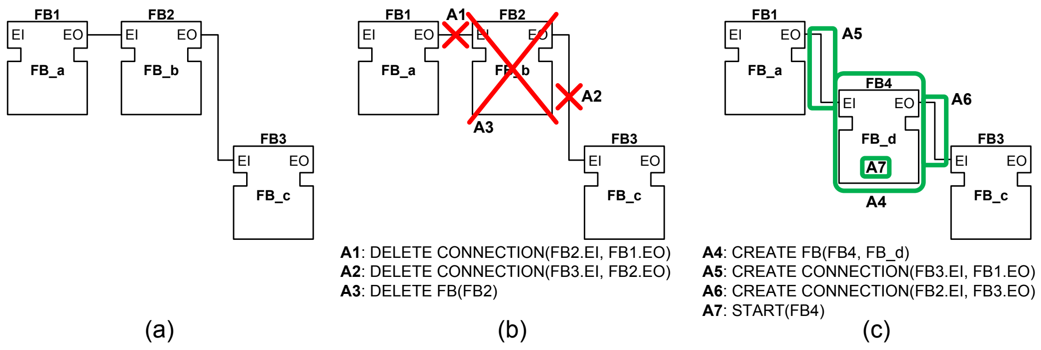
FB1
EI
EO
FB_a
FB2
FB_b
FB3
FB_c
(a)
A1
A2
A3
A1: DELETE CONNECTION(FB2.EI, FB1.EO)
A2: DELETE CONNECTION(FB3.EI, FB2.EO)
A3: DELETE FB(FB2)
(b)
FB4
FB_d
A4
A5
A6
A7
A4: CREATE FB(FB4, FB_d)
A5: CREATE CONNECTION(FB3.EI, FB1.EO)
A6: CREATE CONNECTION(FB2.EI, FB3.EO)
A7: START(FB4)
(c)

Figure 3.18: (a) Initial FB network. (b) Removal of connections and an FB using the structural services "delete connection" and "delete FB". (c) Instantiation of a new FB and wiring of new connections using the structural service "create connection" as well as starting the FB using the execution control service "start".

Figure 3.15b). The following section presents a test of the reconfiguration infrastructure in conjunction with an automation agent having to perform a predefined, simple reconfiguration process.

3.8.3 Test of the Reconfiguration Infrastructure

With the HLC and LLC not located on the same entity, the reconfiguration infrastructure of the LLC is accessed by the HLC using communication via network. Despite the improvements in computer technology in the last decades, the latency of network communication is still a factor to take into account regarding distributed applications [243]. This test investigates the time required for conducting a reconfiguration process via network.

The component-oriented application design that is used for the LLC allows the alteration of a component's functionality by replacing its central FB (see Section 3.5). This test emulates such a reconfiguration by adding one FB into an application. A control module for a CPX valve terminal of the type CPX-CEC-C1 by FESTO [226] is used as target system for this test example. It hosts an Xscale-PXA255 agile Intel microprocessor with 400 MHz, 28 MB Flash and 24 MB RAM, and an output module with 4 digital output ports is attached to it. A Linux operating system is employed on this type of controller and the FORTE is applied as the IEC 61499 runtime environment for executing an application.

Figure 3.19a shows a screenshot of the original IEC 61499 application, which was created using the 4DIAC-IDE and consists of a network of three

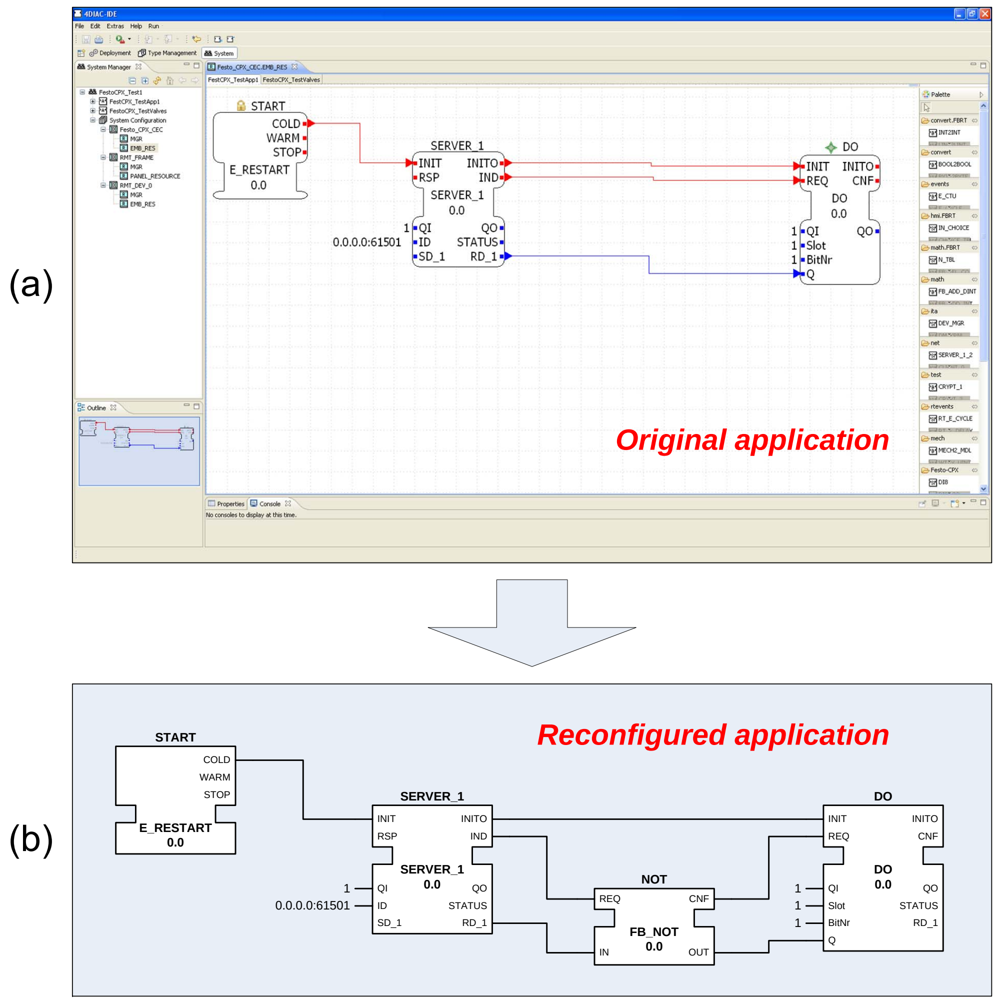

Figure 3.19: Screenshot of the development environment showing the original configuration (a) and diagram of the target configuration (b) [77].

FBs. The obligatory FB `START` of the type `E_RESTART` is responsible for sending the initial event. `SERVER_1` is used for communication to receive a boolean value which shall be written onto one of the digital output ports using the FB `DO`. After the HLC has performed the reconfiguration, the LLC application shall invert the received boolean value before writing it onto the digital output port. This requires the functionality of inverting a boolean value which is provided by the FB type `FB_NOT`. After the reconfiguration, the application shall have a structure according to Figure 3.19b—a screenshot is not possible as the reconfigured application only runs in the controller.

The RCA of this test example within the HLC consists of a sequence of Java commands, which send the reconfiguration requests in XML format to the device management in the FORTE hosting the LLC. Table 3.1 shows the sequence consisting of 8 requests in order to perform the reconfiguration. Before being able to connect the newly instantiated FB `NOT` of the type `FB_NOT`, the corresponding connections between `SERVER_1` and `DO` have to be deleted. New connections can then be created to achieve the desired application. Finally, the instantiated FB `NOT` has to be started to change its operational state from “idle” to “running”. Each time after receiving and executing a request, the LLC sends a response to the HLC. Only after receiving the corresponding response, the HLC issues the next request.

No	**Type of action**	**XML Command**
1	Creating new FB	`<Request ID="1" Action="CREATE"> <FB Name="NOT" Type="FB_NOT"/> </Request>`
2	Deleting connection	`<Request ID="2" Action="DELETE"> <Connection Source="SERVER_1.IND" Destination="DO.REQ"/> </Request>`
3	Deleting connection	`<Request ID="3" Action="DELETE"> <Connection Source="SERVER_1.RD_1" Destination="DO.Q"/> </Request>`
4	Creating new connection	`<Request ID="4" Action="CREATE"> <Connection Source="SERVER_1.IND" Destination="NOT.REQ"/> </Request>`
5	Creating new connection	`<Request ID="5" Action="CREATE"> <Connection Source="NOT.CNF" Destination="DO.REQ"/> </Request>`
6	Creating new connection	`<Request ID="6" Action="CREATE"> <Connection Source="SERVER_1.RD_1" Destination="NOT.IN"/> </Request>`
7	Creating new connection	`<Request ID="7" Action="CREATE"> <Connection Source="NOT.OUT" Destination="DO.Q"/> </Request>`
8	Starting the new FB	`<Request ID="8" Action="START"> <FB Name="NOT" Type=""/> </Request>`

Table 3.1: Issued commands for inserting an FB of the type `FB_NOT` into an application [77].

To measure the time of each reconfiguration step, as well as of the complete reconfiguration, the program Wireshark [244] is used to get the exact time stamp of each request and according response. The time difference indicates the required time for carrying out a request. For gaining representative data, a second reconfiguration sequence for returning the application to its original state is added to run the reconfiguration 100 times in a loop. The average duration and standard deviation of the issued commands are shown in Table 3.2. The average time of a complete reconfiguration sequence with the 8 management commands for adding FB `NOT` to the application is calculated to 75 ms with a mean deviation of 9 ms. Consequently, approximately 95% of the reconfiguration processes for this example will require between 57 and 93 ms. This is confirmed by the fact that the sample showed only 2 outliers making it therefore a representative sample.

No	**Type of action**	**Arithmetic mean (ms)**	**Standard deviation (ms)**
1	Creating new FB	4,49	1,05
2	Deleting connection	7,12	2,49
3	Deleting connection	10,56	3,75
4	Creating new connection	10,43	2,45
5	Creating new connection	10,03	2,81
6	Creating new connection	11,99	2,33
7	Creating new connection	10,94	2,55
8	Starting the new FB	9,49	2,96

Table 3.2: Average duration and standard deviation of the issued commands for inserting an FB of the type `FB_NOT` into an application.

The amount of time required to conduct a simple reconfiguration process as shown in this test example is sufficiently small enough for industrial processes in operation, which do not need to be performed under hard real-time constraints. Evidently, as each command requires a certain amount of time, a reconfiguration process with more commands has a longer duration. For instance, when an FB with a larger interface is inserted, more commands for setting up the connections to the other FBs might be required which leads to a longer reconfiguration process. Consequently, applications and FBs designed for supporting reconfiguration processes should employ adapters that integrate such larger interfaces [224]. A linear increase of required time in the case of more issued commands is shown by the timing measurements for configuring two differently sized applications presented in Section 4.5.1. Therefore, when considering the reconfiguration of an application, this linear increase of required time due to more elements in the application should be taken into account.

The IEC 61499 runtime maintains a list of all instantiated FBs and a list of all established connections with their end points [173]. When requiring information from the lists, the FORTE employs a linear search algorithm by going through the entries one by one. This means that the required time to gather information grows linear with the amount of entries in the lists. Commands that require information from one of those lists will therefore take a longer time the more entries a list has. While deleting an FB requires finding the particular FB in the list of instantiated FBs, the creation of an FB requires no such information as just one entry is added to that list. Therefore, the time for executing the command “delete FB” increases with every entry while the time for executing the command “create FB” does not increase. However, the timing measurements in Section 4.5.1 are not sufficient for investigations on this effect as the network latency is the dominating factor in the performed experiments.

3.9 Summary

This chapter introduces a hybrid architecture for agents controlling physical entities. This agent type is denoted as automation agent and its architecture comprises two software layers.

The HLC, which represents the deliberate layer of the automation agent, relies on a world model, i.e. a symbolic representation of the agent’s environment. The world model is based on ontologies and incorporates two parts: the situation model and the activity model. The agent’s own characteristics as well as its relations to other entities of the system are incorporated in the situation model, which is updated when these relations change. The activity model is employed for detecting inconsistencies (and thereby anomalies and failures) between the expected and the actual state of the agent and its environment, which is shown in a use-case in Chapter 5.

The LLC represents the reactive layer of the automation agent and possesses direct access to the physical world by its interface to the sensors and actuators of the controlled component. It is realized with a network of FBs based on the standard IEC 61499. By applying a component-oriented design according to the subcomponents of the controlled physical entity, the FB network incorporates only a small number of FBs. Furthermore, the design allows an easy reuse of each subcomponent’s according FB as will be shown in Chapter 4. Likewise to the superjacent layer, diagnostic mechanisms are also incorporated in this layer.

Even though the LLC for this work is realized solely on the basis of IEC 61499, the concept of a generic interface is introduced, which is designated for the connection of automation agents to other LLC types. This emphasizes the flexibility of the hybrid agent architecture and might represent a migration path for the application of agent technology in the industry.

Finally, a reconfiguration infrastructure is presented, which provides the basis for allowing an automation agent to modify its LLC. A test example is presented that shows that it is viable to use this reconfiguration infrastructure for a dynamic reconfiguration process despite the need of communicating via network. Chapters 4 and 5 present details on how the reconfiguration infrastructure is used by the agents.

Employing automation agents constitutes a control system with distributed and modular components incorporating the capabilities of diagnosis and self-reconfiguration, of which the latter is elaborated in the following chapters. With its properties and capabilities, the introduced agent architecture complies to the requirements for reconfigurable manufacturing as identified in Section 1.2.2.

CHAPTER 4

Automated System Configuration[1]

Auto-configuration management is essential for reconfigurable manufacturing systems in order to prevent extensive manual efforts, which would decrease the benefits of reconfiguration [23]. Thus, a system can provide support for the dynamic adaptation of its functionality and is made scalable in the presence of changes [17].

Ontologies are suitable for representing the knowledge of agents and the integration of both technologies brings advantages in the context of extensibility and communication [245]. Combining the concept of ontologies and explicit semantics with machine-based reasoning and inference can enable the execution of automated configuration processes [66].

This chapter introduces an approach for automatically configuring the control software of a distributed control system based on automation agents. The approach relies on an ontology for expressing details of the system components combined with the MAS, which is composed of the automation agents each controlling one component. The developed ontology covers the environment structure, characteristics, and component interrelationships enabling the agents to reason about these facts and perform decisions. Based on this knowledge, the agents use the configuration infrastructure presented in Section 3.8 for automatically configuring the hardware-near control layer. Thereby, an executable LLC implementation is generated for each system component within the corresponding controller.

[1]Parts of the contents of this chapter were previously published in [76].

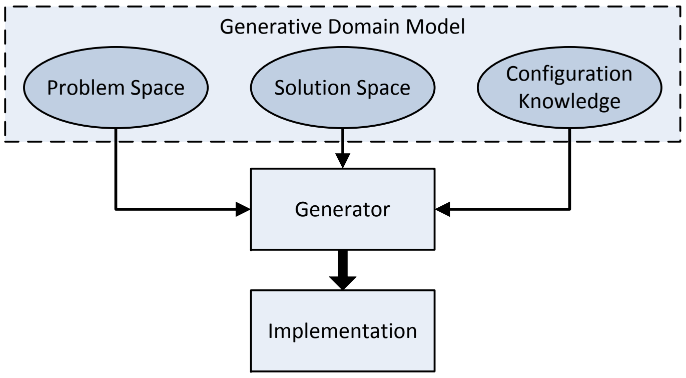

Figure 4.1: Paradigm of generative programming for automatically generating an executable implementation.

4.1 Automated Configuration based on Generative Programming

Generative programming is a paradigm of software engineering, which employs a generator for transforming information that is stored in a higher-level specification into an executable implementation. This means bridging a wide gap as there is commonly an essential difference between the structure of the specification and the structure of the implementation.

The paradigm of generative programming requires a generative domain model which incorporates the following concepts [246]:

- Problem space: contains application-oriented concepts and features which are mapped to the components of the solution space;
- Solution space: contains the implementation components which should be flexibly combinable and reusable with a minimum of code redundancies;
- Configuration knowledge: specifies the rules, dependencies, as well as illegal feature combinations for correctly mapping the features of the problem space to the components of the solution space.

Consequently, the generator utilizes the generative domain model to create the implementation (see Figure 4.1). To ensure a valid generation process, a specification check is performed using the constraints of the configuration knowledge. Finally, the executable implementation is a specific configuration of the implementation components of the solution space.

According to the paradigm of generative programming, the following tasks need to be done for achieving the automated configuration of the control software in a manufacturing system [247]:

1. Definition of the problem space in a format that is understandable and usable for the generator.

2. Definition of the solution space in the form of reusable software components that can be aggregated to form an application.

3. Definition of the configuration knowledge, which includes rules for the specification check ensuring consistency as well as for the actual generation of the application.

The components of the targeted manufacturing system need to be represented in the problem space as $PC = \{pc_0, ..., pc_n\}$ to enable their mapping onto the according software components $SC = \{sc_0, ..., sc_n\}$ of the solution space. Section 4.2 introduces the ontology and its concepts that represent the contents of the problem space. The solution space is constituted by the building bricks of the LLC, i.e. in this work the IEC 61499 FBs as presented in Section 3.5.

For defining the configuration knowledge, a rule base $R = \{r_1, ..., r_n\}$ has to be defined. Each of the rules $r_i : \phi_i(x_i) \rightarrow \psi_i(y_i), i \in \{1, ..., k\}, k \leq n$, which are part of the sub-base R_c for generating control code, consists of a condition function $\phi(\cdot)$ and an action function $\psi(\cdot)$ [248]. On the contrary, each rule $r_j : \phi_j(x_j) \rightarrow \xi_j(y_j), j \in \{1, ..., l\}, l \leq n, k + l = n$ for the specification check in the sub-base R_s incorporates a constraint $\xi(\cdot)$ instead of an action function. The rule base is presented in Section 4.4.

Figure 4.2 depicts the workflow of the configuration process, which is derived from the generative programming paradigm. Using the provided problem space representation in the ontology and the FBs of the solution space, the agent shall act as generator that applies the rules of the configuration knowledge for generating the executable LLC implementation.

4.2 Ontology of the Control System

For the automated configuration of the control software, a type of a model is required, which contains information concerning the structure of the target system. As described in Section 3.4, an automation agent relies on an ontological world model as knowledge base. Consequently, the problem space is represented in the form of an ontology for the automated configuration of control software.

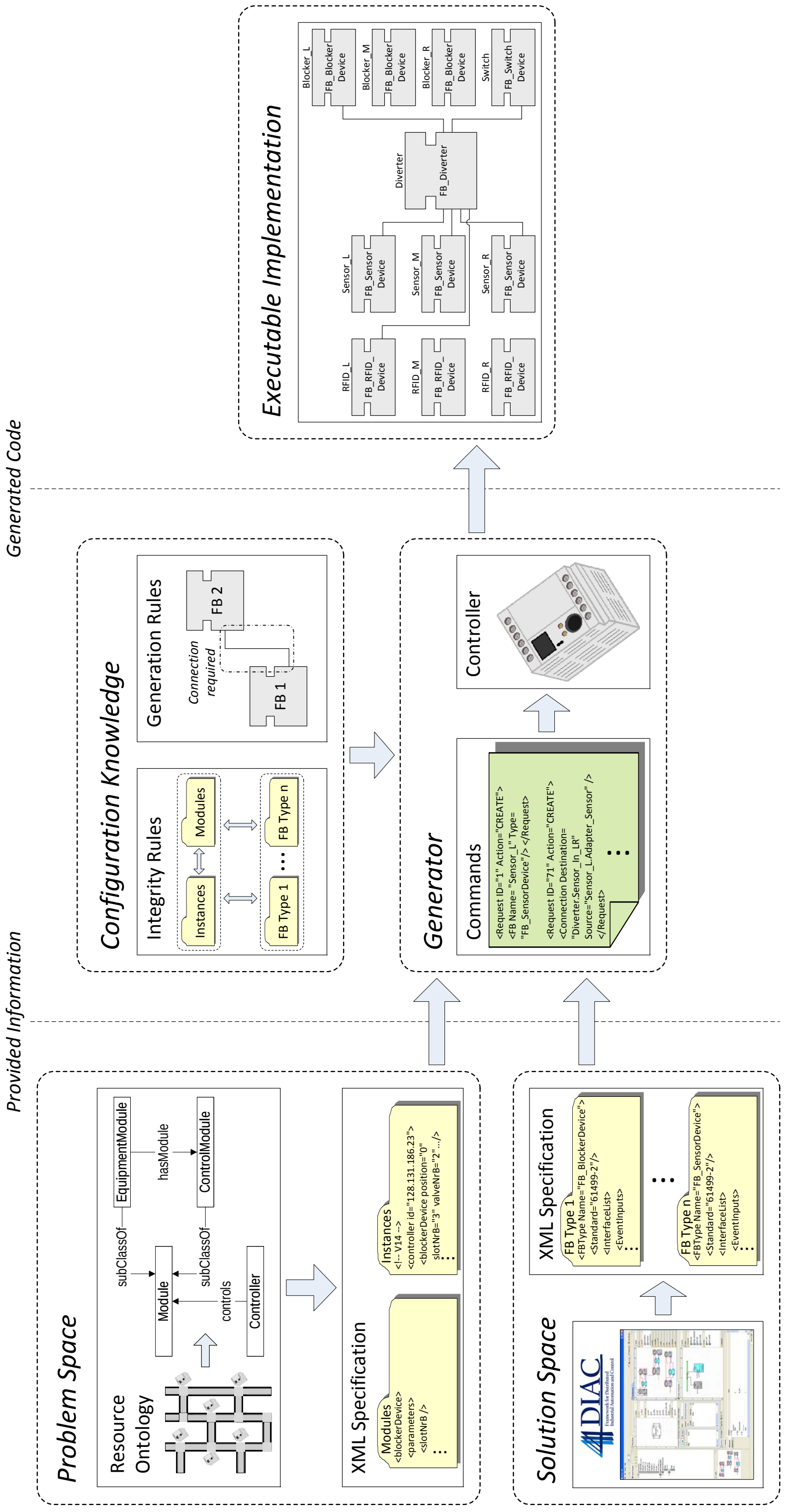

Figure 4.2: Workflow of the configuration process to achieve an executable implementation in the LLC of an automation agent.

In order to be usable for different target systems, the system ontology needs to be structured according to a domain ontology, which defines concepts specific to a domain [249]. In this context, standards represent a consensus on the semantics of terms and definitions in a domain, which means that being compliant with them results in an increased reusability and applicability in industrial practice [250]. Besides, the issue of standardization is considered to be a major challenge concerning the industrial acceptance of semantic and agent technologies [49].

The standard ANSI/ISA-S95 Enterprise-Control System Integration provides a hierarchically structured equipment model in its Part 1: Models and Terminology [251], which is applicable for the manufacturing domain. However, the provided model does not consider entities below the level of work cells. The complete target system of this work represents a work cell by itself and therefore more granularity is required. Consequently, the standard ANSI/ISA-S95 is too abstract for being used in the domain ontology of this work.

Regarding the batch process domain, the standard IEC 61512 Batch Control, Part 1: Models and Terminology [252], respectively ANSI/ISA-88 Batch Control, provides reference models and structures as well as definitions concerning the physical equipment on the field level apart from various other models. Even though designed for batch control, the guidelines of this standard have been adopted to other domains such as discrete manufacturing to apply likewise structural concepts and guidelines in these domains [253].

As mentioned in Section 2.1.4, Lohse et al. report an equipment module ontology with the focus on the functions and behaviors of equipment entities and their connections [136]. In their ontology, the physical object classes of various hierarchical levels are subclasses of a generic class called `Equipment`. This is meaningful, as attributes that all physical objects possess can be inherited from this generic class. However, the compliance with any industrial standard is not reported in this approach.

Following the idea of a generic class for physical objects but combining it with the compliance with the standard IEC 61512 delivers the following classes as core concepts of the hardware-related ontology part (see Figure 4.3, Hardware) for representing the physical components of the target system:

- `Module`: This class represents the abstract class for any actuator, sensor, or aggregation of these components.
- `ControlModule`: This is a subclass of the class `Module`. According to the standard IEC 61512, basic components of a manufacturing system such as blockers and switches, which act as the smallest entities,

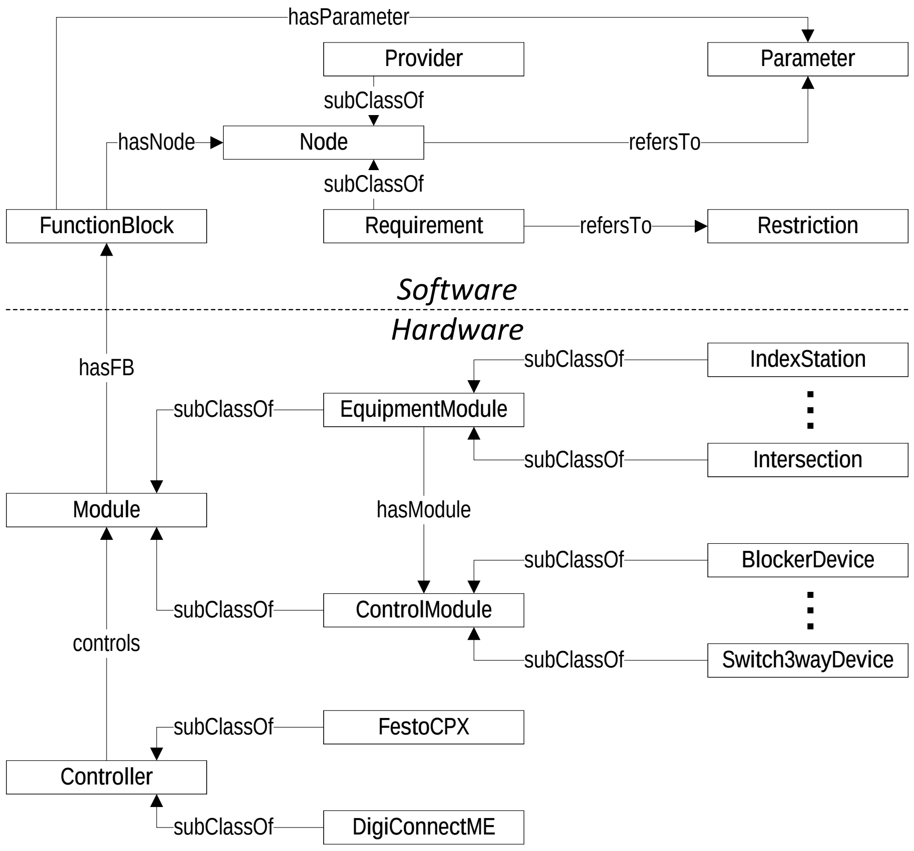

Figure 4.3: Reduced overview of the classes and their relations of the resource ontology describing the components of the target system as well as the software concepts, based on [76].

are referred to as control modules. All types of basic components are therefore subclasses of the class `ControlModule`.

- `EquipmentModule`: This is a subclass of the class `Module`. According to the standard IEC 61512, equipment modules are aggregations of control modules, which is manifested in the ontology with the relation `hasModule` between the classes `EquipmentModule` and `ControlModule`. In the testbed, all types of aggregated components such as intersections, which are composed of sensors, blockers, RFID modules and usually a switch, are subclasses of the class `EquipmentModule`.

Neither the ontology by Lohse et al. nor the standard IEC 61512 takes the controller devices into account. But controllers such as PLCs need to be included for facilitating such an integration process [66]. Regarding the ontologies mentioned in Section 2.1.4, only the work by Alsafi and Vyatkin involves the representation of control devices [140]. Taking this approach into account extends the resource ontology by the class `Controller`, which represents the abstract class for any type of controller. The control of the physical components is represented by the relation `controls` between the classes `Controller` and `Module`. The employed types of controllers are derived as subclasses from the class `Controller`.

For the automated configuration of the LLC, the basic concept of FBs needs to be represented in the ontology. Designating the automatic composition of automation objects as future work, Orozco and Lastra report an ontology that involves the concept of FBs of the standard IEC 61499 [254]. Consequently, further semantic extensions in the form of a software-related part are incorporated for the wiring of the LLC application (see Figure 4.3, Software):

- `FunctionBlock`: This class refers in principle to software components of any type of component-oriented control software and does not necessarily refer to an IEC 61499 FB. This general concept is of importance in the case this approach is adopted for configuring a different type of LLC. Each module type is represented in the control software by a specific type of FB, which is represented by the relation `hasFB`.

- `Node`: The nodes of a software component represent the connection points to the software components of other modules within the same LLC application. The link with the class `FunctionBlock` is represented with the relation `hasNode`. In the case of IEC 61499 FBs, the nodes can represent event and data connection points as well as adapter ports.

The ontology has to include furthermore concepts that enable the automatic matchmaking of the FBs' nodes. In this context, the principle of SOAs involves the matchmaking of entities: service requestors are connected with service providers that can fulfill their request [57]. Valid connections are found in such systems based on descriptions and conditions. Even though not being denoted as SOA, an approach for the matching of equipment modules incorporates the approach of connectivity constraints for linking fitting equipment interfaces with each other [136]. In regard of the standard IEC 61499, the discovery and automatic association of these FBs is rather an open point [255]. Consequently, the additional concepts are represented in the ontology that allow the linking of FBs and their nodes:

- `Provider`: This class categorizes nodes that are offered by a software component. Therefore, the possible link between the classes `Provider` and `Node` is provided with the relation `isA`.

- `Requirement`: This class categorizes nodes of a software component, which require a connection or a value. The possible link to the class `Node` is provided with the relation `isA`.

- `Parameter`: This class refers to the parameters of the software components. On the one hand, values can be required by nodes of a software component. On the other hand, providers are linked with specific parameter values in order to be checked in regard to restrictions. Consequently, both classes `Provider` and `Requirement` can link to a parameter with the relation `refersTo`. For example, the specific pin number at the controller, to which a control module is connected to, is a required parameter for the software component representing that module in the LLC application.

- `Restriction`: This class represents restrictions given by the nodes that require a connection. Therefore, the class `Requirement` is linked to this class with the relation `refersTo`. In order to achieve a valid executable LLC, all required nodes of the LLC's software components need to have connections to provided nodes, which fulfill the given restrictions.

Generally, the providers are linked with their FB's parameters and specific values are expected by the requirements of other FBs, which are described in the requirement's restrictions.

Figure 4.4 depicts selected instances of the example intersection `i1` introduced in Section 3.2.1. The intersection incorporates instances of different control module types (such as `i1_Sensor_L`, `i1_Blocker_L`) that are each represented by distinct FBs. Each FB possesses nodes of either the class `Requirement` or `Provider`. While nodes of the class `Provider` (e.g. `Adapter_Sensor` of the FB `Sensor_Device`) refer to parameters, nodes of the class `Requirement` refer to restrictions. These restrictions are defined as constraints that can be interpreted during the configuration process of the LLC. Besides, intersection `i1` is also linked with the instances of the adjacent conveyors. Such information is used for determining the intersection's operational mode (i.e. diverter in the case of `i1`) as well as for calculating the routing tables (see Section 3.2.2).

The developed ontology represents the problem space of the automated configuration and its contents are stored in an according XML format. Using XML for the design artifacts of the ontology goes along with advantages such as its platform independence as well as a self-explanatory representation [157]. The classes of the resource ontology with their relations are stored in a module specification file. The instances are stored in a corresponding instance specification file.

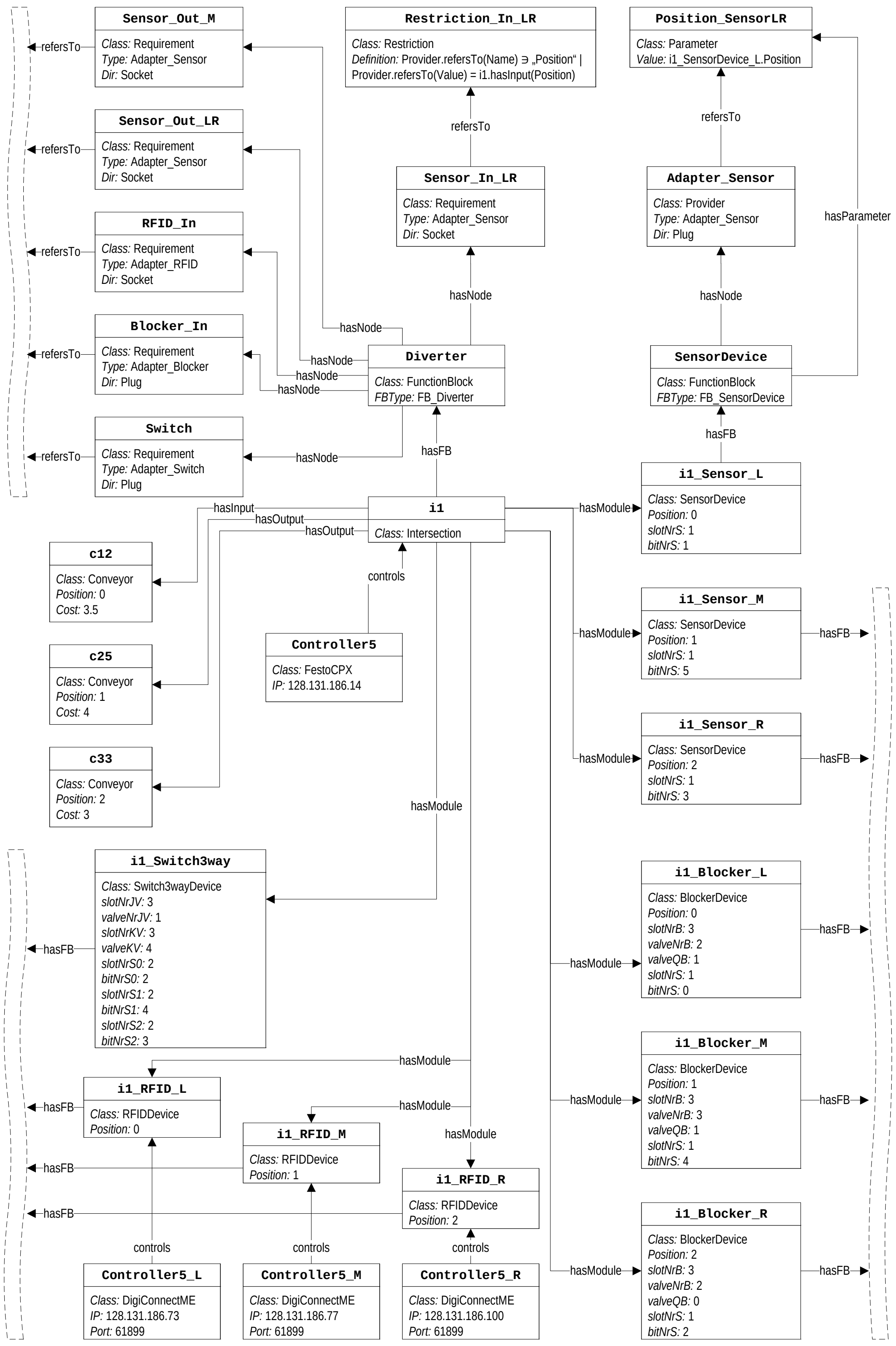

Figure 4.4: Reduced overview of the instances of an intersection representing the physical modules as well as the software components and their nodes.

4.3 Configuration Process of the High Level Control

At system startup, the HLC for each equipment module is created automatically by the agent system based on the target physical system's topology and components. This is done in a few steps and shall be elaborated on the basis of intersection `i1`, which was already used as example in Chapter 3:

1. An automation agent in JADE is instantiated for each equipment module instance in the resource ontology. A local ontology is created for this agent's situation model and the generic activity types, such as "`Routing Pallet`" for an intersection, are added to its activity model.

2. The instance specifications in the ontology deliver information about this agent's own configuration and the neighboring entities in order to setup the facts in the situation model. In this example, it is provided with the information that conveyor `c12` serves as input and the conveyors `c25` and `c33` as outputs.

3. Based on the knowledge in the situation model, more concrete activity definitions in the activity model are derived. For instance the activity "`Routing Pallet from (c12 or c25) to (c25 or c33)`" is created as a specialization of the generic activity "`Routing Pallet`". Based on the fact that conveyor `c12` is the sole input conveyor, the activity "`Routing Pallet from c12 to (c25 or c33)`" is added.

4. After receiving its routing table from the contact agent, the activity "`Routing palletToDS1 from c12 to c25`" is derived, which means that a pallet is routed to output conveyor `c25` in case it needs to reach the target indexstation `DS1`.

The steps were described only briefly as the configuration process of the HLC is beyond the scope of this thesis. More details regarding this process are presented in [207, 238].

4.4 Configuration Process of the Low Level Control

Each automation agent's HLC acts as a generator which has to configure its LLC accordingly. For this purpose the HLC relies on the module and instance specification files to determine the automation agent's physical parts (sensors, blockers, etc.) and how they are connected with the corresponding controller (i.e. specific pin of an I/O module at a Festo CPX). Furthermore, the FB types, which have been developed using the 4DIAC-IDE, are provided in XML format as the components of the solution space. According to the component-oriented design of the LLC (see Section 3.5) each FB incorporates either the functionality of one specific subcomponent (e.g. `FB_BlockerDevice`) or the higher LLC functionalities of a component, which includes the communication with the HLC or the routing of pallets (e.g. `FB_Diverter`). Finally, a set of constraints forming the integrity rules for the specification check (see Section 4.4.1) and generation rules (see Section 4.4.2) for the actual LLC configuration are embedded in a set of Java methods that can be used by the automation agents. The HLC then sends a sequence of commands to the controller, which will host its according LLC, for generating the executable implementation.

For the formalization of the constraints and generation rules, the modules (equipment modules such as intersections as well as control modules such as blocker devices) in the module specification file are denoted as $M = \{m_0, ..., m_n\}$. The designation $modreq_m$ encompasses the FB nodes of a module m that either require connections or values, and $modprov_m$ encompasses the nodes that provide connections for other modules. In the following each of these nodes is denoted either as a $Requirement$ or a $Provider$. Both types of nodes can have parameters of $modpara_m$ linked, which is used for the creation of connections between the FBs in the LLC. Furthermore, for the constraint definition of the specification check the set $ModReq$ represents the $Requirements$ of all specified modules and the set $ModProv$ represents their $Providers$.

The instance specification file contains the instances of the ontology $I = \{i_0, ..., i_n\}$. The class of an instance is denoted by $class_i$ while the set of its parameters and their values are contained in $instpara_i$. The set $InstPara$ represents the parameters of all instances.

Finally, each FB type provided in the solution space is defined as $FB_j = (FBname_j, FBIO_j)$. In this context, $FBname_j$ denotes the name and $FBIO_j$ contains all input and output nodes.

4.4.1 Specification Check

Before issuing the commands for creating the LLC, the HLC performs a specification check by analyzing the XML specification files. This ensures a valid specification, which is especially of importance in the case of a manually created specification. To perform the check, the HLC proofs the validity of the XML specification against a set of constraints.

Firstly, the content of the module file is validated. Several constraints refer to information of the FB type files of the solution space. It needs to be ensured that the software components represented in the ontology of the problem space can be mapped onto the FBs of the solution space. In this context, the fulfillment of Constraint 4.1 assures that all required and provided nodes of a specific software component m of the problem space have a corresponding counterpart in the according FB type definition of the solution space. In the implementation this is done by finding exactly one equivalent among the nodes of an FB type definition for each software component node.

$$\begin{aligned}&\forall x \in (modreq_m \cup modprov_m)\\ &\qquad \rightarrow \exists! y \in \mathit{FBIO}_m | \mathit{Name}(x) = \mathit{Name}(y)\end{aligned} \tag{4.1}$$

The usage of the adapter concept for the nodes of the FBs (see Section 3.5) in the solution space delivers point-to-point connections [173]. Therefore, each node of a software component m can either be a *Requirement* or a *Provider* but not both (Constraint 4.2).

$$\forall x \in modreq_m \rightarrow x \notin modprov_m \tag{4.2}$$

Having nodes with one name but differing types leads to an invalid connection attempt. To ensure consistency between the software component types, *Providers* that share the name but are of different software components have to be of the same type (Constraint 4.3).

$$\begin{aligned}&\forall x, y \in \mathit{ModProv} | (x \neq y) \wedge (\mathit{Name}(x) = \mathit{Name}(y))\\ &\qquad \rightarrow \mathit{Type}(x) = \mathit{Type}(y)\end{aligned} \tag{4.3}$$

The property of being a plug/socket in the case of an adapter interface or an input/output in the case of a standard port is denoted as a node's direction. Likewise to the type, if multiple *Providers* of the same name are existent, then they have to be of the same direction (Constraint 4.4).

$$\begin{aligned}&\forall x, y \in \mathit{ModProv} | (x \neq y) \wedge (\mathit{Name}(x) = \mathit{Name}(y))\\ &\qquad \rightarrow \mathit{Dir}(x) = \mathit{Dir}(y)\end{aligned} \tag{4.4}$$

To ensure that a software component has access to the necessary information for its operation, its required node connections need to have a provided counterpart. While the restrictions are taken into account during the actual wiring of the FBs (see Section 4.4.2), the specification check ensures that at least one *Provider* y exists that shares the name with each *Requirement* x (Constraint 4.5).

$$\forall x \in \mathit{ModReq} \rightarrow \exists y \in \mathit{ModProv} | \mathit{Name}(x) = \mathit{Name}(y) \tag{4.5}$$

Having nodes with one name but differing types leads to an invalid connection attempt in case a *Requirement* and a *Provider* share the name but not the type. In order to be compatible, any found *Providers* sharing the name with *Requirement* x has to also share its type (Constraint 4.6).

$$\begin{split}\forall (x \in \mathit{ModReq}, y \in \mathit{ModProv}) | \mathit{Name}(x) = \mathit{Name}(y) \\ \rightarrow \mathit{Type}(x) = \mathit{Type}(y)\end{split} \tag{4.6}$$

In the case of adapter nodes, *Providers* and *Requirements* cannot be both plugs or sockets. An adapter connection requires one plug and one socket. Also standard connections require both an input and an output node. Consequently, any found *Providers* that shares the name with *Requirement* x has to be of the opposite direction (Constraint 4.7).

$$\begin{split}\forall (x \in \mathit{ModReq}, y \in \mathit{ModProv}) | \mathit{Name}(x) = \mathit{Name}(y) \\ \rightarrow \mathit{Dir}(x) \neq \mathit{Dir}(y)\end{split} \tag{4.7}$$

Then, the content of the instance file is validated with a further set of constraints. For each instance defined in the problem space, an FB type has to exist in the solution space (Constraint 4.8). To avoid ambiguousness during instantiation of the FBs, there cannot be multiple FB types that share the name with one instance. But of course there can be multiple instances created from one FB type.

$$\forall i \in I \rightarrow \exists! m \in M | \mathit{class}_i = \mathit{modname}_m \tag{4.8}$$

Each definition of an instance has to encompass values for the parameters of a software component. The completeness of this set of values is verified with Constraint 4.9, which is used to check if a value exists for all required parameters of a software component m.

$$\begin{split}\forall x \in \mathit{ModPara}_m | \exists (i \in I | \mathit{class}_i = \mathit{modname}_m) \\ \rightarrow \exists! y \in \mathit{instpara}_i | (\mathit{Name}(x) = \mathit{Name}(y)) \wedge (\mathit{Value}(y) \neq \emptyset)\end{split} \tag{4.9}$$

For each *Requirement* x of the instances a corresponding *Provider* y of the same type has to exist among the nodes of the other instances (Constraint 4.10). Specific restrictions based on the instances' parameter values have to be met to assure a valid connection between two instances that are to be connected.

$$\forall x \in \mathit{InstOf}(\mathit{ModReq}) \rightarrow \exists y \in \mathit{InstOf}(\mathit{ModProv}) \\ |(\mathit{Type}(x) = \mathit{Type}(y)) \wedge \mathit{Restrictions} \quad (4.10)$$

To reveal more details about how the constraints are implemented, Listing 4.1 shows an example in pseudocode of the Constraints 4.3 and 4.4. If the specification check is performed with a valid result, the LLC is created by using the configuration services.

```
Create list(name, type, direction);
For i := 1 to number of software_components do
  For j := 1 to number of providers of
     software_components(i) do
    If list contains name of provider(j) of
       software_components(i) then
      If type of provider(j) of software_components(i)
          is not equal to type of named provider in
         list then
        Throw exception;
      If direction of provider(j) of
         software_components(i) is not equal to
         direction of named provider in list then
        Throw exception;
    Else if
      Add name and type of provider(j) of
         software_components(i) to list;
```

Listing 4.1: Example in pseudocode combining Constraints 4.3 and 4.4 in one routine.

4.4.2 Configuration Services

After successful completion of the specification check, the configuration services are used for creating the control application. Both IEC 61131 and IEC 61499 encompass the concept of the "resource" as functional unit and container to host a network of software components. Such a resource is required in the control device before creating the actual control application. Consequently, a resource is created if there is at least one instance defined in the instance specification (Rule 4.11).

$$I \neq \emptyset \rightarrow \mathit{CreateResource} \tag{4.11}$$

Upon creation of the resource, the instances of the software components can be created. For all instances, which are defined in the instance specification, a corresponding FB is created (Rule 4.12).

$$\forall i \in I \rightarrow \mathit{CreateFB}(\mathit{FBname}_i) \tag{4.12}$$

As next step, the according parameters of the created FBs are written as described in the instance specification (Rule 4.13).

$$\forall x \in \mathit{InstPara} \rightarrow \mathit{CreateConnection}(x, \mathit{Value}(x)) \tag{4.13}$$

Each *Requirement* x is then connected with a corresponding *Provider* y of the same type, which meets the *Restrictions* that are defined in the module specification (Rule 4.14).

$$\begin{aligned} &\forall (x \in \mathit{InstOf}(\mathit{ModReq}), y \in \mathit{InstOf}(\mathit{ModProv})) \\ &\quad |(\mathit{Type}(x) = \mathit{Type}(y)) \wedge \mathit{Restrictions} \rightarrow \mathit{CreateConnection}(x, y) \end{aligned} \tag{4.14}$$

Finally, the FB network is initialized by issuing a start command.

Configuring the LLC requires a sequence of the structural reconfiguration services (see Section 3.8.1) for creating the required FBs and connections as well as for setting the according parameters. Moreover, the execution control services are utilized for changing the operational state of the created FBs from “idle” to “running” (for an overview about the operational states see [173]).

Table 4.1 shows a part of the command sequence starting with the resource creation (command 0 in the figure) and ending with the start of the resource for configuring the LLC of a diverter in the pallet transport system. The shown commands between the resource creation and its start, for creating FBs, writing their parameters and creating the connections, are just a selection as the other commands for the configuration process are structured likewise.

Regarding the complete target system (see Section 3.2), the configuration of each agent’s LLC is performed individually by its HLC on the basis of the instance definitions in the resource ontology. Thus, the distinct agents’ configuration processes are logically independent from each other, which is an advantage regarding scalability of the approach.

No	Type of action	XML Command
0	Creating resource	`<Request ID="0" Action="CREATE"> <FB Name="EMB_RES" Type="EMB_RES"/> </Request>`
1	Creating new FB	`<Request ID="1" Action="CREATE"> <FB Name="Sensor_L" Type="FB_SensorDevice"/> </Request>`
...	...	...
31	Setting parameter	`<Request ID="31" Action="WRITE"> <Connection Destination="Sensor_L.SlotNrS" Source="1"/> </Request>`
32	Setting parameter	`<Request ID="32" Action="WRITE"> <Connection Destination="Sensor_L.BitNrS" Source="1"/> </Request>`
...	...	...
71	Creating new connection	`<Request ID="71" Action="CREATE"> <Connection Destination="Diverter.Sensor_In_LR" Source="Sensor_L.Adapter_Sensor" /> </Request>`
...	...	...
76	Starting LLC	`<Request ID="76" Action="START">`

Table 4.1: Issued commands for configuring an automation agent's LLC, which encompasses creating and wiring of FBs as well as starting the resource.

4.5 Performance Results

As already mentioned, the dynamic configuration of an industrial control system is a key technology for achieving flexible production systems. Depending on the timing constraints of the manufacturing environment and to avoid an extensive down-time of the system components, such a process has to be carried out in a specific time frame.

In the target system, each agent's LLC is hosted in a distinct distributed industrial controller of the type CPX-CEC-C1 by FESTO. On the contrary, all HLCs are located in a PC due to the missing integration of the agent framework JADE on the employed controllers. However despite being not physically distributed, the HLCs are still logically distributed acting as distinct entities. Consequently, the reconfiguration infrastructure of the LLC is accessed by the HLC using communication via network.

Consequently, the following experiment investigates on the required time for carrying out the LLC configuration process. Likewise to the experiment presented in Section 3.8.3, the program Wireshark [244] is used to measure the time of each reconfiguration step and thus of the complete reconfiguration process. While the previous experiment was carried out using a predetermined reconfiguration sequence, the present experiment involves the dynamic configuration introduced in this chapter for automatically generating intersection LLCs.

4.5.1 Timing Measurements

The timing measurements show that the execution of each command takes place in average within 13 ms. Compared to the infrastructure test presented in Section 3.8.3, this is slightly longer due to the calculation time for determining the next configuration command. Configuring the LLC of an automation agent acting as a diverter in the pallet transport system requires its HLC to send a sequence of 77 commands over the network to the corresponding Festo CPX controller. Consequently, the complete configuration process for one automation agent requires approximately 1 second (see Figure 4.5a). This duration meets the requirement of achieving a reconfiguration during operation within a maximum of about 1 second as stated in [256]. Evidently, it would not be sufficiently fast for industrial processes, which are performed under hard real-time constraints.

Even though most runs require almost exactly 1 second, run #3 required about 300 ms more while run #4 required about 300 ms less. These significant differences might be caused by network traffic, which indicates that a configuration process could be faster if the HLC resides directly within the same controller as the LLC. However, deploying software agents with their required framework directly onto industrial controllers represents a severe challenge due to their limited computational power and resources compared to a regular PC [228].

Due to the distributed nature of the target system and the logically distributed HLCs, the configuration of the agents' LLC can be performed in parallel. Correspondingly, using parallel threads for configuring several controllers with the presented approach keeps the configuration time at roughly 1 second as measurements show when 5 automation agents configure their LLCs in parallel (see Figure 4.5b). Also here, one configuration run is significantly shorter, which shows the potential of developing a solution that renders the network communication between HLC and LLC obsolete. Nevertheless, further timing measurements with 10 or 15 controllers reveal a total configuration time of approximately 1 second as well (diagrams are omitted as they are likewise to the ones shown in Figure 4.5). Regardless of the number of configured LLCs and thereby configured controllers, the configuration time does not change significantly.

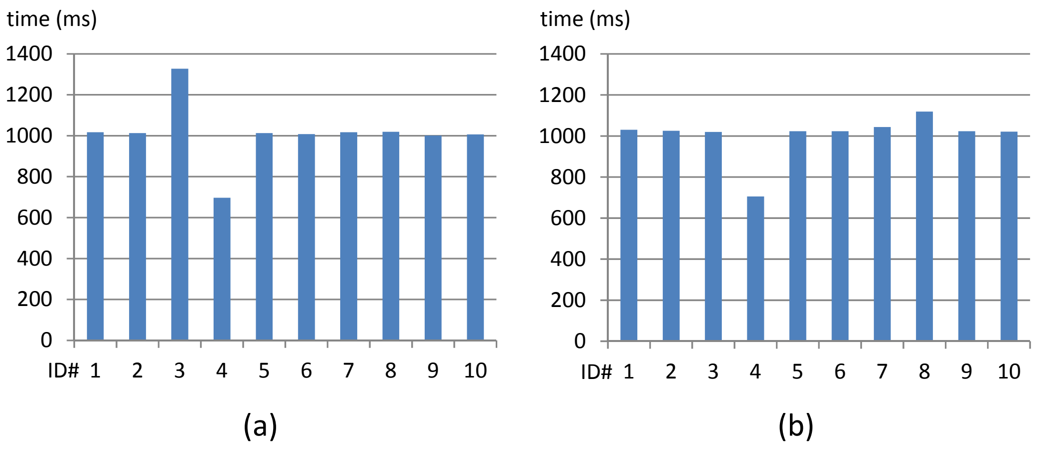

Figure 4.5: Execution time of 10 LLC configuration runs in (a) 1 controller with 77 commands and (b) 5 controllers using parallel threads each with 77 commands.

Likewise to compiling a program for a PLC, the configuration time increases with the number of required commands for each distinct controller. In this context, the component-oriented design of the LLC used for the automation agents brings significant performance advantages compared to a less decoupled application design that is not based on adapter connections between the FBs. In that case, the instance creation would be significantly more complex due to the higher number of FBs and connections between them. This is confirmed when the configuration time of the presented approach is compared with a monolithic LLC approach that was previously used for the pallet transport system before employing the component-based LLC approach that is introduced in this work.

This previous approach involved pre-defined XML files for configuring the agent LLCs in the Festo CPX controllers. Those files incorporated a complete description of the FB network including a list of all connections between the FBs. Even though being distributed on several controllers, the rather monolithic design of the used LLC applications impeded an easy exchange of FBs and thereby the dynamic reconfiguration of the control software. Therefore, an automation agent's LLC incorporated the functionality for all expected system states following thereby a contingencies approach of reconfiguration [186]. For instance an automation agent controlling an intersection incorporated functionality for acting as a diverter as well as a junction already at system startup. Thus, if this intersection started as a diverter, it could switch to the junction functionality as reaction on a direction change of an adjacent conveyor.

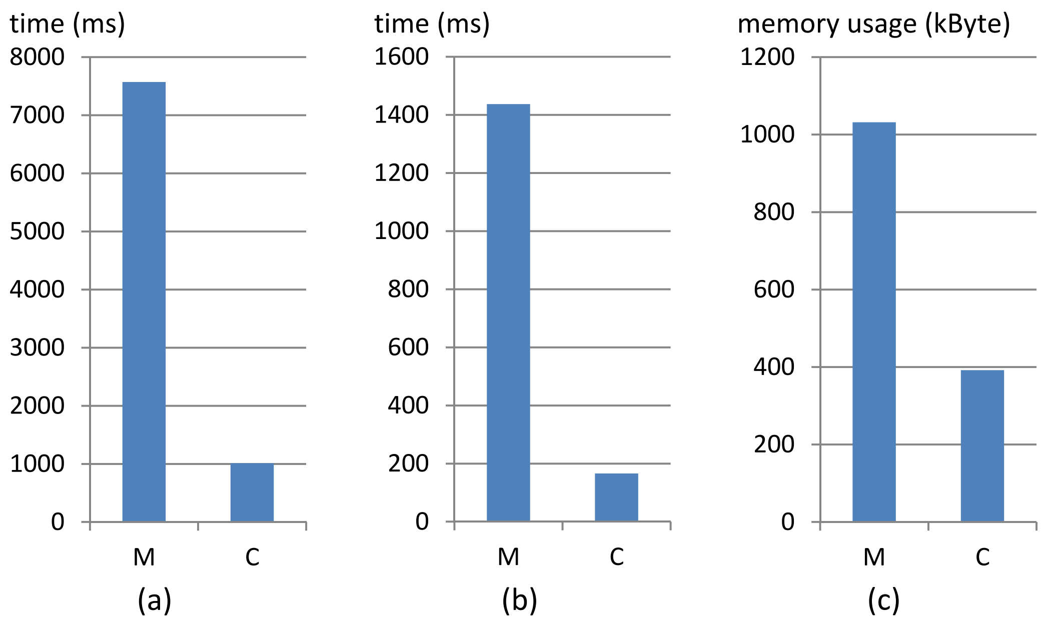

Figure 4.6: Average execution time of the configuration process on the Festo CPX (a) as well as on the Digi Connect ME controllers (b), and memory usage in the Festo CPX (c) of the monolithic LLC design (M) as well as of the component-oriented LLC design (C).

Configuring the FBs of the monolithic design and the connections between them required 500 commands, which is significantly more than the 77 commands required for configuring the component-oriented LLC as mentioned before. The duration of the configuration was measured to be around 7.5 seconds (see Figure 4.6a). Hence, the comparison with the previously used monolithic design of the LLC shows that using the component-oriented design reduces the configuration time by 87%. This amount of reduction confirms the linear correlation between the amount of commands and configuration time as mentioned in Section 3.8.3. Likewise measurements concerning the configuration of the small FB network in the Digi Connect ME controllers, providing access to the RFID modules, also show a reduction of the configuration time by 88% due to the component-oriented design (see Figure 4.6b). Apart from reducing the required configuration time, the component-oriented design of the LLC also brings a significant reduction of the memory usage by 62% in the controllers as can be seen in Figure 4.6c.

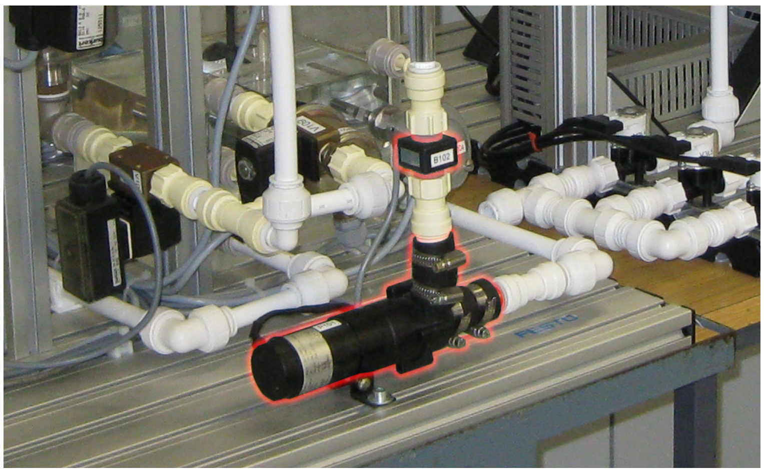

Figure 4.7: Picture highlighting the components of a laboratory batch process plant that are used for validating the configuration approach. Pump and analog flow sensor (both marked in red) compose a flow control equipment module.

4.5.2 Validation of the Approach

In order to show the feasibility of the automated configuration process for another target system, it is also applied for components of a laboratory batch process plant. A pump combined with a flow sensor (see Figure 4.7) delivers the functionality of a flow control entity, which represents a common module in the batch process domain. This flow control entity is represented as an equipment module in the hierarchical structure of the standard IEC 61512. The whole plant is controlled by one industrial controller of the type CX5010 by Beckhoff [257], which comprises a Dual-Core processor with 2 GHz and 2 GB RAM. It is employed for hosting the FORTE as LLC runtime environment.

In order to apply the approach for configuring the LLC of the flow control entity, the resource ontology (see Section 4.2) is extended with the class `FlowControl`, which is a subclass of `EquipmentModule` (see Figure 4.8). A further class is added for the pump, which is derived from the class `ControlModule`. The flow sensor is represented by the class `AnalogSensor`, which is also a subclass of `ControlModule`. This class can also be used for representing other typical analog sensors employed in the batch domain, such as temperature or pressure sensors, that deliver a single analog value. Besides, a representation of the Beckhoff controller is added as subclass of `Controller`.

The ontology is derived into the corresponding module specification file

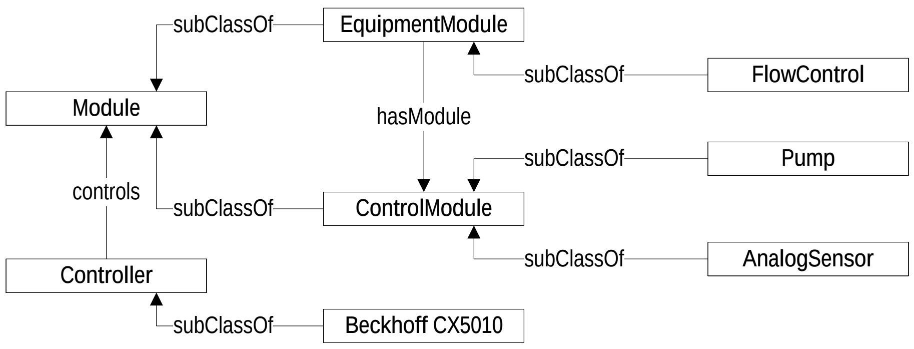

Figure 4.8: Reduced overview of the hardware-related part of the ontology describing the target components of the laboratory batch process plant.

and the instances are defined. Evidently, according IEC 61499 FBs are provided for realizing the functionality of the control modules and the flow control. Using this information, the configuration process is carried out using the rule base as described in Section 4.4. The resulting configuration commands that are sent to the Beckhoff controller are shown in Table 4.2.

Compared to the configuration runs on the Festo CPX controllers, the LLC configuration requires more time for each configuration command. As it turns out, the execution of each configuration command needs about 37 ms in average. This might be due to the much larger overhead in the Beckhoff controller as it employs a Windows environment for more operator convenience and not a small Linux system like the Festo CPX. A total time of 0.7 seconds is therefore measured for the 19 commands, which are required for configuring the complete flow control LLC.

This additional use-case shows that the approach is adoptable for different target systems. Evidently, the corresponding FBs need to follow the design as described in Section 3.5 and have to be provided for the components that shall be configured automatically. Given this prerequisite, the according components can be represented in the ontology for enabling an agent to automatically configure the LLC.

In the case of an LLC that is based on a different paradigm than the standard IEC 61499, the implementation components of the solution space need to be replaced by other corresponding software components, e.g. FBs of the standard IEC 61131. However, in this case also the configuration method of the agents has to be adapted accordingly by providing methods for creating and wiring these software components with another reconfiguration infrastructure than the FORTE.

No	Type of action	XML Command
0	Creating resource	`<Request ID="0" Action="CREATE"> <FB Name="EMB_RES" Type="EMB_RES"/> </Request>`
1	Creating new FB	`<Request ID="1" Action="CREATE"> <FB Name="FlowSensor" Type="FB_AnalogSensor"/> </Request>`
...	...	...
7	Setting parameter	`<Request ID="7" Action="WRITE"> <Connection Destination="FlowSensorSlotNrS" Source="1"/> </Request>`
...	...	...
13	Creating new connection	`<Request ID="13" Action="CREATE"> <Connection Destination="FlowControl.Sensor_In" Source="FlowSensor.Adapter_Sensor" /> </Request>`
...	...	...
18	Starting LLC	`<Request ID="18" Action="START">`

Table 4.2: Issued commands for configuring the flow control LLC.

4.6 Summary

The work presented in this chapter combines the automation agent concept with a generative programming approach for automatically configuring the agents' LLC. Thereby, the components of the manufacturing system and their relations are specified in a resource ontology, which represents the problem space of the domain model. Based on this representation, each agent's HLC utilizes configuration knowledge for performing a specification check and for carrying out the actual configuration by creating and wiring the software components (i.e. the IEC 61499 FBs) of the solution space. Thereby, an executable LLC implementation is generated for each manufacturing system component within the corresponding controller.

Performance measurements show that the automated configuration process is performed within approximately 1 second for the presented LLC applications, which is sufficient for industrial processes without hard real-time constraints [256]. The configuration time remains roughly constant even in the case of several LLC applications being configured in parallel by the automation agents with their HLC residing within one PC. However, the component-oriented design of the LLC is a prerequisite to keep the configuration time within acceptable boundaries as has been shown in the comparison with the configuration time of a monolithic LLC design.

The presented configuration concept can be adopted for different target systems by changing the contents of the ontology, respectively the problem space. Such modifications may encompass not only the physical components of the manufacturing system but also the requirements and restrictions of the software components concerning their wiring to achieve an executable LLC application. This is shown by presenting performance results of configuring the control software of not only one target system, i.e. the pallet transport system, but also of a flow controller for a laboratory batch process system.

CHAPTER 5

Reconfiguration of the Control System[1]

Reconfiguration is regarded as an important capability of a control system to dynamically adapt its behavior and functionality in the case of conditions like a changed manufacturing requirement or the occurrence of a failure [27]. Reported research in literature especially addresses reconfiguration in the context of modifying production plans and schedules as well as the resource allocation [69]. Thus, the reported approaches are mainly concerned with system-level reconfiguration but local self-reconfiguration at agent-level represents a key asset for adaptable functionality in the context of reconfigurable manufacturing [98].

The work presented in this chapter is concerned with the local self-reconfiguration of automation agents on the basis of the configuration approach described in Chapter 4. Reconfiguring the control software of an automation agent relies on the knowledge stored in its world model in conjunction with the ontology describing the system components. Such a reconfiguration process encompasses updating the world model in the HLC as well as a modification of the LLC application.

Likewise to the previous chapters, the reconfiguration of the control system is exemplified with a case study involving the already introduced pallet transport system (see Figure 3.2). Its main objective is to transport pallets between indexstations in a minimum amount of time, usually following the shortest path while avoiding broken components. 15 intersections in the center of the pallet transport system employ three-way switches (see Figure 3.3)

[1]Most contents of this chapter were previously published in [73, 78, 79].

as described in Section 3.2.1. Those offer the flexibility of four operational modes for reacting on changed conveyor directions due to modified routing paths. Section 3.5 desribes the LLC of an automation agent acting as a diverter in one specific direction, which represents one operational mode. This LLC has to be adapted in the case the intersection needs to provide a different operational mode. Consequently, a global topology reconfiguration with changed conveyor directions requires certain automation agents controlling intersections to adapt their functionality accordingly.

5.1 Reconfiguration Process

The following sections are concerned with a reconfiguration process, which is triggered due to a detected component breakdown. In the given case, a defect outgoing conveyor is detected by an automation agent of an intersection using its diagnostic mechanisms in the HLC (see Section 3.7.2). Consequently, this agent notifies a contact agent about the detected failure.

5.1.1 Determination of Reachability

After receiving the notification from the affected automation agent, the contact agent updates its representation of the system topology with the fact of the unusable route. For calculating routing paths and verifying the reachability of all destinations in the system, the contact agent relies on a shortest path algorithm, which is based on Dijkstra's algorithm [213] and was developed in the frame of a previous research project [214]. If routes are found from and to each destination, the automation agents controlling the intersections update their routing tables and store them in the LLC.

In the case of one or several unreachable destinations, the contact agent employs a change direction algorithm for determining necessary direction changes of conveyors. This algorithm uses the system representation from the ontology and derives the structure of intersections and conveyors into a matrix, but omitting the failed conveyor as a valid path. Only conveyors that are located between two intersections with three-way switches represent possible candidates for direction changes as these intersections are able to change their operational modes. As such a conveyor direction change might require additional direction changes of other conveyors, the algorithm is programmed recursively. If a solution is found, the contact agent requests from the identified conveyors to change their directions for enabling the reachability of the unreachable destinations. If no solution is found, the destinations affected by the failure remain unreachable. More details regarding the change

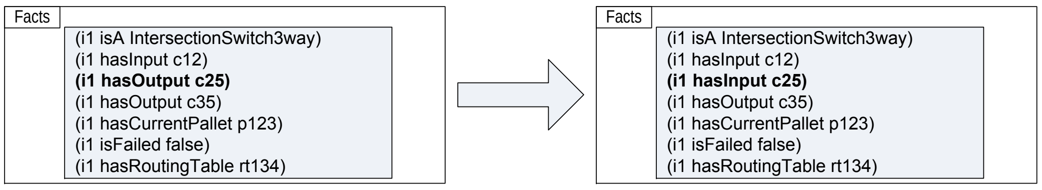
Facts
(i1 isA IntersectionSwitch3way)
(i1 hasInput c12)
(i1 hasOutput c25)
(i1 hasOutput c35)
(i1 hasCurrentPallet p123)
(i1 isFailed false)
(i1 hasRoutingTable rt134)
Facts
(i1 isA IntersectionSwitch3way)
(i1 hasInput c12)
(i1 hasInput c25)
(i1 hasOutput c35)
(i1 hasCurrentPallet p123)
(i1 isFailed false)
(i1 hasRoutingTable rt134)

Figure 5.1: Update of facts in the situation model, based on [73].

direction algorithm, which was developed in the frame of a previous research project, are presented in [30].

Hence, the automation agent of the conveyor changes its direction and informs the automation agents controlling the adjacent components (i.e. the adjacent intersections) about the direction change.

5.1.2 Updating the World Model in the High Level Control

Based on the information an automation agent controlling the intersection receives from its neighboring conveyors, it updates its world model. The resulting world model reflects the current knowledge about the state of the world. During the update, the HLC can detect inconsistencies between the world model and the new knowledge about the environment. Reconfiguring the automation agent's functionality involves several steps:

- The first step encompasses updating the facts in the situation model according to the received information. This ensures that the received knowledge is expressed correctly in its situation model. Figure 5.1 illustrates the modification of the facts in the situation model. In this case, the fact `(i1 hasOutput c25)` is replaced by the fact `(i1 hasInput c25)`.

- The second step handles the update of the activity types in the activity model (see Figure 5.2). As the fact `(i1 hasOutput c25)` no longer exists, the activity "`Routing a Pallet from c12 to (c25 or c33)`" and its derived activity types can no longer be carried out and are removed from the activity model. However, as the fact `(i1 hasInput c25)` is now existent, the automation agent is now able to perform the activity "`Routing a Pallet from (c12 or c25) to c33`". This procedure follows the HLC configuration process as described in Section 4.3 and in more detail in [207, 238].

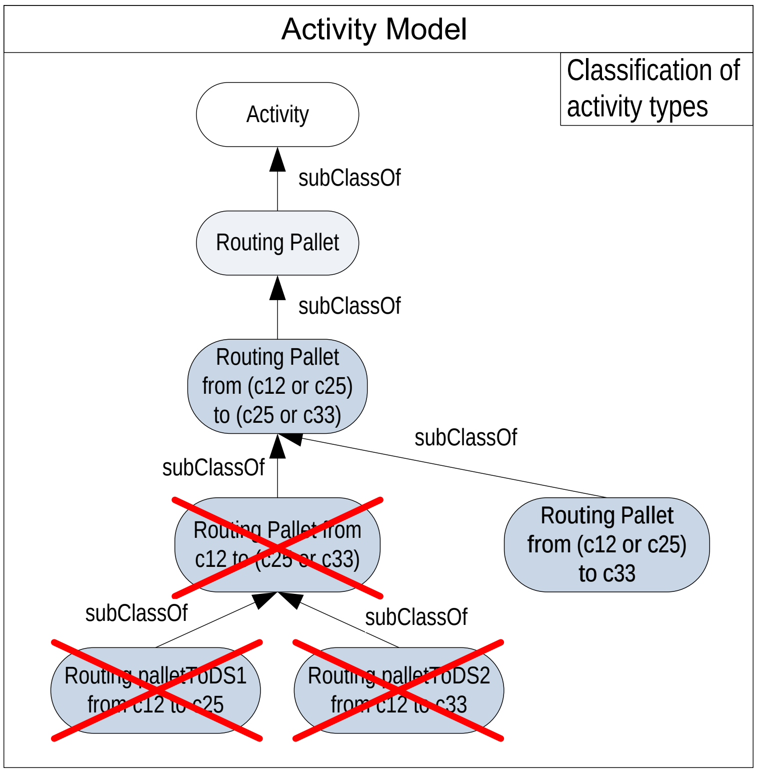
Activity Model
Classification of activity types
Activity
subClassOf
Routing Pallet
subClassOf
Routing Pallet from (c12 or c25) to (c25 or c33)
subClassOf
subClassOf
Routing Pallet from c12 to (c25 or c33)
Routing Pallet from (c12 or c25) to c33
subClassOf
subClassOf
Routing palletToDS1 from c12 to c25
Routing palletToDS2 from c12 to c33

Figure 5.2: Update of the activity types in the activity model encompassing the removal of nonexecutable activities and the addition of henceforward executable activities [73].

- The third step is concerned with the currently expected and observed activities. Kramer and Magee note that the affected part of a system needs to be in a consistent application state before the reconfiguration and that the change causes a minimum of disruption [258]. In this context, this step ensures that the reconfiguration of the LLC does not interfere with an ongoing process and therefore is not performed during an ongoing observation, of which the corresponding activity type is removed during the previous step. Figure 5.3 gives an example regarding this issue based on the activity update of the previous step, which happens at time `T`. The upper part of the figure shows the change of activities at time `T`. The lower part of the figure shows the observed activity `o1_1` of the type "`Routing p123 from c12 to c25`", which started at time $\mathtt{T}_s$`(o1_1)` and is expected to finish at time $\mathtt{T}_e$`(o1_1)`. At the current time `T` the observed activity `o1` is ongoing, which means that the expected activity `e1_1` needs to exist until the completion of `o1`.

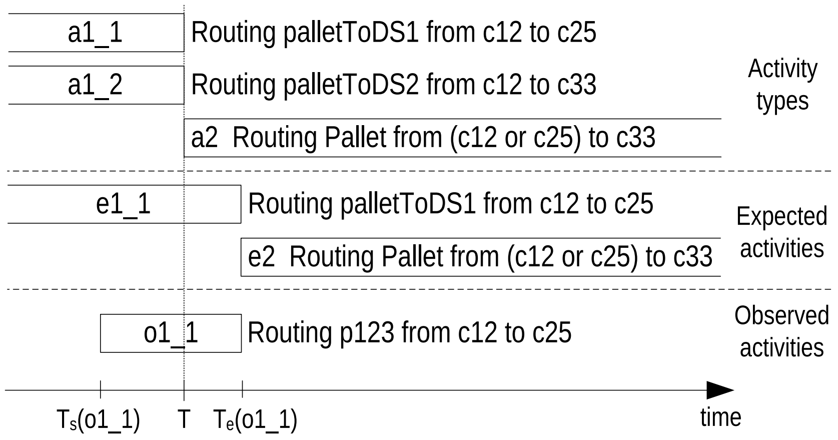
a1_1
Routing palletToDS1 from c12 to c25
a1_2
Routing palletToDS2 from c12 to c33
a2 Routing Pallet from (c12 or c25) to c33
Activity types
e1_1
Routing palletToDS1 from c12 to c25
e2 Routing Pallet from (c12 or c25) to c33
Expected activities
o1_1
Routing p123 from c12 to c25
Observed activities
Ts(o1_1)
T
Te(o1_1)
time

Figure 5.3: Determining the timing constraints for expected and observed activities, based on [73].

Consequently, the expected activity `e2`, which is derived from the activity type "`Routing Pallet from (c12 or c25) to c33`", can only start after $\texttt{T}_e\texttt{(o1\_1)}$. Thereby, the HLC ensures that the LLC reconfiguration from diverter to junction is performed only after the pallet currently inside the intersection has left.

- The fourth and final step is to infer the goal configuration to be attained on the basis of the facts in the situation model. Regarding an intersection, two types of configurations exist:

  (i) `routingAsDiverter(inCon,outCon1,outCon2)`: This configuration represents a diverter having one input conveyor (`inCon`) and two output conveyors (`outCon1` and `outCon2`). It requires an FB of the type `FB_Diverter` as pivotal functionality in the LLC. This is represented by Constraint 5.1, which sets the cardinality of input and output conveyors in relation to the required pivotal FB.

$$(\#inCon = 1) \wedge (\#outCon = 2) \rightarrow FBtype_{req} = \text{“}\texttt{FB\_Diverter}\text{”} \tag{5.1}$$

  (ii) `routingAsJunction(inCon1,inCon2,outCon)`: This configuration represents a junction with two input conveyors (`inCon1` and `inCon2`) and one output conveyor (`outCon`). It requires an FB of the type `FB_Junction` as pivotal functionality in the LLC, which is represented in Constraint 5.2.

$$(\#inCon = 2) \land (\#outCon = 1) \rightarrow FBtype_{req} = \text{“}\texttt{FB\_Junction}\text{”} \tag{5.2}$$

In the considered example, the automation agent has to achieve the goal configuration `routingAsJunction(c12,c25,c33)`.

After updating the world model, the agent determines the operations to perform in order to constitute this configuration in its LLC, which is explained in the following section.

5.1.3 Reconfiguration of the Low Level Control

Figure 5.4 depicts the workflow of the LLC reconfiguration process, which is related to the configuration process as described in Chapter 4. The current LLC application is provided as knowledge in the ontology representing the problem space. Likewise to the configuration process, the solution space provides the software components, i.e. the IEC 61499 FBs. The configuration knowledge differs as it incorporates constraints related to the components of the solution space as well as an additional set of rules, which are based on the principles as explained in Section 4.4. Both the constraints as well as the additional rules are explained in the following.

As first step the agent infers if its pivotal FB corresponds to the requirement of the goal configuration. This is done by comparing the configuration's FB requirement, expressed on the left side in Constraint 5.3 (obtained from Constraint 5.1 or 5.2), with the pivotal FB i currently represented in the ontology. FB i can be identified as it is the only FB representing an equipment module (right side of Constraint 5.3).

$$FBtype_{req} \rightarrow FBtype_i | \exists i \in I | i = InstOf(EquipmentModule) \tag{5.3}$$

In the considered example, the required FB is of the type “`FB_Junction`” (Equation 5.4).

$$FBtype_{req} = \text{“}\texttt{FB\_Junction}\text{”} \tag{5.4}$$

However, the current pivotal FB, as represented in the ontology, is of the type “`FB_Diverter`” (Equation 5.5).

$$FBtype_i = \text{“}\texttt{FB\_Diverter}\text{”} \tag{5.5}$$

Consequently, Constraint 5.3 is not satisfied and the pivotal FB needs to be replaced.

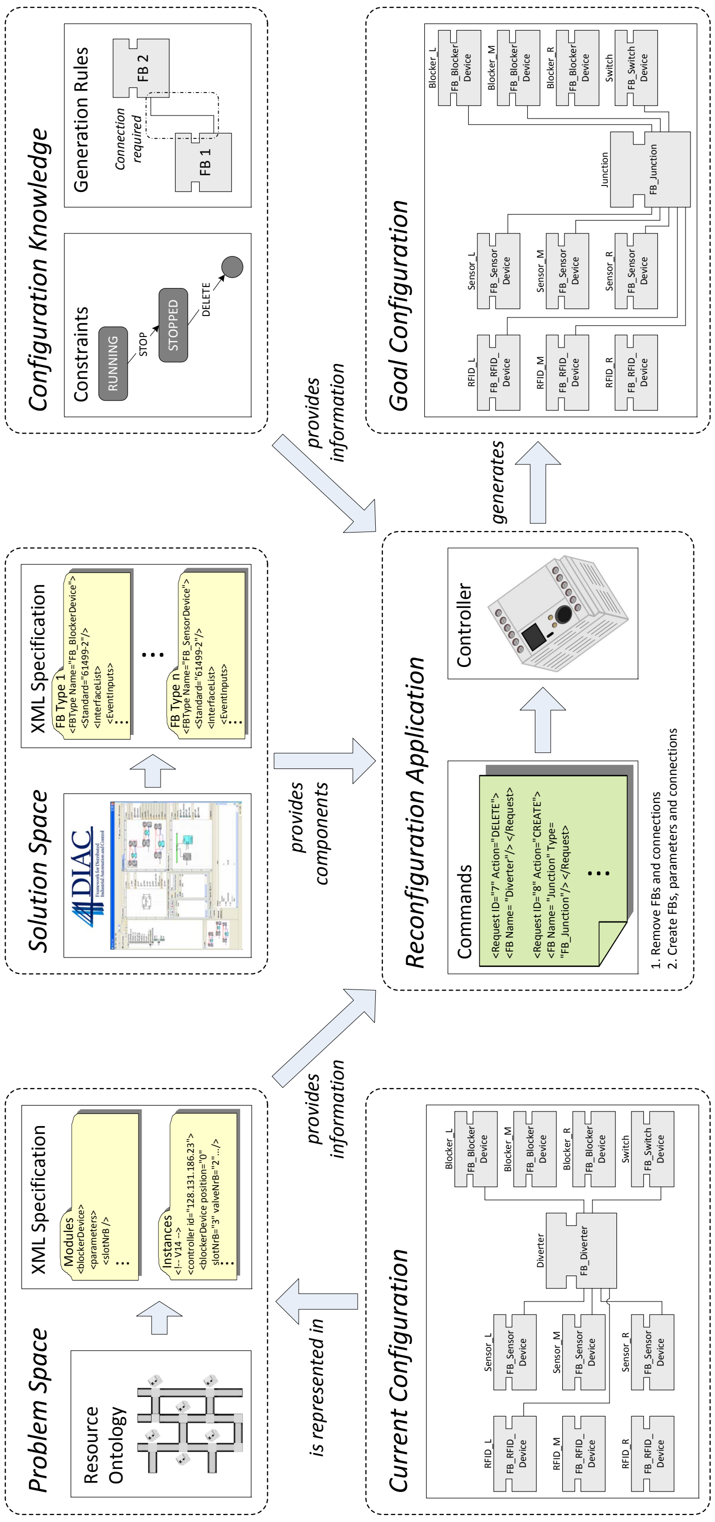

Figure 5.4: Workflow of the reconfiguration process to achieve an LLC application according to the goal configuration.

The further steps for deleting the current pivotal FB are determined using constraints for the configuration knowledge that are specific for the solution space. The LLC is based on the standard IEC 61499 and consequently the configuration knowledge encompasses Constraint 5.6, which is derived from the operational state machine of an FB [173]. There it is specified that for the deletion of an FB, it needs to be in the operational status "`stopped`".

$$DeleteFB(FBname_i) \rightarrow Status_i = \text{“}\texttt{stopped}\text{”} \tag{5.6}$$

Apart from the standard, also the implementation of the LLC runtime environment, i.e. the FORTE, has a constraint regarding the deletion of an FB. As the FORTE does not remove connections of a deleted FB automatically, they need to be removed prior to the FBs deletion to ensure system consistency. Having no connections of its nodes to any other FBs' nodes prior to its deletion is expressed by Constraint 5.7.

$$DeleteFB(FBname_i) \rightarrow \forall x \in FBIO_i | \nexists Connection(x, FBIO_n) \tag{5.7}$$

Constraints 5.6 and 5.7 represent configuration knowledge specific for the solution space in order to define the logical order of operations for deleting an FB.

The pivotal FB i can be identified as it is the only FB representing an equipment module (see left side of Rule 5.8). To make this FB deletable according to Constraint 5.6, its operational state needs to be changed from "running" to "stopped" (see right side of Rule 5.8).

$$\exists i \in I | i = InstOf(EquipmentModule) \rightarrow StopFB(FBname_i) \tag{5.8}$$

Any previously set up parameters do not require to be removed before the deletion of an FB. As stated above in Constraint 5.7, its connections to the other FBs need to be removed before it can be deleted. In order to do so, the connections, which have been previously set up with this FB, need to be determined and removed using Rule 5.9 on the basis of the agent's ontology.

$$\begin{gathered}(\exists i \in I | i = InstOf(EquipmentModule)) \wedge (\forall((x \in InstOf(modreq_i),\\ y \in InstOf(ModProv)) \cup (x \in InstOf(modprov_i), y \in InstOf(ModReq)))\\ |(Name(x) = Name(y)) \wedge Restrictions) \rightarrow DeleteConnection(x, y)\end{gathered} \tag{5.9}$$

Consequently, after the deletion of the connections the FB itself can be removed from the LLC (Rule 5.10).

$$\exists i \in I | i = \mathit{InstOf}(\mathit{EquipmentModule}) \rightarrow \mathit{DeleteFB}(\mathit{FBname}_i) \tag{5.10}$$

As soon as the deletion of the no longer required FB i is finished, its representation is removed from the agent's ontology. Afterwards, a representation of the required pivotal FB j is added. Its instance in the LLC is created, parameterized and wired to the other FBs using derivatives of the Rules 4.12-4.14 of the configuration services as described in Section 4.4.2. Finally, the newly created FB j is started by issuing a start command (Rule 5.11).

$$\exists j \in I | j = \mathit{InstOf}(\mathit{EquipmentModule}) \rightarrow \mathit{StartFB}(\mathit{FBname}_j) \tag{5.11}$$

The resulting reconfiguration sequence for the given case is presented in Table 5.1. It consists of the following steps: stopping the diverter FB, disconnecting all sensor and actuator FBs, deleting the diverter FB, creating the junction FB, connecting the relevant sensors and actuators, and finally starting the junction FB.

5.1.4 Result of the Reconfiguration

Figure 5.5 shows the resulting LLC of the automation agent, which provides the functionality of a junction. `FB_Diverter` is substituted by `FB_Junction`, which incorporates the basic junction functionality and the communication interface to the HLC. Accordingly, the sensor and blocker FBs as well as the three-way-switch FB are wired with adapter connections to the pivotal FB, which is now represented by `FB_Junction`. An intersection acting as a junction has two input conveyors and therefore the RFID readers of the two incoming conveyors need to be accessed for identifying the incoming pallets. Hence, two RFID communication blocks are connected with `FB_Junction` in contrast to the case of a diverter, which only relies on one RFID reader. Likewise, the two FBs accessing the blockers at the input conveyors are wired to the pivotal FB.

For the presented reconfiguration process of the intersection, the following basic reconfiguration services, as presented in Section 3.8.1, are used: structural services and execution control services. The modifications of the adapter connections in the FB network as well as the actual substitution of `FB_Diverter` by `FB_Junction` are performed by using the corresponding structural services. In order to stop the functionality of the diverter and to initialize and start the functionality of the junction, the according execution control services are utilized.

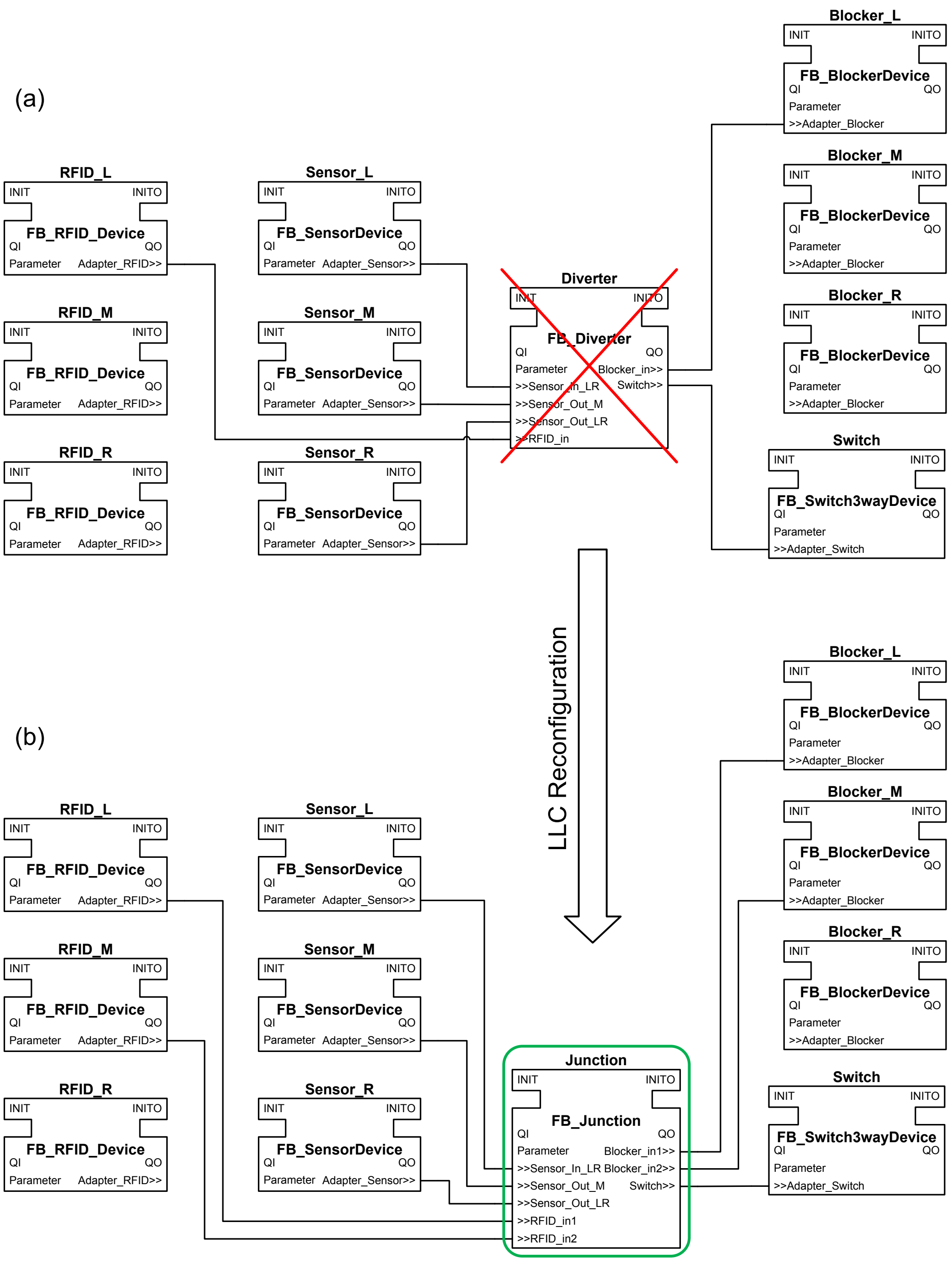

Figure 5.5: Original LLC providing the functionality of a diverter (a) and reconfigured LLC realizing the functionality of a junction (b), based on [73].

No	Type of action	XML Command
0	Stop diverter FB	`<Request ID="1" Action="STOP"> <FB Name="Diverter" Type=""/> </Request>`
1	Delete adapter connection from left sensor FB to diverter FB	`<Request ID="2" Action="DELETE"> <Connection Source="Sensor_L.data" Destination="Diverter.Sensor_in"/> </Request>`
2-6	Deletion of further adapter connections to diverter FB	...
7	Delete diverter FB	`<Request ID="8" Action="DELETE"> <FB Name="Diverter"/> </Request>`
8	Create junction FB	`<Request ID="9" Action="CREATE"> <FB Name="Junction" Type="FB_Junction"/> </Request>`
9	Create adapter connection from left sensor FB to junction FB	`<Request ID="10" Action="CREATE"> <Connection Source="Sensor_L.data" Destination="Junction.Sensor_in1"/> </Request>`
10-16	Creation of further adapter connections to junction FB	...
17	Start junction	`<Request ID="18" Action="START"> <FB Name="Junction" Type=""/> </Request>`

Table 5.1: Reconfiguration sequence for switching from diverter to junction, based on [73].

5.2 Evaluation of the Approach

Simulation experiments have shown that the application of reconfiguration mechanisms are beneficial for the pallet transport system's performance in case of component failures and resulting unavailable destinations and workstations [209, 259]. As these were pure simulation experiments, they did not rely on actually implemented local reconfiguration mechanisms. On the contrary, this section presents experiments carried out on the real pallet transport system. In order to investigate the advantages of the system reconfiguration using the local reconfiguration of the control software in the case of a detected failure, a set of test cases is evaluated. Thus, this involves the usage of the reconfiguration process as described in Section 5.1.

5.2.1 Test Cases

The target system (see Section 3.2) encompasses indexstations, which are located in the 3 outer loops of the system. The indexstations represent the locations at which pallets or their load can be manipulated by workstations. Consequently, one indexstation of each outer loop is used in this evaluation as possible target destination.

Due to their role to connect different parts of a production system and to carry and route goods and materials, a transportation system represent a production system's backbone. Two factors influence the efficiency (i.e. the throughput) of such a system [260]:

- Transport time: The time to transport material is a significant factor, especially when the transportation times between machines are considerably longer than the machine processing times. In this context, delay times due to queues have a large impact on the transport duration.
- Machine workload: Balancing machine workload is of importance for avoiding bottlenecks and thus queues and delay times.

The process flow for the performed tests consists of two process steps that involve transportation between indexstations. At first the pallets start at a workstation at indexstation D1 where they have to be processed for a period of 10 seconds. As a second step they need to be processed at another workstation at a different indexstation `D2` for a longer period than at `D1`, which is 20 seconds. In order to avoid a bottleneck at this second workstation, a redundant workstation is used at indexstation `D3` within the testbed. Hence, each of the redundant two workstations only needs to process half of the pallets in the system. After the second step, each pallet shall return to the workstation at `D1` to go through another process cycle. The processing times were defined in order to balance the factors of transport times and balanced machine workload. For instance previous tests with halved processing times showed no necessity for employing a redundant workstation for the second process step, which therefore limited the significance of the machine workload factor for the throughput in relation to the transport time. Consequently, this process flow encompasses the factors transport time as well as balanced machine workload. Figure 5.6a depicts the initial paths calculated by the contact agent for routing the pallets from `D1` to `D2` (marked green) and from `D1` to `D3` (marked red) as well as back to `D1` (marked shaded green and red).

For 10 pallets, the size of the testbed is large enough so that the pallets are evenly distributed on the paths between the indexstations for ensuring a high throughput with the given processing times. The performance of the system is evaluated using the following three test cases:

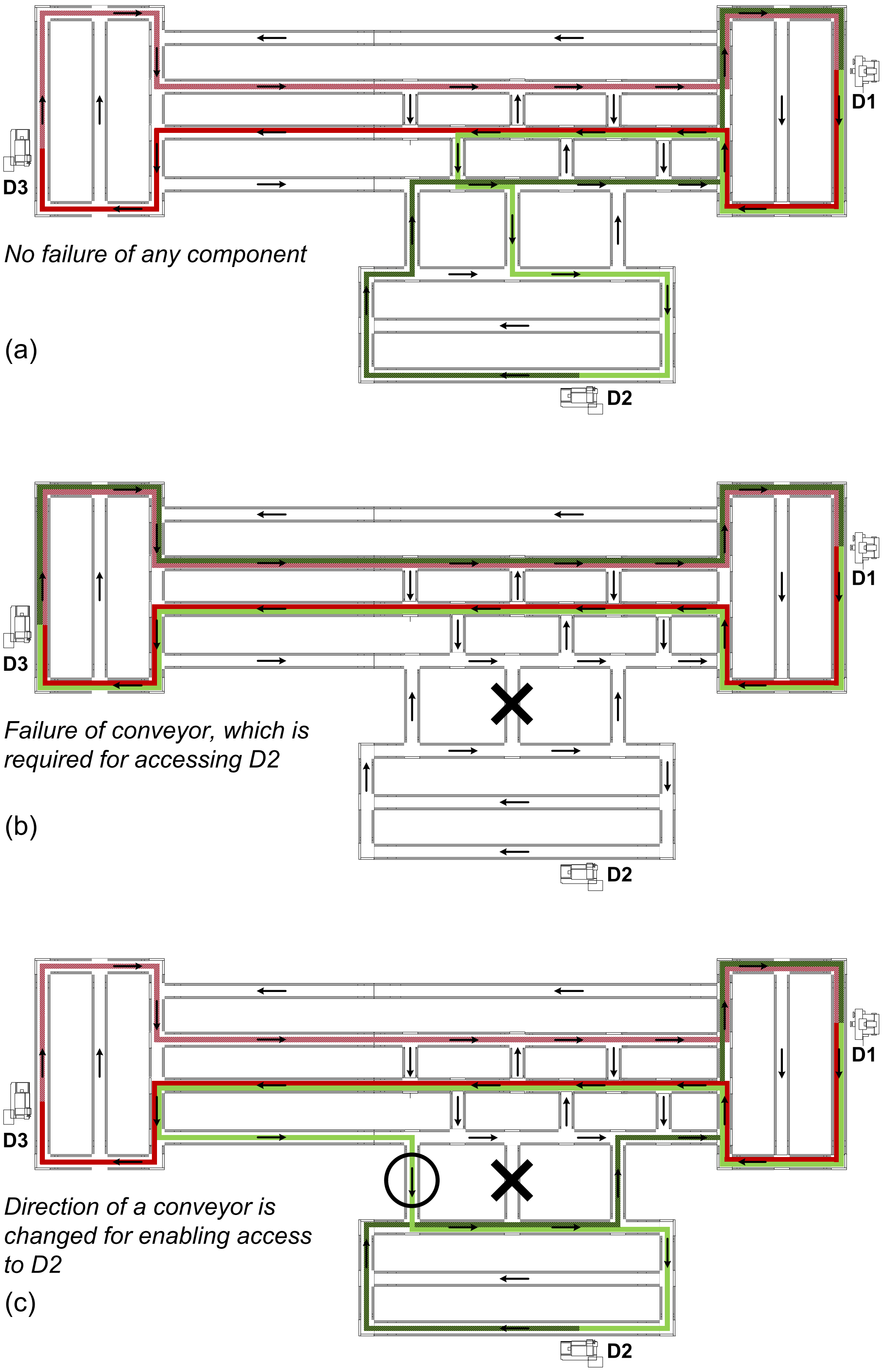
D1
D2
D3
No failure of any component
(a)
D1
D2
D3
Failure of conveyor, which is required for accessing D2
(b)
D1
D2
D3
Direction of a conveyor is changed for enabling access to D2
(c)

Figure 5.6: Paths through the pallet transport system in the failure free case (a) as well as in the cases of a conveyor failure without (b) and with (c) reconfiguration mechanisms available [78].

(a) Test case without conveyor failures: The transport system is in normal operation and all destinations are reachable.

(b) Test case, when one critical conveyor fails: A specific conveyor fails that is required for reaching indexstation `D2`. For this test case, the reconfiguration mechanisms are turned off in the control software. Hence, all pallets need to be sent to `D3` after being processed at `D1` in order to achieve a complete process cycle.

(c) Test case, when one critical conveyor fails but a resolving solution can be achieved by the system: The same conveyor as in test case (b) fails. However, the system can react on this failure by changing the direction of one conveyor and reconfiguring the control software of the adjacent intersections. Therefore, indexstation `D2` is accessible again.

According to test case (c), the following assumption is defined in the context of the performance tests: Applying the reconfiguration mechanisms enhances fault tolerance. Thus, in the case of a failure of a critical component, a throughput is achievable that is comparable to the case without failures.

In order to gain measurement data, all indexstations log the processed pallets using their ID and a time stamp. This allows the calculation of the following characteristic numbers that relate to the defined assumption:

- Average travel duration along a path from one indexstation to a particular other indexstation, which includes also waiting times at intersections and indexstations due to other pallets as well as wrongly taken routes due to false RFID readings;
- Average duration of a complete process cycle starting at indexstation `D1` until a pallet returns to `D1` for beginning the next process cycle; and
- Average throughput per hour, which represents the number of produced products per hour in the manufacturing system.

Shorter travel durations result in a shorter duration of a complete process cycle. Consequently, a shorter process cycle duration leads to a higher throughput per hour and thus a higher performance of the system. Hence, for proving the assumption made, the average duration of a complete process cycle in test case (c) should be likewise to the one in test case (a).

5.2.2 Discussion of Results

Executing the test cases delivers the following paths for the pallets calculated by the contact agent:

(a) Test case without conveyor failures: As mentioned before, Figure 5.6a shows the calculated shortest paths the pallets take. At `D1`, one half of the pallets receives `D2` as next destination while the other half is sent to `D3`. Therefore, those pallets with `D2` as next destination follow the green path and afterwards return to `D1` along the shaded green path. On the other hand, those pallets with `D3` as next destination follow along the red path to D3 to return afterwards along the shaded red path to `D1`.

(b) Test case, when one critical conveyor fails: Due to the unusable conveyor, `D2` is not reachable. Hence, all pallets are sent to `D3` after being processed at `D1`. It can be seen in Figure 5.6b that both the green and red path are similar as all pallets have to move from `D1` to `D3` and back.

(c) Test case, when one critical conveyor fails but a resolving solution is achieved by the system: Using the reconfiguration mechanisms, a conveyor's direction is switched and its adjacent intersections adapt their functionality. Thus, new routing paths are calculated and indexstation `D2` is accessible again. Figure 5.6c shows that the red path for one half of the pallets to indexstation `D3` and back is unchanged, but the green path to indexstation `D2` and back to `D1` for the other half of the pallets is different and a bit longer compared to the one in test case (a).

Figure 5.7 depicts the travel time of the pallets between the indexstations representing therefore the duration of the paths for all test cases. It can be seen that in test case (a) the returning path from `D2` and `D3` to `D1` takes in average a longer time of about 20 seconds. This can be explained with occurring traffic jams at `D1` when pallets return from `D2` and `D3` at the same time.

As indexstation `D2` is not reachable in test case (b), there is no data concerning paths to and from `D2`. The duration for the path from `D1` to `D3` is obviously significantly longer than in the other test cases. This is evident, as `D3` represents a bottleneck in this test case due to the longer processing time leading therefore to serious traffic jams. On the other hand, the return path to `D1` takes in average a slightly shorter time than in test case (a) as the pallets return from `D3` with enough time difference leading therefore to no traffic jams at `D1`.

As the path in test case (c) from `D1` to `D2` is longer, so is the travel time, which can be clearly seen in the diagram. The path from `D1` to `D3` takes roughly the same time as in test case (a). However, the returning paths from `D2` and `D3` to `D1` are slightly shorter, which might be explained by less traffic jams at `D1` due to a longer path to `D2` resulting in a better distribution of the pallets on the testbed.

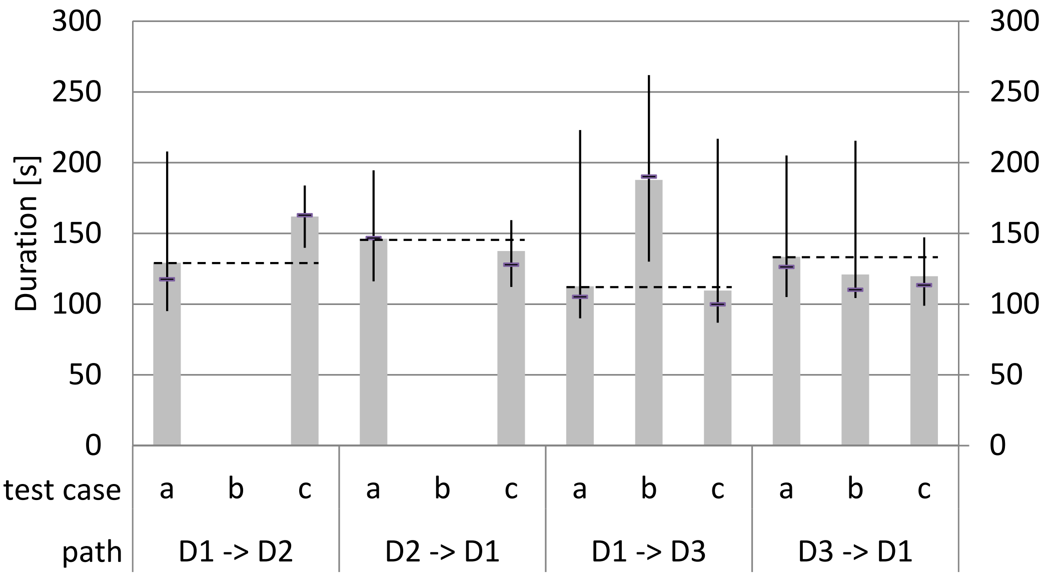

Figure 5.7: Average durations for all test cases of each path shown as grey bars (dashed lines represent reference values from the failure free test case) with the vertical black strokes depicting the spread from the minimum to the maximum value and the small horizontal black strokes representing the median value, based on [78].

Figure 5.8a shows the average duration of one process cycle for 10 pallets which encompasses travel times and processing times. While in test case (a) the processing time is in average 260.7 seconds, test case (b) reveals a processing time of 308.7 seconds resulting therefore in an increase of 18.4%. Due to this, the average throughput per hour with 10 pallets drops from 138.1 pieces to 116.6 pieces as can be seen in Figure 5.8b, which is a loss of 15.6%.

On the contrary, when the system is reconfigured by the agents to restore the reachability of `D2`, one process cycle takes in average 264.4 seconds, which is an increase of only 1.4% compared to the failure free test case (a). With 136 pieces per hour the throughput is correspondingly a bit lower compared to test case (a), but only by 1.4%. Thus, the application of the described approach significantly improves the system efficiency compared to a system without reconfiguration capabilities.

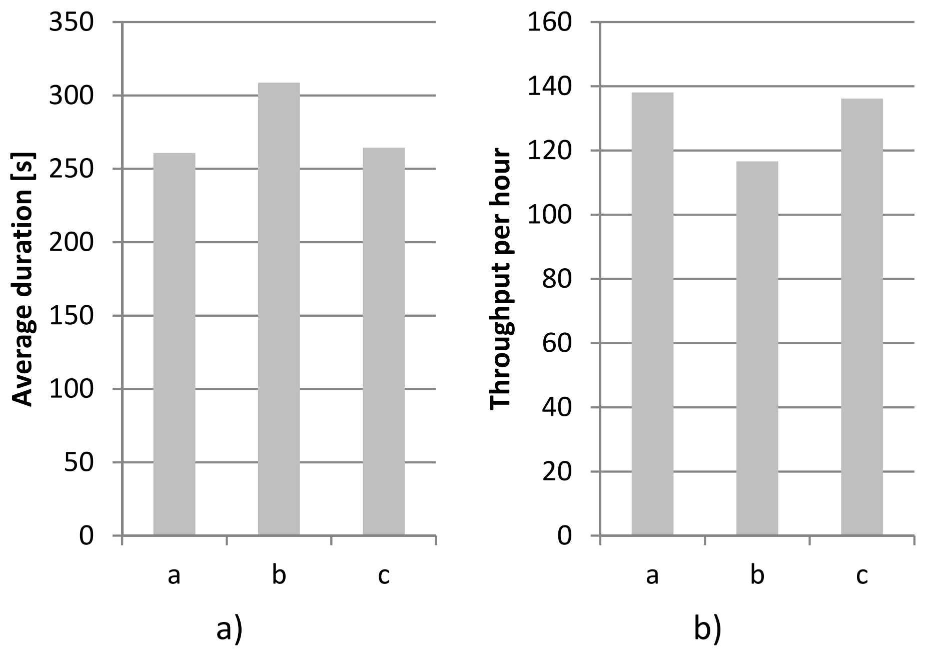

Figure 5.8: Average duration of a process cycle (a) and throughput per hour (b) for all test cases [78].

5.3 Summary

While Chapter 4 is concerned with the automated configuration process during system startup, this chapter extends this work for achieving the local self-reconfiguration of each automation agent as an autonomous process. By combining this local reconfiguration ability with global observations and modifications, the performance of the system in the context of failure tolerance is significantly improved.

Upon the acquisition of information from the component's environment (i.e. the adjacent components and their automation agents), the HLC determines the necessity of a reconfiguration process. After updating the facts in the world model of the HLC and detecting that the current control software configuration is no longer in compliance with the requirements set by the environment, the HLC initiates the reconfiguration activities. On the one hand, the activity model is updated in the HLC, which allows the automation agent to perform the according diagnostic tasks in the new configuration. On the other hand, based on the knowledge provided by the resource ontology and using the reconfiguration services, the HLC is able to modify the LLC. Once the reconfiguration process is finished, the newly configured functionalities of the LLC are initialized and can be executed.

The results of the experiments show the benefits of the MAS in the case of a conveyor breakdown of a pallet transport system. Identifying an alternative route and reconfiguring the system accordingly results in a better performance for most cases. However, if the processing times are too short and the number of pallets in the system is rather small, jams will not occur even though certain destinations of the system are unreachable making therefore such a reaction obsolete. On the contrary, in the case of an unreachable system part and longer processing times or a higher number of pallets, a direction change of a conveyor can significantly increase the performance back to the failure free case due to the regained reachability of those destinations.

CHAPTER 6

Conclusion and Outlook

6.1 Conclusion

The analysis in Chapter 1 shows that a distributed control architecture answers well to the requirements of reconfigurable manufacturing. In this context, agent technology represents a suitable approach for controlling the entities of such a distributed control system. By integrating a knowledge representation based on ontologies, the agents' domain of application can be specified explicitly, which increases the flexibility of such an approach. The comparison of the common hardware-near control software standards in Section 2 reveals that the standard IEC 61499 is well suited for reconfiguration as it supports distribution and comes with a basic reconfiguration interface.

Consequently, the aim of this thesis is the introduction and implementation of an agent architecture with reconfiguration capabilities for controlling the physical components of a manufacturing system. In regard to the contributions presented in Section 1.4, the following work is presented in this thesis:

1. A hybrid agent architecture is introduced, which is suitable for controlling the physical components of manufacturing systems—see Chapter 3.

2. A resource ontology is presented, which is suitable for the automated configuration and reconfiguration of the control software—see Chapter 4.

3. Reconfiguration mechanisms are integrated in the agent architecture, which allows an agent to adapt its functionality in regard to changes in its environment—see Chapter 5.

4. An implementation and evaluation of the approach on the "Test-bed for Distributed Holonic Control" is presented to show the feasibility and benefits of the described work—see Chapter 5.

Based on the results of this work the research questions can be answered as follows:

Research Question 1: Is it possible to realize self-contained components of a manufacturing system and incorporate diagnostic mechanisms by using agent technology?

Chapter 3 introduces the concept of the automation agent, which is structured according to a hybrid architecture composed of two software layers. The HLC layer is responsible for the control in regard to the achievement of the agent's own goals as well as for those in regard of the complete system, which encompasses the coordination with other entities. For processing information, the HLC relies on a symbolic representation of the environment in the form of a world model based on an ontology. It incorporates two parts: a situation model for storing the agent's characteristics and its relation to the environment as well as an activity model for the detection of anomalies. Both world model parts can be modified during runtime for aligning the representation of the environment with its actual state. For providing the basic functionality and accessing the sensors and actuators, the automation agent incorporates the LLC, which is realized in a component-oriented design with a network of function blocks. The LLC is based on the standard IEC 61499 and encompasses a reconfiguration infrastructure, which allows the modification of the provided functionality during runtime. Both HLC and LLC incorporate diagnostic mechanisms for detecting failures. The hybrid architecture of an automation agent with its flexible HLC knowledge base and the reconfigurable LLC constitutes a logically self-contained component for a manufacturing system.

Research Question 2: Is an ontology apt for representing the concepts about the physical system and the control software, so that an agent is able to automatically configure its hardware-near control layer?

Chapter 4 presents an approach for the automated configuration of the control software. The approach relies on a system ontology, which contains sufficient information for automatically generating an FB network in the hardware-near LLC. In order to represent the topology of the system, concepts about the employed physical components are included in the ontology, which determine their interrelations. This includes also the used controllers for hosting the control software. For the instantiation of the correct FBs representing the physical components in the LLC, corresponding linkage information is provided, which maps each physical component onto an FB type. Each FB instantiation is accompanied by the setting of parameters, which relies on provided data such as the connection point of a physical component with its controlling entity. Furthermore, for wiring the FBs to form an operating network, concepts based on requirements and restrictions are included, which determine the nodes of the FBs that need to be connected. In the presented work, only control software based on FBs can be represented in the ontology but it can be of any programming language or standard that supports such a concept. For increasing the generality and to encompass other control software programming paradigms (e.g. ladder logic or textual languages), the ontology needs to be extended with corresponding concepts. Nonetheless, a hardware-near control layer based on FBs can be automatically configured using this ontology.

Research Question 3: Is it possible to integrate reconfiguration mechanisms in both layers of a hybrid agent architecture for achieving self-reconfiguration?

Section 5.1 describes the reconfiguration process of an automation agent, which has to modify its functionality. After updating the facts in its world model according to the received information about its environment, the agent adapts its performable activities and determines a suitable LLC configuration. The reconfiguration process is based on a set of rules for removing FBs from and for adding FBs to the LLC application. Executing these rules is based on the information provided in the system ontology as well as on a set of Java methods embedded in the HLC, which are used for sending commands in XML format to the according controller. These commands are received by the external reconfiguration interface of the LLC runtime, which modifies the application accordingly. Thus, reconfiguration mechanisms are integrated in both layers of the automation agent, which is therefore capable of reconfiguring itself. Likewise to the ontology, the implemented reconfiguration mechanisms are designated only for reconfiguring control software based on FBs. Integrating additional rules for other control software programming paradigms would therefore be a meaningful extension.

Research Question 4: Is the developed approach feasible and beneficial on a real (laboratory) system?

The performance measurements presented in Section 4.5 show that the automated configuration process is performed sufficiently fast for industrial processes without hard real-time constraints. For the given implementation, experiments indicate a linear correlation between the amount of commands and configuration time with the network latency being the dominating factor (see Section 3.8.3). Consequently, a component-oriented design of the LLC is a prerequisite to keep the configuration time within acceptable boundaries. Besides, Section 5.2 presents an evaluation of the approach on a laboratory pallet transport system. The evaluation is based on a performance comparison of three test cases: (i) test case without any conveyor failure, (ii) test case with a broken critical conveyor with no resolving means, and (iii) test case with a broken critical conveyor with the application of the presented approach. The experiment shows that the throughput of the system evidently decreases in the case of a broken critical conveyor (ii) compared to the failure free case (i). However, if the developed reconfiguration mechanisms are available when the failure is detected (iii), the system is able to adapt itself and its throughput is almost as high as in case (i). Thus, the system's tolerance to failures is increased by the developed approach.

6.2 Outlook

In order to apply the presented control software reconfiguration approach in a system with hard real-time constraints, the reconfiguration services of the agents need to be extended for being able to create an RCA within the agent's LLC. Thus, it can create an RCA without interfering with the process under control and consequently the LLC application can then be reconfigured sufficiently fast. Finally the RCA can be removed from the LLC again with no interference of the process. In this context, also the topic of transition management, which represents a research domain of its own, needs to be considered more closely to ensure a smooth transition during the reconfiguration process [159].

Furthermore, in the case of an LLC that is based on a different paradigm than IEC 61499, the implementation components of the solution space can also be replaced by other software components, such as FBs of the standard IEC 61131. However, in this case also the configuration method of the agents needs to be adapted accordingly by providing methods for creating and wiring these software components with another reconfiguration infrastructure than the FORTE. For improving the generality of the configuration approach, the concepts of the solution space and their relations could also be provided in the form of an ontology, as the one presented in [254] for representing FBs of the standard IEC 61499. As the concept of adapter connections is missing within this ontology, it would have to be extended accordingly. Thus, the solution space concepts would be described in a more declarative manner, which could ease the adoption of this configuration approach to other reconfiguration infrastructures.

In the context of diagnosis, the presented agent architecture provides the basis for diagnostic algorithms which combine information from multiple entities to deduce the cause of an anomaly. One possible case is briefly mentioned in Section 3.7.2 but this approach still needs to be validated regarding further and more complex failure cases on various manufacturing systems. Another possibility for diagnostic mechanisms is the integration of learning capabilities in the agents, e.g. by extending the agent approach with evolving data models based on the behavior of the monitored system [261].

Regarding the implementation on the pallet transport system, the presented MAS application calculates new routing tables only in the case of a detected failure, which forces the system to provide alternative paths. However, such a system's throughput performance could be enhanced also in the failure free case by applying a routing algorithm for the determination of conveyor direction changes based on the momentary location of pallets and the destinations they are heading to.

Generally, agent technology is not yet widely spread in the industry despite its potential. Apart from the commercially important factor regarding the return on investment, doubts regarding the reliability of an agent-based system cannot be ignored especially in the case of agents, which might not be under direct human supervision [262]. Reported research on the stability of MAS is mainly concerned with rather homogeneous entities and swarm theory (e.g. [263, 264]) and therefore more advanced models and means for a stability analysis might be required, which take the individual complex behaviors of agents in a heterogeneous MAS into account.

Apart from the radical paradigm shift from a controller-centric view to modularization and service orientation, which requires an appropriate education of the engineering staff [49], another factor hindering the wide application of agent technology might be that generally the integration of new technologies in the industry requires according design tools for supporting their applicability [265]. For instance the agent's HLC models in the presented work are designed and implemented manually in the Java classes but they could be created with tools such as Protégé [266] with a suitable code parser. Also the XML files incorporating the resource ontology could be exported from a commercial tool such as EPLAN Electric P8 [267] if an according XML export would be provided. Consequently, agent-based solutions realized as easily configurable black boxes with transparent interfaces are advisable [49]. This includes the deployment of the complete automation agents onto the controllers located in the field level, including also the resource intensive HLC of each agent, which should become more feasible due to the recent advances in the controller technology and the increasing computational power.

6.3 Epilogue

In regard of the sheer endless amount of available publications, it is a proven fact that new knowledge is likely based on existing approaches—evidently, there are some references included in this thesis. Hence, likewise to the diploma thesis finished by the author a few years ago, also this work shall be concluded with an appropriate quotation:

"The dwarf sees farther than the giant,
when he has the giant's shoulder to mount on." [268]

Bibliography

[1] N. N. Chokshi and D. C. McFarlane. *A Distributed Coordination Approach to Reconfigurable Process Control*. Springer Series in Advanced Manufacturing. London, UK: Springer, 2008. ISBN: 978-1-84800-059-9.

[2] V. Mařík, M. Fletcher, and M. Pěchouček. „Holons & Agents: Recent Developments and Mutual Impacts“. In: *Multi-Agent Systems and Applications II*. Vol. 2322. Lecture Notes in Computer Science. Springer, 2002, pp. 89–106.

[3] G. Da Silveira, D. Borenstein, and F. S. Fogliatto. „Mass customization: Literature review and research directions“. In: *International Journal of Production Economics* 72.1 (2001), pp. 1–13.

[4] S. Davis. *Future Perfect: Tenth Anniversary Edition*. Basic Books, 1997. ISBN: 978-0201327953.

[5] B. J. Pine II. *Mass Customization: The New Frontier in Business Competition*. Harvard Business School Press, 1993. ISBN: 0-87584-372-7.

[6] F. S. Fogliatto, G. Da Silveira, and D. Borenstein. „The mass customization decade: An updated review of the literature“. In: *International Journal of Production Economics* 138.1 (2012), pp. 14–25.

[7] M. Polke. *Process Control Engineering*. VCH, 1994. ISBN: 3-527-28689-6.

[8] B. Favre-Bulle. *Automatisierung komplexer Industrieprozesse, Systeme, Verfahren und Informationsmanagement*. Vienna, Austria: Springer, 2004. ISBN: 3-211-21194-2.

[9] J. M. K. Szmrecsányi. „On the Historicity of the Second Industrial Revolution and the Applicability of its Concept to the Russian Economy Before 1917“. In: *Economies et Sociétés* 62 (2008), pp. 619–646.

[10] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren. „Reconfigurable manufacturing systems: Key to future manufacturing“. In: *Journal of Intelligent Manufacturing* 11.4 (2000), pp. 403–419.

[11] D. A. Hounshell. *From the American System to Mass Production, 1800-1932: The Development of Manufacturing Technology in the United States*. The Johns Hopkins University Press, 1985. ISBN: 978-0801831584.

[12] H. A. ElMaraghy. „Flexible and reconfigurable manufacturing systems paradigms“. In: *International Journal of Flexible Manufacturing Systems* 17.4 (2006), pp. 261–276.

[13] R. Shah and P. T. Ward. „Lean manufacturing: context, practice bundles, and performance“. In: *Journal of Operations Management* 21.2 (2003), pp. 129–149.

[14] Y. Koren and M. Shpitalni. „Design of reconfigurable manufacturing systems“. In: *Journal of Manufacturing Systems* 29.4 (2010), pp. 130–141.

[15] W. Covanich, D. McFarlane, and A. M. Farid. „Guidelines for Evaluating the Ease of Reconfiguration of Manufacturing Systems“. In: *Proceedings of the 6th IEEE International Conference on Industrial Informatics*. 2008, pp. 1214–1219.

[16] S. Wang and K. G. Shin. „Constructing Reconfigurable Software for Machine Control Systems“. In: *IEEE Transactions on Robotics and Automation* 18.4 (2002), pp. 475–486.

[17] J. Kramer and J. Magee. „Self-Managed Systems: an Architectural Challenge“. In: *Future of Software Engineering*. 2007, pp. 259–268.

[18] D. Gouyon, J.-F. Pétin, and G. Morel. „A product-driven reconfigurable control for shop floor systems“. In: *Studies in Informatics and Control* 16 (2007).

[19] P. Kopacek and B. Kopacek. „Intelligent, flexible disassembly“. In: *International Journal of Advanced Manufacturing Technology* 30.5–6 (2006), pp. 554–560.

[20] E. Muhl, P. Charpentier, and F. Chaxel. „Optimization of physical flows in an automotive manufacturing plant: some experiment and issues“. In: *Engineering Application of Artificial Intelligence* 16.4 (2003), pp. 293–305.

[21] J. Padayachee and G. Bright. „Modular machine tools: Design and barriers to industrial implementation“. In: *Journal of Manufacturing Systems* 31.2 (2012), pp. 92–102.

[22] Z. M. Bi, S. Y. T. Lang, M. Verner, and P. Orban. „Development of reconfigurable machines“. In: *International Journal of Advanced Manufacturing Technology* 39.11–12 (2008), pp. 1227–1251.

[23] G. Reinhart, S. Krug, S. Hüttner, Z. Mari, F. Riedelbauch, and M. Schlögel. „Automatic configuration (Plug & Produce) of Industrial Ethernet Networks“. In: *Proceedings of the 9th IEEE International Conference on Industry Applications.* 2010, pp. 1–6.

[24] P. A. Vicaire, E. Hoque, Z. Xie, and J. A. Stankovic. „Bundle: A Group-Based Programming Abstraction for Cyber-Physical Systems“. In: *IEEE Transactions on Industrial Informatics* 8.2 (2012), pp. 379–392.

[25] M. Lanthaler. *Self-healing wireless sensor networks.* URL: http://www.cs.helsinki.fi/u/niklande/opetus/SemK07/paper/lanthaler.pdf. Access date October 2013.

[26] L.-C. Wang and S.-K. Lin. „A multi-agent based agile manufacturing planning and control system“. In: *Computers & Industrial Engineering* 57.2 (2009), pp. 620–640.

[27] P. Vrba and V. Mařík. „Capabilities of Dynamic Reconfiguration of Multiagent-Based Industrial Control Systems“. In: *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 40.2 (2010), pp. 213–223.

[28] P. Tichý, P. Šlechta, R. J. Staron, F. P. Maturana, and K. H. Hall. „Multiagent Technology for Fault Tolerance and Flexible Control“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 36.5 (2006), pp. 700–705.

[29] Y. Zhang and J. Jiang. „Bibliographical review on reconfigurable fault-tolerant control systems“. In: *Annual Reviews in Control* 32.2 (2008), pp. 229–252.

[30] G. Koppensteiner, M. Merdan, I. Hegny, and G. Weidenhausen. „A Change-Direction-Algorithm for distributed Multi-Agent Transport Systems“. In: *Proceedings of the IEEE International Conference on Mechatronics and Automation.* 2008, pp. 1030–1034.

[31] J. H. Christensen. „Holonic manufacturing systems: initial architecture and standards directions“. In: *Proceedings of the 1st European Conference on Holonic Manufacturing Systems.* 1994.

[32] R. W. Brennan. „Toward Real-Time Distributed Intelligent Control: A Survey of Research Themes and Applications“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 37.5 (2007), pp. 744–765.

[33] M. Seitz. *Speicherprogrammierbare Steuerungen für die Fabrik- und Prozessautomation.* Munich, Germany: Carl Hanser Verlag, 2012. ISBN: 978-3-446-43325-0.

[34] W. Bolton. *Programmable Logic Controllers.* Oxford, UK: Newnes, 2009. ISBN: 978-1-85617-751-1.

[35] J. Heidepriem. *Prozessinformatik 2.* Oldenbourg, 2004. ISBN: 3-486-27050-8.

[36] C. Ou-Yang and J. S. Lin. „The development of a hybrid hierarchical/heterarchical shop floor control system applying bidding method in job dispatching“. In: *Robotics and Computer-Integrated Manufacturing* 14.3 (1998), pp. 199–217.

[37] P. Leitao. „An Agile and Adaptive Holonic Architecture for Manufacturing Control“. PhD thesis. University of Porto, Portugal, 2004.

[38] P. Timmermans and L. Szakalc. „A comparative experiment of control architectures“. In: *Computers in Industry* 28.3 (1996), pp. 185–193.

[39] S. Bussmann and K. Schild. „An Agent-based Approach to the Control of Flexible Production Systems“. In: *Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation.* 2001, pp. 481–488.

[40] P. Leitao, J. Barata, L. Camarinha-Matos, and R. Boissier. „Trends in agile and cooperative manufacturing“. In: *Proceedings of the Low Cost Automation Symposium.* 2001, pp. 156–165.

[41] S. S. Heragu, R. J. Graves, B.-I. Kim, and A. St. Onge. „Intelligent Agent Based Framework for Manufacturing Systems Control“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans* 32.5 (2002), pp. 560–573.

[42] L. Monostori, J. Váncza, and S. R. T. Kumara. „Agent-Based Systems for Manufacturing“. In: *CIRP Annals – Manufacturing Technology* 55.2 (2006), pp. 697–720.

[43] R. F. Babiceanu and F. F. Chen. „Development and applications of holonic manufacturing systems: a survey“. In: *Journal of Intelligent Manufacturing* 17.1 (2006), pp. 111–131.

[44] P. R. Wurman, R. D'Andrea, and M. Mountz. „Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses“. In: *AI Magazine* 29.1 (2008), pp. 9–19.

[45] C. E. Pereira and L. Carro. „Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control“. In: *Annual Reviews in Control* 31.1 (2007), pp. 81–92.

[46] N. R. Jennings and S. Bussmann. „Agent-Based Control Systems: Why Are They Suited to Engineering Complex Systems?“ In: *IEEE Control Systems Magazine* 23.3 (2003), pp. 61–73.

[47] E. Börger. „High Level System Design and Analysis using Abstract State Machines“. In: *Applied Formal Methods – FM-Trends 98*. Vol. 1641. Lecture Notes in Computer Science. Springer, 1999, pp. 1–43.

[48] G. Kiczales. „Towards a new model of abstraction in software engineering“. In: *Proceedings of the 1991 International Workshop on Object Orientation in Operating Systems*. 1991, pp. 127–128.

[49] P. Leitao, V. Mařík, and P. Vrba. „Past, Present, and Future of Industrial Agent Applications“. In: *IEEE Transactions on Industrial Informatics* 9.4 (2013), pp. 2360–2372.

[50] L. Ribeiro, J. Barata, and P. Mendes. „MAS and SOA: Complementary Automation Paradigms“. In: *Innovation in Manufacturing Networks*. Vol. 266. IFIP Advances in Information and Communication Technology. Springer, 2008, pp. 259–268.

[51] M. Wooldridge. „Intelligent Agents: The Key Concepts“. In: *Multi-Agent Systems and Applications II*. Vol. 2322. Lecture Notes in Computer Science. Springer, 2002, pp. 3–43.

[52] N. R. Jennings, K. Sycara, and M. Wooldridge. „A Roadmap of Agent Research and Development“. In: *Autonomous Agents and Multi-Agent Systems* 1.1 (1998), pp. 7–38.

[53] W. A. Gruver, D. B. Kotak, E. H. van Leeuwen, and D. Norrie. „Holonic Manufacturing Systems: Phase II“. In: *Holonic and Multi-Agent Systems for Manufacturing.* Vol. 2744. Lecture Notes in Computer Science. Springer, 2003, pp. 1–14.

[54] A. Koestler. *The Ghost in the Machine.* 1967.

[55] J. H. Christensen. „HMS/FB architecture and its implementation“. In: *Agent Based Manufacturing: Advances in the Holonic Approach.* Ed. by S. M. Deen. Springer, 2003, pp. 53–87. ISBN: 3-540-44069-0.

[56] N. Komoda. „Service Oriented Architecture (SOA) in Industrial Systems“. In: *Proceedings of the 4th IEEE International Conference on Industrial Informatics.* 2006, pp. 1–5.

[57] A. Ricci, C. Buda, and N. Zaghini. „An Agent-Oriented Programming Model for SOA & Web Services“. In: *Proceedings of the 5th IEEE International Conference on Industrial Informatics.* 2006, pp. 1059–1064.

[58] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer. „Service-Oriented Computing Research Roadmap“. In: *International Journal of Cooperative Information Systems* 17.2 (2008), pp. 223–255.

[59] M. Abel and P. Klemm. „Flexible SOA Based Platform for Research on Start-Up Procedures for Reconfigurable Production Machines“. In: *Advances in Sustainable and Competitive Manufacturing Systems.* Lecture Notes in Mechanical Engineering. Springer, 2013, pp. 489–501.

[60] N. Kaur, C. S. McLeod, A. Jain, R. Harrison, B. Ahmad, A. W. Colombo, and J. Delsing. „Design and simulation of a SOA-based system of systems for automation in the residential sector“. In: *Proceedings of the IEEE International Conference on Industrial Technology.* 2013, pp. 1976–1981.

[61] P. R. Wurman, R. D'Andrea, and M. Mountz. „Research Directions for Service-Oriented Multiagent Systems“. In: *IEEE Internet Computing* 9.6 (2005), pp. 65–70.

[62] O. Etzioni. „Moving Up the Information Food Chain: Deploying Softbots on the World-Wide Web“. In: *AI Magazine* 18.2 (1997), pp. 11–18.

[63] M. Wooldridge. „Agent-based software engineering“. In: *IEE Proceedings on Software Engineering* 144.1 (1997), pp. 26–37.

[64] A. H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence.* New York, USA: Morgan Kaufmann, 1998. ISBN: 978-0-93461-363-7.

[65] F. Zambonelli, N. R. Jennings, A. Omicini, and M. J. Wooldridge. „Agent-Oriented Software Engineering for Internet Applications“. In: *Coordination of Internet Agents: Models, Technologies and Applications.* Ed. by A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf. Springer, 2001, pp. 326–346. ISBN: 978-3-642-07488-2.

[66] J. L. M. Lastra and I. M. Delamer. „Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap“. In: *IEEE Transactions on Industrial Informatics* 2.1 (2006), pp. 1–11.

[67] D. Weyns, H. Van Dyke Parunak, F. Michel, T. Holvoet, and J. Ferber. „Environments for Multiagent Systems State-of-the-Art and Research Challenges“. In: *Environments for Multi-Agent Systems.* Vol. 3374. Lecture Notes in Computer Science. Springer, 2005, pp. 1–47.

[68] S. Borgo and P. Leitao. „The Role of Foundational Ontologies in Manufacturing Domain Applications“. In: *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE.* Vol. 3290. Lecture Notes in Computer Science. Springer, 2004, pp. 670–688.

[69] W. Shen, L. Wang, and Q. Hao. „Agent-Based Distributed Manufacturing Process Planning and Scheduling: A State-of-the-Art Survey“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 36.4 (2006), pp. 563–577.

[70] P. Leitao. „Agent-based distributed manufacturing control: A state-of-the-art survey“. In: *Engineering Applications of Artificial Intelligence* 22.7 (2009), pp. 979–991.

[71] V. Mařík and D. McFarlane. „Industrial Adoption of Agent-Based Technologies“. In: *IEEE Intelligent Systems* 20.1 (2005), pp. 27–35.

[72] M. G. Mehrabi, A. G. Ulsoy, Y. Koren, and P. Heytler. „Trends and perspectives in flexible and reconfigurable manufacturing systems“. In: *Journal of Intelligent Manufacturing* 13.2 (2002), pp. 135–146.

[73] W. Lepuschitz, A. Zoitl, M. Vallée, and M. Merdan. „Towards Self-Reconfiguration of Manufacturing Systems using Automation Agents“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 41.1 (2011), pp. 52–69.

[74] W. Lepuschitz, M. Vallée, M. Merdan, P. Vrba, and J. Resch. „Integration of a Heterogeneous Low Level Control in a Multi-Agent System for the Manufacturing Domain“. In: *Proceedings of the 14th IEEE International Conference on Emerging Technologies and Factory Automation*. 2009, pp. 1–8.

[75] W. Lepuschitz, V. Jirkovsky, P. Kadera, and P. Vrba. „A Multi-Layer Approach for Failure Detection in a Manufacturing System based on Automation Agents“. In: *Proceedings of the 9th International Conference on Information Technology: New Generations*. 2012, pp. 1–6.

[76] W. Lepuschitz, A. Zoitl, and M. Merdan. „Ontology-Driven Automated Software Configuration for Manufacturing System Components“. In: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. 2011, pp. 427–433.

[77] W. Lepuschitz, M. Vallée, A. Zoitl, and M. Merdan. „Online Reconfiguration of the Low Level Control for Automation Agents“. In: *Proceedings of the 36th Annual Conference of the IEEE Industrial Electronics Society*. 2010, pp. 1359–1364.

[78] W. Lepuschitz, B. Groessing, M. Merdan, and G. Schitter. „Evaluation of a Multi-Agent Approach for a Real Transportation System“. In: *Proceedings of the IEEE International Conference on Industrial Technology*. 2013, pp. 1273–1278.

[79] W. Lepuschitz, B. Groessing, and M. Merdan. „Automation Agents for Controlling the Physical Components of a Transportation System“. In: *Industrial Agents: Emerging Applications of Software Agents in Industry*. Ed. by S. Karnouskos P. Leitao. Elsevier, 2015, pp. 323–339. ISBN: 978-0-12-800341-1.

[80] R. Bénabou. „Tax and Education Policy in a Heterogeneous-Agent Economy: What Levels of Redistribution Maximize Growth and Efficiency?“ In: *Econometrica* 70.2 (2002), pp. 481–517.

[81] C. Shilling. „The Undersocialised Conception of the Embodied Agent in Modern Sociology“. In: *Sociology* 31.4 (1997), pp. 737–754.

[82] M. Wooldridge and N. R. Jennings. „Intelligent agents: theory and practice“. In: *Knowledge Engineering Review* 10.2 (1995), pp. 115–152.

[83] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, Second Edition*. Prentice Hall, 2003. ISBN: 81-297-0041-7.

[84] W. Shen, D. H. Norrie, and J.-P. A. Barthès. *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*. London, UK: Taylor & Francis, 2001. ISBN: 0-7484-0882-7.

[85] T. Finin, R. Fritzson, D. McKay, and R. McEntire. „KQML as an Agent Communication Language“. In: *Proceedings of the 3^{rd} International Conference on Information and Knowledge Management*. 1994, pp. 456–463.

[86] Foundation for Intelligent Physical Agents. *FIPA Agent Communication Language Specification*. URL: http://www.fipa.org/repository/aclspecs.html. Access date March 2014.

[87] Y. Labrou. „Standardizing Agent Communication“. In: *Multi-Agent Systems and Applications*. Vol. 2086. Lecture Notes in Computer Science. Springer, 2001, pp. 74–97.

[88] A. L. Symeonidisa, K. C. Chatzidimitriou, I. N. Athanasiadis, and P. A. Mitkas. „Data mining for agent reasoning: A synergy for training intelligent agents“. In: *Engineering Applications of Artificial Intelligence* 20.8 (2007), pp. 1097–1111.

[89] P. Maes. „The agent network architecture (ANA)“. In: *ACM SIGART Bulletin* 2.4 (1991), pp. 115–120.

[90] M. Scheutz and B. Logan. „Affective vs. Deliberative Agent Control“. In: *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing*. 2001, pp. 1–10.

[91] M. Wooldridge and N. R. Jennings. „Agent Theories, Architectures, and Languages: A Survey“. In: *Intelligent Agent Systems Theoretical and Practical Issues*. Vol. 890. Lecture Notes in Computer Science. Springer, 1995, pp. 1–39.

[92] J. P. Müller. „Control Architectures for Autonomous and Interacting Agents: A Survey“. In: *Intelligent Agent Systems*. Vol. 1209. Lecture Notes in Computer Science. Springer, 1997, pp. 1–26.

[93] K. P. Sycara. „Multiagent Systems“. In: *AI Magazine* 19.2 (1998), pp. 79–92.

[94] M. R. Genesereth and S. P. Ketchpel. „Software Agents“. In: *Communications of the ACM* 37.7 (1994), pp. 48–53.

[95] W. Shen. „Distributed Manufacturing Scheduling Using Intelligent Agents“. In: *IEEE Intelligent Systems* 17.1 (2002), pp. 88–94.

[96] R. G. Smith. „The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver“. In: *IEEE Transactions on Computers* 29.12 (1980), pp. 1104–1113.

[97] H. Van Dyke Parunak and R. S. VanderBok. „Managing Emergent Behavior in Distributed Control Systems“. In: *Proceedings of the ISA Tech '97.* 1997.

[98] M. Pěchouček and V. Mařík. „Industrial deployment of multi-agent technologies: review and selected case studies“. In: *Autonomous Agents and Multi-Agent Systems* 17.3 (2008), pp. 397–431.

[99] S. Bussmann, N. R. Jennings, and M. Wooldridge. *Multiagent Systems for Manufacturing Control: A Design Methodology.* Springer Series on Agent Technology. Springer, 2004. ISBN: 978-3-540-20924-9.

[100] P. Stone and M. Velose. „Multiagent Systems: A Survey from a Machine Learning Perspective“. In: *Autonomous Robots* 8.3 (2000), pp. 345–383.

[101] M. Merdan. „Knowledge-based Multi-Agent Architecture Applied in the Assembly Domain“. PhD thesis. Vienna University of Technology, Austria, 2009.

[102] N. R. Jennings and M. Wooldridge. „Applications of Intelligent Agents“. In: *Agent Technology.* Ed. by N. R. Jennings and M. Wooldridge. Springer, 1998, pp. 3–28. ISBN: 978-3-662-03678-5.

[103] K. Hall, R. Staron, and P. Vrba. „Experience with Holonic and Agent-Based Control Systems and Their Adoption by Industry“. In: *Holonic and Multi-Agent Systems for Manufacturing.* Vol. 3593. Lecture Notes in Computer Science. Springer, 2005, pp. 1–10.

[104] A. Helsinger, M. Thome, and T. Wright. „Cougaar: a scalable, distributed multi-agent architecture“. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.* 2004, pp. 1910–1917.

[105] P. Valckenaers. „Challenges of Next Generation Manufacturing Systems“. In: *Integration of Software Specification Techniques for Applications in Engineering.* Vol. 3147. Lecture Notes in Computer Science. Springer, 2004, pp. 23–28.

[106] D. Vasko, F. P. Maturana, A. Bowles, and S. Vandenberg. „Autonomous Cooperative Factory Control“. In: *Proceedings of the 3rd Pacific Rim International Workshop on Multi-Agents: Design and Applications of Intelligent Agents.* 2000, pp. 156–169.

[107] V. Mařík, P. Vrba, K. H. Hall, and F. P. Maturana. „Rockwell Automation Agents for Manufacturing“. In: *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems*. 2005, pp. 107–113.

[108] S. Jacobi, C. Madrigal-Mora, E. León-Soto, and K. Fischer. „Agent-Steel: An agent-based Online System for the Planning and Observation of Steel Production“. In: *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems*. 2005, pp. 114–119.

[109] K. Schild and S. Bussmann. „Self-Organization in Manufacturing Operations“. In: *Communications of the ACM* 50.12 (2007), pp. 74–79.

[110] M. Obitko and V. Mařík. „Ontologies for Multi-Agent Systems in Manufacturing Domain“. In: *Proceedings of the 13th Int. Workshop on Database and Expert Systems Applications*. 2002, pp. 597–602.

[111] T. ten Berge and R. van Hezewijk. „Procedural and Declarative Knowledge“. In: *Theory & Psychology* 9.5 (1999), pp. 605–624.

[112] R. Sun, E. Merrill, and T. Peterson. „From implicit skills to explicit knowledge: a bottom-up model of skill learning“. In: *Cognitive Science* 25.2 (2001), pp. 203–244.

[113] F. Nickols. „The Knowledge in Knowledge Management“. In: *The Knowledge Management Yearbook 2000-2001*. Ed. by J. A. Woods and J. Cortada. Butterworth-Heinemann, 2000, pp. 12–21.

[114] J. Mylopoulos and H. Levesque. „An Overview of Knowledge Representation“. In: *GWAI-83*. Vol. 76. Informatik-Fachberichte. Springer, 1983, pp. 143–157.

[115] P. F. Patel-Schneider. „A Decidable First-Order Logic for Knowledge Representation“. In: *Proceedings of the 9th international joint conference on Artificial intelligence*. 1985, pp. 455–458.

[116] L. A. Zadeh. „Knowledge Representation in Fuzzy Logic“. In: *IEEE Transactions on Knowledge and Data Engineering* 1.1 (1989), pp. 89–100.

[117] S. C. Shapiro. „Path-based and node-based inference in semantic networks“. In: *Proceedings of the 1978 workshop on Theoretical issues in natural language processing*. 1978, pp. 219–225.

[118] H. Chen. „Machine Learning for Information Retrieval: Neural Networks, Symbolic Learning, and Genetic Algorithms“. In: *Journal of the American Society for Information Science* 46.3 (1995), pp. 194–216.

[119] R. P. Lippmann. „An Introduction to Computing with Neural Nets“. In: *IEEE ASSP Magazine* 4.2 (1987), pp. 4–22.

[120] D. D. Dankel. „Expert Systems“. In: *Organization and Decision Theory*. Ed. by I. Horowitz. Vol. 18. Recent Economic Thought Series. Springer, 1990, pp. 255–276. ISBN: 978-94-009-2514-4.

[121] B. Nebel. „Frame-Based Systems“. In: *The MIT Encyclopedia of the Cognitive Sciences*. Ed. by R. A. Wilson and F. C. Keil. MIT Press, 1999.

[122] M. Stefik. „An examination of a frame-structured representation system“. In: *Proceedings of the 6th international joint conference on Artificial intelligence*. 1979, pp. 845–852.

[123] R. Davis, H. Shrobe, and P. Szolovits. „What Is a Knowledge Representation?“ In: *AI Magazine* 14.1 (1993), pp. 17–33.

[124] D. C. Schmidt. „Model-Driven Engineering“. In: *Computer* 39.2 (2006), pp. 25–31.

[125] T. Kühne. „What is a Model?“ In: *Language Engineering for Model-Driven Software Development, Number 04101 in Dagstuhl Seminar Proceedings*. 2005.

[126] K. Thramboulidis and G. Frey. „Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation“. In: *Journal of Software Engineering and Applications* 4.4 (2011), pp. 217–226.

[127] A. Belkin, A. Kuwertz, Y. Fischer, and J. Beyerer. „World Modeling for Autonomous Systems“. In: *Innovative Information Systems Modelling Techniques*. Ed. by C. Kalloniatis. InTech, 2012, pp. 137–158. ISBN: 978-953-51-0644-9.

[128] W. A. Woods. „Important Issues in Knowledge Representation“. In: *Proceedings of the IEEE* 74.10 (1986), pp. 1322–1334.

[129] B. Kulvatunyou, H. Cho, and Y. J. Son. „A semantic web service framework to support intelligent distributed manufacturing“. In: *International Journal of Knowledge-based and Intelligent Engineering Systems* 9.2 (2005), pp. 107–127.

[130] T. R. Gruber. „A Translation Approach to Portable Ontology Specifications“. In: *Knowledge Acquisition* 5.2 (1993), pp. 199–220.

[131] N. Noy and D. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology.* URL: http://liris.cnrs.fr/alain.mille/enseignements/Ecole_Centrale/What is an ontology and why we need it.htm. Access date October 2013.

[132] S. Lemaignan, A. Siadat, J.-Y. Dantan, and A. Semenenko. „MASON: A Proposal For An Ontology Of Manufacturing Domain". In: *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications.* 2006, pp. 195–200.

[133] G. Antoniou and F. van Harmelen. „Web Ontology Language: OWL". In: *Handbook on Ontologies, Part I.* Ed. by S. Staab and R. Studer. Springer, 2004, pp. 67–92. ISBN: 978-3-540-24750-0.

[134] P. Vrba, M. Radakovič, M. Obitko, and V. Mařík. „Semantic Extension of Agent-Based Control: The Packing Cell Case Study". In: *4th International Conference on Industrial Application of Holonic and Multi-Agent Systems.* 2009, pp. 47–60.

[135] M. Hadzic, P. Wongthongtham, D. Tharam, and E. Chang. *Ontology-Based Multi-Agent Systems.* Springer Publishing Company, Incorporated, 2009. ISBN: 978-3-642-01904-3.

[136] N. Lohse, H. Hirani, and S. Ratchev. „Equipment ontology for modular reconfigurable assembly systems". In: *International Journal of Flexible Manufacturing Systems* 17.4 (2006), pp. 301–314.

[137] O^{3}neida. *Network of Networks to Advance Distributed Industrial Automation.* URL: http://www.oooneida.org. Access date April 2014.

[138] V. Vyatkin, J. H. Christensen, and J. L. M. Lastra. „OOONEIDA: An Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation". In: *IEEE Transactions on Industrial Informatics* 1.1 (2005), pp. 4–17.

[139] O. J. López Orozco and J. L. M. Lastra. „Using Semantic Web Technologies to Describe Automation Objects". In: *International Journal of Manufacturing Research* 1.4 (2006), pp. 482–503.

[140] Y. Alsafi and V. Vyatkin. „Ontology-based reconfiguration agent for intelligent mechatronic systems in flexible manufacturing". In: *Robotics and Computer-Integrated Manufacturing* 26.4 (2010), pp. 381–391.

[141] L. Bunch, M. Breedy, J. M. Bradshaw, M. Carvalho, N. Suri, A. Uszok, J. Hansen, M. Pěchouček, and V. Mařík. „Software Agents for Process Monitoring and Notification". In: *Proceedings of the 2004 ACM Symposium on Applied computing.* 2004, pp. 94–100.

[142] M. Vallée, M. Merdan, and P. Vrba. „Detection of Anomalies in a Transport System using Automation Agents with a Reflective World Model“. In: *Proceedings of the IEEE International Conference on Industrial Technology.* 2010, pp. 489–494.

[143] F. Heck, T. Längle, and H. Wörn. „A Multi-Agent Based Monitoring and Diagnosis System for Industrial Components“. In: *Proceedings of the 9th International Workshop on Principles of Diagnosis.* 1998, pp. 63–69.

[144] R. Frei, G. Di Marzo Serugendo, and J. Barata. „Designing Self-Organization for Evolvable Assembly Systems“. In: *Proceedings of the 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems.* 2008, pp. 97–106.

[145] A. Shui, W. Chen, P. Zhang, S. Hu, and X. Huang. „Review of Fault Diagnosis in Control Systems“. In: *Proceedings of the Chinese Control and Decision Conference.* 2009, pp. 5324–5329.

[146] Y.-H. Bae, S.-H. Lee, H.-C. Kim, B.-R. Lee, J. Jang, and J. Lee. „A real-time intelligent multiple fault diagnostic system“. In: *International Journal of Advanced Manufacturing Technology* 29.5–6 (2006), pp. 590–597.

[147] E. M. Davidson, S. D. J. McArthur, J. R. McDonald, T. Cumming, and I. Watt. „Applying Multi-Agent System Technology in Practice: Automated Management and Analysis of SCADA and Digital Fault Recorder Data“. In: *IEEE Transactions on Power Systems* 21.2 (2006), pp. 559–567.

[148] M. Merdan, M. Vallée, W. Lepuschitz, and A. Zoitl. „Monitoring and diagnostics of industrial systems using automation agents“. In: *International Journal of Production Research* 49.5 (2011), pp. 1497–1509.

[149] M. Albert, T. Längle, H. Wörn, M. Capobianco, and A. Brighenti. „Multi-agent systems for industrial diagnostics“. In: *Proceedings of the 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes.* 2003, pp. 483–488.

[150] Z. M. Bi, S. Y. T. Lang, W. Shen, and L. Wang. „Reconfigurable manufacturing systems: the state of the art“. In: *International Journal of Production Research* 46.4 (2008), pp. 967–992.

[151] M. Bruccoleri, M. Amico, and G. Perrone. „Distributed intelligent control of exceptions in reconfigurable manufacturing systems“. In: *International Journal of Production Research* 41.7 (2003), pp. 1393–1412.

[152] M. Bruccoleri, Z. J. Pasek, and Y. Koren. „Operation management in reconfigurable manufacturing systems: Reconfiguration for error handling“. In: *International Journal of Production Economics* 100.1 (2006), pp. 87–100.

[153] P. Leitao and F. J. Restivo. „Implementation of a Holonic Control System in a Flexible Manufacturing System“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 38.5 (2008), pp. 699–709.

[154] J. Zhang, F. T. S. Chan, P. Li, H. C. W. Lau, R. W. L. Ip, and P. Samaranayak. „Investigation of the reconfigurable control system for an agile manufacturing cell“. In: *International Journal of Production Research* 40.15 (2002), pp. 3709–3723.

[155] M. K. Lim and D. Z. Zhang. „An integrated agent-based approach for responsive control of manufacturing resources“. In: *Computers & Industrial Engineering* 46.2 (2004), pp. 221–232.

[156] R. M. Lima, R. M. Sousa, and P. J. Martins. „Distributed production planning and control agent-based system“. In: *International Journal of Production Research* 44.18 (2006), pp. 3693–3709.

[157] V. Vyatkin. „IEC 61499 as enabler of distributed and intelligent automation SOTA review“. In: *IEEE Transactions on Industrial Informatics* 7.4 (2011), pp. 768–781.

[158] A. Zoitl, T. Strasser, C. Sünder, and T. Baier. „Is IEC 61499 in Harmony with IEC 61131-3?“ In: *IEEE Industrial Electronics Magazine* 3.4 (2009), pp. 49–55.

[159] A. Zoitl. *Real-Time Execution for IEC 61499*. USA: International Society of Automation, 2009. ISBN: 978-1-934394-27-4.

[160] K.-H. John and M. Tiegelkamp. *IEC 61131-3: Programming Industrial Automation Systems*. Springer, 2001. ISBN: 3-540-67752-6.

[161] IEC 61131-3. *Programmable controllers—Part 3: Programming languages*. Geneva: International Electrotechnical Commission, 1993.

[162] A. W. Colombo, R. Schoop, and R. Neubert. „An Agent-Based Intelligent Control Platform for Industrial Holonic Manufacturing Systems“. In: *IEEE Transactions on Industrial Electronics* 53.1 (2006), pp. 322–337.

[163] A. W. Colombo, R. Neubert, and R. Schoop. „A solution to holonic control systems“. In: *Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation.* Vol. 2. 2001, pp. 489–498.

[164] R. Schoop, R. Neubert, and A. W. Colombo. „A Multiagent-based Distributed Control Platform for Industrial Flexible Production Systems“. In: *Proceedings of the 27th Annual Conference of the IEEE Industrial Electronics Society.* Vol. 1. 2001, pp. 279–284.

[165] G. Cândido and J. Barata. „A Multiagent Control System for Shop Floor Assembly“. In: *Holonic and Multi-Agent Systems for Manufacturing.* Vol. 4659. Lecture Notes in Computer Science. Springer, 2007, pp. 293–302.

[166] P. Vrba, V. Mařík, and M. Merdan. „Physical Deployment of Agent-based Industrial Control Solutions: MAST Story“. In: *Proceedings of the IEEE International Conference on Distributed Human-Machine Systems.* 2008, pp. 133–139.

[167] F. P. Maturana, P. Tichý, P. Šlechta, F. Discenzo, R. J. Staron, and K. Hall. „Distributed multi-agent architecture for automation systems“. In: *Expert Systems with Applications* 26.1 (2004), pp. 49–56.

[168] T. Cucinotta, A. Mancina, G. F. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusinà. „A Real-Time Service-Oriented Architecture for Industrial Automation“. In: *IEEE Transactions on Industrial Informatics* 5.3 (2009), pp. 267–277.

[169] R. W. Brennan, P. Vrba, P. Tichý, A. Zoitl, C. Sünder, T. Strasser, and V. Mařík. „Developments in dynamic and intelligent reconfiguration of industrial automation“. In: *Computers in Industry* 59.6 (2008), pp. 533–547.

[170] S. Lohmann, O. Stursberg, and S. Engell. „Comparison of Event-Triggered and Cycle-Driven Models for Verifying SFC Programs“. In: *Proceedings of the American Control Conference.* 2007, pp. 3606–3611.

[171] A. Zoitl and V. Vyatkin. „IEC 61499 Architecture for Distributed Automation: The “Glass Half Full” View“. In: *IEEE Industrial Electronics Magazine* 3.4 (2009), pp. 7–23.

[172] A. Zoitl, T. Strasser, K. Hall, R. Staron, C. Sünder, and B. Favre-Bulle. „The Past, Present, and Future of IEC 61499“. In: *Holonic and Multi-Agent Systems for Manufacturing.* Vol. 4659. Lecture Notes in Computer Science. Springer, 2007, pp. 1–14.

[173] IEC 61499-1. *Function blocks—Part 1: Architecture.* Geneva: International Electrotechnical Commission, 2005.

[174] R. Lewis. *Modelling Control Systems Using IEC 61499.* UK: Institution of Engineering & Technology, 2001. ISBN: 0-85296-796-9.

[175] V. Vyatkin and V. Dubinin. „Refactoring of Execution Control Charts in Basic Function Blocks of the IEC 61499 Standard". In: *IEEE Transactions on Industrial Informatics* 6.2 (2010), pp. 155–165.

[176] C. Sünder, A. Zoitl, J. Christensen, H. Steininger, and J. Fritsche. „Considering IEC 61131-3 and IEC 61499 in the context of component frameworks". In: *Proceedings of the 6th IEEE International Conference on Industrial Informatics.* 2008, pp. 277–282.

[177] S. M. Deen. *Agent Based Manufacturing: Advances in the Holonic Approach.* Berlin: Springer, 2003. ISBN: 978-3540440697.

[178] O. J. López Orozco and J. L. M. Lastra. „A Real-Time Interface for Agent-Based Control". In: *Proceedings of the International Symposium on Industrial Embedded Systems.* 2007, pp. 49–54.

[179] O. J. López Orozco and J. L. M. Lastra. „Agent-Based Control Model for Reconfigurable Manufacturing Systems". In: *Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation.* 2007, pp. 1233–1238.

[180] V. Vyatkin and C. Peniche. „How does the IEC61499 Architecture Fit the Requirements of Intelligent Automation Systems?" In: *Proceedings of the 2nd International Conference on Industrial Informatics.* 2004, pp. 575–580.

[181] A. Zoitl, C. Sünder, and I. Terzic. „Dynamic Reconfiguration of Distributed Control Applications with Reconfiguration Services based on IEC 61499". In: *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications.* 2006, pp. 109–114.

[182] R. W. Brennan, X. Zhang, Y. Xu, and D. H. Norrie. „A reconfigurable concurrent function block model and its implementation in real-time Java". In: *Integrated Computer-Aided Engineering* 9.3 (2002), pp. 263–279.

[183] M. Metzger and G. Polaków. „A Survey on Applications of Agent Technology in Industrial Process Control". In: *IEEE Transactions on Industrial Informatics* 7.4 (2011), pp. 570–581.

[184] X. Zhang, R. W. Brennan, Y. Xu, and D. H. Norrie. „Runtime Adaptability of a Concurrent Function Block Model for a Real-Time Holonic Controller". In: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics.* 2001, pp. 164–168.

[185] R. W. Brennan, M. Fletcher, and D. H. Norrie. „A Holonic Approach to Reconfiguring Real-Time Distributed Control Systems". In: *Multi-Agent Systems and Applications II.* Vol. 2322. Lecture Notes in Computer Science. Springer, 2002, pp. 323–335.

[186] R. W. Brennan, M. Fletcher, and D. H. Norrie. „An Agent-Based Approach to Reconfiguration of Real-Time Distributed Control Systems". In: *IEEE Transactions on Robotics and Automation* 18.4 (2002), pp. 444–451.

[187] S. Olsen, J. Wang, A. Ramirez-Serrano, and R. W. Brennan. „Contingencies-based reconfiguration of distributed factory automation". In: *Robotics and Computer-Integrated Manufacturing* 21.4-5 (2005), pp. 379–390.

[188] K. Thramboulidis and A. Zoupas. „Real-Time Java in Control and Automation: A Model Driven Development Approach". In: *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation.* 2005, pp. 39–46.

[189] K. Thramboulidis and N. Papakonstantinou. „An IEC 61499 Execution Environment for an aJile-based Field Device". In: *Proceedings of the 11th IEEE Conference on Emerging Technologies and Factory Automation.* 2006, pp. 989–992.

[190] M. Khalgui and O. Mosbahi. „Intelligent Distributed Control Systems". In: *Information and Software Technology* 52.12 (2010), pp. 1259–1271.

[191] M. Khalgui, O. Mosbahi, Z. Li, and H.-M. Hanisch. „Reconfigurable Multiagent Embedded Control Systems: From Modeling to Implementation". In: *IEEE Transactions on Computers* 60.4 (2011), pp. 538–551.

[192] M. Rausch and H.-M. Hanisch. „Net condition/event systems with multiple condition outputs". In: *Proceedings of the INRIA/IEEE Symposium on Emerging Technologies and Factory Automation.* 1995, pp. 592–600.

[193] M. Khalgui, O. Mosbahi, Z. Li, and H.-M. Hanisch. „Reconfigurable of Distributed Embedded-Control Systems". In: *IEEE/ASME Transactions on Mechatronics* 16.4 (2011), pp. 684–694.

[194] Y. Alsafi and V. Vyatkin. „An Ontology-Based Reconfiguration Agent for Intelligent Mechatronic Systems“. In: *3rd International Conference on Industrial Application of Holonic and Multi-Agent Systems.* 2007, pp. 114–126.

[195] Y. Alsafi and V. Vyatkin. „A Deployment of an Ontology-based Reconfiguration Agent for Intelligent Mechatronic Systems“. In: *Proceedings of the IEEE International Symposium on Industrial Electronics.* 2008, pp. 1780–1785.

[196] A. Zoitl, W. Lepuschitz, M. Merdan, and M. Vallée. „A Real-Time Reconfiguration Infrastructure for Distributed Embedded Control Systems“. In: *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation.* 2010, pp. 1–8.

[197] W. Dai, V. Vyatkin, J. H. Christensen, and V. Dubinin. „Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Interoperability“. In: *IEEE Transactions on Industrial Informatics* 11.3 (2015), pp. 771–781.

[198] M. Sorouri and V. Vyatkin. „Intelligent Product and Mechatronic Software Components Facilitating Mass Customization in Collaborative Manufacturing Systems“. In: *Collaboration in a Hyperconnected World.* Ed. by H. Afsarmanesh, L. Camarinha-Matos, and A. Lucas Soares. Vol. 480. IFIP Advances in Information and Communication Technology. Springer, 2016, pp. 394–407. ISBN: 978-3-319-45390-3.

[199] M. Sorouri, V. Vyatkin, and Z. Salcic. „MIRA: Enabler of Mass Customization through Agent-Based Development of Intelligent Manufacturing Systems“. In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2014, pp. 258–263.

[200] M. Sorouri, V. Vyatkin, and Z. Salcic. „Rule-Based Composition of Intelligent Mechatronic Components in Manufacturing Systems Using Prolog“. In: *Proceedings of the 11th IEEE International Conference on Industrial Informatics.* 2013, pp. 242–247.

[201] R. Priego, D. Schütz, B. Vogel-Heuser, and M. Marcos. „Reconfiguration Architecture for Updates of Automation Systems During Operation“. In: *Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation.* 2015, pp. 1–8.

[202] C. Legat, D. Schütz, and B. Vogel-Heuser. „Automatic generation of field control strategies for supporting (re-)engineering of manufacturing systems“. In: *Journal of Intelligent Manufacturing* 25.5 (2015), pp. 1101–1111.

[203] J. Yan, C. Pang, C.-W. Yang, and V. Vyatkin. „Adaptable Software Components: Towards Digital Ecosystems and Software Evolution in the Industrial Automation Domain“. In: *Proceedings of the 40th Annual Conference of the IEEE Industrial Electronics Society.* 2014, pp. 2512–2518.

[204] nxtControl. *nxtSTUDIO.* URL: http://www.nxtcontrol.com/engineering. Access date August 2017.

[205] A. Streit, S. Rösch, and B. Vogel-Heuser. „Redeployment of Control Software during Runtime for Modular Automation Systems Taking Real-Time and Distributed I/O into Consideration“. In: *Proceedings of the 16th IEEE International Conference on Emerging Technologies and Factory Automation.* 2014, pp. 1–8.

[206] C. Sünder, A. Zoitl, and C. Dutzler. „Functional structure-based modelling of automation systems“. In: *International Journal of Manufacturing Research* 1.4 (2006), pp. 405–420.

[207] M. Vallée, H. Kaindl, M. Merdan, W. Lepuschitz, E. Arnautovic, and P. Vrba. „An Automation Agent Architecture with A Reflective World Model in Manufacturing Systems“. In: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics.* 2009, pp. 305–310.

[208] W. Shen, Q. Hao, H. J. Yoon, and D. H. Norrie. „Applications of agent-based systems in intelligent manufacturing: An updated review“. In: *Advanced Engineering Informatics* 20.4 (2006), pp. 415–431.

[209] M. Merdan, T. Moser, P. Vrba, and S. Biffl. „Investigating the robustness of re-scheduling policies with multi-agent system simulation“. In: *International Journal of Advanced Manufacturing Technology* 55.1–4 (2011), pp. 355–367.

[210] Telecom Italia Lab. *JADE - Java Agent DEvelopment Framework.* URL: http://jade.tilab.com/. Access date March 2014.

[211] FESTO. *Festo Didactic.* URL: http://www.festo-didactic.com. Access date May 2016.

[212] Digi International. *Digi Connect ME.* URL: http://www.digi.com/de/products/embeddedsolutions/digiconnectme. Access date October 2013.

[213] E. W. Dijkstra. „A Note on Two Problems in Connexion with Graphs.“ In: *Numerische Mathematik* 1 (1959), pp. 269–271.

[214] M. Merdan, G. Koppensteiner, I. Hegny, and B. Favre-Bulle. „Application of an Ontology in a Transport Domain“. In: *Proceedings of the IEEE International Conference on Industrial Technology.* 2008, pp. 1–6.

[215] P. Tichý, P. Šlechta, F. Maturana, and S. Balasubramanian. „Industrial MAS for Planning and Control“. In: *Multi-Agent Systems and Applications II.* Vol. 2322. Lecture Notes in Computer Science. Springer, 2002, pp. 137–154.

[216] E. Gat. „On three-layer architectures“. In: *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems.* AAAI Press, 1998, pp. 195–210. ISBN: 0262611376.

[217] J. P. Müller. *The Design of Intelligent Agents: A Layered Approach.* Vol. 1177. Lecture Notes in Computer Science. Springer, 1996, p. 227. ISBN: 978-3-540-62003-7.

[218] F. Baader, I. Horrock, and U. Sattler. „Description Logics“. In: *Handbook on Ontologies.* International Handbooks on Information Systems. Springer, 2004, pp. 3–28.

[219] J. F. Allen. „Maintaining Knowledge about Temporal Intervals“. In: *Communications of the ACM* 26.11 (1983), pp. 832–843.

[220] A. Zoitl, R. Smodic, C. Sünder, and G. Grabmair. „Enhanced Real-Time Execution of Modular Control Software based on IEC 61499“. In: *Proceedings of the IEEE International Conference on Robotics and Automation.* 2006, pp. 327–332.

[221] A. Drumea and C. Popescu. „Finite State Machines and their applications in software for industrial control“. In: *Proceedings of the 27th International Spring Seminar on Electronics Technology: Meeting the Challenges of Electronics Technology Progress.* Vol. 1. 2004, pp. 25–29.

[222] A. Zoitl and H. Prähofer. „Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modeling Language“. In: *IEEE Transactions on Industrial Informatics* 9.4 (2013), pp. 2387–2396.

[223] A. Zoitl and H. Prähofer. „Building hierarchical automation solutions in the IEC 61499 modeling language“. In: *Proceedings of the 9th International Conference on Industrial Informatics.* 2011, pp. 557–564.

[224] I. Hegny, T. Strasser, M. Melik-Merkumians, M. Wenger, and A. Zoitl. „Towards an Increased Reusability of Distributed Control Applications Modeled in IEC 61499“. In: *Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation.* 2012, pp. 1–8.

[225] L. H. Yoong, P. S. Roop, and Z. Salcic. „Efficient Implementation of IEC 61499 Function Blocks“. In: *Proceedings of the IEEE International Conference on Industrial Technology.* 2009, pp. 1–6.

[226] FESTO. *Controller CPX-CEC-C1.* URL: http://www.festo.com/cms/en-us_us/14523.htm. Access date October 2013.

[227] 4DIAC Iniative. *FORTE—4DIAC Runtime Environment.* URL: http://www.eclipse.org/4diac/en_rte.php. Access date May 2016.

[228] G. Koppensteiner, M. Merdan, I. Hegny, W. Lepuschitz, S. Auer, and B. Groessing. „Deployment of an ontology-based agent architecture on a controller“. In: *Proceedings of the 8th IEEE International Conference on Industrial Informatics.* 2010, pp. 890–895.

[229] J. Yick, B. Mukherjee, and D. Ghosal. „Wireless sensor network survey“. In: *Computer Networks* 52.12 (2007), pp. 2292–2330.

[230] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac. „Internet of things: Vision, applications and research challenges“. In: *Ad Hoc Networks* 10.7 (2012), pp. 1497–1516.

[231] M. A. Taleghan, A. Taherkordi, M. Sharifi, and T.-H. Kim. „A Survey of System Software for Wireless Sensor Networks“. In: *Generation Communication and Networking.* 2007, pp. 402–407.

[232] P. Vrba and V. Mařík. „Simulation in Agent-based Manufacturing Control Systems“. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.* 2005, pp. 1718–1723.

[233] COPA-DATA. *zenon Driver Information.* URL: https://www.copadata.com/en-nl/support-services/zenon-driver-information. Access date March 2017.

[234] G. Koppensteiner, M. Merdan, W. Lepuschitz, and I. Hegny. „Hybrid based approach for fault tolerance in a multi-agent system“. In: *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics.* 2009, pp. 679–684.

[235] P. Fröhlich and W. Nejdl. „Resolving Conflicts in Distributed Diagnosis“. In: *Proceedings of the 12th European Conference on Artificial Intelligence, Workshop on Modelling Conflicts in AI.* 1996.

[236] M. Albert, T. Längle, and H. Wörn. „Development Tool for Distributed Monitoring and Diagnosis Systems“. In: *Proceedings of the Thirteenth International Workshop on Principles of Diagnosis*. 2002.

[237] P. Fröhlich, I. Móra, W. Nejdl, and M. Schröder. „Diagnostic Agents for Distributed Systems“. In: *Formal Models of Agents*. Lecture Notes in Artificial Intelligence. Springer, 1999, pp. 173–186.

[238] H. Kaindl, M. Vallée, and E. Arnautovic. „Self-Representation for Self-Configuration and Monitoring in Agent-Based Flexible Automation Systems“. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43.1 (2013), pp. 164–175.

[239] K. Collins. *PLC Programming for Industrial Automation*. Exposure, 2007. ISBN: 978-1846855986.

[240] N. Faci, Z. Guessoum, and O. Marin. „DimaX: A Fault-Tolerant Multi-Agent Platform“. In: *Proceedings of the 5th International Workshop on Software Engineering for Large-scale Multi-Agent Systems*. 2006, pp. 13–20.

[241] J. Kramer and J. Magee. „Dynamic Configuration for Distributed Systems“. In: *IEEE Transactions on Software Engineering* 11.4 (1985), pp. 424–435.

[242] 4DIAC Iniative. *Framework for Distributed Industrial Automation and Control*. URL: http://www.fordiac.org. Access date June 2016.

[243] S. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. Ousterhout. „It's Time for Low Latency“. In: *Proceedings of the 13th Workshop on Hot Topics in Operating Systems*. 2011.

[244] Wireshark Foundation. *Wireshark*. URL: http://www.wireshark.org. Access date October 2013.

[245] E. J. González, A. F. Hamilton, L. Moreno, R. L. Marichal, and V. Munoz. „Software experience when using ontologies in a multi-agent system for automated planning and scheduling“. In: *Software: Practice and Experience* 36.7 (2006), pp. 667–688.

[246] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Boston, MA, USA: Addison-Wesley, 2000. ISBN: 978-0201309775.

[247] K. Czarnecki and U. W. Eisenecker. „Components and Generative Programming“. In: *Software Engineering – ESEC/FSE '99*. Vol. 1687. Lecture Notes in Computer Science. Springer, 1999, pp. 2–19.

[248] M. Steinegger and A. Zoitl. „Automated Code Generation for Programmable Logic Controllers based on Knowledge Acquisition from Engineering Artifacts: Concept and Case Study“. In: *Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation*. 2012, pp. 1–8.

[249] Q.-N. N. Tran and G. Low. „MOBMAS: A methodology for ontology-based multi-agent systems development“. In: *Information and Software Technology* 50.7–8 (2008), pp. 697–722.

[250] E. Munoz, E. Capón-García, A. Espuna, and L. Puigjaner. „Ontological framework for enterprise-wide integrated decision-making at operational level“. In: *Computers & Chemical Engineering* 42 (2012), pp. 217–234.

[251] ANSI/ISA-S95. *Enterprise-Control System Integration—Part 1: Models and Terminology*. Research Triangle Park, North Carolina, USA: ISA The Instrumentation, Systems, and Automation Society, 2000.

[252] IEC 61512-1. *Batch Control—Part 1: Models and Terminology*. Geneva: International Electrotechnical Commission, 1997.

[253] A. Zoitl, G. Kainz, and N. Keddis. „Production Plan-Driven Flexible Assembly Automation Architecture“. In: *Industrial Applications of Holonic and Multi-Agent Systems*. Vol. 8062. Lecture Notes in Computer Science. Springer, 2013, pp. 49–58.

[254] O. J. López Orozco and J. L. M. Lastra. „Adding Function Blocks of IEC 61499 Semantic Description to Automation Objects“. In: *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*. 2006, pp. 537–544.

[255] J. M. Mendes, P. Leitao, and A. W. Colombo. „Service-oriented Computing in Manufacturing Automation: A SWOT Analysis“. In: *Proceedings of the 9th International Conference on Industrial Informatics*. 2011, pp. 346–351.

[256] D. Regulin, D. Schütz, T. Aicher, and B. Vogel-Heuser. „Model Based Design of Knowledge Bases in Multi Agent Systems for enabling Automatic Reconfiguration Capabilities of Material Flow Modules“. In: *Proceedings of the IEEE International Conference on Automation Science and Engineering*. 2016, pp. 133–140.

[257] Beckhoff. *Controller CX5010*. URL: http://www.beckhoff.com/english.asp?embedded_pc/cx5010_cx5020.htm. Access date October 2013.

[258] J. Kramer and J. Magee. „A Model for Change Management“. In: *Workshop on the Future Trends of Distributed Computing Systems in the 1990s*. 1988, pp. 286–295.

[259] M. Vallée, M. Merdan, W. Lepuschitz, and G. Koppensteiner. „Decentralized Reconfiguration of a Flexible Transportation System“. In: *IEEE Transactions on Industrial Informatics* 7.3 (2011), pp. 505–516.

[260] M. D. Byrne and P. Chutima. „Real-time operational control of an FMS with full routing flexibility“. In: *International Journal of Production Economics* 51.1–2 (1997), pp. 109–113.

[261] V. Jirkovský, P. Kadera, and P. Vrba. „Semantics for Self-configurable Distributed Diagnostics“. In: *Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation.* 2012, pp. 1–6.

[262] P. Vrba, P. Tichý, V. Mařík, K.H. Hall, R. J. Staron, F. P. Maturana, and P. Kadera. „Rockwell Automation's Holonic and Multiagent Control Systems Compendium“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 41.1 (2011), pp. 14–30.

[263] L. Moreau. „Stability of Multiagent Systems With Time-Dependent Communication Links“. In: *IEEE Transactions on Automatic Control* 50.2 (2005), pp. 169–182.

[264] P. De Wilde, H. S. Nwana, and L. C. Lee. „Stability, Fairness and Scalability of Multi-Agent Systems“. In: *International Journal of Knowledge-Based Intelligent Engineering Systems* 3 (1999), pp. 84–91.

[265] R. D'Andrea. „A Revolution in the Warehouse: A Retrospective on Kiva Systems and the Grand Challenges Ahead“. In: *IEEE Transactions on Automation Science and Engineering* 9.4 (2012), pp. 638–639.

[266] Stanford University. *The Protégé Ontology Editor and Knowledge Acquisition System.* URL: http://protege.stanford.edu. Access date October 2013.

[267] EPLAN. *EPLAN Electric P8.* URL: http://www.eplan.at/at-de/loesungen/elektrotechnik/eplan-electric-p8/. Access date October 2013.

[268] S. T. Coleridge. *The Friend; A Series of Essays.* London, UK: Gale and Curtis, 1812.

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.
Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, im April 2018

Wilfried Lepuschitz