UNIVERSITY OF SOUTHAMPTON

AN AGENT-ORIENTED DESIGN METHODOLOGY FOR PRODUCTION CONTROL

by

Stefan Karl BUSSMANN

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Engineering and Applied Science

Department of Electronics and Computer Science

September, 2003

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

AN AGENT-ORIENTED DESIGN METHODOLOGY FOR PRODUCTION CONTROL

by Stefan Karl BUSSMANN

This thesis presents and evaluates the DACS methodology for designing agent-based systems for (non-real-time) production control. This methodology is appropriate and sufficiently prescriptive for a control engineer with only minimal training in agent concepts and with no prior experience in agent development to design an agent-based production control system. This was achieved by deriving the concepts of the methodology from an analysis of the application domain and by specifying all relevant agent-oriented design rules in terms of these concepts. In particular, the methodology consists of a method for analysing the decision making necessary to control a production system; a method for identifying appropriate production control agents, which also includes a rule specifying when to abandon an agent-oriented design approach; and a method for selecting interaction protocols to resolve any decision dependencies between the agents which allows to re-use existing interaction protocols. To evaluate the methodology, several case studies and reviews were performed. In particular, two case studies with students applying the methodology to a realistic test case and two reviews by engineers, who design control systems, were conducted. The feedback was unequivocally that the methodology is appropriate and sufficiently prescriptive for designing agent-based production control systems and that it could be applied in industrial projects in order to gain more experience with the methodology.

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I would like to thank my supervisors Professor Nick Jennings and Professor Mike Wooldridge for their excellent supervision. Their frequent and constructive feedback has been very helpful and has contributed significantly to the success of my PhD. I would also like to thank Nick for accepting me as a part-time PhD student, despite all the difficulties that go with it, and Mike for being the second supervisor, despite the long time it took me to finish the PhD.

I would also like to thank Dr Kurt Sundermeyer who motivated me to do my PhD in England and who supported my PhD at work. Also thanks to DaimlerChrysler who made it possible for me to do the PhD in parallel to my job.

Furthermore, I would like to thank all those who helped me with the evaluation of the methodology. In particular, I would like to thank Ilka Lehweß-Litzmann and Laura Obretin for their engagement during the case studies. This thanks also goes to Jörg Sieverding who struggled with a very early version of the methodology in the Holomobiles project. And I would like to thank Ralf Neubert, Armando Walter Colombo, Böris Süssmann, Dirk Hofmann, and Raimund Krieg for reviewing the methodology. Special thanks goes to Walter Colombo who even reviewed part of the PhD as well as to Duncan McFarlane with whom I had many fruitful discussions over the years about the pros and cons of agent-based systems for manufacturing.

Here, I would also like to thank those who made developing agent-based control systems a fun and successful activity: Klaus Schild, Hartwig Baumgärtel, Sven Brückner, and Harald Windisch from our lab; Christian Anders, Gerold Winz, and Christoph Siegel from DaimlerChrysler; as well as Ralf Neubert and Ronald Schoop from Schneider Electric. It was a great time and I hope that we can develop many more systems like these in the future.

And most importantly, I owe a big thanks to my wife who supported me all the way, even when the PhD got very time-consuming and stressful, as well as to my children who – even though not knowing why – have not always seen as much of me as they would have liked. This PhD is therefore dedicated to them.

To my wife Martina

and my children Anabel and Silvia

Chapter 1

Introduction

Software agents offer a new approach to designing and building complex distributed systems that significantly extends previous approaches like object-oriented or distributed computing (Jennings 2000, Zambonelli 2003). Instead of modelling distributed systems as software programs exchanging data and commands, agent technology creates autonomous decision makers which communicate their preferences, negotiate sub-goals, and co-ordinate their intentions in order to achieve the individual or system goals (O'Hare 1996, Weiss 1999). This decision- and interaction-based approach to computing makes it possible to build systems that can dynamically react to unforeseen events, incorporate different preferences and attitudes, exploit different capabilities of components, and adapt flexibly to changes in the environment. The ability of agents to adapt their behaviour during computation reduces the need for the designer to foresee all possible scenarios and changes the system will encounter (Jennings 2000). Moreover, an agent-oriented design is often a natural fit to the distributed nature of decision making in many application domains and thus increases the understandability and maintainability of the software system.

The advantages of agent technology have been widely recognised and have led to a wide range of application studies in many different domains. The applications reported include industrial applications (process control, air traffic control), commercial applications (information management, electronic commerce, business process management), medical applications (patient monitoring, health care) and entertainment (games, interactive theatre and cinema) (Jennings 1998). One of these application domains that has been the target of many agent applications in the past is *production control* (Parunak 1999, Parunak 2000). Production control has been attractive to agent researchers for several reasons. First of all, production is an important industrial activity, responsible for a large portion of the gross national product in industrial nations (Eurostat 2002). Second, production systems are inherently distributed and dynamic systems exhibiting many changes and disturbances during operation (Parunak

1987). Third, and perhaps most importantly, agent technology promises to meet the forthcoming challenges in (non-real-time) production control which will go beyond the capabilities of current control technology. The following sections therefore evaluate the trends in production control and analyse the requirements on agent technology to meet these challenges.

1.1 Motivation

At the beginning of the 21st century, production faces a fundamental change from a vendor's to a customer's market. The growing surplus of industrial capacity provides the customer with a greater choice, and increases competition between vendors. Aware of his power, the customer becomes more demanding and less loyal to a brand. He demands constant product innovation, low-cost customisation, better service, and chooses the product which meets his requirements best (McFarlane 2003). The consequences for the industry are manifold. Companies must shorten product-life cycles, reduce time-to-market, increase product variety, quickly satisfy demand, reduce investment costs, and so on. For production, these consequences imply more complex products, faster changing products, faster introduction of products, a volatile order volume, and reduced investment (McFarlane 2003). The effects on production can be summarised as *increasing complexity* and *constant change* with *decreasing investment.*

The above trends have motivated researchers in academia and industry to create and exploit new production paradigms on the basis of autonomy and co-operation because both concepts are necessary to create flexible behaviour and thus to adapt to the changing production conditions (Tharumarajah 2003). Holonic manufacturing, for instance, as one of these new production paradigms, proposes the introduction of autonomous and co-operative building blocks, called holons, which organise themselves into flexible hierarchies, called holarchies (van Leeuwen 1997). Agent technology can be regarded as a key technology for realising the information processing of such systems (Bussmann 1998). In this context, agents are autonomous and co-operative units which due to their decision making and interaction capabilities are able to create and continuously adapt flexible process behaviour in the face of constant changes and disturbances. The ability of agent technology to address these challenges, as well as the industrial feasibility of an agent-based control approach have already been demonstrated with a number of industrial prototypes (Parunak 1996, Parunak 2000). But despite the numerous advantages of agent technology, installations of agent-based control systems are still rare in practice.

The reason for the slow industrial take-up are manifold, and in part certainly typical for the introduction of any new technology. Some reasons for the slow take-up, however, are technology-specific. Arguably the most important technology-specific reason for

this is that agent technology provides a large set of decision and interaction capabilities which are able to create a wide variety of system behaviours (Parunak 1997, Weiss 1999, Wooldridge 2002). Engineering such a system thus requires that the designer applies these capabilities very carefully in order to achieve the intended system behaviour. Without such a careful design, the designer may produce an agent-based system that shows a different, non-intended behaviour, or that achieves the intended behaviour in an inefficient way. In particular, the design may be more difficult to implement or maintain than necessary. A design, however, which is more difficult to implement or maintain will surely be more costly. In a domain such as production control, where investment must be continuously reduced, this is a "knock-out" criterion against using a new technology. The designer of a control system must therefore very carefully analyse and decide which agent-oriented mechanisms he requires in order to achieve the system requirements. The designers of control systems, however, are usually not computer scientists experienced at developing agent-based systems. The designer of a control system is typically an engineer with a background in production or control engineering, but with only minimal training in agent concepts at best and no experience with agent-based development in most cases. These designers thus require a methodology explaining how to perform the analysis and design of an agent-oriented production control system. In particular, such a methodology must include all the agent-oriented design criteria necessary to arrive at a well-designed agent-based system. To provide such a methodology, the next section looks at existing design methodologies and assesses whether these are sufficient to support a control engineer in developing agent-based control systems.

1.2 State-of-the-art

Many methodologies for designing software systems have been proposed in the past, some even for designing control systems (see chapter 3). The methodologies proposed include object-oriented, manufacturing control, and agent-oriented methodologies. A careful analysis of the wide variety of methodologies, however, reveals that existing methodologies are either not appropriate or not sufficiently prescriptive for designing agent-based production control systems. Object-oriented methodologies mainly fail to support the design of such systems because the modelling concepts of these methodologies are not appropriate for modelling the decision making of a control system (see section 3.3). In particular, objects are generally passive in nature, do not encapsulate behaviour activation, exchange only data or commands, and provide only minimal support for structuring organisations, whereas agents pro-actively follow their own goals, initiate interactions with other agents that were not foreseen at design time, and are able to adapt their organisational relationships (Jennings 2001). Object-oriented methodologies therefore miss many important aspects of an agent-based (control)

system. Manufacturing control methodologies, in turn, provide elaborated models for capturing the actual production process and the associated control decisions. The decision making, however, is modelled in a centralised or hierarchical form which conflicts with the autonomous and co-operative approach required for future production systems (see section 3.4). A designer can therefore rely on existing manufacturing methodologies to model the actual production process. But for the control system, the designer requires a methodology that prescribes how to design the agent-oriented aspects of such a system.

The limitations of object-oriented and manufacturing control methodologies have prompted many agent researchers to develop specifically agent-oriented design methodologies (see section 3.5). These methodologies are obviously able to adequately model agent-based systems. The existing agent-oriented design methodologies, nevertheless, are either inappropriate or not sufficiently prescriptive for modelling agent-based *production control* systems (see subsection 3.5.7). This is mainly due to the fact that most agent-oriented methodologies have been developed for applications other than production control. These methodologies therefore focus on concepts like roles, goals, and organisations, and thus miss the most important aspect of control, the decision making. Furthermore, existing agent-oriented methodologies are not sufficiently prescriptive for identifying production control agents. Most methodologies do provide criteria for agent identification. However, these are either too vague or lead to an inappropriate set of agents for production control. Finally, only few methodologies provide methods for designing or even re-using interaction protocols. The few that do consider the design of interactions only cover some aspects of the design process.

In summary, to date there does not exist a design methodology for agent-oriented production control systems that is appropriate and sufficiently prescriptive to be applied by a control engineer. However, such a design methodology, as argued above, is absolutely necessary for promoting the wide-spread use of agent technology in industry. Against this background, the aim of this research is therefore to develop such a methodology.

1.3 Contributions of the thesis

The goal of this thesis is to develop a methodology for the design of agent-based production control systems which can be successfully applied by a control engineer with only minimal training in agent technology and no prior experience in agent development. To this end, this thesis proposes a methodology for *Designing Agent-based Control Systems*, called DACS. This methodology starts with a specification of the control problem including

(i) a specification of the (physical) *production process* to be controlled,

(ii) a specification of the *production operation conditions*, and

(iii) a specification of the *production goals and requirements*.

An agent-based control system satisfying the above specification is then developed with the help of three methods which build upon each other:

1. The method for the *analysis of decision making* analyses the specification of the control problem in order to identify the decision tasks necessary to solve the control problem and any dependencies between these decision tasks.
2. The method for the *identification of agents* clusters the decision tasks according to agent-oriented criteria and assigns each cluster to an agent. If necessary, the decision tasks are re-organised before clustering in order to facilitate the identification of agents. Furthermore, the method includes a rule for assessing the applicability of an agent-oriented approach.
3. The method for the *selection of interaction protocols* classifies each decision dependency between different agents and matches this classification against a library of existing interaction protocols, in order to identify a protocol that is able to resolve the dependency. This protocol is then customised to the specific dependency situation arising between the control agents.

The output of the methodology is thus a list of agents with their distinct decision responsibilities and the interaction protocols required to resolve any decision dependencies. Due to the explicit specification of the dependencies and the associated interaction protocols able to resolve the dependencies, the resulting design is sufficiently modular so that each agent can be implemented independently.

The DACS methodology thus covers all agent-oriented design steps from analysing the production control problem, through identifying the control agents, to re-using existing interaction protocols. In particular, the DACS methodology extends the state-of-the-art in at least three respects:

- The methodology provides a method for analysing the production control problem that creates an agent-oriented decision model specified only in terms of domain concepts. In contrast to current approaches described in the literature, the methodology thus bridges the gap between the domain of production control and agent-based systems.
- The methodology provides a set of criteria for identifying agents – based on the decision model developed during the analysis – which either leads to an appropriate set of control agents or else suggests abandoning an agent-oriented approach. In contrast to the approaches in the literature, the methodology thus

provides a set of design rules that capture agent-oriented design knowledge in terms of rules directly related to the domain concepts.

- The methodology provides a classification scheme for re-using interaction protocols that is defined only in terms of the decision situation arising during the production process. In contrast to the literature, selecting an interaction protocol can thus be performed without any knowledge of the interaction protocols, and is therefore scalable to a large set of existing protocols.

Altogether, the DACS methodology thus significantly reduces the knowledge of and the experience in agent technology required during the design process and consequently enables a control engineer with only minimal training in agent technology and no prior experience in agent development to successfully design an agent-based production control system.

1.4 Structure of the thesis

The remainder of this thesis is structured as follows. Chapter 2 gives an overview of agent-based production control. It defines the terms "production" as well as "production control" and discusses the new requirements on future production systems. The chapter furthermore reviews agent technology and presents the state-of-the-art in agent-based production control.

Chapter 3, in turn, specifies the requirements on a design methodology for agent-based production control systems and reviews the state-of-the-art concerning design methodologies. In particular, it reviews object-oriented, manufacturing control, and agent-oriented methodologies, and shows in which respects these methodologies fail to meet the requirements on an agent-based design of production control systems. Furthermore, chapter 3 discusses the state-of-the-art in re-use and reviews the extent to which re-use has already been applied to the design of agent-based systems.

Chapter 4 then presents the main contribution of this work: the *DACS design methodology* for agent-based production control systems. The chapter specifies the input and output of the methodology, as well as each step of the methodology, namely the *analysis of decision making*, the *identification of agents*, and the *selection of interaction protocols*.

An evaluation of the methodology is given in chapter 5. This chapter provides initial evidence that the methodology is applicable by a control engineer with no prior experience in agent development. To this end, the chapter discusses two industrial case studies carried out by the author, a comparison of the methodology with the state-of-the-art, and two case studies performed by students as well as two reviews of the

methodology performed by control engineers. Chapter 6 then concludes the thesis and points to future work.

Finally, three appendices provide additional information. Appendix A lists the characterisations of several interaction protocols according to the classification scheme developed in chapter 4. Appendix B presents the application of the design methodology to an industrial test case. And appendix C lists the questionnaires used in the third-party reviews and the feedback received during these reviews.

Chapter 2

Agent-Based Production Control

An agent-oriented design methodology for production control is at the intersection of a new software technology, namely software agents, and an application domain, in this case production control. In order to understand the motivation for this particular design methodology, it is necessary to answer a set of questions:

1. What is production control? Why is it important and what is challenging about designing production control systems?
2. What is agent technology? And why does it help to meet the challenges in production control?
3. How is agent technology applied to production control? And why is it necessary to develop a design methodology for agent-based control systems?

All these questions will be answered in the following sections. First, section 2.1 will explain what production control is, why it is important, and what is challenging about it. This section will give basic definitions for what *production*, a *production system*, and *production control* is. It will furthermore review the state-of-the-art in production control and discuss why existing control techniques are not able to meet current challenges in the production industry. Section 2.2, in turn, will give an overview of software agent technology. It will provide basic definitions of an *agent* and a *multi-agent system*, and will review the agent-oriented techniques necessary to meet the challenges of modern production control. Finally, section 2.3 will show how agent technology can be applied to production control problems and will give examples of control applications using software agents. In particular, this section will discuss why there is no universal design for all production control problems and why, consequently, a design methodology must be developed.

2.1 Production control

Production is defined as the transformation of (physical) goods (Hoitsch 1993, Groover 1987, Hitomi 1994). The goods which are consumed by the transformation process are called *raw materials* and the goods produced are referred to as *products*. The process of providing raw material is called *procurement* (or supply) and the products are distributed through *shipping*. Procurement and shipping thus define the boundaries of the production process (see figure 2.1).

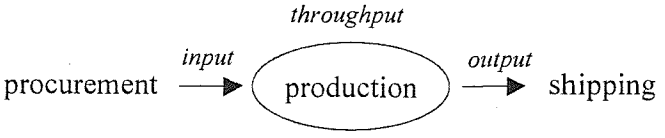

Figure 2.1: Production as a transformation process.

Production processes can be classified in many different ways. A common classification distinguishes production processes with respect to the types of products produced. The Federal Statistical Office of Germany, for instance, divides the production industry into the following main product branches: automobiles, chemicals, clothing, computing, electrical equipment, glass, machinery, food, furniture, metals, oil refinement, paper, publishing, textiles, tobacco, and wood (Federal Statistics Office of Germany 2001). The statistics also show how important this industry is. In 1999, the production industry has created a gross added-value of 1.342 billion Euros within the European Union[1], and 410 billion Euros in Germany (Eurostat 2002). The automotive industry alone accounts for 143 billion Euros in Europe[2], and 60 billion Euros in Germany. The production industry is thus an important economic factor in Europe (creating for example 25 % of the gross national product of Germany (Federal Statistics Office of Germany 2001)).

Another important classification of production processes is the distinction between ***discrete manufacturing*** and ***process industry***. This distinction refers to the general mode of operation. Discrete manufacturing is concerned with the processing of solid goods, such as metal, textiles, or wood which are processed individually, whereas the process industry takes liquids, such as milk, liquefied sugar, or oil, as input and then processes them continuously (Moore 1991). The distinction between discrete manufacturing and the process industry is obviously important to production control (Chokshi 2002): Solid goods are usually processed with tools and transported on conveyors. Liquids, on the other hand, are processed in tanks and transported through pipes. This work clearly focuses on discrete manufacturing (which created in 1999 at least 48% of the turnover of the production industry in Germany (Federal Statistical

[1] 15 European states without Spain and Ireland for which the data were not available yet.
[2] dito.

Office of Germany 2001)). In section 6.2, though, this focus will be revisited and the contribution of this work will be evaluated with respect to its applicability to other types of production processes.

A third common classification which is relevant to this work is the classification of production processes with respect to the product quantity made. The two extreme cases here are: job-shop and mass production (Groover 1987, p. 18). In ***job-shop production***, the quantities made of one product are usually small, often of size one. The products are typically created according to specific customer requirements and, as a consequence, a job-shop factory must be able to produce a wide variety of such products. Typical products produced in a job-shop factory are heavy machinery and ships. In ***mass production***, the product is produced in large quantities, sometimes with more than one million units per year. A factory for mass production is usually dedicated to a specific product in order to achieve the required quantity and to benefit from economies of scale due to the mass production. If the quantity produced is high, but the product varies slightly, mass production is often referred to as ***large-series production***. A typical example for a mass product is mineral water, typical examples for products manufactured in large-series are automobiles and household appliances.

To transform raw material into products, any of the above production types must implement certain production functions. Obviously, the main task of production is to process the raw material and transform it into an intermediate or finished product. But processing alone is not sufficient to create physical products. For example, the processing requires material which must be supplied somehow to the processing operations. For discrete manufacturing, Groover lists four basic functions that are necessary in modern production ((Groover 1987, p. 20), see also (Hitomi 1994, pp. 415)):[3]

- processing
- assembly
- material handling and storage
- inspection and testing

Processing and ***assembly*** are the operations that add value to the product by either changing the properties of the material (through processing) or by combining several workpieces into one (through assembly). ***Material handling*** and ***storage*** is necessary in order to (physically) provide the material at the right time and in the right quantity to the processing and assembly functions, whereas ***inspection*** and ***testing***, also called ***quality assurance***, is needed to verify that the processing and assembly operations meet the required quality standards. Naturally, there may be more functions necessary for

[3] (Groover 1987) also lists control as a basic production function. Control, however, will be dealt with separately in this section.

specific production types, but the above functions can be found in any (discrete) manufacturing system.

To perform the above functions, a production process obviously requires more input than the mere raw material (Groover 1987, p. 22). First of all, a production process requires (physical) ***equipment*** to perform its functions. The physical equipment may include machines, tools, and fixtures for processing; conveyors, forklifts, and buffers for material handling; and measuring tools and machines for quality assurance. Secondly, any factory requires ***labour***, either because a production function must be performed manually, or because the equipment must be set up and maintained manually. And thirdly, any production process consumes energy, sometimes also other ***auxiliary material***, such as water or machine oil. All these inputs (including the raw material) are referred to as *the production factors* (Hoitsch 1993). They constitute everything that is physically necessary to run a production process.[4] The production functions and factors are schematically summarised in figure 2.2.

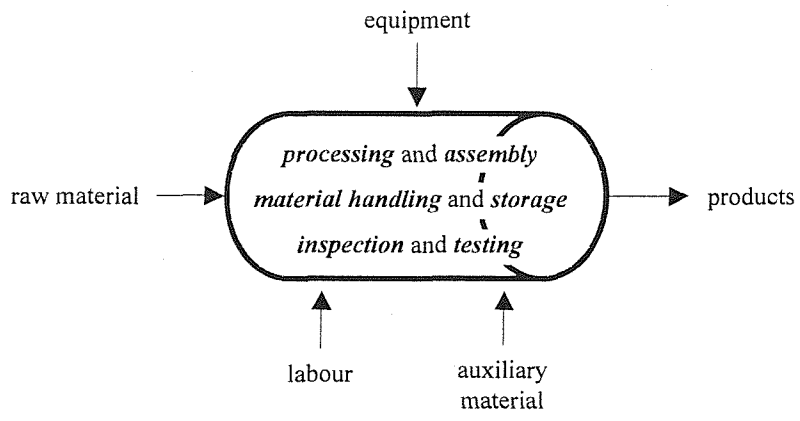

Figure 2.2: Production functions and inputs.

Production factors that are non-consumable, like machines or conveyors, are also called ***production resources***. Production resources are particularly important because they usually require a major investment and must be installed before any production can start. Because of their (largely) static nature, the production resources and their arrangement on the factory floor also determine the structure of the production process (Hitomi 1994). The structure of a production process, commonly referred to as the *production system*, is particularly relevant to control and is therefore explicitly defined in the following:

[4] Again, any planning and control aspects are omitted here because they will be addressed later in this section.

Definition: A set of production resources and a (spatial) arrangement of these resources is called a ***production system***. The arrangement is called the ***production layout*** (or the *plant layout*).

The production system is sometimes also referred to as the *factory floor* or the *shop floor*. These terms are used in particular to emphasise the difference between planning and reality (on the shop floor).

Two examples of typical production systems in discrete manufacturing will be given in the following subsection: one for job-shop and one for large-series production. Subsection 2.1.2 will then discuss what is necessary to operate a production system. In particular, it will define the term *production control*. Subsections 2.1.3 and 2.1.4 will review the state-of-the-practice in production control, analyse the current limitations, and infer the requirements for future production processes. Finally, a summary of this section is given in subsection 2.1.5.

2.1.1 Examples of production systems

An example of a production system is the *flexible manufacturing system* (FMS) (Hartley 1984). A typical FMS consists of several machines, a loading and unloading station, a transportation system, and a system buffer. Figure 2.3 shows a FMS with a circular layout, i.e., the stations are arranged in a circle and the workpieces move around this circle with the possibility of either entering a machine or continuing on the circle. The operation of this manufacturing system is as follows. A worker loads a workpiece on a pallet at the loading station and feeds it into the manufacturing system. The workpiece is transported to a machine over the circular conveyor system and processed (automatically) by the machine. The processing is continued at other machines until the workpiece is finished, i.e., has received all operations prescribed by its process plan. As a final step, the workpiece is returned to the unloading station and taken off the pallet. Occasionally, if different workpieces are competing for the same machines, some workpieces have to be temporarily stored in the buffer until their machines are free to process them.

A flexible manufacturing system is used for job-shop production in highly automated environments. Machines are usually computerised numeric control (CNC) machines which are able to perform almost any operation if programmed accordingly. The greatest disadvantage of FMS, however, is the low volume and the high costs per product.

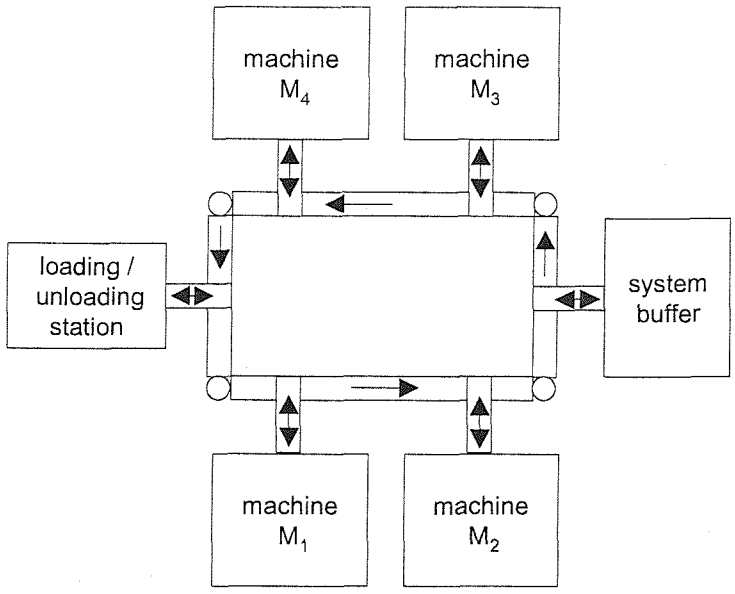

Figure 2.3: A flexible manufacturing system.

Because of their flexibility, CNC machines are slower and more expensive than dedicated machines. Large-series and mass production is therefore organised as a so-called ***line production***. The new V-engine assembly plant (NVM) of DaimlerChrysler in Stuttgart (Germany) is a typical example of a line production system (Weber 1997, Bussmann 2001). The assembly system consists of approximately 60 stations which are mostly linearly connected, performing engine block assembly, cylinder head assembly, final assembly, and testing. Around half of the stations are automatic, the remaining require workers to perform the assembly step. At several stations, mostly for heavy parts, material is supplied through automated buffers. Block and cylinder head assembly are schematically shown in figure 2.4.

The assembly starts with the crankcase, which is put on a pallet at the first station. The pallet then runs nearly linearly through every station until it reaches the shipping station at the end of the assembly system. Since all stations are dedicated to a single operation for the given product, the stations operate at a short cycle time (about 60 seconds). Because of this short cycle time, the assembly system is able to produce a much higher volume at comparably lower costs than the flexible manufacturing system. The disadvantage of the assembly system, though, is that it has almost no product or volume flexibility. It can only produce a specific product family at a fixed volume.

Because of the relative advantages and disadvantages of the two production systems, there is no dominant type of production - both types (and many more) can be found in industry.

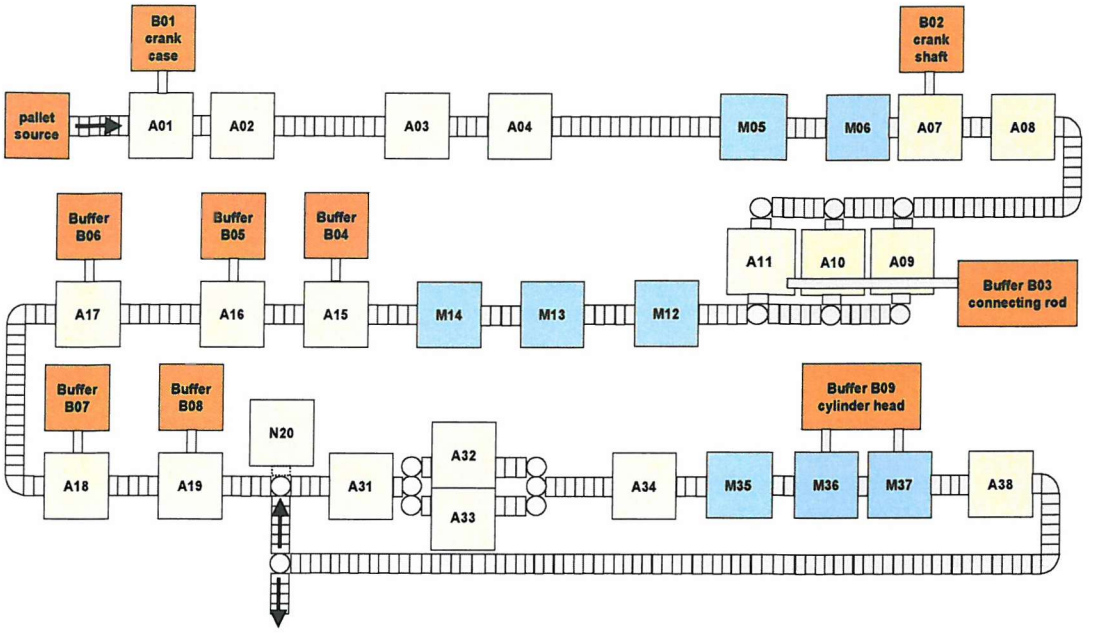

Figure 2.4: The block and cylinder head assembly of NVM with automatic (A), manual (M) and rework (N) stations.

2.1.2 Production operation

A production system, as defined above, consists only of the physical resources necessary to perform the production functions. To actually run a production process, however, it is also necessary to perform several management functions that prepare or guide the production process. These management functions are commonly distinguished into production planning and production control activities (Groover 1987, Hoitsch 1993, Hitomi 1994).[5] ***Production planning*** includes any activity for planning the *production program* (how many products of what type to produce), the *production process* (in which steps to create the product) and the *production factors* (how much equipment, labour, and auxiliary material is needed). ***Production control***, on the other hand, takes orders from the production program and assigns them to resources (*resource allocation*), initiates processing and handling of workpieces (*execution*), and supervises the performance of the production system (*monitoring*). Production control is thus the link between the planning processes of a manufacturing firm and the actual execution of the plans at the shop floor. As Groover and Monden emphasise, the main responsibility of production control is to achieve the production plans:

[5] Naturally, there are more management activities necessary in a manufacturing firm (such as product development or sales). This work, though, focuses on those functions that are closely related to the actual production process.

> *Manufacturing control* is concerned with managing and controlling the physical operations in the factory to implement the manufacturing plans. (Groover, p. 26)
>
> Controls the production of the necessary products in the necessary quantities at the necessary time in every process of a factory and also among companies. (Monden 1983)

To achieve the production plans, production control must decompose the production program into instructions for the shop floor. The following definition of production control emphasises this task:

> Production control is the function of management which plans, directs, and controls the materials supply and processing activities of an enterprise. Where, planning is the process of deciding what to do in the future, directing comprises the operation of issuing orders, and control can be described as the constraining of events to follow plans. (Burbridge 1978)

What makes production control challenging, however, is that it is not enough to simply decompose a production program into single shop floor actions and to issue these actions as instructions. Physical processes fundamentally carry the risk of failure. Tools may break, operations may miss tolerances, transportation devices may jam, and so on. These contingencies must be taken into account when running a production process (Parunak 1991). Some researchers have even defined (production) control to be only a monitoring and correcting activity:

> [The task of production control:] Whenever the actual production progress and performances deviate from the production standards (plans and schedules) set at the planning stages [...], such deviations are measured and modifications are appropriately made. (Hitomi 1994, p. 420)
>
> *Shop floor control* is concerned with the problem of monitoring the progress of the product as it is being processed, assembled, moved, and inspected in the factory. (Groover, p. 27)

To optimally run a production process, it is thus necessary to choose the production instructions such that these fulfil the production plans. But also vice versa, the production instructions should be chosen with respect to the reality on the shop floor and what can be reasonably expected to be executed. Production control should thus take the production plans, monitor the production performance, and issue those production instructions that optimally achieve the production plans given the current situation on the shop floor (see figure 2.5). Within the context of this work, production control is therefore defined as follows (see also (Dean 1991)):

Definition: ***Production control*** is the process of choosing, initiating, and monitoring actions in a production system in order to achieve or optimise a given production program.

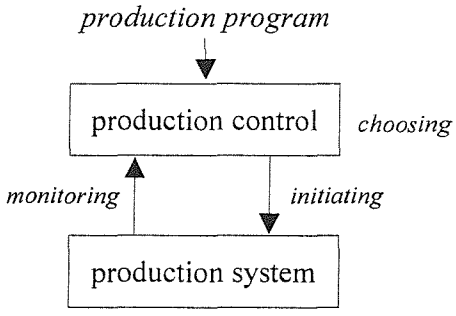

Figure 2.5: Production control loop.

Note that the above definition – as all other definitions of production control in the literature – deliberately excludes the actual execution of an action (the action is only *initiated*). The reason is that the execution of an action requires an actuator to physically perform a movement (or some other physical state change) in the production system. There is a large body of technology, commonly referred to as *control theory* (Dorf 1998), that is able to control these state changes (at least for discrete manufacturing). With this technology, machines can drill holes, robots can assemble parts, and switches can move pallets to their exits. The focus of this work, and thus of the above definition, though, is to determine which hole to drill, which parts to assemble, and where to move the pallet (i.e., how to decompose the production plans into instructions for the shop floor). It can be safely assumed that, with the existing control technology, the instructions can be executed. Note that the above definition also excludes most of the real-time issues because in discrete manufacturing these can be handled by existing control technology at lower levels when executing the instructions.

To clearly identify the interface between production control – as defined above – and the control components actually executing the production actions, the latter components will be called ***(local) controllers*** in the rest of this work (see also (Veeramani 1994, p. 553)). That is, it is assumed that every physical component supposed to execute actions is associated with a control component providing an appropriate interface to the production control system (cf. figure 2.6).

2.1.3 State-of-the-art and limitations

The classical approach to production control is best characterised as *hierarchical* and *schedule-driven*. First of all, control systems are organised in a command hierarchy, in which sub-ordinate units are only supposed to execute the commands given by the

super-ordinate levels (see also proper hierarchies in (Dilts 1991)). Secondly, the control process starts at the top with an abstract scheduling of the production program (Hitomi 1994, Becker 1994) and consecutively details and distributes this production schedule on each level until executable actions reach the controllers (see figure 2.6). In particular, each level of the hierarchy creates production schedules for its subordinate units with only minimal feedback from the lower levels. An adaptation of the schedule is only done within one unit with hardly any consultation of neighbouring or superior units. Failures to achieve a production schedule are fixed within the next scheduling cycle.

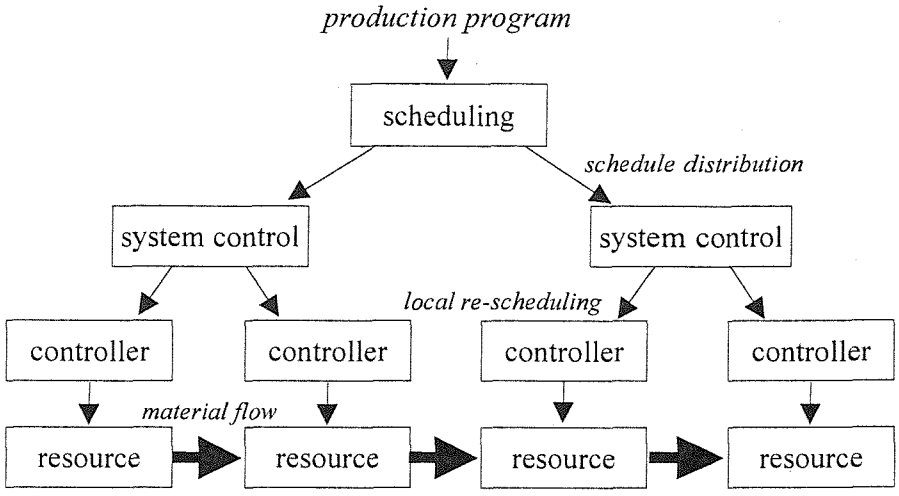

Figure 2.6: Classical approach to production control.

This approach works optimally if actions are executed as planned, but fails completely otherwise. In case of a disturbance, a controller is not able to execute its actions or has to postpone them. Since production operations are optimised in order to maximise productivity and minimise costs, resource capacities are fully utilised and buffer sizes are reduced to an absolute minimum. As a consequence, any deviation from the schedule quickly affects neighbouring units resulting in a cascading effect of the disturbance. Since the hierarchical and schedule-driven control organisation does not support system-wide re-scheduling, the impact of a disturbance on production cannot be constrained. As every real production system is regularly affected by disturbances (Parunak 1991), production operations soon deviate from the production schedule. It is even "proverbial among shop foremen that the schedules produced by the front office are out of date the moment they hit the [shop] floor" (Parunak 1987, p. 303). This gap between the "reality of planning" and the "reality of the shop floor" (Scherer 1998) will increase even further as production is faced with accelerating business trends towards more product complexity and volatile markets (Chokshi 2002, McFarlane 2003).

To overcome the limitations of the current approach to production control, the planning

process must be pushed down the hierarchy and interleaved with execution (Hatvany 1985, Scherer 1998b). Each unit must have the freedom to choose the right actions depending on the current situation. To achieve this freedom, the production program must be decomposed and distributed as subgoals to the subordinate units. Each unit then pursues its assigned goals and, in doing so, co-operates with other units to ensure that the fulfilment of the subgoals leads to the fulfilment of the production program. The production units thus retain their freedom in order to react to unforeseen events, while the production program emerges from the interaction of the previously created subgoals (see figure 2.7).

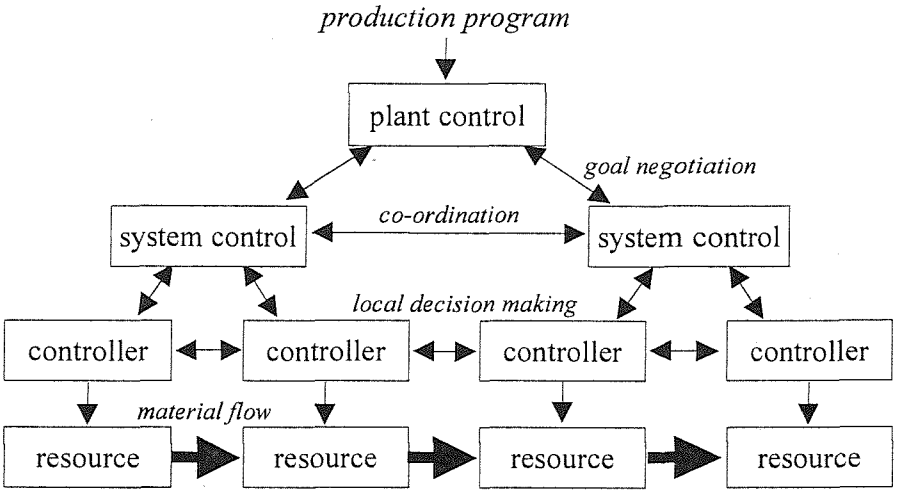

Figure 2.7: Goal-driven production control.

Such a goal-driven and co-operative approach to production control not only supports robustness, but also increases flexibility and reconfigurability of the production process (Duffie 1988) – two requirements which are becoming even more important than robustness in a global economy with decreasing product life-cycles. A distribution of goals implies the distribution of the corresponding decision making, and thus empowers the controllers not only to react to disturbances on the shop floor, but also to adapt to changes of customer orders. It becomes transparent to the controller where changes come from and it can apply the same techniques for coping with disturbances to handle changes. Furthermore, the distribution of control functionality to local controllers increases their ability to act irrespective of the context a controller is placed in. Changes in the configuration of the production system are treated just like all other internal or external changes to which a controller constantly adapts its goal achievement. The combination of robustness, flexibility, and reconfigurability – enabled by the goal-driven approach – thus leads to a truly agile performance in an increasingly dynamic production environment.

2.1.4 New requirements on production control

A goal-driven approach, however, places new requirements on the design of production control systems (Brennan 2003, McFarlane 2003). First of all, the control should be distributed to physical components of the production system instead of being divided into central control functions (Duffie 1988). A local controller will require all the decision capabilities necessary to choose the optimal actions under varying production conditions given its local goal. Scheduling, for instance, will no longer exist as a single central control function, but will be distributed over the factory floor with every controller being able to do its own (local) scheduling.

Secondly, a local controller must be equipped with reactive and goal-directed decision making capabilities (Valckenaers 1999). While the reactiveness ensures that the controller is able to adapt to any change or disturbance, the goal-directedness guarantees that the controller eventually reaches its goal despite the dynamics of its environment. In particular, the controller must be goal-directed in that it is able to decompose its goals into the necessary actions and to initiate these actions at its own or other resources.

Thirdly, the local controllers must co-operate in a flexible manner (Hatvany 1985, Duffie 1988). If a disturbance cannot be constrained within the responsibility of a single controller, this unit has to co-ordinate its actions with neighbouring units to reduce the impact of the disturbance or to re-negotiate its goals with superior units if the goals can no longer be met. Co-operation thus makes it possible to treat a failure to achieve a goal immediately at the appropriate level of the hierarchy, and therefore keeps the deviation from the overall goals to a minimum. The co-operation process, however, cannot be fixed at design time because changes may require a controller to adapt its interactions just as it adapts its actions.

Finally, a controller should follow the strategy of *low and late commitment* (Valckenaers 1994). An early or over-constraining commitment to an action may turn out to be sub-optimal or even counter-productive if the situation changes in the meantime. A decision that is made at the latest moment possible with the least commitment minimises the probability that events change the situation. A low and late commitment thus increases robustness, but also supports flexibility as controllers gain the maximum freedom to adapt to the changing production conditions.

A production control system that fulfils these requirements will operate quite differently in comparison to the hierarchical control systems (Dilts 1991). This is particularly true for the operation of the local controllers. While in the hierarchical and schedule-driven approach a controller only executes the given schedule (determined by a central scheduler), the new controller must autonomously choose an appropriate action in its current situation, and, at the same time, be able to co-operate with the

neighbouring controllers in order to optimise the overall system performance (Duffie 1988). In short, the new controller must be an *autonomous and co-operative decision maker.*

2.1.5 Summary

Current production control systems are hierarchically organised and separate scheduling from execution. Because of this separation, there exists a constant gap between schedule and shop floor reality that leads to a significant decrease of the performance. To overcome this gap, future control systems require controllers that are capable of goal-based decision-making and co-operation with other controllers. To guide these controllers, the production program must be decomposed and distributed in terms of goals instead of schedules. The production program is then achieved through the flexible decision making and interaction of the local controllers. In the next section, it will be shown that agent technology is ideally suited for modelling and implementing such a goal-driven approach.

2.2 Software agent technology

Software agent technology has been a very active field of research for more than two decades now (see (Wooldridge 2002) for a brief history of the field). It started in the early 1980's as a sub-field of artificial intelligence, namely as distributed artificial intelligence, focusing its research on aspects of multi-agent planning and distributed problem solving (Bond 1988). Since then, it has constantly attracted more researchers who have expanded the field to cover now the whole range from rational over emotional to social agenthood (AAMAS 2002). Today, even though the research field of agent technology is still very heterogeneous and lacks a unifying umbrella, it has established itself as a major stream of computer science and artificial intelligence in particular.

One stream of agent research, often referred to as *multi-agent systems*, is of particular relevance to the domain of production control. Multi-agent systems research investigates how to model and implement individual and social behaviour in distributed systems. It explores on the one hand notions like autonomy, reactivity, and goal-directed reasoning in order to model and implement (rational) individual behaviour (see for example (O'Hare 1996, Huhns 1998, Müller 1996)). On the other hand, it examines aspects of co-operation, co-ordination, negotiation, coalition formation, role assignment, and self-organisation in order to create social behaviour (see for example (O'Hare 1996, Weiss 1999)). To date, multi-agent systems research has already developed theories for social and individual behaviour, agent architectures,

communication and co-operation techniques, as well as new programming languages (Wooldridge 1995, Meyer 2002).

Despite the significant amount of research undertaken in multi-agent systems, however, there is still no universally accepted definition of an agent or a multi-agent system. Some researchers define an agent in terms of mental states such as beliefs, capabilities, choices, and commitments (Shoham 1993); others stress the ability of an agent to act autonomously in a dynamic environment (Weiss 1999); while still others include in their spectrum of agenthood properties such as adaptability, personality, or mobility (see also (Bradshaw 1997, Franklin 1997, Nwana 1997, Huhns 1998, d'Inverno 2001) for a selection of agent definitions and viewpoints). In the context of this work, the definition of an (intelligent) agent by Wooldridge and Jennings (Wooldridge 1995) is adopted because their definition equally stresses the ability of an agent to autonomously make decisions in a dynamic environment as well as the ability to flexibly interact with other decision makers. Other aspects, like adaptability or mobility, are not relevant to all control applications and should therefore not be included in a general definition of an agent.

Definition: An ***agent*** is a software process with the following properties (Wooldridge 1995):

- "*autonomy:* agents operate without the direct intervention of humans or others, and have control over their actions and internal state [...];
- *social ability:* agents interact with other agents (possibly humans) via some kind of *agent-communication language* [...];
- *reactivity:* agents perceive their environment [...], and respond in a timely fashion to changes that occur in it;
- *pro-activeness:* agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by *taking the initiative*."

A multi-agent system is then simply defined as a collection of agents that somehow interact.

Definition: A ***multi-agent system*** is a collection of interacting agents.

The remainder of this section will review techniques to model and implement multi-agent systems. Since multi-agent systems research has produced (and is still producing) a large variety of techniques, it is not possible within the limits of this work to review all results. This review will therefore focus on those aspects of multi-agent systems which are relevant to production control. In particular, subsection 2.2.1 will review

existing agent models for building production control agents, whereas subsection 2.2.2 will focus on techniques suitable for the interaction between such agents.

2.2.1 Agent models

According to the above definition, the first fundamental property of an agent is its ability to act *autonomously*. That is, an agent is situated in an environment, in which it can sense and act (see figure 2.8), and has complete control over its own actions in that environment (Wooldridge 1999, p. 29).

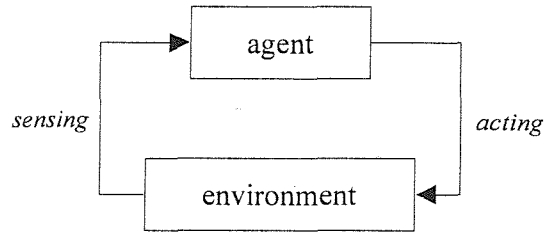

Figure 2.8: Basic model of an agent.

The property of autonomous action, though, furthermore implies that the agent is able to choose its actions on its own – without the direct intervention from outside. The question – around which much of agent research is centred – is thus how can and should an agent choose its actions. As Wooldridge notes: "The key problem facing an agent is that of deciding *which* of its actions it should perform in order to best satisfy its design objectives." (Wooldridge 1999, p. 30). To solve this decision problem, several agent models and architectures have been proposed. Here, three main types of agent architectures are briefly reviewed:

- reactive agents
- deliberative agents
- hybrid agents

2.2.1.1 Reactive agents

In reactive agent architectures, the sensory input is directly linked to the action capabilities of an agent. That is, the designer of a reactive agent specifies, for each possible sensory input, which action the agent should (immediately) perform upon this input. During execution, the agent thus only needs to repeatedly take its sensory input and match this against the conditions for each action. A problem, though, arises if more than one condition matches the same sensory input, and the actions associated with these conditions are in conflict. To overcome this problem, Brooks proposed the

subsumption architecture (Brooks 1986). Put simply, each condition, also called behaviour in this architecture, is associated with a list of lower-prioritised behaviours which it inhibits (see figure 2.9). A behaviour thus only executes its action if no other behaviour inhibits this behaviour (for more details see (Brooks 1986)).

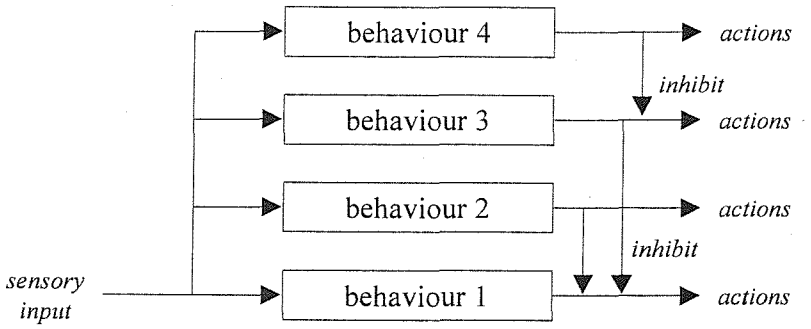

Figure 2.9: Subsumption architecture.

Reactive agent architectures clearly implement autonomous agents. The agents choose on their own – even if this choice is hardwired – how to react to a specific situation in the environment. Obviously, these agent architectures ensure the reactiveness of an agent. The main disadvantage of these architectures, however, is that it is difficult to implement pro-activeness and goal-directed behaviour in such an architecture (Wooldridge, p. 53). Reactive architectures only look at the current situation and have no means to initiate behaviour or take into account longer term goals. In the extreme, purely reactive agents do not act unless something in the environment changes.

2.2.1.2 Deliberative agents

Deliberative agent architectures explicitly represent goals and form plans about how the agent wants to behave in the future in order to achieve its goals. Probably the most prominent deliberative agent architecture is the belief-desire-intention (BDI) architecture of Rao and Georgeff (d'Inverno 1998, Rao 1992). In this architecture, the agent takes its beliefs, i.e., the sensory input accumulated over time, and its desires, i.e., its goals, and forms intentions about what it is going to do in the future (see figure 2.10). Intentions are courses of actions to which an agent commits itself, i.e., the agent will execute these actions unless certain specified situations arise in which the agent has to abandon its intention. The current set of intentions formed then determines which actions the agent should perform.

In contrast to reactive architectures, BDI agents are able to follow their goals pro-actively and, at the same time, to react to their environment. In case the beliefs (derived from the sensory input) no longer support an intention, the intention is either changed or abandoned and thus the behaviour of the agent is adapted. The BDI agent

architecture is thus able to implement autonomous, reactive, and pro-active agents. A disadvantage of BDI agents, however, is that they can only react after the new sensory information has run through the different steps necessary to form an intention (i.e., it is necessary to form a belief about the change and to adapt the intentions before the corresponding actions can be executed). For complex agents, this may take a long time, in highly dynamic environments possibly too long. To avoid this problem, hybrid architectures were proposed.

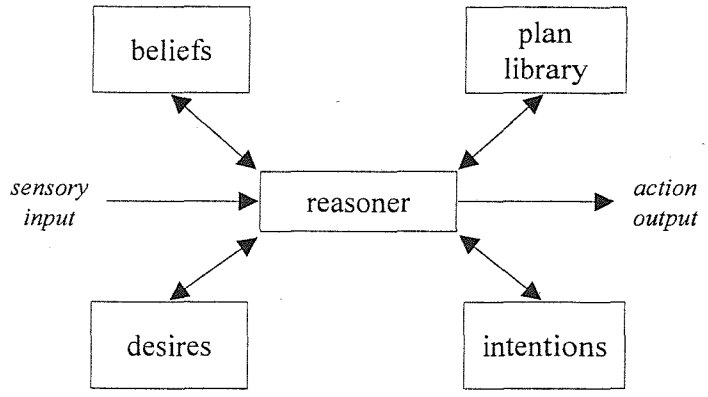

Figure 2.10: BDI architecture.

2.2.1.3 Hybrid agents

Hybrid agent architectures incorporate both reactive and deliberative mechanisms into one architecture (typically by introducing a layer for each mechanism). The InteRRaP architecture (Müller 1996), for instance, consists of three layers (each equipped with a database): a behavioural layer for reactive situation-action rules, a plan layer for goal-directed pro-active planning, and a co-operation layer for modelling and handling interactions with other agents (see figure 2.11). In this architecture, the sensory input is first provided to the behavioural layer. If one of the situation-action rules is applicable to the input, the rule 'fires' its actions. If no rule matches the input, the sensory input is handed to the next higher layer. Here again, the layer decides whether it can handle the input (e.g., through planning) or hands it further to the next layer. When a higher layer chooses a certain course of action, this information is handed down the hierarchy to the lower levels for execution. That is, whatever any of the layers decide, the behavioural layer must execute the corresponding actions.

Hybrid architectures fulfil all requirements imposed on an agent. A hybrid agent is autonomous, reactive, pro-active, and – in many cases because of a co-operation layer – also capable of social behaviour. The only disadvantage of hybrid architectures is that it is difficult for the designer of such an agent to co-ordinate the different layers in order to produce a coherent agent behaviour. In particular, there is still no clear semantics or

methodology for programming such a kind of architecture, as it exists for example for BDI architectures (Wooldridge 1999).

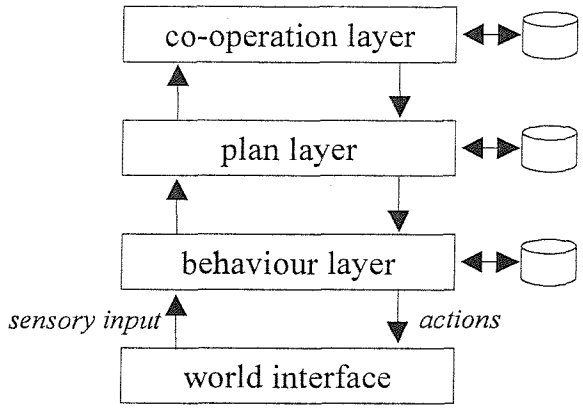

Figure 2.11: InteRRaP architecture.

2.2.2 Agent interaction

As defined above, the second fundamental property of an agent is its ability to interact with other agents. The environment an agent is situated in is usually not purely passive. There are often other agents which also act autonomously and pro-actively. To meet its goals, an agent may either have to avoid negative or be able to exploit positive interactions with other agents. In particular, certain goals may not be achievable with the limited capabilities of a single agent, but only if a whole set of agents works co-operatively towards these goals. Interaction may thus be indispensable to the goal achievement of an agent.

In general, ***interaction*** is any kind of information exchange that somehow influences the actions of another agent (Bond 1988). Interactions can thus take many different forms. This subsection, though, will only briefly review those main types of interaction techniques, namely *co-ordination* and *negotiation*, that are most commonly used in agent-based production control. Other forms and in particular a more thorough treatment of interactions can be found in (O'Hare 1996, Weiss 1999, Wooldridge 2002).

2.2.2.1 Co-ordination

Co-ordination is the process by which agents ensure that their community acts in a well-defined manner (Jennings 1996, Bond 1988). While co-ordination itself is nearly invisible, the lack of co-ordination becomes immediately apparent. In a well-run conference everything happens as expected, while in a badly organised conference the

participants immediately recognise if something goes wrong; if, for example, the conference rooms are not clearly marked, the presentation is not set up in time, or the coffee is not served at the beginning of the breaks. The problem of co-ordination becomes particularly difficult in a multi-agent system where there is no central control and every agent can autonomously choose its actions. In such a system, co-ordination is not an inherent system property, but can only be achieved through an explicit effort of the agents. Jennings (Jennings 1996) gives three main reasons why co-ordination may be difficult to achieve:

1. The actions of the agents may interfere.

   Two robots reaching for the same workpiece, for instance, will collide.

2. There may be global constraints to be met.

   The processing of a workpiece at different machines may have to be scheduled such that the workpiece is finished by a certain deadline.

3. No individual agent has sufficient capabilities or resources to achieve its or the system's goals.

   A single machine is usually not sufficient to perform all the operations required by a workpiece. In contrast, to process a workpiece, several machines, handling systems, and possibly other resources must co-ordinate their actions.

All of the above cases have in common that there is some kind of dependency between the agents. The agents are dependent on each other for performing (or avoiding) certain actions in order to arrive at the desired overall system behaviour. Malone and Crowston therefore define co-ordination to be simply "managing dependencies between activities" (Malone 1994, p. 90). Or, as they say, "if there is no interdependence, there is nothing to coordinate" (Malone 1994, p. 90). To achieve system co-ordination, it is therefore necessary to first understand the possible dependencies that may exist and then to derive interaction techniques that are able to handle these dependencies.

Several researchers have modelled and classified dependencies between agents. Castelfranchi et al. (Castelfranchi 1992) as well as Sichman et al. (Sichman 1994) have modelled the goals and plans of agents and have investigated how these can be dependent on other agents' goals and plans in order to analyse how co-operation can evolve from these dependencies (see also (Jennings 1996, d'Inverno 2001, Sichman 2002, Yu 2002)). In particular, they have distinguished:

- *unilateral dependence*: One agent is dependent on another, but not vice versa.
- *mutual dependence*: Two agents are dependent on each other for the same goal or plan.

- *reciprocal dependence*: Two agents are dependent on each other, but for different goals or plans.

Malone, Crowston et al. (Malone 1994, Crowston 1994, Malone 1999) have intensively studied co-ordination and the underlying dependencies in organisational processes. To study these processes, they have adopted a task/resource model and have identified the types of dependencies that can arise in such a process model. Malone et al. (Malone 1999), for instance, distinguish between fit, flow, and sharing dependencies (see figure 2.12).

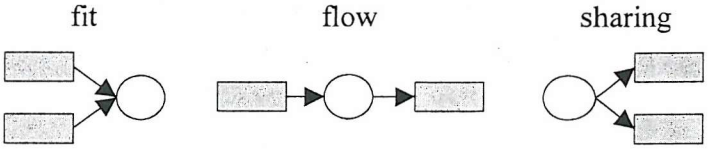

Figure 2.12: Task/resource dependencies.

In a *fit dependency*, multiple activities collectively produce a single resource and have to make sure that whatever is created fits together. A *flow dependency* arises whenever one activity produces a resource that is to be used by another activity. This resource must then be provided at the right time ('prerequisite' dependency), at the right place ('accessibility' dependency), and in the right form ('usability' dependency). Finally, there may be a *sharing dependency* between activities, i.e., two (or more) activities use the same resource. In this case, Crowston (Crowston 1994) further distinguishes whether the resource is *shareable* or *non-shareable* by more than one activity, and whether it is *re-usable* or *non-reusable*; in the latter case, the resource is totally or partly consumed after its usage.

Von Martial (von Martial 1992) and Decker and Lesser (Decker 1992b) have studied dependencies that may arise between multi-agent goals or plans. Von Martial has proposed a set of plan relations that capture positive or negative dependencies between several agents. Obviously, agents executing their individual plans may run into conflicts concerning the usage of resources or the achievement of (incompatible) states in their common environment. But agents may also encounter positive plan relations. Their planned actions may be identical or may subsume each other so that the agents save time and effort if they co-ordinate their plans. Von Martial also identified the *favour* relationship in which a plan of one agent does not make the plan of another agent obsolete, but significantly reduces the effort the other agent has to put into its plans.

Similarly, Decker and Lesser studied possible relationships in augmented goal structures – an abstract representation of task and goal activities – and derived a whole range of possible relationships in (Decker 1992, Decker 1992b) (for a formal model of plan relationships see also (Ossowski 1999)):

- basic domain relationships: *enables*, *facilitates*, *hinders*, *precedes*, *causes*, *share-results*, *cancels*, and *favours*;
- goal relationships: *overlaps*, *necessary*, *sufficient*, *extends*, *subsumes*, and *competes*;
- temporal relationships: *before*, *equal*, *meets*, *overlaps*, *during*, *starts*, *finishes*, and their inverses; and
- resource constraints: *use*.

To handle these goal and plan relationships, Decker and Lesser (Decker 1995, Decker 1992) generalised the *partial global planning* (PGP) approach of Durfee (1996). In PGP, distributed planners co-ordinate their actions by abstracting from their plans and exchanging these abstractions. Given the different local plan abstractions, each agent is then able to identify common goals to which the local goals of the agents contribute. Since these common goals may be only partially known to the agents, they are called *partial global goals*. Once a partial global goal has been identified, the local plans can be integrated into partial global plans. In its original design, though, PGP provides only two mechanisms to perform this integration: redundant tasks are avoided, and tasks are performed earlier if this facilitates the work of other agents. Decker and Lesser therefore extended PGP into *generalised partial global planning* (GPGP). GPGP generalises PGP first of all because it uses the domain-independent goal relationships described above. But GPGP also provides a set of domain-independent co-ordination mechanisms:

1. *Updating non-local viewpoints*
   The agents exchange local information in order to detect new co-ordination relationships.
2. *Communicating results*
   The agents communicate all the results of their activities ('all' policy) or only those which are necessary to satisfy any commitments to other agents ('minimal' policy).
3. *Handling simple redundancy*
   Once redundant actions planned by more than one agent are detected, the agents randomly choose one agent to execute the action and communicate the results.
4. *Handling hard co-ordination relationships*
   The agents schedule activities such that any hard temporal ordering constraints are obeyed.
5. *Handling soft co-ordination relationships*
   As for the previous co-ordination mechanism, the agents try to schedule

activities such that ordering constraints, like for example *facilitates*, are obeyed if the corresponding schedule is acceptable to the agent.

Despite the domain-independent approach, GPGP is not – and was not intended to be – a complete set of co-ordination mechanisms. In contrast, the co-ordination relationships as well as the co-ordination mechanisms can be extended to accommodate new applications, as was done in (Decker 2000) by introducing the mutual exclusion relationship and a co-ordination mechanism for mutual exclusion based on bidding.

Other co-ordination techniques that have been developed include, for example, distributed problem solving (Durfee 1999), search algorithms (Yokoo 1999), and social laws (Shoham 1995, Fitoussi 2000). Distributed problem solving is concerned with distributing a system task or sharing distributedly computed results in order to make use of the capabilities of several problem solvers. Search algorithms explore a search space in a distributed manner. Distributed constraint satisfaction algorithms, for instance, are able to solve a constraint net if variables (or constraints) of the net are distributed to different agents. Social laws, finally, attempt to avoid the interactions of agents in the first place. They define behavioural conventions which make any explicit co-ordination obsolete. Robots, for example, which always move on the right side of a (sufficiently large) passage, never collide – even without communication. If such conventions exist and the agents can be expected to obey them, social laws significantly reduce the co-ordination requirements. Dynamically identifying suitable social laws and agreeing upon them, though, can be a co-ordination problem in itself.

2.2.2.2 Negotiation

As for co-ordination, negotiation is a common form of interaction between human beings and has therefore been extensively studied in sociology (see for example (Pruitt 1981)). In sociology, any (human) interaction is regarded as a negotiation in which the participants of the interaction, usually called *parties*, have a conflict of interests, but must come to a joint decision. Pruitt (1981), for instance, defines ***negotiation*** as follows:

> Negotiation is a process by which a joint decision is made by two or more parties. The parties first verbalize contradictory demands and then move towards agreement by a process of concession or search for new alternatives.

Conflicts are a natural part of our (human) life, since everyone has their own special interests. But with autonomous agents, conflicts also become a natural phenomenon in software systems. If each software agent has its own interests and goals, conflicts between software agents are inevitable. One of the objectives of multi-agent systems research is therefore to enable software agents to perform negotiations in order to

resolve any conflicts (Wooldridge 2002, p. 129). To achieve this objective, many of the negotiation forms existing in human societies have been adopted or extended for software agents. The most prominent examples of these negotiation techniques will be reviewed in the following. This will include auctions, general equilibrium market mechanisms, service negotiation, and conflict resolution techniques.

Auctions are trade mechanisms for exchanging commodities (Friedman 1991). The commodities can be any kind of goods, but usually some good is exchanged for money, as for example in a stock exchange. There are one-sided and two-sided auctions, as well as continuous auctions. In a *one-sided* auction, there is one auctioneer that either only accepts bids – if he wants to sell a good – or only accepts asks – if he wants to buy a good. The result is always a single trade (for the good that was auctioned off). In *two-sided* auctions, the auctioneer allows the agents to place bids and asks and then matches these in order to create several trades. Finally, an auction is called *continuous* if it allows bids and asks to arrive over time (and matches both over time). For each of the above auction categories there already exists a large set of possible auction protocols, and probably many more will be developed in the future (see (Sandholm 1999, Friedman 1991)). In the following, only three typical examples of common auction protocols are briefly characterised, namely the English auction, the continuous double auction, and the contract-net protocol.

The standard example for a one-sided auction is the *English auction.* In the English auction, an auctioneer wants to sell a (specific) good for the highest price possible and requests a set of potential buyers to make bids. The (potential) buyers respond with bids and increase their bids until no bidder offers a higher price. The auctioneer then sells the good to the bidder with the highest bid for the price of the highest bid.

In the *continuous double auction* (CDA) participants may pose bids or asks. The bids and asks are continuously matched by a neutral auctioneer according to a set of market rules. The rules basically state that bids and asks are matched if the price of the bid is higher than the price of the ask. The actual auction rules, though, depend on the specific CDA installation (see (Friedman 1991) for a discussion).

The *contract-net protocol* (CNP) is a simple, but efficient protocol for assigning tasks to individual nodes in a network (Smith 1980). It assumes that one node has a task that needs to be executed (by another node) and that there are (potentially) several nodes that are able to execute this task. The node with the task is called the *manager* and the other nodes are (potential) *contractors* (see figure 2.13). The manager initiates the protocol and proceeds as follows. First, it announces the task to the potential contractors. The contractors answer with a bid. The necessary information provided in the bid was specified by the manager in the announcement message. The manager compares the bids and chooses the best bid according to its preferences. The node which has sent the best bid then receives an award message and is said to have a

contract with the manager about the execution of the task. The other nodes may or may not receive a reject message.

Even though the CNP was originally designed for task distribution, it is actually a *one-sided first-price sealed-bid auction.* First of all, it is irrelevant to the protocol whether the agents exchange a task or some other kind of good. Secondly, the CNP simply stops after the auctioneer received the first bid from each bidder ('first-price'), instead of repeating the auction until no bidder changes its bid. And thirdly, the bid is 'sealed', i.e., not made known to the other bidders. In the English auction, it is a fundamental prerequisite that the bids are made 'open-cry', i.e., are broadcast to all agents because otherwise the other bidders do not know whether they need to raise their bid in order to win the auction. The contract-net protocol can thus be viewed as an auction mechanism, even if there may not necessarily be a conflict between the manager and the potential contractors.

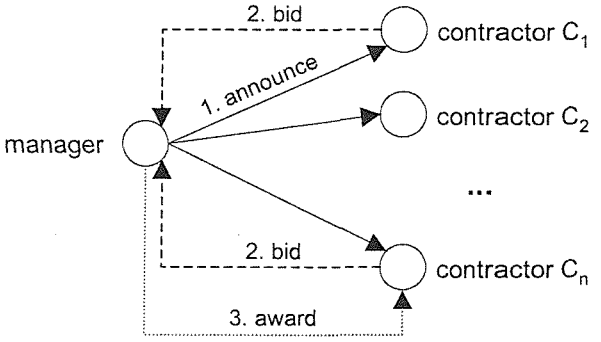

Figure 2.13: The contract-net protocol (and its phases).

All of the above auctions have in common that they are able to match buyers and sellers, but, in doing so, hardly take into account any global constraints or optimisation criteria. ***General equilibrium market (GEM) mechanisms*** (Sandholm 1999), on the contrary, are able to optimise the allocation of goods and resources among agents through the identification of a *market clearing price.* That is, the market price for each good is varied until an optimal allocation of goods is found. More precisely, a GEM consists of producers and consumers which create or consume goods. The actual production or consumption of each agent, however, depends on the market price for each good. The higher the price, the less the consumers will consume and the more the producers will wish to produce. And, vice versa, the lower the price, the more the consumers will consume and the less the producers will wish to produce. The market is thus in equilibrium if:

(i) the amount of goods consumed is equal to the amount of goods produced;

(ii) the consumers are able to consume the maximum amount of goods they demand given the prices; and

(iii) the producers maximise their profit (which is price multiplied by the amount produced).

The task of a market clearing algorithm is to find such a clearing price. A common algorithm for clearing markets is the distributed price tâtonnement algorithm (Sandholm 1999). In this algorithm, each producer and consumer determines, given the current prices, how much it will produce or consume in order to maximise its profit or utility function. A central price controller then adjusts the prices according to the difference between the production and consumption plans. This process is repeated until production and consumption plans reach an equilibrium. Only then may production and consumption start. The GEM markets are thus a static approach: the solution must be completely known before execution can start.

Another drawback of auctions, and also of the GEM mechanisms, is that they can only negotiate prices (or the combination of prices and amount), whereas the actual good to be exchanged is fixed. In many settings, though, it is also necessary to alter other aspects of the negotiation subject in order to find an agreement. For instance, several agents planning a joint holiday will have to consider different ways of making holidays (staying at a beach hotel, hiking through the mountains, visiting an exciting city, and so on) in order to devise a trip that satisfies the interests of all agents. For these settings, it is therefore necessary to employ more sophisticated negotiation techniques. Two examples of such negotiation techniques are briefly reviewed.

Faratin (Faratin 1998) developed a sophisticated interaction model for bilateral negotiation of services. In this model, the two negotiation agents basically exchange new proposals until either both agents accept the last proposal or one of the agents withdraws from the negotiation. This interaction process is complemented by a set of negotiation tactics and strategies for generating new proposals. Faratin defined tactics for reacting to time or resource constraints, for imitating an opponent's behaviour, for trading off different aspects, or for manipulating the set of negotiation issues (Faratin 2000). These tactics are combined into strategies by assessing the influence of a tactic on the generation of the next proposal. With these negotiation techniques, an agent is able to exhibit a more flexible negotiation behaviour than to merely change prices. In particular, the agent can trade off different aspects of the negotiation subject.

Hollmann et al. (Hollmann 2000) proposed an interaction technique for resolving conflicts between agents that are supposed to find a common proposal (on some matter). The task of the agents is thus to search for a proposal which satisfies all the agents. An agent is 'satisfied' with a proposal if the evaluation result of the proposal passes a given threshold. To find such a proposal, each 'unsatisfied' agent is requested

to put forward a new proposal that only minimally changes an already existing proposal such that the evaluation function of the agent is satisfied. These new proposals are generated until either a proposal is found satisfying all the agents or it can be asserted that there is no solution. The advantage of this interaction technique is that it is able to negotiate complex matters between several agents. However, the disadvantage is that it is not able to reconcile opposing interests. The agents make new proposals, but there is no explicit process of concession (of course, the agents could always concede by themselves). This negotiation technique can thus only be applied if there is a solution to which all agents would agree.

There are certainly many more negotiation techniques that could be reviewed here. This review, though, was only intended to give an impression of the different techniques that exist and are potentially relevant to production control. For more complete overviews of negotiation techniques see (Sandholm 1999, Jennings 2001b, Wooldridge 2002).

2.3 Agent-based control systems

The review of agent technology has shown that software agents are an appropriate technology to meet most of the new requirements for modern production control systems (see subsection 2.1.4). First of all, multi-agent systems distribute the decision capabilities of the control system to the individual agents of a production system, usually to the local controllers. Secondly, an agent is by definition a reactive and pro-active decision maker (see subsection 2.2.1). And thirdly, also by definition, agents are able to co-operate in a flexible manner, either to improve their own or the system's performance (see subsection 2.2.2.1). Only the requirement of low and late commitment does not immediately follow from the definition and concepts of agent technology, although agents can be designed to pursue such a strategy. Thus, agent technology provides many of the (software) techniques that are necessary to design and implement modern production control systems. This potential was recognised as early as 1985 when the first control systems based on agent-oriented concepts were proposed (Parunak 1985, Shaw 1985). Since then, much work has been invested into developing agent-based control systems. This work as well as its context will be reviewed in this section.

This section is organised as follows. It will review the basic concepts of agent-based production control (see subsection 2.3.1); look at the relation of these concepts to other trends in production research (see subsection 2.3.2); present industrial applications of agent-based control systems (see subsection 2.3.3); and finally discuss the design of such systems (see subsection 2.3.4).

2.3.1 Basic concepts

One of the very first applications of agent-oriented concepts to manufacturing control was the prototype factory control system YAMS of Parunak et al. (Parunak 1985, Parunak 1986, Parunak 1987). In YAMS, the manufacturing enterprise is modelled as a hierarchy of production units, of which the smallest units are called *workcells*. An engine plant, for example, may consist of a block, an oil pump, a head and an assembly flexible manufacturing system (FMS). Each FMS may, in turn, consist of a set of workcells which perform the actual operations like drilling or milling (see figure 2.14). The hierarchy, however, records only composition, not control. Task distribution down the hierarchy is done through a negotiation process that is based on the contract-net protocol (see subsection 2.2.2.2). A node announces a task, units capable of performing the task reply with a bid, and the node in turn assigns the task to the unit with the best bid. Because of this negotiation process between superior and subordinate units, the assignment process is able to take into account any changes or disturbances that may occur during the planning process. A superior unit may even repeat the negotiation if the unit that was assigned to the task fails to execute it.

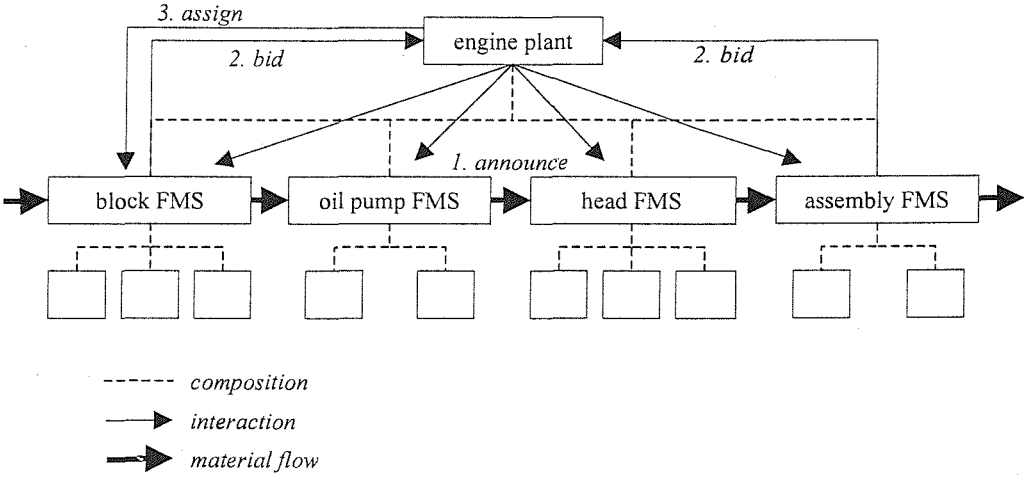

Figure 2.14: An example factory hierarchy in YAMS (Parunak 1985).

Once a manufacturing task has been assigned to a production unit, this unit must request the corresponding material from the material handling system. The material handling system in YAMS consists of a mover agent for each workcell and for each part of the transportation system. These movers keep track of each pallet in their physical vicinity and forward a pallet if it is requested by a neighbouring unit or request it themselves from their neighbours if the pallet is not available yet. A pallet, however, is only forwarded if the number of pallets, i.e., the *work-in-process*, at the target mover does not exceed a certain physical limit. If the limit is exceeded, the pallet is delayed until the work-in-process of the target cell decreases again. Through this mechanism,

the control system ensures that the material flow is automatically adjusted to the current capacity and workload of each workcell. In case of disturbances, for instance, the material flow is slowed down exactly to the reduced capacity of the disturbed workcell. Both control mechanisms, the top-down negotiation and the work-in-process limitation, thus enable the manufacturing system to adapt its assignment and execution process to any changes and disturbances in the manufacturing system (cf. subsection 2.1.3).

In parallel to Parunak's work, Shaw and Whinston (Shaw 1985, Shaw 1987) also developed a distributed scheduling method based on the contract-net protocol. As in YAMS, manufacturing tasks are contracted out to workcells through a bidding process in which the workcells bid for the tasks. The work in (Shaw 1987), though, focussed on evaluating different decision rules for choosing the winner of the bidding process, in particular comparing these to centralised scheduling using the same decision rules. The comparison included a centralised scheduler preferring the shortest processing time when assigning a manufacturing task, and two bidding schemes, one choosing the bid with the shortest processing time and one the bid with the earliest finish time. In a set of simulations with randomly created manufacturing tasks, the decentralised bidding scheme outperformed the centralised scheduler in each case. This is because the bidding algorithm is based on more accurate information. Instead of duplicating the shop-floor information in a centralised scheduler, the bidding scheme lets the workcells, which have the most up-to-date information, bid themselves for the tasks. This leads to more realistic schedules and thus to a better performance.

Since this early work on agent-based control systems, the contract-net-based approach to scheduling has been very popular (see (Tilley 1992, Maturana 1996, Saad 1997, Maley 1998, Sousa 1998)) and was further evaluated and extended by several researchers. Tilley and Williams (Tilley 1992, Tilley 1996), for instance, studied the communication performance of a distributed bidding scheme, while Saad et al. (Saad 1997) evaluated the performance of four additional decision rules for choosing the winner of the bidding process. Approaches similar to the contract-net were also proposed in (Bussmann 1996) and (Hahndel 1996). In (Bussmann 1996) a co-ordination algorithm for scheduling transportation tasks was presented. As in YAMS, tasks are announced to the transport units. In contrast to YAMS, however, these units perform only a local analysis of the task and return the result of the analysis to a co-ordinator. The co-ordinator then synthesises the results into a global schedule. Hahndel et al. (Hahndel 1996) developed a completely decentralised approach to scheduling assembly tasks consisting of several steps. As in previous work, each step of a manufacturing task is assigned to a workcell through a contract-net-based negotiation. Once the first step is assigned, though, the responsibility for assigning the next step goes to the workcell that is assigned to the last step. This workcell is then not only supposed to execute the assigned step, it is also responsible for finding the next workcell. In case of assembly steps, there are several workcells for the same tasks which are looking for the

next workcell. To ensure that these workcells choose the same assembly cell, the workcells select a synchronisation agent. This synchronisation agent, in essence, chooses the assembly cell for the next step.

The contract-net-based scheduling approach also inspired new approaches, such as the market-driven control scheme (see for example (Baker 1996, Markus 1996)). In (Markus 1996, Vancza 1998), for instance, tasks are assigned to machine agents through a bidding scheme based on prices. That is, a superior unit, called 'management' in (Markus 1996), still announces the manufacturing tasks, but the machine agents now bid for tasks by returning only the costs for executing the tasks. The management then tries to maximise the 'profit' of the manufacturing system by choosing the bid offering the lowest costs. The advantage of this approach is that the performance of all production units, as well as of the production system itself, can be (constantly) evaluated on the basis of one single measure, namely profit. Components which underperform because their earnings, for instance, are significantly less than their actual costs can be removed from the system. From the work presented to date, however, it is not clear whether a production system making greater 'profit' really performs better in terms of the company's goals. The 'manufacturing profit' proposed in (Markus 1996) is a only virtual concept and does not directly correspond to the company's real profit. Furthermore, it is also not clear whether this approach outperforms the classical contract-net-based approach. The market-driven approach certainly requires more evaluation.

An exception to the predominant contract-net and market based approaches is the team-based approach of Fischer (Fischer 1994). In this approach, the central shop floor control announces the manufacturing jobs to the workcells by adding these to a global task list. The workcells examine the list and ask other resources to join a team for a particular job. Once the team is complete, i.e., the team can perform the job autonomously, the team leader requests the job. To avoid any resource conflicts between parallel team formation attempts, Fischer proposed a protocol that is conflict-free.

To summarise, research into agent-based production control systems has focussed much on the decentralised assignment of manufacturing tasks to production units. In applying negotiation techniques, like the contract-net protocol, this process becomes more flexible with respect to changes and disturbances in the production system. The following section will show that this research is part of a much larger effort to introduce flexibility into production.

2.3.2 Trends in production research

The limitations of the pre-dominant approach to production control, discussed in

subsection 2.1.3, have also motivated the development of new control concepts in production research. The starting point of these developments, though, was not the application of a (software) technology to specific control tasks, but the design of new control, or even new production paradigms. These new developments consequently take a more holistic view on production control, encompassing all control tasks necessary to run a production facility. The two developments which are (most) relevant to this work are *heterarchical control architectures* and *holonic manufacturing systems*. Each development will be reviewed in the following and a comparison of both with agent-based control systems is given at the end of this subsection. Other approaches, which are less relevant to this discussion, are for example random (Iwata 1994), fractal (Warnecke 1993), or bionic (Okino 1993, Ueda 1993) manufacturing (for a comparison of holonic, fractal, and bionic manufacturing see also (Tharumarajah 2003)).

2.3.2.1 Heterarchical control

Heterarchical control was motivated by the limitations of centralised and hierarchical control architectures (Dilts 1991). Both centralised and hierarchical control architectures result in quite complex control systems because they centralise distributed information. The information thus becomes quickly out-of-date and the control logic is difficult to implement due to the enormous number of cases that arise in a larger production system. As a consequence, these systems are difficult to modify or extend and exhibit a very low fault-tolerance. Heterarchical control, in contrast, eliminates any centralised elements and increases the autonomy of each controller (Duffie 1987). That is, all controllers are connected via a communication network and are solely responsible for any control tasks within their vicinity (see figure 2.15).

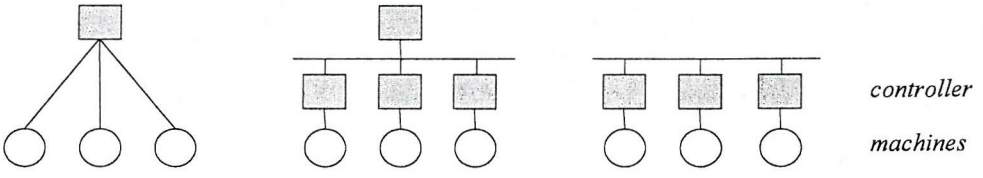

Figure 2.15: Centralised, hierarchical, and heterarchical control architectures (Dilts 1991).

Autonomy, however, is not enough to make heterarchical control systems work. A control system in which each controller only pursues its own goals without regard to the goals of others will result in a chaotic, sub-optimal production process. To optimise a production process, the controllers must co-operate with respect to the overall production goals (cf. subsection 2.1.3). Hatvany therefore proposed 'co-operative heterarchies' (Hatvany 1985), and Duffie stressed the concept of 'co-operating autonomous entities' (Duffie 1988). This co-operation is usually achieved by the same

interaction techniques as in agent-based control systems. Duffie, for instance, uses a protocol similar to the contract-net protocol (Duffie 1988).

2.3.2.2 Holonic manufacturing systems

Holonic manufacturing, in turn, takes a much larger perspective than heterarchical control by looking at the whole manufacturing process instead of only the control. Originally, *holonic manufacturing systems* (HMS) were proposed in the early 1990's as a new manufacturing paradigm to address the upcoming challenges of the 21st century (Suda 1989, 1990). These challenges are mainly globalisation and industrial over-capacity which result in a shift from a vendor's to a customer's market and in particular in shorter product life-cycle, reduced time-to-market, mass-customisation, volatile demand, and constant cost pressure (McFarlane 2003, Brennan 2003). As a consequence, production operations are faced with increasing complexity and constant change under decreasing investments (cf. subsection 2.1.4). In addressing these challenges, holonic manufacturing was inspired by the work of the philosopher Arthur Koestler who tried to explain the evolution of biological and social systems (Koestler 1989). On the one hand, these systems develop stable intermediate forms during evolution that are self-reliant. On the other hand, it is difficult in living and organisational systems to distinguish between 'wholes' and 'parts': almost everything is both part and whole at once. These observations led Koestler to propose the word "holon", which is a combination of the Greek word 'holos' meaning whole and the Greek suffix 'on' meaning particle or part (as in proton or neutron).

Following the holonic concept, a holonic manufacturing system thus consists of autonomous and self-reliant manufacturing units, called *holons*, which operate in a flexible hierarchy. In such a system, any unit can be a holon as long as the unit is able to create and control the execution of its own plans and/or strategies (see the HMS definition of autonomy (Christensen 1994, van Leeuwen 1997)). Machines, conveyors, and automated guided vehicles as well as human workers are obviously holons, but also orders can be holons as long as the orders are able to pro-actively pursue their own processing. For this reason, Christensen (Christensen 1994) foresees three major interfaces to an artificial holon: a physical processing interface, an inter-holon interface, and a human interface (see figure 2.16). The physical processing interface is optional as holons, such as an order holon, need not contain any physical components at all.

Holons co-operate with other holons during the production process in order to accomplish the production goals. Co-operation, in the form of co-ordination and negotiation, develops wherever and whenever necessary, usually along material and information flow. A system of holons which can co-operate to achieve a goal or objective is called a *holarchy* (van Leeuwen 1997). Holarchies are recursive in the sense that a holon may itself be an entire holarchy that acts as an autonomous and co-

operative unit in the first holarchy. Holons within a holarchy may dynamically create and change hierarchies. Moreover, holons may engage in multiple hierarchies at the same time. In contrast to hierarchical control systems, though, HMS creates only loose and flexible communication hierarchies which never force a holon to perform a certain task (Valckenaers 1994).

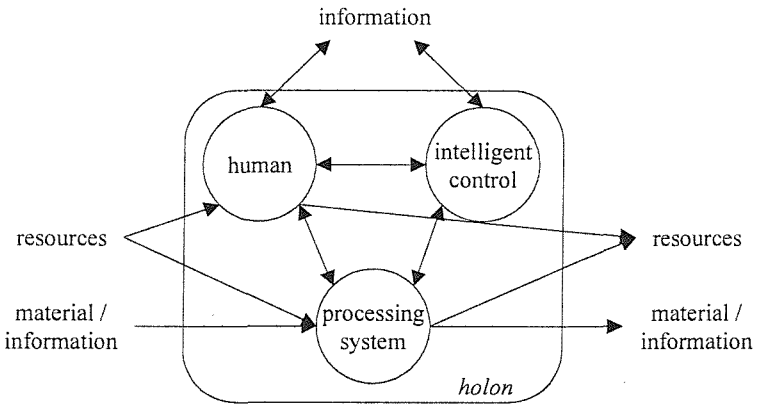

Figure 2.16: Holon interfaces (Christensen 1994).

Since its conception in the early 1990's, holonic manufacturing has received a lot of attention in academia and industry (see for example (van Leeuwen 1997, Deen 2003, Marik 2003)). In particular, holonic manufacturing was one of the six test cases in the international Intelligent Manufacturing Systems (IMS) feasibility study that was set up in 1992 (Hayashi 1993). The success of the holonic manufacturing test case, both with respect to the feasibility of the holonic concept as well as to the international collaboration, led to the endorsement of holonic manufacturing as an international IMS project in 1994, with a 10 year research program (van Leeuwen 1997). Over 30 academic and industrial partners from the IMS regions Australia, Canada, Europe, Japan, and the United States are now participating in this international pre-competitive research collaboration (Holonic Manufacturing Systems Consortium 2002).

To date, a large amount of research into holonic manufacturing has been conducted inside and outside of the HMS project. This research includes the development of generic technologies for designing and implementing holonic manufacturing systems as well as the application of these concepts to machining units, workpiece fixturing, material handling, and resource management, i.e., planning and control of manufacturing systems (van Leeuwen 1997). Concerning planning and control, HMS research has addressed issues such as:

- distributed decomposition of orders into manufacturing tasks;
- distributed scheduling of manufacturing tasks between autonomous and co-operative units, i.e., holons;

- autonomous execution of manufacturing tasks that interfaces with the scheduling process; and
- machine control architectures consisting of co-operative devices.

For a complete overview of the research into planning and control aspects of HMS, see also (McFarlane 2000).

2.3.2.3 Summary

As for agent-based control, autonomy and co-operation are the key concepts in both heterarchical control and holonic manufacturing systems. Both paradigms, however, use autonomy and co-operation to realise a certain organisation of the production process: Heterarchical control proposes a flat hierarchy of local controllers, while holonic manufacturing systems envision flexible hierarchies which emerge from the production process. Agent-based control, in contrast, is a software technology that can be used to implement any kind of autonomous and co-operative control behaviour, be this hierarchically or heterarchically organised. In fact, it can be argued that the information processing part of a holonic manufacturing system, and certainly that of a heterarchical control system, is an agent-based system (Bussmann 1998). Agent-based control is consequently an enabling (software) technology for both manufacturing paradigms, in particular for holonic manufacturing systems (McFarlane 1995, Marik 2002).

2.3.3 Industrial applications

This section reviews two industrial applications of agent-oriented production control systems.

2.3.3.1 Production 2000+

Probably the first full-scale industrial agent-based production system that has brought agent-oriented concepts into operation is the cylinder head manufacturing system *Production 2000+* (P2000+) of DaimlerChrysler (Bussmann 2001b). This manufacturing system consists of flexible CNC machines which are configured to process a range of products. To achieve robustness, each operation of the manufacturing system is provided by at least two machines. In case of a single machine failure, there is thus at least one other machine able to process the workpieces. The above flexibility and robustness requires a flexible transportation system such that a workpiece may be moved from any machine to any other machine. This flexible transportation is provided by a system of forward conveyors, backward conveyors, and

shifting tables moving the workpieces in and out of the machines (see figure 2.17).

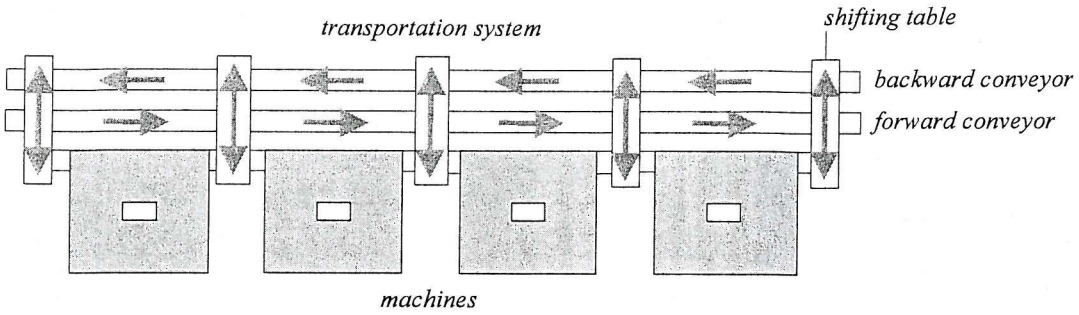

Figure 2.17: Production layout P2000+.

To operate such a manufacturing system, the associated control system must decide during the production process – among other aspects – how and when the workpieces are assigned to the individual machines. To this end, DaimlerChrysler and Schneider Electric have designed and implemented an agent-based control system consisting of agents for each machine, each transportation switch, and each workpiece (see figure 2.18). These agents interact in order to achieve a robust and flexible material flow through the manufacturing system: The workpiece agent manages the state of the workpiece and searches for machines to process the workpiece. The machine agent controls the workload of the machine and bids for suitable workpieces. The workpiece agent in turn chooses the best machine for the next operation based on processing as well as workload criteria. Finally, the transportation agent chooses a route to the next machine taking into account the current load of the transportation system (more details and some properties of the control behaviour are given in (Bussmann 2000)).

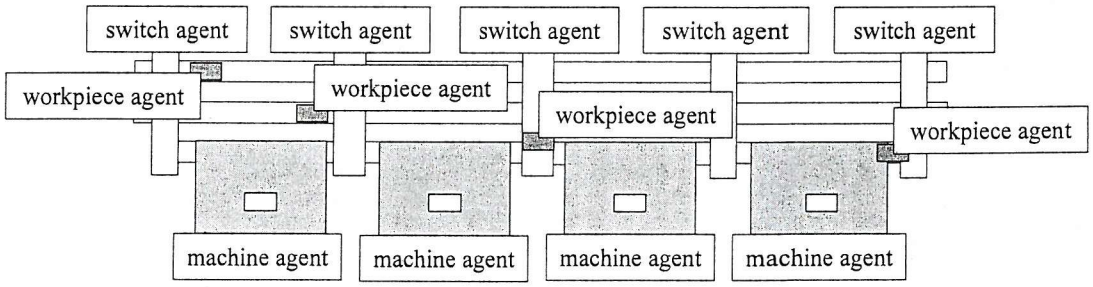

Figure 2.18: Control agents of P2000+.

The manufacturing system consisting of six machines was installed in 1999 as a bypass to an existing large-series manufacturing line at the DaimlerChrysler engine plant in Stuttgart-Untertürkheim, Germany (see figure 2.19). After a suite of performance tests which demonstrated the industrial feasibility and performance of the agent-based approach, this prototype was extended by two machining units and an automatic loading machine in order to produce in series the cylinder head for a four cylinder

diesel engine (offered in the Mercedes-Benz C- and E-class 220 CDI). The prototype has now been in operation since 1999 and, during that time, has proven that it is able to hold its high standard even in day-to-day operation. This prototype can thus be viewed as a proof for the industrial feasibility of agent technology (for manufacturing operations).

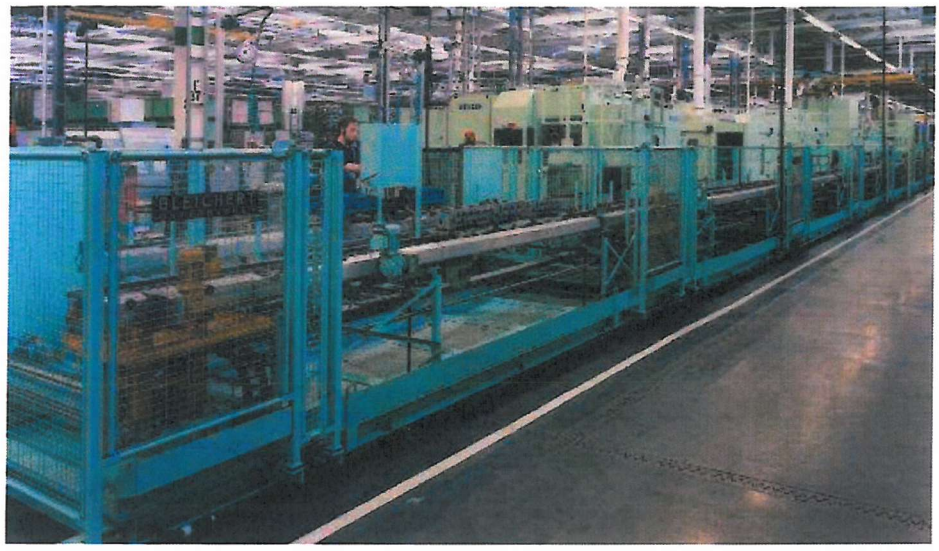

Figure 2.19: DaimlerChrysler Prototype P2000+.
(Courtesy of Bleichert, Osterburken, Germany)

2.3.3.2 Holomobiles

Another example of an agent-based control system was developed by the holomobiles workpackage of the HMS project (Bussmann 2001). The workpackage analysed the limitations of existing engine assembly systems in the automotive industry and designed a new material flow concept that improves the robustness and volume flexibility of the assembly process. To achieve the volume flexibility, the new assembly concept included a migration path from the existing assembly system to the new design, in which resources are added step by step. This migration path required that the control is able to incrementally incorporate additional resources into the assembly process, and that the control itself is scalable in the same steps as the assembly system – two requirements that are easily fulfilled by holonic manufacturing systems. The development of the new assembly concept therefore included the design of a holonic control system for the new assembly process. The new assembly process, the holonic control systems as well as the results of a feasibility study are briefly discussed in the following.

The new assembly process consists of a main assembly line, as it was described in subsection 2.1.1, and a pool of additional resources that are located along the main line

(see figure 2.20). The main line is a standard assembly line covering the whole engine assembly process. During the scaling process, this line is extended with additional resources at bottleneck stations whenever the volume of the assembly system should be increased. There are two possibilities to perform this extension. The first is to add flexible buffers in order to de-couple failure-prone stations. And the second is to add multi-functional (MF) stations that are able to perform part of the assembly process in order to increase capacity at specific bottleneck stations. To connect the main line with the additional resources, automated guided vehicles (AGVs) take engines off the main line at so-called docking stations, transport these engines to the flexible buffers or the MF-stations, and bring the engines back to the same docking station (in case they have only been buffered) or to succeeding docking stations (in case they have been processed). The task of the control system is therefore to decide when to take an engine off the main line, whether to buffer or process an engine, how many operations to apply to an engine, and when to bring an engine back to the main line.

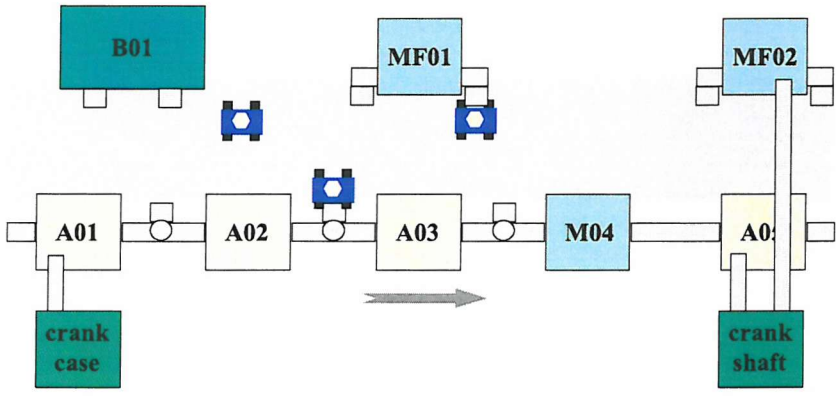

Figure 2.20: Assembly layout of Holomobiles.

To solve the control task and, at the same time, to meet the requirement for scalability of the control system, Holomobiles introduced a control holon for each docking station (DS), each engine buffer (EB), each MF-station (MF), and each AGV (see figure 2.21). An important aspect in this design was that no holon was introduced for components of the main assembly line. Such holons were avoided in order to allow the main assembly line to operate without the existence or the operation of the additional resources. This has the clear advantage that the main assembly line can be built solely with standard (control) technology. The main line must only provide enough space to add docking stations in later steps of the scaling process.

The control holons basically perform the control tasks identified above. The docking station holon determines – based on the situation around it – whether it should send an engine off the main line. It does so if two conditions hold. The first condition holds when both its exit and its entry are blocked. In this case there is obviously a bottleneck behind the docking station. The second condition is that the docking station is closest to

the bottleneck. To verify this, the docking station agent runs an election protocol with the docking stations following it. It wins the election, and may thus divert engines, if there is no docking station directly behind it that has also detected a bottleneck. To divert an engine, a DS holon must find either an MF-station that will process the engine, or a buffer currently capable to store the engine. It does so by requesting capacity from the corresponding holons. Once it has received the required capacity, the DS holon requests an AGV holon to do the transportation and waits for the engine to be picked up. An MF holon decides for each arriving engine where to send it to next. To do so, it requests DS holons, other MF holons, or EB holons to accept the engine for further processing (or buffering). Depending on the holon that accepts the engine, the MF-station performs the assembly operations necessary to put the engine in the correct processing state. Similarly, the EB holon decides when and where to send the engine to next. These decisions are co-ordinated by the EB holons in order to achieve a buffering strategy that optimally uses the system-wide buffering capacity. Finally, the AGV holons offer their transportation capacity to any holon that requests it.

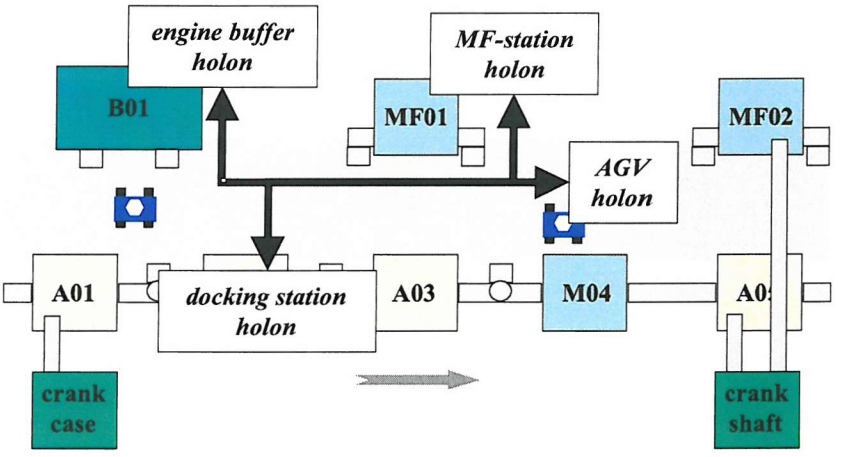

Figure 2.21: Control holons in Holomobiles.

To evaluate the new assembly as well as the holonic control approach, the performance of the new assembly system was compared to that of an existing industrial assembly line. The comparison was based on a set of common scenarios that were simulated with data taken from the existing line. The main results are briefly reviewed here (for more details see (Bussmann 2001)). First of all, the holonic system showed a more robust behaviour than the existing assembly system because the additional resources are used to de-couple failure-prone stations. Secondly, and more importantly, the holonic system can be scaled up in small steps as additional resources are added to the system. The introduction of flexible buffers not only increases robustness, it also increases, as a side effect, the throughput and thus scales up the volume of the assembly system. Likewise, the introduction of MF-stations increases the capacity of assembly sections and thus also increases the volume. The new assembly system designed by Holomobiles thus

offers robustness and scalability which are unprecedented in existing assembly systems.

The above examples of industrial applications provide a glimpse at the variety of agent-based production control systems that are required in industry. More examples of different industrial control systems can be found in (Parunak 1999, Parunak 2000).

2.3.4 Design of agent-based control systems

Agent-based control techniques address the need for more flexible production control systems and have demonstrated their industrial flexibility in several applications. The applications, however, have also shown that there is no universal design for an agent-based control system. The two applications presented in the previous section, for instance, have a completely different architecture and employ (partly) different control algorithms. While the P2000+ control system consists of workpiece, machine, and switch agents, the Holomobiles control system has no workpiece agents (that represent the engines). This is due to the fact that in Holomobiles the main assembly line must be operated by the traditional control system in which engines are not explicitly represented. However, it is also not necessary to represent engines because in Holomobiles the engines all receive the same set of assembly operations. Consequently, there is no need for an engine agent to ensure that the processing graph of the engine is respected.[6]

The architectures of P2000+ and Holomobiles also differ with respect to the type of resource agents used. P2000+ has machine and switch agents, whereas Holomobiles requires agents for the MF-stations, the engine buffers, the AGVs, and the docking stations. This difference is not only a matter of terminology, but manifests itself in a quite different functionality of each agent. In Holomobiles, the MF-station and engine buffer agents pro-actively manage the processing (and buffering) of the engines; the AGV agents offer a transportation task instead of simply forwarding workpieces; and the docking stations supervise the main line by identifying bottlenecks.

Finally, Holomobiles requires additional interaction protocols. It uses, as the P2000+ control system, a contract-net protocol to request resources such as MF-stations or AGVs. But Holomobiles agents must also run an election protocol between the docking stations in order to identify those docking stations directly in front of a bottleneck and a co-ordination protocol between the resource agents and the docking station agents in order to determine in which state to transform the engines and when to return them to the main line.

All in all, the agent-based control design of P2000+ and Holomobiles share common aspects, but there are also significant differences between both designs that make it

[6] The point here is not whether it is possible to introduce engine agents, but which design is easier to implement and to maintain.

impossible to derive a common control design for both applications. In general, this is true for most control applications. The overall architectures of most agent-based control systems are similar in that they all require resource, transportation, and sometimes workpiece agents, but in detail these agents have quite different functionality and employ quite different interaction protocols to achieve their goals. These differences are mainly due to the different nature of the production processes that need to be controlled. A cylinder head manufacturing process is obviously different to an engine assembly process. But the differences may also be due to the product being produced or the requirements on the production process. For instance, the P2000+ system was supposed to be flexible and robust, while the Holomobiles system was intended to be robust and scalable from the existing traditional assembly system. Different control applications will thus require different agent-based control designs. This will be true even if a universal design of an agent-based control system existed or will be developed in the future. Such a universal design would have to cover all the different control aspects of any possible application and would thus require the realisation and test of a large set of functionality. However, the more functionality a control system consists of, the more costly it is to develop and maintain it. Given the constantly increasing cost pressure in industry, it is therefore imperative to avoid unused functionality. A control system will thus always be tailored to the specific needs of a production process.

2.4 Conclusions

Agent technology has the potential to meet the future challenges in production control. It provides conceptual models and implementation architectures for goal-based decision-making as well as for negotiation and co-ordination of goals and actions, which – as pointed out in section 2.1.4 – will be indispensable features of future control systems. Many aspects of agent-based production control systems have been investigated in the literature over the last fifteen years, and the potential of this approach has been demonstrated in several prototypical applications. Recently, even the industrial feasibility and the economic benefits of this approach have been proven in probably the first full-scale industrial manufacturing system for large-series automotive production. The technology of agent-based production control is consequently ready for exploitation in industry.

Recent applications of agent technology, however, have also shown that there is no universal design of an agent-based production control system that can be re-used for every production control problem. On the contrary, the required design of a control system may vary significantly depending on the product to be produced, the production process necessary to produce the product, and the business requirements on the production process (see subsection 2.3.4). Even the two supposedly similar production

tasks of engine manufacturing and assembly in the automotive industry require quite different control systems because the production process and the business requirements differ in both cases. The lack of a universal design for agent-based production control thus implies that for most production control problems a new design effort is necessary in order to develop a cost-efficient agent-based design for the control problem at hand. This recurrent design effort, however, hampers the exploitation of the agent-based production control technology in industry because to date this design effort requires the designer of a control system to be an expert in agent technology: designing agent-based production control systems is still a research activity (cf. the international Holonic Manufacturing Systems project in subsection 2.3.2.2). But no matter how important, agent-based control is only one aspect in the design of an industrial production control system, which also includes the design and optimisation of machining and robotic control programs, of transportation device and buffer programs, of human worker integration, of performance and quality monitoring systems, of material supply and conditioning processes (Dorf 1994, Groover 1988). A control engineer thus cannot be expected to specialise in agent technology (and it is also not economically reasonable to install specialists for this design aspect because these specialists would only be required in certain phases of the development). Consequently, to facilitate the use of agent technology, there must be some kind of design methodology that allows even a non-expert in agent technology to design an agent-based production control system given the specification of a production control problem. With such a design methodology the recurrent design effort for agent-based production control systems is no longer an obstacle to the industrial exploitation of agent technology.

The ultimate goal of this thesis is therefore to provide a design methodology for agent-based production control systems that can be successfully applied by a non-expert in agent technology. From the point of view of agent technology, the deliberate focus of this design methodology on production control is justified for two reasons. First of all, production control is an application domain with special requirements on the agent-based design because an agent-based production control systems needs to interface with a physical system (of a specific structure) to achieve its goals. This puts special requirements on the agent-based system to be designed (Parunak 1987, Parunak 1991). Secondly, production is an important economical factor in industrialised countries (see section 2.1). Optimising agent-based systems towards these special requirements is in particular justified when considering the added value produced by some of the production systems (an automotive car plant, for instance, creates an added value of several billion Euros per year). A design methodology for agent-based production control systems is thus at once scientifically and economically worthwhile. But before a design methodology tailored to production control is developed, it should first be verified that existing methodologies fail to address the needs of production control. This is done in the next chapter.

Chapter 3

Design Methodologies

The need for methodologies in software development was already recognised in the late 1960s (Dijkstra 1968, Wirth 1971, Parnas 1972) and led to the introduction of the field of *software engineering* (Sommerville 1995). Since then, many methodologies have been proposed for the different phases of software development. In particular, there are a large number of methodologies for designing software systems, the most prominent probably being structured and object-oriented design methodologies. Many of these design methodologies claim to be applicable to any (software) design problem and must consequently also be applicable to the design of agent-based production control systems. This chapter will therefore review the main existing design methodologies that are potentially applicable to agent-based production control systems and will assess to what extent these methodologies are able to adequately support the design of such systems.

Generally speaking, design methodologies are classified according to their underlying programming paradigms. In 1992, Fichman and Kemerer (1992) classified the existing methodologies into (i) structured approaches, (ii) data-oriented approaches, and (iii) object-oriented approaches. For the purpose of this review, this classification is extended by approaches from (iv) manufacturing control, and (v) agent-based systems. Manufacturing control methodologies are obviously relevant to this review because they were specifically developed for designing manufacturing control systems. Agent-oriented methodologies, in turn, are a recent development in software engineering that tailors the design process to the specific needs of agent-based systems. These methodologies are thus also potential candidates for designing agent-based production control systems, even though most of these were not developed for this particular type of application domain. Finally, the classification will be complemented by (vi) re-use approaches. Design methodologies generally focus on designing a software system from scratch. A more efficient and usually more reliable approach, though, is to (at least partly) re-use existing designs (Krueger 1992, Coulange 1998). Naturally, re-use may

not be the sole approach because re-use requires that some designs have already been created. Nevertheless, re-use has the potential to significantly reduce the required design effort and should therefore be included in this review.

The review is thus organised as follows. Each of the above classes of design methodologies will be discussed in one of the following sections: structured and data-oriented methodologies in section 3.2; object-oriented methodologies in section 3.3; manufacturing control methodologies in section 3.4; and agent-oriented methodologies in section 3.5. Furthermore, in section 3.6, this review will look at recent developments in re-use and to what extent these developments can facilitate the design of agent-based systems. Finally, section 3.7 will summarise the review and draw some conclusions. But before the review may start, it is necessary to first define the term *methodology* and identify criteria which assess whether a methodology adequately supports the design of agent-based production systems. This is done in section 3.1.

3.1 What is a methodology?

A methodology is a "recipe" that enables a designer to find a solution to a specified set of problems. This recipe supports the designer by specifying many aspects of the design process, while leaving some of the design issues to the creativity of the designer. This is in contrast to a *procedure* which fully specifies how the solution to a specific problem is determined (and which can therefore be implemented on a computer). To capture this distinction, a methodology is defined as follows (see (Hußmann 1997, pp. 13; Budgen 1994, pp. 143)):

Definition: A ***methodology***[7] always consists of the following components:

- An (optional) definition of the *problem space* to which the methodology is applicable.
- A set of *models* which represent different aspects of the problem domain or the solution at different stages.
- A set of *methods* which transform instances of one model into another model.
- A set of *procedural guidelines* which define an order for the systematic application of the methodological steps.

The application of a methodology starts with a problem statement (which must belong to the problem space specified) and ends with a solution to the problem. Methods and

[7] Some authors, like Hußmann (1997), use the term *method* instead of *methodology*. In this thesis, however, the term *methodology* is preferred because *method* will be used for a single model transformation.

guidelines tell the designer how to create an initial model of the design problem and how to transform this model – with possibly many intermediate models – into a model representing a solution to the design problem (see figure 3.1). The set of models thus contains at least one model to represent the problem statement and one model to specify the solution.

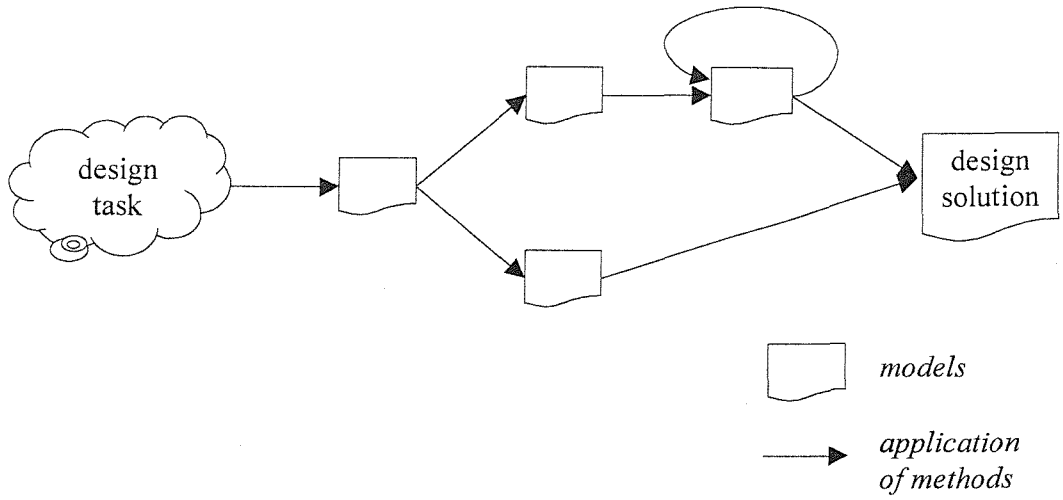

Figure 3.1: Application of a methodology.

Most of the existing design methodologies provide models, methods, and procedural guidelines for the design process, but do not specify the problem space to which the methodology is applicable, or alternatively they claim that it is generally applicable (i.e., applicable to any software design problem). In practice, however, any methodology will be more suitable for some problems than for others (Sommerville 1995, p. 216). Budgen even doubts "whether there can ever be a 'right' method that would be appropriate for all problems ..." (Budgen 1994, p. 365). To assess the suitability of a methodology for a particular class of design problems, it is therefore necessary to determine whether the application of a methodology is appropriate rather than possible. The following subsection will therefore identify a set of criteria which assess the suitability of a methodology for the design of agent-based production control systems.

3.1.1 Requirements for a methodology

A design methodology for agent-based production control systems should obviously provide models and methods that (somehow) capture the key agent-oriented aspects of the design (see section 2.2) because otherwise they are not able to specify the agent-based solution (see also (Fisher 1997)). Modelling agent-oriented aspects, though, is not a sufficient criterion for the appropriateness of a methodology. Methodologies, by definition, are supposed to support human designers in developing a design solution. A

methodology is thus appropriate only when the designer is able to apply the models and methods of the methodology to his design problem. Therefore, it is also necessary to look at the designer and his capabilities to apply the design methodology.

The design of (agent-based) production control systems will be primarily performed by production engineers who are also charged with many other engineering tasks, such as robot programming and optimisation, device programming, or quality engineering. Consequently, these designers cannot be expected to be experts in agent-based systems because agent-oriented control – no matter how important – will not become effective without the simultaneous implementation of the other engineering aspects (see also section 2.4). Rather, the designer of a production control system can only be expected to satisfy the following minimal requirements:

- The designer has an education in production or control engineering, but none in computer science.
- The designer has only basic training in agent technology.
- The designer has no or little experience with agent-based development.

The methodology should thus enable a designer with a minimal qualification in agent technology to derive an agent-based solution from the specification of a production control problem (i.e., to transform the problem specification into an agent-based solution). This transformation, however, must bridge the gap between the problem and the solution domain. As Kaindl (1999) points out, objects of the problem domain are inherently different to those of the solution domain. In the case of designing agent-based production control systems, the objects of the problem domain are physical components, such as machines and workpieces, whereas the solution domain consists of software artefacts that are supposed to show autonomous and co-operative behaviour. The difficulty of the design is thus the transition from the problem to the solution objects (Kaindl 1999). Since this transition is accomplished by the creation of models (see figure 3.1), the models must reflect this transition. In other words, each model created must be somehow related to the previously created models such that any new concepts are put into relation to previously introduced concepts. This is particularly true for agent-oriented concepts which are not part of the problem domain (machines are not per se autonomous and co-operative entities!). Any agent-oriented concepts must thus be introduced by relating these either to production or previously introduced concepts. This translates, in turn, into the following requirement on the design methodology:

- **Model appropriateness.** The models of a methodology should be clearly related to the relevant concepts of the problem domain. The initial model should be based on domain concepts and any new concepts should be related to those already introduced. This applies in particular to the introduction of agent-oriented concepts.

The appropriateness of the models used in a design process, however, is not sufficient to enable a novice (in agent technology) to apply the methodology, because the designer must also know *how* to instantiate the models for his particular design problem. This instantiation is performed by the methods of the methodology – they tell the designer how to parameterise the models (see figure 3.1). To support the designer in parameterising the models, however, the methods must be sufficiently prescriptive – otherwise it is left to the designer to decide how to actually instantiate the models. The prescriptiveness of the methods is particularly important with respect to the agent-oriented concepts. If the designer is not familiar with introducing or defining agents, he will have difficulties to do so even if the methodology tells him that the next model to be created should include agents: he will not know *which* agents to introduce. The methods of the methodology therefore need to provide all the (agent-related) rationales necessary to derive the agent-oriented design aspects. This translates into the second requirement on the methodology:

- **Method prescriptiveness.** The methods of the methodology should be prescriptive in the sense that they prescribe each step the designer has to go through, and for each step clearly identify what the task of the designer is and – at least for any agent-oriented design aspect – explain how the task should be performed.

A methodology fulfilling the above requirements will enable a designer with only minimal qualification in agent technology to perform the design of agent-based production control systems. The first requirement ensures that the designer understands the links between the design problem and the models he is creating on behalf of the methodology. He can start the design process by modelling the domain he is familiar with, and any new model he is creating clearly identifies new concepts and explains them in terms of the concepts already used. This straight transition between analysis and design models allows the designer to incorporate more sophisticated agent concepts into the control design despite his lack of training in agent technology. The second requirement ensures that when the designer moves from one model to another he knows how to perform this step. Either the methodology clearly identifies the required results if the design step involves domain reasoning (it is assumed that the designer knows how to perform domain reasoning)[8], or the methodology provides all the design rules that are necessary to perform the design step if it involves agent-related reasoning. The latter ensures that the design process does not require experience in agent design (even though experience is always beneficial).

Recently, there have been several attempts to provide frameworks for evaluating agent-based design methodologies (O'Malley 2002, Cernuzzi 2002, Sturm 2003, Dam 2003), mostly motivated by the diverse range of existing agent-oriented methodologies. These

[8] It can be safely assumed that the designer knows how to perform domain reasoning, if he wants to design a production control system. This work therefore focuses on the agent-oriented reasoning.

evaluation frameworks, however, are not applicable to the evaluation of agent-oriented design methodologies for production control. First of all, all of the above frameworks presuppose a specific agent or even multi-agent model. In particular, they evaluate to what extent concepts such as beliefs, desires, intentions, roles, norms, organisational relationships, and so on, are modelled. As shown in subsection 2.3.3, however, an agent-based system may be successfully deployed with fewer or different concepts. The frameworks are thus restricted to specialised agent-oriented approaches. Secondly, there is as yet no consolidation between the evaluation frameworks. Each framework proposes a different set of criteria, and, generally speaking, these criteria are not well justified. It is therefore difficult to see which framework should be used and whether this framework is really able to identify the "best" methodology. Given this fact, the work described in this thesis focuses on the two criteria identified above, i.e., model appropriateness and method prescriptiveness, since these criteria must be satisfied in order for a methodology to be suitable.

To summarise, a methodology is suitable for the design of agent-based production control systems if it fulfils the following requirements.

Requirement I:
It meets the *definition of a methodology.*

Requirement II:
It is able to *model the agent-oriented aspects* of a production control system.

Requirement III:
Its models provide a straight and comprehensible transition from domain concepts to the agent-based system (*model appropriateness*).

Requirement IV:
The methodology provides all the necessary methods and criteria to perform agent-related design decisions (*method prescriptiveness*).

To assess the suitability of existing methodologies for the design of agent-based production control systems, the following review is performed in two steps. First, it is assessed whether a methodology is generally suitable for the design of an agent-based system (i.e., is it able to model agent-oriented concepts?). If yes, the methodology is additionally assessed with respect to the above requirements of model appropriateness and method prescriptiveness in order to determine its suitability for the design of an agent-based production control system. To this end, the review will focus on two basic aspects that are central to the design of agent-based systems: (i) the modelling and identification of agents; and (ii) the identification and design of interactions between the agents (see sections 2.2 and 2.3.4).

Even though chapter 2 has already shown that the requirements on future production control will require an agent-oriented approach, the following review nevertheless starts

with conventional approaches to software design. This is done for two reasons. First of all, conventional approaches, in particular object-oriented approaches, are necessary to understand the manufacturing control and some of the agent-oriented design methodologies. And second, some aspects of structured and object-oriented approaches have been incorporated into the DACS design methodology.

3.2 Data-oriented and structured design methodologies

The very first design methodologies developed in the 1970's were based on the contemporary programming concepts, namely data and functions. In order to create a program design, these methodologies either started from the input and output data (data-oriented methodologies) or from the functions necessary to process the data (structured methodologies). Both classes of methodologies are shortly reviewed in the following subsections.

3.2.1 Data-oriented methodologies

Jackson (1975) developed a data-oriented approach to designing functional programs, i.e., programs which convert well-defined input data into the corresponding output data as specified by the program function (Budgen 1994, p. 178). This design approach, called the *Jackson Structured Programming* (JSP) method, starts with the data structures of the program input and output and derives a hierarchical program structure by first merging the input and output of the program into a common (hierarchical) data structure. This common data structure is then transformed into a hierarchy of programming instructions by assigning to each node of the program hierarchy the data operations that are necessary to convert the input data of the node into the corresponding output data (see figure 3.2). The motivation for this data-oriented approach was to create a program whose internal structure reflects the structure of the input and output data, because this structure was expected to change less frequently than the programming instructions.

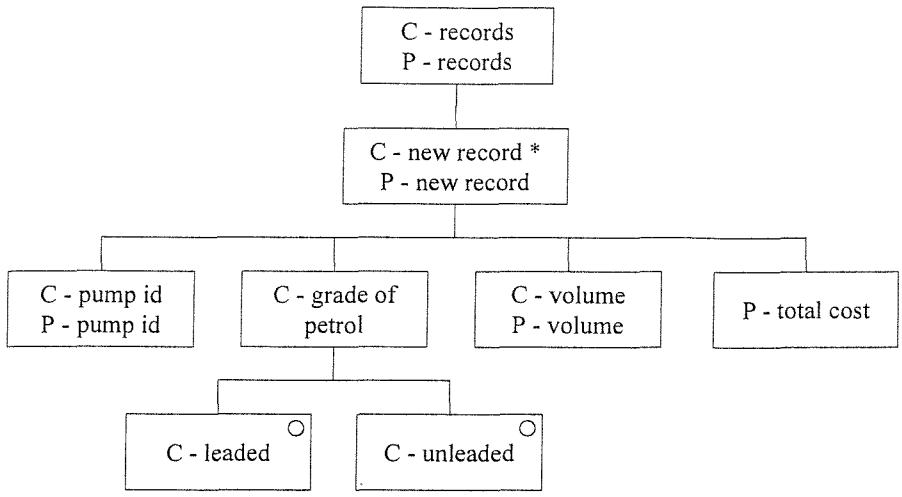

Figure 3.2: Example of a Jackson Structured Diagram
with the data structures consumed (C) and produced (P) by each node
(Budgen 1994, p. 187).

In later work, Jackson and co-workers generalised JSP into an analysis and design method, which was called the *Jackson System Development* (JSD) approach (Jackson 1983, Cameron 1986, Sutcliffe 1988). JSD shares with JSP the philosophy of deriving the basic system structure from a model of the real world in order to reduce the effects of environment changes on the program. The expressiveness of the real world model, however, was significantly enhanced in JSD. Instead of modelling only functional tasks transforming data structures, JSD starts with a model of 'long-running' interacting concurrent processes performing actions in their environment and derives the functional tasks of the desired system from this analysis model (Budgen 1994, p. 240). Because of this increased emphasis on modelling, the development process in JSD is divided into three main phases: "the Model phase, in which the model processes are selected and defined, the Network phase in which the rest of the specification is developed, and the Implementation phase in which the processes and their data are fitted on to the available processors and memory" (Cameron 1986, p. 222). More specifically, the three development phases consist of the following steps:

- **Modelling phase:** In this phase, the real world "entities and the actions they perform or suffer" are identified (Jackson 1983, pp. 39). In order to express the time-ordering of actions, processes (i.e., sequential sets of actions), are "described by structure diagrams – tree structures whose leaves are the actions" (Cameron 1986, p. 228). These structure diagrams allow sequence, iteration, and selection of actions to be expressed.
- **Network phase:** The network phase elaborates the analysis model into a design specification by identifying the external triggers to the system, linking the processes that need to be executed upon the triggers, and determining the

outputs that are to be generated in the particular situation (Cameron 1986, pp. 228, Budgen 1994, pp. 252). This phase may also specify the timing of the processes by assigning priorities or specifying scheduling rules.

- **Implementation phase:** This phase finally focuses on data design, the mapping of tasks onto available processors, and the actual scheduling of the tasks.

Martin et al. also developed a data-oriented analysis and design methodology. This methodology, though, focused on the modelling of an enterprise and its mapping onto a (distributed) database (Martin 1989). The methodology therefore includes many additional data representation forms, such as decomposition diagrams, dependency diagrams, state-transition diagrams, and entity-relationship diagrams (Martin 1985), as well as several business and data analysis methods (Martin 1989), but introduces no additional methods for analysing or designing software. The applicability of the methodology is thus restricted – as intended by the authors – to information engineering.

3.2.2 Structured methodologies

In contrast to the bottom-up approach of data-oriented methodologies, structured methodologies – in their original form – proposed a top-down functional decomposition strategy for designing software (Budgen 1994, pp. 211). A typical representative of this kind of methodology is the *structured design* (SD) approach, initially proposed by Yourdon and Constantine (Yourdon 1979) – other examples are SSADM (Ashworth 1988) and SADT (Ross 1977, Ross 1977b) (SADT is discussed in subsection 3.4.1). The basic idea of SD is to partition the computational system into black boxes, which are organised in a hierarchy of control (see also (Page-Jones 1988)). The modularization of the system is intended to lead to manageable and cost-minimal systems. To assess the optimality of a modularization, Yourdon and Constantine introduced the notions of *coupling* and *cohesion*. According to (Yourdon 1979), coupling is "a measure of the *strength* of interconnection", whereas cohesion is the "intramodular functional relatedness" (p. 85 and 106). A system is considered to be well designed if the cohesion is strong and the coupling is weak.

To arrive at a good "structured" design, Yourdon and Constantine propose two design strategies: (i) transform analysis and (ii) transaction analysis. In the transform analysis, the computational problem is first restated as a data flow diagram. The analysis then identifies data flow elements which read, process, or write system data. These are – depending on their position in the data flow diagram – organised into a hierarchical structure chart which prescribes in which order subordinate modules are called (see figure 3.3). The design is said to be *completely factored* if the actual processing of data

is accomplished only by bottom-level modules in the structure chart. In the transaction analysis, several transformation processes are organised into a single structure chart by using a dispatcher at the highest level of the chart. The dispatcher chooses the appropriate module to process the current transaction.

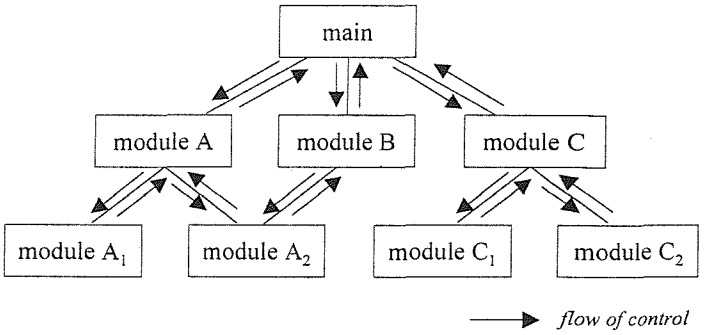

Figure 3.3: An example of a structure chart.

SD was complemented by *structured systems analysis* (SSA) in order to provide a well-structured input to the design process of SD (see for example (De Marco 1978, Gane 1979, Yourdon 1989)). As an analysis method, SSA is concerned with modelling the problem-oriented aspects of the system to be developed and producing a functional specification of what the system is supposed to do (Budgen 1994, p. 212). To create this functional specification, SSA starts with modelling the desired system as a single entity interfacing with its environment. This "context diagram" is then refined by expanding the single entity "system" into a data flow diagram which models the necessary flow of information through the system (Budgen 1994, p. 97). For the refinement step, SSA basically proposes two alternative strategies ((Budgen 1994, p. 214), see also (Yourdon 1989)):

- Top-down functional decomposition: The system function is repeatedly divided into sub-functions until a sub-function is considered sufficiently simple to be specified.
- Event partitioning: The different threads of system actions associated with external events are identified and represented as information flows in the data flow diagram. These information flows are then either grouped to create the overall system structure or further refined by the functional decomposition strategy.

The resulting data flow diagram then represents the functional specification of the system to be developed and is handed over to the SD phase. In addition to the data flow diagram, the SSA may also produce data dictionaries, process specifications (structured English, decision tables, or decision trees), and data store descriptions to complete the functional specification.

An important extension of SSA/SD for control applications was proposed by Gomaa (1984) with the *Design Approach for Real-Time Systems* (DARTS). DARTS introduces the concepts of concurrency and system states into the analysis and design process of SSA/SD in order to meet the specific requirements of real-time systems. Concurrency is introduced by separating the functions of the data flow diagrams into tasks which can be executed concurrently. Criteria for separating tasks are the dependence on specific I/O devices, time criticality, high computational requirements, or periodic execution, while functions should be grouped into one task if they are functionally or temporally cohesive. To design the concurrent tasks, DARTS extends SD in that the resulting transaction dispatcher not only chooses the right function for some input data, but also manages the corresponding system states and passes these to the function. Furthermore, DARTS provides four communication interfaces to enable tasks to interact during execution: message-based communication with/without a message queue, task synchronisation through events, and a shared data repository.

3.2.3 Evaluation

Data-oriented and structured methodologies were the first systematic design methodologies to be proposed, and therefore represent important milestones for software engineering. In particular, these methodologies defined models for capturing a design, clearly separated analysis and design activities, and introduced design measures, such as coupling and cohesion, in order to assess a design. Despite these achievements, however, these methodologies are insufficient to design agent-based production control, or even agent-based systems (see requirement II). The basic computational model of all these methodologies is a hierarchically organised, functional program transforming a distinct set of input data into the desired output data. Some methodologies, like JSD and DARTS, do introduce distribution and concurrency into the computational model. A node of the distributed system, however, is still a functional program, and the communication between the nodes is modelled as pure message passing or shared data access. The underlying computational model of the data-oriented and structured methodologies is thus clearly inappropriate for modelling and designing the autonomy and flexible interactions of agent-based systems – a deficit that is partly remedied by object-oriented programming.

3.3 Object-oriented design methodologies

In many ways, object-oriented methodologies break with conventional approaches[9] to software development (Fichman 1992). This methodological revolution is caused by the

[9] The term *conventional* approaches or methodologies is used by object-oriented experts to refer to pre-object-oriented work (as discussed in section 3.2).

underlying object-oriented programming paradigm, which is characterised by four basic concepts: *object*, *class*, *inheritance*, and *method invocation* (see for example (Korson 1990, Wirfs-Brock 1990)). Objects are the basic building blocks of an object-oriented model or program. An object contains data and is associated with operations on this data. In contrast to the programming concepts of data-oriented and structured approaches, an object thus encapsulates both state and behaviour. A class defines a set of objects with the same data structures and operations. By declaring data structures and implementations of operations as private to a class, object-oriented programming supports the design principle of information hiding. Inheritance, in turn, makes it possible to derive the definition of a class from an existing class, and thus to re-use the code of a class. The derived class inherits all data structures and operations, but may alter or extend both. Finally, method invocation allows objects to interact. An object can request a (public) operation from another object by sending it a message corresponding to a call of this method. The receiving object then immediately executes this request. As with conventional procedures or functions, the method simply returns the result of a function or a notification upon termination. Method invocation can thus be best characterised as client-server interaction (Wirfs-Brock 1990). The requesting object is the client and the executing object is the server. The server always executes the request and terminates the interaction after execution.

The object-oriented programming paradigm prompted the development of new analysis and design methodologies because conventional methodologies were felt to be no longer appropriate. This section therefore evaluates the main object-oriented design methodologies with respect to the design of agent-based (production control) systems.

3.3.1 Object-Oriented Design and Object-Oriented Analysis and Design

The methodologies *Object-Oriented Design* (OOD) proposed by Booch (1991) and *Object-Oriented Analysis and Design* (OOAD) proposed by Coad and Yourdon (1991, 1991b) are both bottom-up design approaches that create an object-oriented system by first identifying the objects (and the classes) of the system and then elaborating other aspects of the design, such as inheritance or interaction. The OOD methodology of Booch, for instance, consists of four major steps (pp. 190):

1. *Identify classes and objects.* Identify the key abstractions in the problem space and label them as candidate classes and objects. To identify the key abstractions, collect typical "objects", such as (physical) things, people, roles, events, and so on, from a description of the problem domain (pp. 141).
2. *Identify the semantics of classes and objects.* Establish the meaning of the classes and objects by analysing "scripts" that define the life cycle of each

object from creation to destruction. The result is captured in object and class diagrams.

This step is also supposed to identify the mechanisms between the objects, i.e., the messages the objects should exchange to solve the system task. OOD, though, does not specify how these interactions should be identified or designed.

3. *Identify relationships between classes and objects.* Relationships can be either inheritance or visibility between objects and classes.
4. *Implement classes and objects.*

Similarly to OOD, the OOAD methodology of Coad and Yourdon prescribes five major steps to design the object-oriented system bottom-up from the objects and classes (p. 34):

1. *Finding classes and objects.* Find the key abstractions of the problem domain (with basically the same methods as proposed by OOD).
2. *Identify structures.* Identify generalisation and specialisation relationships, in particular inheritance and composition.
3. *Identify subjects.* Identify entities which are too large for objects (containing several different object types).
4. *Define attributes of objects.*
5. *Define services to other objects and specify the computation of a service.*

With step 5, OOAD goes beyond OOD in that it explicitly identifies the services an object should provide to other objects. In OOAD, services (and thus interactions) are identified by looking at the computations an object is able to perform (and is thus able to provide as a service) and the need of the other objects to receive this service in order to fulfil their own computational tasks. Whenever such a link is identified, the corresponding service, i.e., the object method interface, is specified.

During the design phase of OOAD, the analysis results are mapped onto a problem domain, a human interaction, a task management, and a data management component, and the results are modified according to implementation considerations, if necessary (Coad 1991b).

3.3.2 Object Modelling Technique

The *Object Modelling Technique* (OMT) proposed by Rumbaugh et al. (1991) is an elaborated object-oriented analysis and design methodology. It produces three models which each capture a different view of the software system (pp. 17).

- The *object model* describes the structure of objects in a system, in particular their identity, their relationships to other objects, their attributes, and their operations.
- The *dynamic model* describes those aspects of a system concerned with time and the sequencing of operations, in particular events that mark changes, sequences of events, states that define the context for events, and the organisation of events and states.
- The *functional model* describes those aspects of a system concerned with transformations of value, in particular functions, mappings, constraints, and functional dependencies.

Each model is associated with a rich set of modelling tools in order to represent different aspects of object-oriented systems: (i) the object model consists of object and class diagrams with attributes, operations, links between objects/classes, link attributes, roles, aggregation, multiple inheritance, modules, meta-data, and constraints (on objects and classes); (ii) the dynamic model consists of (nested) state diagrams with events, conditions, actions, and concurrency; and (iii) the functional model consists of data flow diagrams with processes, data flows, actors, data stores, and control flows.

To derive the different models, OMT proposes an analysis phase with a sequence of steps for each model (pp. 148). For the object modelling, potential objects are identified through a linguistic analysis of the problem statement (i.e., by listing the nouns found in the written description of the problem). For the dynamic modelling, the interactions between objects are identified through the analysis of typical scenarios in which the object-oriented system is used. In this analysis, the external events received by the system are traced through the objects in order to determine which object needs to pass or request information from another object. Finally for the functional modelling, the required computations of each object are specified.

After the analysis phase, details of the implementation are determined (this phase is called design in OMT). The design includes, among other things, the organisation of the system into subsystems, the identification of concurrency, the allocation of subsystems to processors, and the implementation of each object (p. 199).

3.3.3 Hierarchical Object-Oriented Design

In contrast to the previously discussed object-oriented methodologies, the *Hierarchical Object-Oriented Design* (HOOD) method proposed by Robinson (1992) takes a hierarchical, top-down approach to the design of object-oriented systems. The method starts with a specification of the program to be developed, regarded as the *root object*, and successively decomposes an object of level `i` into a set of child objects at level

`i+1` until an object is marked as a *terminal object*. The result of this design method is a design tree with the root object as the root of the tree and terminal objects as the leaves.

Each design step leading to the introduction of a new level of the design tree is structured into four phases (pp. 14):

1. **Problem definition:** The context of the object to be refined is stated, with the goal of organising and structuring the data from the requirement analysis phase.
2. **Development of solution strategy:** The outline solution of the problem stated above is described in terms of objects at a high level of abstraction.
3. **Formalisation of the strategy:** The objects and their associated operations are defined. This phase has five sub-phases:
   a) Object identification
   b) Operation identification
   c) Grouping of objects and operations
   d) Graphical description
   e) Justification of design decisions
4. **Formalisation of the solution:** The solution is formalised through the formal definition of provided object interfaces and the formal definition of object and operation control structures.

The identification of objects and operations in phase 3 is based on a linguistic analysis of the requirements specification, as in most other object-oriented methodologies. Candidate objects are identified by looking at the nouns and noun phrases, and possible operations are identified by examining the verbs of the specification. An alternative approach is proposed if the specification is given in terms of data flow diagrams. In this case, objects are identified in the data flow diagram by grouping either external interfaces or internal data stores with the corresponding data flows.

For the specification of the resulting design, HOOD provides an object description skeleton covering interfaces, data flows and control structures; a class skeleton; and a diagram representing "include" relationships.

3.3.4 Responsibility Driven Design

Wirfs-Brock, Wilkerson, and Wiener (1990) propose a design process on the basis of the concepts responsibility and collaboration, called *Responsibility Driven Design* (RDD). An object is said to be responsible for providing its service, i.e., its operations, to any object which may request it. A collaboration is said to be present when an object requests a service from another object; two collaborating objects are said to have a

service contract. On the basis of this terminology, RDD suggests three initial steps in the design process (p. 29):

1. Find the classes of the system (through a linguistic analysis of the requirements specification).
2. Determine what operations each class is responsible for performing, and what knowledge it should maintain (by analysing typical scenarios).
3. Determine the way in which objects collaborate with other objects in order to discharge their responsibilities (by identifying operations an object is unable to perform on its own and the corresponding objects that can provide the missing knowledge or capabilities).

To support the design process, the methodology provides schemata for recording class responsibilities and collaborations, as well as graphs for defining inheritance hierarchies, visualising collaborations, and defining subsystems. In later steps of the design, Wirfs-Brock et al. also talk of protocols between objects. Protocols, however, only define the specific signatures for method invocation.

3.3.5 Evaluation of object-oriented methodologies

Because of the encapsulation of both state and behaviour in a single object, the object-oriented approach provides a more powerful model for capturing the characteristics of real-world domains than conventional approaches (Johnson 2000). This model, however, still misses many essential aspects of domains which consist of interacting agents. Jennings points out that (i) objects are generally passive in nature; (ii) objects do not encapsulate behaviour activation; (iii) object-oriented concepts and mechanisms are too fine-grained to model complex systems; and (iv) object-oriented approaches provide only minimal support for structuring organisations (Jennings 2001, p. 39; see also Wooldridge 1997). On the contrary, agents are conceptualised in terms of goals, high-level interactions, and organisational relationships: (i) agents pro-actively follow their own goals and decide autonomously what to do and with whom to co-operate; (ii) they are able to initiate and execute interactions with other agents that were not foreseen at design time; and finally (iii) agents are able to take into account and also to adapt their organisational relationships (Jennings 2000, pp. 280). Objects are thus an inappropriate concept for modelling interacting agents (Booch 1994, Wooldridge 1997). These limitations of objects are only partly remedied by concurrent objects (Agha 1990) which only add pro-activeness to the objects without addressing the other limitations (Wooldridge 1997).

Because of the general limitations of objects to capture essential features of agents, the object-oriented approach falls equally short to model the production control systems

envisioned in section 2.3. Production control agents are assigned production goals, such as bringing a workpiece into a specified product state, and must autonomously decide how to achieve this goal state irrespective of the current situation in the production system. To achieve these goals, the production control agents must flexibly interact with other agents of the production system either to use their services, such as processing and transportation, or to avoid conflicts with other independently pursued goals. In doing so, the agents may have to form teams or other kinds of organisational structures in order to create an overall coherent system behaviour.

The general inability of the underlying programming concepts to model agent-oriented aspects also makes the associated design methodologies inappropriate for designing agent-based (production control) systems. At first glance, identifying "objects" and their interactions is the same design task that is necessary for agent-based systems. But the tools for identifying objects and their interactions miss many aspects of agents and thus lead to a wrong design for agent-based systems. Object-oriented methodologies identify objects either through the identification of key abstractions (OOD, OOAD), or a linguistic case analysis of the problem description (OMT, HOOD). Interactions, in turn, are identified through an analysis of the relationships of objects (OOD), through the definition of services (OOAD, RDD), through the analysis of typical scenarios (OMT), or through a linguistic case analysis (HOOD). These identification techniques are clearly insufficient for agent-based systems. First of all, these techniques may identify too many objects which are not agents. Many entities in a production system are relevant to the control system, but only a few are able to make decisions about the course of the production process. Secondly, these techniques may even fail to identify necessary agents. Instead of identifying an agent, the identification step may find objects which are part of the agent or overlap with it because of common data structures. Finally, even though some methodologies identify services or patterns of co-operation among objects, the modelling tools do not provide sufficient support for modelling complex multi-stage interactions. For interaction modelling, the methodologies only provide use-relationships, message connections, or event and state diagrams, but no means to model purpose, reasoning, or flexibility of agent interactions (see section 2.2). This overall critique equally applies to more recent object-oriented design methodologies, such as Fusion (Coleman 1994) or OPM (Dori 2002).

The use of an agent-oriented terminology in some methodologies does not remedy these limitations. The anthropomorphic approach of RDD to design, for instance, can only be viewed as "an aid to conceptualization" (Wirfs-Brock 1990, p. 7). A close examination of the concepts reveals that the terms "responsibility" and "collaboration" are merely metaphors for ordinary object-oriented concepts, like public methods and method invocation. Similarly, the term "protocol" is equated with method signature. Even though the metaphors may be helpful in ordinary object-oriented design, they are misleading in the design of an agent-based system, as these terms have a different

meaning in agent technology. Deprived of the metaphoric terms, the RDD methodology operates on the same level of abstraction as other object-oriented methodologies. The basic building block is an object which encapsulates information and operations, and objects interact through method invocation. The critique put forward above therefore equally applies to this methodology.

All in all, despite the prescriptiveness of the methodologies from the object-oriented point of view and the agent-like terminology in some of the methodologies, object-oriented methodologies fail to adequately model agent-oriented concepts and to provide the rationales related to an appropriate agent-oriented design. The object-oriented methodologies are therefore not only inappropriate because of their underlying model, but are also insufficiently prescriptive for designing agent-based systems.

Despite their limitations with respect to agent-oriented design, object-oriented methodologies, however, have made their contribution to the state-of-the-art in designing software systems. Object-oriented methodologies start by identifying the entities of the domain and transforming these into analysis and design objects. Object-oriented methodologies thus provide a more natural and appropriate design process than earlier methodologies. In conjunction with the encapsulation of state and behaviour, object-oriented methodologies have also clearly separated the design of the objects (and their functions) from the design of the interactions between the objects. These two important improvements should therefore be honoured in an agent-oriented design approach.

3.4 Manufacturing control design methodologies

The general software design methodologies discussed in the previous sections should in principle be also applicable to designing manufacturing control systems as these are basically software systems. Manufacturing control, however, exhibits some peculiarities that require special attention during the design process (Parunak 1987). In particular, the software system to be developed is always supposed to control a physical system, namely the manufacturing system (cf. figure 2.5). The control system must therefore have interfaces for sensing and acting in this physical system, and these control interfaces must be served in (soft) real-time because the processes behind these interfaces have their own physical dynamics. Moreover, a control system must take into account the fact that physical actions may fail or unexpected events may occur. Basically, control software must meet the requirements discussed in subsection 2.1.4. To address the above peculiarities of control software, manufacturing research has therefore developed methodologies tailored to the development of manufacturing control systems. Most of these methodologies, though, are concerned with modelling and specifying the control system (see for example (Castillo 2002, Booth 1998)). This

section will therefore review only those methodologies that cover at least some aspects of the design of control systems.

3.4.1 SADT/IDEF-based design of manufacturing systems

The *Structured Analysis and Design Technique* (SADT), originally developed for software systems (Ross 1977, Ross 1977b), has repeatedly been used to model manufacturing systems. SADT is a structured analysis methodology that follows a functional decomposition strategy in order to create a hierarchical model of the system to be designed (Budgen 1994). The basic building block of this model is a black box representing an operation or function of the system. This operational box has four types of interfaces that are distinguished graphically: (i) inputs to the operation are shown entering on the left side; (ii) control flows are shown at the top; (iii) outputs emerge on the right side; and (iv) mechanisms, such as tools, are provided at the bottom (see figure 3.4).

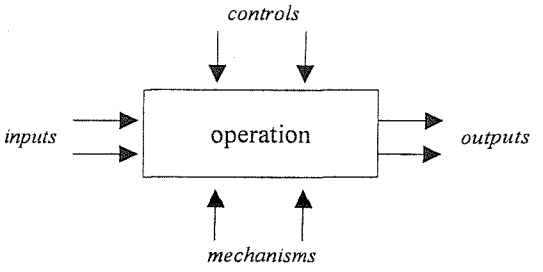

Figure 3.4: The SADT box representing an operation.

Different operations are combined by linking their interfaces in a diagram (see figure 3.5). A link, though, does not necessarily imply a flow of material or information. It only states that there is a dependency between the operations, and thus the goal operation of the link is constrained by the source of the link. Furthermore, an operation can be refined by creating a new diagram representing the "parent box". For the new diagram, the set of external interfaces must be identical to those of the parent box. This refinement mechanism allows a hierarchy of diagrams to be created with the top diagram representing the system to be developed and the main steps of the SADT methodology are actually concerned with creating this hierarchy of operations (Budgen 1994). Strictly speaking, SADT is thus only a specification and not a design methodology, as its name suggests.

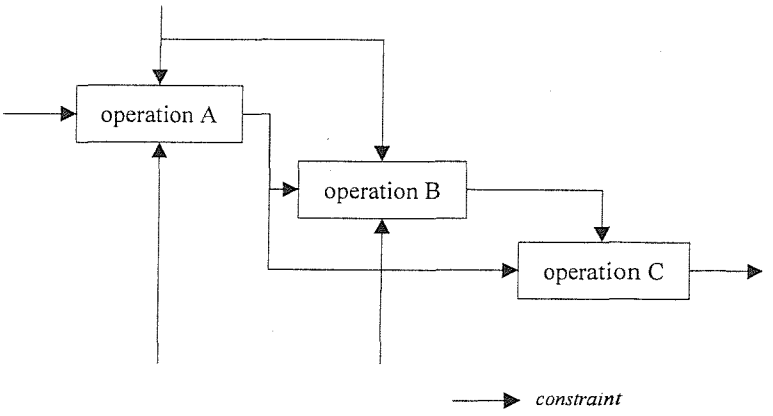

Figure 3.5: An SADT functional network.

SADT was adopted in the 1970s by the *Integrated Computer Aided Manufacturing Definition* (IDEF) initiative as the standard IDEF0 for modelling the functions of a manufacturing system (Bravoco 1985). Since then, both SADT and IDEF0 have been used in several methodologies for designing manufacturing (control) systems. One such example of a design methodology for control systems employing SADT is the methodology for designing logic controllers proposed by Zaytoon (1996) (see also subsection 3.5.2 for other SADT/IDEF0-based methodologies). This design methodology integrates SADT with Petri Nets and Grafcets[10] by transforming an SADT design into a Petri net and then into a set of Grafcets programs for the local controllers. To achieve this transformation, Zaytoon defines a set of temporal relationships for synchronising the control operations which are added to the initial SADT design of the control system. Once added, the augmented SADT design can be automatically transformed into a Petri net that executes the control operations in the same order as specified by the temporal relationships. The advantage of the Petri net representation is that certain properties of the system behaviour can be verified automatically. After verification, the Petri net is again converted automatically into a set of Grafcet skeletons which then must be filled with Grafcet commands specifying the actual control steps.

The strength of the methodology proposed by Zaytoon is certainly the verification step and the matching onto a programming language for logic controllers. Just like SADT, however, this methodology does not provide any techniques for creating the initial control design. For the initial design, Zaytoon relies completely on SADT which, as pointed out above, is only a specification methodology. Neither SADT nor the methodology of Zaytoon specify how to identify the necessary control operations, how to structure these into a functional hierarchy or map these onto a set of local controllers, and how to identify and design the interactions of these controllers. Zaytoon does

[10] Grafcet is a controller programming language.

define a set of temporal relationships for synchronising the control operations. His methodology, though, does not provide any criteria for when and where to add these relationships to the design model.

3.4.2 Modelling control with discrete event systems

Fanti et al. (1996) provide a modelling framework for designing generic control software of flexible manufacturing systems, with the focus on the job release and material flow decisions. The model is based on the theory of discrete event systems (Zeigler 1984) and essentially captures the entities of the manufacturing system (resources, operations, and jobs), the associations between these entities (static and dynamic relations specifying the configuration of the manufacturing system and its state changes), and in particular the decision rules necessary to control the flow of material in the manufacturing system. This model, however, is only a basic architecture for generic control software which must be instantiated for a given manufacturing system by specifying the actual decision rules with which the manufacturing system should be controlled. And just like the SADT-based methodologies, the framework of Fanti et al. does not provide any techniques for identifying the necessary control rules. Furthermore, the envisioned (generic) control system is centralised. It consists only of a knowledge base capturing the system's state, one job release manager and one job flow manager. The modelling framework is thus not even able to model the distributed nature of agent-based control systems.

3.4.3 Petri net-based methodologies

A third common formalism to model and design control systems are Petri nets. A Petri net is a directed graph whose nodes are either places or transitions (Murata 1989). Places are able to hold tokens, while transitions move tokens from the input places to the output places of the transition once all input places are occupied by a token. An example of a Petri net controlling a simple manufacturing workcell with two machines and a loading device that are connected via a roundtable is depicted in figure 3.6. First, the roundtable is moved to the next position (places P1/P2). Then, each machine may process its workpiece concurrently until the machines have to synchronise again in order to advance the roundtable further.

With respect to control design, Petri nets have the advantage that they are able to naturally model concurrency in production systems. All transitions are checked for possible movements in parallel and execute their movements as soon as their input places are occupied. Furthermore, as a rigorous formalism, Petri nets lend themselves to automated analysis and verification. A large body of theory has already been developed for verifying network properties such as reachability, absence of deadlocks, liveness,

and so on, which is very important in control design (see for example (Narahari 1985, Zhou 1992)). For this reason, several researchers have adopted the Petri net formalism as a basis for their methodology. The following subsections discuss the major work in this area.

3.4.3.1 Synthesising Petri-net-based control programs

Ferrarini (1992) has developed a methodology for synthesising control programs from smaller ones. A control program consists of elementary control tasks which implement a (cyclic) sequence of control actions. These elementary control tasks must be combined into a larger control program by managing the interactions between the elementary control tasks. This can be done through three possible co-operation mechanisms: condition, inhibition, and synchronisation.[11] In his methodology, Ferrarini has shown that the application of the co-operation mechanisms to the specific types of control programs considered preserves certain properties of the control program. These properties include the correctness as a control program (e.g., there must always be exactly one token in an elementary control task), absence of deadlocks, liveness, and so on. The methodology of Ferrarini, however, does not consider how to identify the elementary control tasks, how to identify the need for co-operation, and how to map the elementary control tasks to the available controllers in a production system. Ferrarini's methodology is thus only applicable once the overall design of the control program has been created.

[11] Condition is the presence, and inhibition the absence of a token.

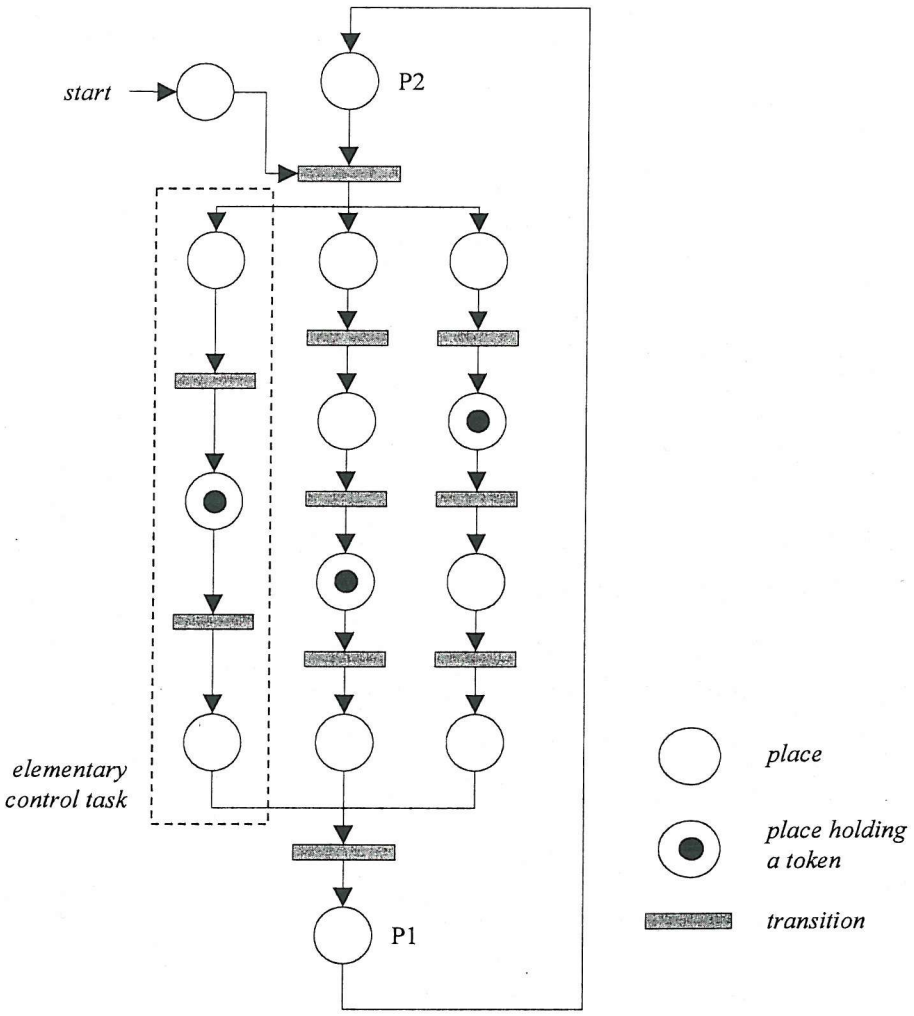

Figure 3.6: Example Petri net-based control program (Ferrarini 1992).

3.4.3.2 Knowledge-based design of control programs

Caselli et al. (1992) have developed a knowledge-based methodology for designing manufacturing workcells which uses Petri nets to model the control aspects of the workcell. The methodology starts by creating a structural and functional model of the workcell with relations expressing generalisation (G), structural aggregation (A), functional aggregation (F – meaning sequential execution), and iteration. Figure 3.7 shows an example of a functional knowledge schema for the processing of a component.

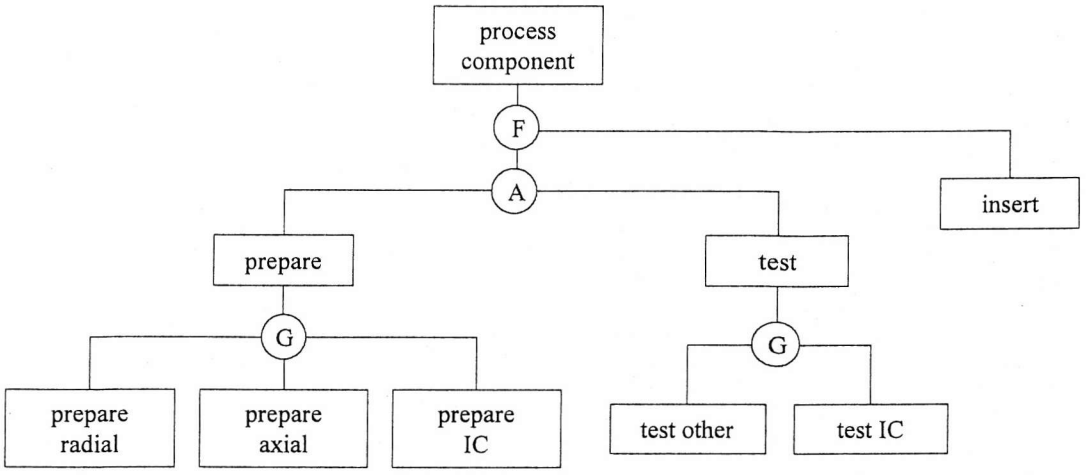

Figure 3.7: Functional knowledge schema (Caselli 1992).

The control program in the form of a Petri net is synthesised from this schema. That is, a functional aggregation is converted into a sequence of places and transitions covering all steps included in the aggregation (see figure 3.8); a generalisation is converted into a selection construct executing only one path in the Petri net; and so on. The conversion of each relation then leads to the Petri net executing the functional knowledge schema.

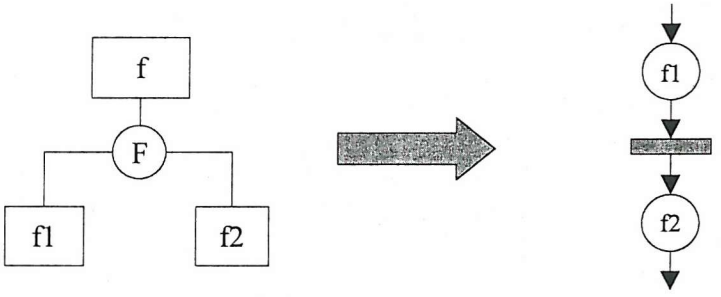

Figure 3.8: Conversion of functional aggregation.

This approach, however, is clearly insufficient for designing a real control program. In contrast to Ferrarini, this methodology does identify the elementary control tasks by deriving these from the functional description of the product and the production system. However, it does not address any interactions between the elementary control tasks which may arise from the parallel processing of several workpieces and does not perform the mapping of the Petri net onto a control architecture for the production system. This methodology therefore covers only the very first step of the control design, namely converting the manufacturing plan of a product into a set of control commands.

3.4.3.3 Conflict-driven design of control programs

Feldmann, Schnur, and Colombo (1996) have proposed a Petri net-based methodology that creates the control design by analysing the conflicts that arise during the production process. The first step of this methodology is therefore to model the production process as a Petri net (see also (Feldmann 1998)). This model is then analysed with respect to the occurrences of structural and behavioural conflicts. A structural conflict is given if a place has two (or more) outgoing transitions of which only one may be chosen. In this case, there is a choice as to which transition may fire. A behavioural conflict is given if two (or more) transitions "model the work from two resources which compete for exclusive permission to use a shared resource" (Feldmann 1996, p. 1069). For example, two adjacent robots (each represented by a transition) want to put a workpiece into a common buffer. Both types of conflicts must be resolved in order to operate the production system.

To resolve structural and behavioural conflicts, Feldmann et al. (1996) took a hierarchical approach (see figure 3.9). First, structural conflicts are resolved by assigning a local controller (LC), which makes the necessary decision, to each structural conflict. Then, in a second step, each behavioural conflict is resolved by a complex controller (CC). The complex controller receives a request from a local controller whenever a behavioural conflict with another local controller arises. In such a case, the complex controller resorts to a global production schedule that is produced by higher level production management functions and allows only one of the controllers in conflict to perform its actions. The other controllers must wait or choose a different action.

This design methodology does go beyond the previous manufacturing control methodologies in that it identifies the necessary control decisions, assigns most of these to local controllers, and handles their interactions. But the methodology still lacks some important aspects of agent-based manufacturing control design. First of all, the necessary control decisions identified in the Petri net are assigned to the physical resources at which they arise. This results in a purely resource-based control architecture which does not allow other types of agents, such as workpiece or tool agents, to be explored. Also, it is up to the designer to decide at which granularity agents will be identified. A structural conflict could be resolved at the level of a device, a machine, a station, or a cell agent. The optimal granularity, though, depends on the application and may even differ within an application.

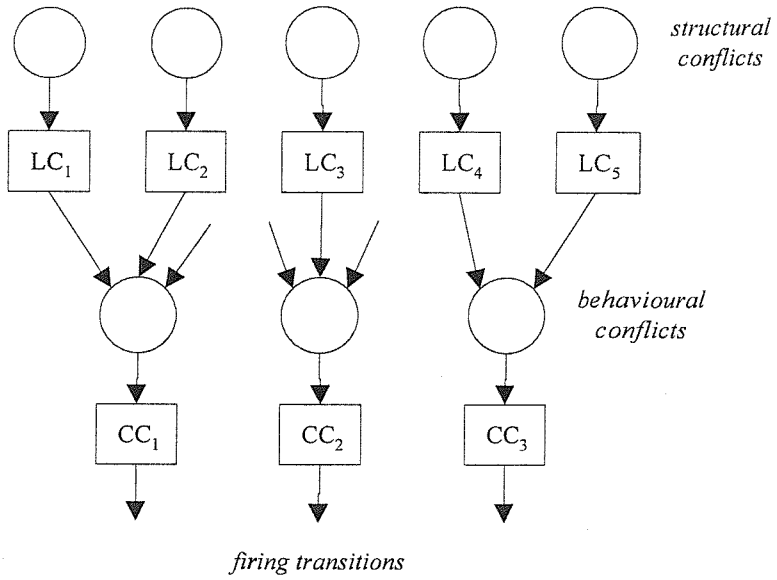

Figure 3.9: Hierarchical procedure for resolving conflicts.

Furthermore, the methodology does handle interactions between the local controllers, but only in a centralised manner. Any (behavioural) conflict between the local controllers is resolved by a complex controller dedicated to the specific conflict type, and the complex controller resolves the conflict only by resorting to a global schedule. The resulting design is thus actually a hierarchical and schedule-driven control system (cf. subsection 2.1.3). Moreover, the resulting control system may not identify all possible interactions between the local controllers. Behavioural conflicts, for instance, do not cover the synchronisation of the material flow in assembly systems when two parts must choose the same assembly station in order to be assembled into one product. To arrive at a truly agent-based design for manufacturing control, the above design methodology (as well as the other control methodologies) must therefore be significantly extended with respect to the design of the agents and their interactions.

3.5 Agent-oriented design methodologies

The difficulties in designing agent-based systems using conventional, object-oriented, or even manufacturing control methodologies, as discussed in the previous sections, has led to the development of agent-oriented design methodologies which explicitly model agent-oriented aspects. At first, these methodologies were based on existing methodologies, in particular knowledge-oriented and object-oriented approaches. However, with the growing maturity of agent-oriented concepts many methodologies have been proposed that are based on purely agent-oriented concepts, such as roles, autonomous behaviour, or organisations. Because of the explicit use of agent-oriented

concepts during the design, these methodologies are presumably appropriate for the development of agent-based systems. This section will therefore focus its review on the suitability of these methodologies for the design of agent-based *production control* systems (cf. subsection 3.1.1). In particular, this section will examine whether the design models used are appropriate for the design of agent-based production control systems (requirement III), and to what extent the design methods are sufficiently prescriptive with respect to the agent-oriented aspects of such a control system (requirement IV).

3.5.1 Extensions of knowledge-oriented methodologies

The knowledge-oriented methodologies proposed for designing agent-based systems are mostly extensions of the knowledge-engineering methodology CommonKADS (Schreiber 1994). These methodologies adopt the CommonKADS approach and add agent-oriented concepts to it. There are two examples of knowledge-oriented methodologies for agent-based systems, *CoMoMAS* and *MAS-CommanKADS*, which will both be reviewed in the following.

3.5.1.1 CoMoMAS development methodology

CoMoMAS is an agent-oriented analysis methodology (Glaser 1997). The aim of CoMoMAS is to build a set of conceptual models for the desired multi-agent system. The development process is therefore regarded as a modelling activity which consists of five analysis steps, each taking a different view of the system. The five analysis steps, which can be performed in parallel, are the following (Glaser 1997, pp. 7):

- **Requirement Analysis:** The purpose of requirements analysis is to determine the design requirements on the multi-agent system.
- **Functional Analysis:** The purpose of the functional analysis is to determine the set of tasks the multi-agent system has to solve.
    1. Analyse the application domain and problem space.
    2. Determine goals, tasks, and their decompositions.
    3. Determine data and control flow.
    4. Build and validate the initial task model.
- **Competence Analysis:** The purpose of the competence analysis is to identify the competences which the system should provide in order to solve the goals of the task model.
    1. Determine the problem-solving methods.

2. Determine the necessary resources.
3. Determine the strategies.
4. Determine the behaviours and patterns of behaviour.

- **Social Analysis:** The purpose of social analysis is to identify the social competencies that are required by the agents.
  1. Identify conflicting actions and goals between agents.
  2. Identify goal and data dependencies.
  3. Determine the agent organisation and associate roles to agents.
  4. Determine and associate intentions to actions; desires to goals; commitments to goals; and beliefs to goals.

- **Co-operative Analysis:** The purpose of co-operative analysis is to identify the co-operation protocols, the co-operation methods, and the conflict resolution methods for the agents.
  1. Identify the needs and levels of co-operation between agents.
  2. Identify conflict raising facts in agent interaction.
  3. Determine co-ordination and negotiation strategies.

Agent models are derived from the five conceptual models constructed in the above analysis phase through the integration of reactive, co-operation, and social knowledge and competencies. An agent model is constructed as follows (Glaser 1997, p. 9):

1. Analysis of task model to build sub-trees or subsets of tasks.
2. Identification of agents based on categorisation of tasks.
3. Construction of agents with cognitive and/or reactive knowledge.
4. Integration of co-operation knowledge into agent model(s).
5. Integration of social knowledge into agent model(s).

The resulting agent models represent the design result of the CoMoMAS development process.

3.5.1.2 MAS-CommonKADS analysis and design

Iglesias et al. (1998) also extend the CommonKADS methodology to the analysis and design of multi-agent systems. Their methodology adds techniques from various object-oriented methodologies (e.g., OMT (see subsection 3.3.2)) in order to capture agent-oriented aspects. The methodology starts, after a conceptualisation phase in which the

requirements specification for the multi-agent system is developed, with the construction of various models (Iglesias 1998, pp. 315).

- **Agent modelling:** Develop initial instances of the agent model for identifying and describing agents.
- **Task modelling:** Decompose tasks and determine goals and ingredients of tasks.
- **Co-ordination modelling:** Develop the co-ordination model for describing the interactions and co-ordination protocols between the agents.
- **Knowledge modelling:** Model the knowledge of the domain.
- **Organisation modelling:** Develop an organisational model.

In the agent modelling step, agents are identified according to the following heuristics (Iglesias 1998, p. 316).

- Identify actors in the use cases of the conceptualisation phase.
- Syntactically analyse the problem statement and identify active objects.
- Assign entities to different agents if they are geographically, logically, or organisationally distributed or the required knowledge is distributed.
- Analyse the initial task model in order to identify the necessary functions that can be assigned to agents.
- Analyse the usage of agents by other agents and the different roles played in the domain.

The next phase, the *co-ordination modelling*, defines the communication channels, builds a prototype, and analyses the prototype in order to optimise the interactions. This is accomplished in five steps (Iglesias 1998, p. 318).

1. Describe the prototypical scenarios between agents.
2. Represent the events or messages between agents in event flow diagrams.
3. Model the data exchanged in each interaction.
4. Model each interaction with state transition diagrams.
5. Analyse each interaction and determine its synchronisation type (e.g., synchronous or asynchronous).

The models derived in the analysis phase are transformed into a design model. The design model specifies the infrastructure of the agent system (network, knowledge, and co-ordination facilities), the design of the agents (choice of a suitable agent architecture), and the choice of an implementation platform.

3.5.1.3 Evaluation

Because of the underlying knowledge-engineering approach, both methodologies view an agent system as a problem solving system decomposing the system task into subtasks according to a task hierarchy. In production control, however, goals are decomposed and distributed depending on the current situation, and local goals derived during the negotiation process allow controllers to autonomously choose their actions when unplanned situations arise on the shop floor. Instead of modelling a pre-defined task hierarchy, it is therefore necessary to model the decision situations that may arise during control, as all manufacturing methodologies do (see section 3.4), and to enable the agents to incorporate the goals into the control decisions at run-time. This is not easy with a top-down approach (cf. subsection 2.1.3).

Furthermore, the identification of agents is either based on the task hierarchy (CoMoMAS) or on heuristics (MAS-CommonKADS). CoMoMAS proposes identifying agents based on sub-trees of the task model. As pointed out above, a pre-defined task hierarchy is inappropriate for modelling the control process and thus also for identifying agents. Moreover, CoMoMAS is also not prescriptive with respect to the identification of agents because it does not elaborate which sub-trees should be assigned to agents. MAS-CommonKADS, on the other hand, provides several heuristics for identifying agents, such as linguistic case analysis, use cases, or conceptual analysis. Even though the heuristics help to identify the agents, the heuristics are not sufficiently prescriptive (see also section 3.3) and there are no guidelines on when and how to apply which heuristic. In particular, the methodology does not provide a clear (abstract) agent model which is required to provide criteria for identifying agents. Agent identification in MAS-CommonKADS therefore relies primarily upon the experience of the designer, despite the heuristics proposed.

Finally, both methodologies provide only general guidelines for interaction design. CoMoMAS proposes identifying the need for interaction based on resource sharing or goal conflicts, but does not elaborate how the need can be identified and how co-operation methods and protocols can be derived from the identified need for co-operation. MAS-CommonKADS uses scenario analysis to identify interaction patterns, but does not provide any guidelines to identify relevant scenarios or to ensure that the scenarios are complete. Interaction design is thus also based on intuition.

3.5.2 Extensions of object-oriented and manufacturing methodologies

This subsection reviews extensions of object-oriented and manufacturing design methodologies. Some of these extensions are general-purpose agent-oriented methodologies, some are dedicated to the design of agent-based production control systems.

3.5.2.1 Burmeister – Agent-oriented analysis methodology

Burmeister (1996) extends the object-oriented methodologies to agent-oriented analysis by introducing mental and co-operation concepts. As in object-oriented analysis, she proposes three models which can be developed in parallel (Burmeister 1996, p. 10).

- The *agent model* contains agents and their internal structure, described in terms of mental notions such as goals, plans, and beliefs or similar concepts.
- The *organisational model* specifies the relationships among agents and agent types. Organisational relationships can be inheritance or role-based as in real organisations.
- The *co-operation model* describes the interaction (or more specifically the co-operation) among agents.

To create these models, Burmeister proposes three methods. For the agent modelling, the designer should identify the agents, their motivations, their behaviours, and their beliefs. To identify the agents, Burmeister proposes to look at the "live" and "active" entities in the domain, i.e., at entities which can change their own states and affect the environment through actions. For the organisational modelling, the designer should identify the roles in the scenarios and organise these roles into a hierarchy. Finally, for co-operation modelling, the designer should identify the necessary interactions, list the message types, and specify the necessary co-operation protocols. For the definition of co-operation protocols, Burmeister suggests a specific protocol framework (Burmeister 1995, Haddadi 1995).

3.5.2.2 Agent-oriented development methodologies for manufacturing

Kendall et al. (1996) propose a methodology for developing agent-based systems that builds upon object-oriented methodologies, such as OMT (see subsection 3.3.2) and OOSE (Jacobson 1992), and the IDEF methodology for modelling manufacturing systems (see subsection 3.4.1). The methodology first creates an object-oriented, as well as an IDEF, model of the system to be designed, and then identifies agents and interactions in these two models. Agents are viewed as autonomous decision makers and are identified in the object-oriented model whenever actors appear, and in the IDEF model whenever IDEF functions create control information as output. Identified agents are then modelled as a belief-desire-intention architecture, in particular as a procedural reasoning system (Rao 1992). Interactions, in turn, are identified whenever two or more IDEF resources are involved in an information exchange. The interaction pattern is then taken from the corresponding use case (Jacobson 1992). The agents and interactions identified are further refined with the help of object-oriented design techniques.

3.5.2.3 The PROSA methodology

Van Brussel et al. (1999) propose a methodology for identifying manufacturing agents[12] based on the object-oriented approach and the PROSA framework. The PROSA framework consists of product, resource, and order agents, which may be supported by staff agents. These agents can be aggregated or specialised in the object-oriented sense in order to create a taxonomy of agents for a specific manufacturing application. The first step of the proposed methodology is the identification of the agents and their responsibilities in the specific manufacturing application (other steps are detailed agent design, implementation, and system operation). The identification process is guided by the PROSA framework and starts with an object-oriented model of the manufacturing system. The designer then selects the objects in this model which should become agents, even though the methodology does not provide any criteria for identifying suitable objects. Finally, the agents are classified in the PROSA framework by aggregating and specialising agents. Van Brussel et al. propose to apply this approach to the resource allocation hierarchy and the process planning and execution hierarchy in a manufacturing system.

3.5.2.4 Manufacturing-oriented agentification methods

Ritter et al. (2002) propose an agentification method for manufacturing (i.e., a method for identifying agents in a manufacturing system). The agentification process is based on the aggregation of physically and logically dependent manufacturing objects (taken from an inventory list). The method, however, does not provide precise criteria for the presence of physical or logical dependencies (apart from an example). Furthermore, the method does not include any guidelines specifying to which manufacturing objects to apply the aggregation process and in particular for how long to apply it. The agentification process is thus highly intuitive.

Colombo, Neubert, and Süssmann (2002) extend the Petri net-based modelling approach of Feldmann et al. (see subsection 3.4.3) to the modelling of agent-based production control systems. In their methodology, agents are identified on the basis of a Petri net model of the manufacturing process and its relevant control decisions. Although this methodology provides a more rigorous model for the agentification step than the previous method, it likewise does not provide any criteria for identifying the agents in the Petri net model (other than agents are decision makers and are thus responsible for the control decisions).

[12] Van Brussel et al. (1999) actually identify manufacturing holons (see subsection 2.3.2.2). In (Bussmann 1998), however, it was shown that from an IT point of view, agents and holons can be regarded as similar concepts.

3.5.2.5 Evaluation

Kendall et al. view an agent as an autonomous decision maker and thus capture exactly the key aspect of agent-based production control systems (cf. Subsection 2.3.1). To identify these agents, however, Kendall et al. base their analysis on object-oriented methodologies and the IDEF0 method, both of which do not explicitly model decision making. The IDEF0 method in particular only models manufacturing functions and their information exchange. The control interface which is used in the methodology of Kendall et al. to identify decision processes – and thus agents – is defined by IDEF0 as "the conditions required for the function to produce correct outputs" (IDEF 1993, p. 10). This abstract definition may include decisions, but also information or data (cf. the examples in (IDEF 1993)). The control interface is consequently not a suitable criterion for identifying decision points in the manufacturing process. Furthermore, the IDEF0 method is only suitable for the description of manufacturing processes, it is not a design method providing criteria for structuring the design (cf. subsection 3.4.1). An IDEF0 model therefore cannot be created for new control systems, unless the result of the design process is anticipated which leads to a cyclic dependency in the design process: the agents must be known in order to identify them.

The use of object-oriented models for the identification of agents is also inadequate (see subsection 3.3.5). Burmeister, for instance, uses the "activeness" of an object to identify agents. This criterion, though, is not a distinguishing feature of agents in control design. A production system consists of many components, nearly all of which can actively change their state. A conveyor belt, for example, can turn its rollers in order to transport pallets. For the design of control systems, though, the only relevant components are those whose decisions have a significant impact on the production process. Production agents are therefore more than simply active components. Even though Kendall et al. and Colombo et al. do have a decision-oriented model of an agent, they also provide no criteria other than activeness to identify agents in object-oriented models. Likewise, Van Brussel et al. rely on object-oriented criteria to identify the initial set of agents. The definition of an agent type framework by Van Brussel et al. only partly remedies this deficit. The PROSA framework defines only abstract agent types such as product, resource, or order agents, and provides no criteria on how these can be specialised to capture the decision processes of a specific manufacturing system.

Ritter et al. take a different approach to agent identification by aggregating physical components into agents. But, as already mentioned, their method does not include any criteria specifying how to perform the aggregation. In particular, because the method does not define physical and logical dependencies and does not specify for how long to apply the aggregation, the aggregation process may easily end up with only one agent since in a manufacturing process nearly everything is somehow dependent on everything else. Süssmann et al. (2002) therefore propose to create "proxy" agents for

aggregating physically dependent agents. This creates a hierarchy of agents in which the dependencies decrease towards the top of the hierarchy. But Süssmann et al. still do not provide criteria for when and how to perform the aggregation. Furthermore, taking an inventory list as a starting point for the aggregation without the possibility of discarding objects will lead to a control system in which every physical object will become an agent, irrespective of the relevance of this object to the control process. Therefore, the methodology of Ritter et al., as well as the other methodologies, does not provide sufficient support for the identification of agents.

The identification and design of interactions is even less developed in all these methodologies. Kendall et al. propose to identify and design interactions on the basis of the information exchange in IDEF0 diagrams as well as patterns in use cases, but do not elaborate either approach. Van Brussel et al., Ritter et al, and Colombo et al. do not address this issue in detail because their work focuses on the identification of agents or on the formal specification of agent-based systems. Only Burmeister provides a co-operation model which is based on a representation framework for co-operation protocols. But even Burmeister provides only very abstract criteria for identifying the need of interaction (e.g., to share resources, to synchronise actions, or to co-ordinate behaviour), and does not explain how the required interaction process can be designed.

3.5.3 Role-based methodologies

The (general) limitations of methodologies that are based on concepts from other fields, such as object-orientation, to model agent-based systems led to the development of methodologies that are purely (or mostly) based on agent-oriented concepts. The dominant agent-oriented concept used in such methodologies is that of a role. Kendall (1998, 2001) defines a role as an abstraction of agent behaviour modelled in terms of responsibilities, possible collaborators, required expertise, and co-operation mechanisms used. The most important advantage of the concept of a role is that it can be freely assigned and reassigned to agents, as long as the agents fulfil the role's requirements. Since the concept of a role has been adopted in many methodologies, this subsection only reviews methodologies in which roles are central to the design process. Other purely agent-oriented design methodologies will be reviewed in later subsections.

3.5.3.1 MASB Methodology

Moulin and Brassard (1996) propose a methodology which identifies roles of (human or artificial) agents in an application by analysing scenarios in which the agents interact with a potential user (see also (Moulin 1994)). In the analysis phase, the user (textually) describes a typical scenario emphasising "the roles played by humans and artificial agents, the typical information exchange, events that occur in the course of the scenario

and the actions performed by agents" (Moulin 1996, p. 219). A role is then characterised by a behaviour diagram specifying the activities, the knowledge, and the interactions involved in that particular role. The analysis phase is finished by modelling the local data, static and dynamic descriptions of the world, and system user interactions.

In the design phase, agents are identified and roles are assigned to agents. It is argued that the designer knows how to perform this step because of his analysis of scenarios and roles in the previous phase (Moulin 1996, p. 226). Once the agents are identified, the designer specifies knowledge structures characterising the agents, i.e., beliefs, decisions, actions, and reasoning involved in playing a role. Finally, conversations between agents are specified on the basis of plans an agent can execute.

3.5.3.2 The methodology of Kinny and Georgeff

Kinny and Georgeff (1997) propose a methodology on the basis of roles and responsibilities in a multi-agent system (see also (Kinny 1996)). The analysis of responsibilities leads to the identification of services provided by the agents. The model proposed is divided into (i) an external viewpoint which models the purpose, the responsibilities, the services, and interactions of an agent; and (ii) an internal viewpoint which is based on a specific agent architecture, modelling beliefs, goals, and plans.

The external viewpoint consists of two models which are independent of the architecture used for the internal viewpoint (Kinny 1997, p. 3):

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

Agent classes can be derived from other agent classes and can thus inherit properties from these classes (even though this requires the modelling of beliefs, goals, and plans, and thus the internal viewpoint), and agent classes can be aggregated into new agent classes.

The external models are derived in four major analysis steps (Kinny 1997, p. 17):

1. Identify the roles of the application domain. Elaborate an agent class hierarchy.
2. For each role, identify its associated responsibilities, and the services provided to fulfil those responsibilities. Decompose agent classes to the service level.
3. For each service, identify the interactions associated with the provision of the service, the speech acts required for those interactions, and their information

content. Identify events and conditions to be noticed, actions to be performed, and other information requirements. Determine the control relationships between agents.

4. Refine the agent hierarchy and the control relationships.

The roles initially identified guide the identification of the agents. The concrete agents, however, are not defined until the roles have been decomposed to the level of services. At that level, the agents may be regrouped in order to optimise the system structure.

After or in parallel to the development of the external viewpoint, each agent is modelled in terms of the goals to be achieved, the beliefs it may adopt, and the plans that achieve the goals. This model is based on a corresponding agent architecture (Rao 1992).

3.5.3.3 The Gaia methodology

Wooldridge, Jennings, and Kinny (1999b, 2000) also propose a methodology for the analysis and design of agent systems based on the key abstractions of roles and responsibilities. Like the Kinny and Georgeff methodology, it deals with both the societal and the agent level of a design, but in contrast to the previous methodology it does not assume any particular agent architecture.

Wooldridge et al. model an agent-based system in terms of agent roles. Each role is characterised by three attributes: responsibilities, permissions, and protocols. The responsibilities of a role define what the agent is supposed to do (i.e., its functionality). The permissions associated with a role define the resources an agent may spend when carrying out a role. And finally, the protocols define how agents interact in order to fulfil their roles.

The role models are created in three analysis steps which can be iterated to improve the models (Wooldridge 1999b, p. 72).

1. Identify the roles in the system.
2. For each role, identify and document the associated protocols.
3. Using the protocol model as a basis, elaborate the role models.

After the analysis phase, the models are transformed into a set of design models: an agent model, a services model, and an acquaintances model (see figure 3.10). The agent model defines agents by associating roles and creating an agent type tree. The services model identifies the services associated with each role and specifies properties of these services, such as pre-conditions and the result of a service. Finally, the acquaintance model defines which agent can directly communicate with which other agent.

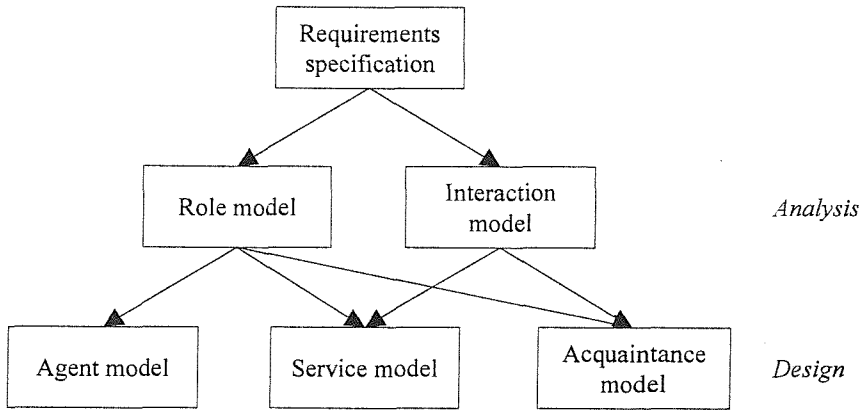

Figure 3.10: The Gaia models.

These design models are developed in three steps (Wooldridge 1999b, p. 74).

1. Create an agent model.
   a) Aggregate roles into agent types and refine these to form an agent type hierarchy.
   b) Document the instances of each agent type using instance annotations.
2. Develop a services model, by examining protocols as well as safety and liveness properties of roles.
3. Develop an acquaintance model from the interaction and the agent model.

3.5.3.4 Extensions of the Gaia methodology

The Gaia methodology has been extended in several directions, although without consolidating the different improvements into a revised Gaia methodology. An obvious extension is to introduce social roles. Gaia only models individual roles assigned to a single agent. Omicini (2001) therefore proposes in his SODA (*Societies in Open and Distributed Agent spaces*) methodology to distinguish between individual and social roles. An individual task is assigned to an individual role and thus to a single agent, while a social task is assigned to a group of agents. Each agent in the group then plays a social role, such that the interactions of these social roles achieve the social task. In order to address the same problem, Juan, Pearce, and Sterling (2002) propose in their ROADMAP methodology to identify a hierarchy of roles. Roles at the bottom of the hierarchy are individual roles, as in Gaia, while all other roles are social roles that aggregate several roles into a composite role. As a further extension, ROADMAP also allows agents to change their roles at run-time, although it does not discuss how role adjustment can be computed or managed in practice.

Zambonelli, Jennings, and Wooldridge (2001) add three organisational abstractions to

the Gaia methodology: organisational rules, organisational structures, and organisational patterns. Organisational rules are identified in the analysis phase of Gaia in order to capture the social rules "that the organisation should respect and/or enforce in its global behaviour". Zambonelli et al. express these rules as formulae in temporal logic which constrain the behaviour and the assignment of roles in the agent system. The first step of the design phase is then to choose an organisational structure which defines the organisational topology and the control regime of the agent system and which is able to enforce the organisational rules identified in the analysis phase. To increase efficiency, the designer may re-use here any of the organisational patterns already existing. For all subsequent design steps, the adopted organisational structure provides an explicit framework into which, in particular, roles and interactions must fit.

Finally, Omicini (2001) as well as Zambonelli et al. (2001b) propose to use co-ordination spaces (called agent spaces or co-ordination models by the authors) for the interactions of the agents in Gaia. Co-ordination spaces provide a medium for the agent interactions which is managed by a set of mechanisms out of control of the agents and which can thus enforce co-ordination or social laws even if the agents are self-interested (and may therefore not be willing to comply with the social rules). Furthermore, a co-ordination space can increase the openness of an agent system if the space provides standardised interfaces and captures some of the social reasoning.

3.5.3.5 The MaSE methodology

DeLoach, Wood, and Sparkman (2001) propose the elaborated *Multiagent Systems Engineering* (MaSE) methodology that derives the agent roles from the goals of the application. The first step of the analysis phase in MaSE is thus to capture the goals of the system on the basis of use cases and to represent these in a goal hierarchy. In a second step, these goals are transformed into roles by associating a new role to each goal. In general, MaSE makes it possible to map more than one goal to a role, but it does not provide any criteria for determining when this is appropriate. With every role MaSE then associates a task that details how the goal should be achieved. In the design phase of MaSE, (i) roles are assigned to agent classes, (ii) interaction protocols are defined, (iii) the internal architecture of each agent is specified, and (iv) the deployment of the agents is sketched. But here again, MaSE does not specify any criteria for performing these steps. For the assignment of roles to agents, MaSE proposes the standard software engineering concept of cohesion, but does not elaborate what cohesion means in terms of roles. Concerning the design of interactions, MaSE provides only a protocol specification model that interacts well with the specification of tasks in MaSE, but provides no method to actually design the protocols.

3.5.3.6 Evaluation

In comparison to extensions of knowledge-based or object-oriented methodologies, role-based methodologies use purely agent-oriented concepts to model software systems. The elaborated role model of the Gaia Methodology, for instance, is able to capture the flexibility of agents by specifying responsibilities and permissions of a role. Most role-based methodologies, however, require that the designer is able to directly identify the roles in an application, as it is for example straightforward in a (human) organisation, or to derive the roles from the goals in an application (see also (Zhang 2002)). This is not possible in production control. A requirements specification of a production control system only consists of a description of the physical components of a production system and the overall production goals to be achieved. The specification of the physical components, in turn, only describes a sensor and actuator interface to each component. To identify roles in a specific production application, it is therefore necessary to first derive an understanding of the production goals *and* in particular the required production process and thus of the decision making necessary to control the production process such that the production goals are achieved. Roles, however, define aspects like responsibilities, permissions, services, and interaction patterns. None of these concepts explicitly model decision making and can only be identified at later stages of a control design. In contrast, it is the main task of the control design to decide what the responsibilities and services of the local controllers will be. In control design, roles are therefore not appropriate as an analysis concept (even if they are derived from the production goals). Although, roles could be used as a design concept to increase the flexibility of the agent system once the basic control design has been created.

Because of the underlying assumption that roles can be intuitively identified in an application, none of the methodologies provide strong criteria for identifying (individual or social) roles, nor for assigning roles to agents. In production control, though, it is not clear how decisions should be combined to roles and assigned to agent types. Even though future challenges in production control require control architectures to be component-oriented (cf. subsection 2.1.4), they do not generally prescribe which physical components should be regarded as a unit and thus should be controlled by a single agent. Assigning decisions to roles or agents is therefore a major design step which must be guided by design rules in order to be sufficiently prescriptive. Here, it is also not sufficient to look at use cases. Use cases only describe – from an external viewpoint – what the (physical) system should do, but not how this can be done. Furthermore, none of the methodologies provide criteria for identifying patterns of interactions (or organisational structures). Again, it is assumed that the patterns can be easily derived from the roles of the agents and the services they provide. Without the roles or tasks naturally given in a production control problem, however, it is not possible to identify interaction patterns (or useful organisational structures and rules) from the problem description.

To summarise, role-based methodologies are neither able to bridge the conceptual gap between the domain of production control and agent-based concepts, nor sufficiently prescriptive to identify agents or interactions between agents. As already mentioned, from the point of view of control design, roles are only a concept that is useful at later stages of the design process.

3.5.4 System-oriented methodologies

Several methodologies have tried to integrate all the various aspects of agent-based systems (such as goals, tasks, roles, responsibilities, interaction, and organisation) into their analysis and design models. To integrate the different models, a number of methodologies have introduced different views of the system for each basic concept (see Massive and MESSAGE), while others have created a single model of an agent-based system somehow relating all these concepts to one another (see Elammari and Lalonde, Tropos, and Prometheus). This subsection starts its review with those methodologies that have introduced views.

3.5.4.1 The MASSIVE methodology

Lind (2001) proposes a design methodology based on multiple views representing different features of a multi-agent system. The different views cover (i) the tasks of the system; (ii) the environment of the system; (iii) the roles that agents should perform in the system; (iv) the interactions between the agents; (v) the structure of the agent system (i.e., the society view); (vi) the software architecture of the system and its agents; and finally (vii) technical aspects of implementation and deployment (i.e., the system view). The following discussion focuses on the views that are relevant to the goal of this review, namely the task, the role, and the interaction view.

The task view identifies the system task and decomposes it into several sub-tasks according to the different functions required to achieve the sub-task. This functional decomposition is integrated, and leads to a task tree.

The role view defines a role to be "a logical grouping of atomic activities according to the physical constraints of the operational environment of the target system" (Lind 2001, p. 95). To identify roles, functions are firstly grouped into a coherent cluster with high coupling and low cohesion. Then, in a second step, the roles are adapted to reflect the physical distribution of the system. How roles can be grouped or adapted to the physical structure of the system is not elaborated. The description of MASSIVE only refers to the notion of coupling and cohesion as proposed by (Collis 1998) which specifies that an "area of responsibility" should be assigned to only one agent and that the "access point" for information or services should also be assigned to only one agent. Finally, in a third step, roles are assigned to agents. But again, MASSIVE does not

prescribe how the agents are identified or roles are assigned to agents.

Concerning interactions, the interaction view only provides a very general characterisation scheme for interactions in order to support the designer in selecting the most appropriate interaction type. The scheme consists of (i) the purpose of the interaction (either co-operation or competition); (ii) the mode of interoperation (interacting either directly through message passing or indirectly through the environment); (iii) the structure of the problem domain (i.e., to what extent the problem space can be handled independently); and (iv) scalability requirements. The design of the interaction and even the actual selection of the interaction type is left to the designer. After the (manual) design, the interaction is described in the form of a protocol specifying roles in the interaction as finite state machines (including state transitions and messages to be sent).

3.5.4.2 The MESSAGE methodology

The Eurescom Project MESSAGE (*Methodology for Engineering Systems of Software Agents*) proposes an analysis methodology for the development of agent-oriented systems (Caire 2002). The analysis model to be created in MESSAGE consists of agents performing services; organisations defining power relationships between agents; roles; goals; tasks; resources; interactions; and information entities (see figure 3.11). On the basis of this analysis model MESSAGE defines different views: an organisation view, a goal/task view, an agent/role view, an interaction view, and a domain view. To create such an analysis model, MESSAGE starts by building the organisation and the goal/task view, and then the agent/role, the domain, and the interaction views. This model may then be refined by following one of several possible strategies:

(i) *organisation-centred:* focus on system structure, services, and global tasks;

(ii) *agent-centred:* identify agents first and then derive the other concepts;

(iii) *interaction-oriented:* focus on the interaction scenarios and derive goals, tasks, and required resources; and

(iv) *goal/task decomposition:* start with a functional decomposition of the system goals.

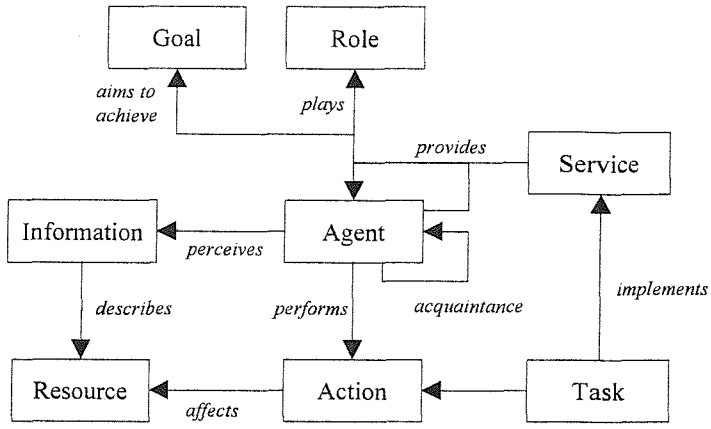

Figure 3.11: The MESSAGE model of agent-based systems.

3.5.4.3 Elammari and Lalonde – An agent-oriented methodology

Elammari and Lalonde (1999) propose a methodology that moves from high-level to implementable models through a discovery and definition phase. The methodology generates five models: (i) the high-level model identifies agents and their high-level behaviour; (ii) the internal agent model describes the internal structure and behaviour of agents; (iii) the relationship model captures dependencies and jurisdictional relationships; (iv) the conversational model describes co-ordination between agents; and finally (v) the contract model defines a structure for commitments between agents.

The discovery phase develops the high-level model with the help of use case maps (Buhr 1998). A use case map (UCM) provides a high level view of causal sequences in the system, called paths, which are comparable to scenarios. On the basis of the UCMs, the discovery phase identifies agents by extracting nouns from the problem description that play an active role throughout the application. Furthermore, the discovery phase identifies roles and responsibilities present in the paths analysed.

The definition phase creates the remaining four models. The internal agent model describes the internal structure of an agent in the form of a table including the goals, beliefs, plans, and tasks of the agent. The relationship model, in turn, describes the relationships between agents and consists of two sub-models: the dependency diagram and the jurisdictional diagram. The dependency diagram relates an agent providing a service to an agent requesting this service. These dependencies are classified into four categories: (i) a goal dependency indicates that an agent is dependent on another agent to achieve a certain goal or state; (ii) a task dependency indicates that an agent requires a specific task to be performed by another agent; (iii) a resource dependency indicates that an agent is dependent on an agent for providing it with a specific resource; and (iv) a negotiated dependency indicates that an inter-agent negotiation is required to fulfil the dependency. The jurisdictional diagram of the agent relationship model describes

authority status of agents with respect to other agents.

The conversational model identifies the necessary messages that must be exchanged between agents in order deal with dependencies. Each type of dependency as well as each jurisdictional relationship has a set of pre-defined messages associated with it that define the possible patterns of interactions. A task dependency, for instance, is associated with an *execute* message, while a negotiated dependency is resolved by *proposal*, *counter proposal*, *accept* and *reject* messages. The contract model, finally, describes obligations and authorisations between different agents. It helps to define expectations of agents about their relationships to other agents. A contract specifies the participants, the authorisations, the obligations, beliefs, and policies. The policies defines constraints on the service to be provided.

3.5.4.4 The Tropos methodology

Tropos is an agent-oriented software engineering methodology that covers the software development process from requirements engineering to detailed design (Bresciani 2001). The methodology starts with capturing the system requirements through modelling goals, tasks, resources, and actors of the system as well as the dependencies between the actors. During the design, this requirements model is elaborated by (i) refining the system actors, (ii) identifying the capabilities necessary to fulfil the system goals, and (iii) assigning these to agents. In the first design step, the system actors diagram can be extended with the help of design patterns, even though it is not explained how appropriate patterns can be identified, as for most design steps the methodology misses prescriptive methods for performing these steps.

3.5.4.5 The Prometheus methodology

The Prometheus methodology was explicitly developed to be taught to industry practitioners and undergraduate students who do not have a background in agent technology (Padgham 2002). The methodology thus pursues a similar goal as the one stated in section 3.1. Prometheus, however, is supposed to be applicable to any application domain and therefore uses modelling concepts, namely goals and functionalities, that are very similar to those of role-based approaches (see subsection 3.5.3). The authors themselves state that their concept of functionality is similar, but not identical to the concept of roles used in many other methodologies (Padgham 2002).

The Prometheus methodology starts with an analysis phase in which the basic goals and functionalities of the system to be developed are identified (Padgham 2003). To identify goals, functionalities, as well as the interactions between the functionalities, the methodology proposes to analyse typical use cases (as in object-oriented methodologies, see subsection 3.5.4.5). Once identified, the functionalities are then

grouped and assigned to agents according to the criteria of coherence and coupling (see subsection 3.2.2). In particular, Prometheus provides the following strategies for grouping functionalities (see also (Bussmann 2001c)):

- *group* if functionalities use the same data or require the same information;
- *group* if this creates fewer interaction links between the agents;
- *do not group* if the functionalities are "unrelated" or should reside on different hardware platforms;
- *do not group* if data of one functionality should not be available to another functionality for security or privacy reasons; and
- *do not group* if functionalities will change, or will be modified by different people.

To apply these general strategies, Prometheus only provides a data coupling and an acquaintance diagram. The former diagram shows which functionality requires or shares data with which other functionality. The latter diagram shows which functionality interacts with which other functionality (as specified in the analysis phase of Prometheus). The grouping process is thus only supported with respect to the data and the interaction coupling. But both aspects are not appropriate for guiding the grouping process. Data could easily be shared between two agents if they only read the information. But even if they also want to change the data, there might exist an effective mechanism to co-ordinate the write attempts. Likewise, it is not clear whether an interaction link should be avoided at all costs because this might lead to large agents. Again, it may be possible to co-ordinate many interaction links in an efficient manner. Data and interaction links alone are therefore an inappropriate means for grouping functionalities into agents.

3.5.4.6 Evaluation

With respect to control design, system-oriented methodologies (including methodologies like PASSI (Cossentino 2002) or ODAC (Gervais 2003)) suffer from the same weaknesses as the methodologies discussed in subsections 3.5.1 and 3.5.3. To model an agent-based system, these methodologies use concepts like goals, roles, functionalities, or tasks that have been shown in previous sections to be inappropriate for designing agent-based production control systems. Likewise, these methodologies provide comparatively little support for performing the analysis or design steps. A notable exception to this are the criteria for the grouping of functionalities provided in Prometheus. These criteria, however, operate on the level of data and interactions which is inappropriate for production control and probably even for agent-based systems in general, as argued in subsection 3.5.4.5. A second notable exception is the

modelling of dependencies in the methodology of Elammari and Lalonde. These dependencies are interpreted as a need for interaction and a classification of dependencies is directly mapped to interaction patterns. For each dependency type, however, the methodology only provides a simple interaction pattern. These interaction patterns may be extended by the designer, but the methodology does not elaborate how this can be achieved and, once extended, how the designer may then choose between alternative interaction patterns.

3.5.5 Interaction-oriented methodologies

In most agent-oriented design approaches, aspects like agents, tasks, or roles are designed first. Only a few researchers have proposed to start with the design or the programming of the interactions of an agent-based systems (see for instance (Huhns 2001, Singh 1996, Ciancarini 2000)). Not surprisingly, there is to date only one elaborated design methodology that follows this approach. This methodology is reviewed in this subsection.

3.5.5.1 Agent interaction analysis

Miles, Joy, and Luck (2001, 2002) propose a design methodology, called *Agent Interaction Analysis*, that derives the necessary interactions from a goal and preference analysis of the system requirements. The methodology starts with identifying the system goals in the requirements specification and elaborating these goals into a goal hierarchy. Each goal is assumed to be solved through an interaction of some agents. In a second step, the designer thus chooses an interaction mechanism for each independent goal of the goal hierarchy while using only place-holders (i.e., roles) to refer to the agents eventually participating in the interaction (see figure 3.12). During system execution, a goal is acquired by a "real" agent who then tries to find agents to take on the other roles in the interaction. In case the agent fails to find agents to engage in the interaction, it may use the goal hierarchy to decompose the goal into sub-goals which it then posts to initiate smaller interactions achieving the same overall goal. The actual agents to take on the roles at run-time are derived from the roles required in the interactions. These roles are grouped according to the preferences derived from the system requirements and general considerations, such as limited complexity or functionality of the agents, single agents for resolving conflicts between goals, or optimisation of the computational load on a single agent.

To assign an appropriate interaction mechanism to a goal, Miles et al. (2002) propose to re-use existing co-ordination mechanisms by searching for a corresponding interaction pattern that matches the preferences identified in the requirements specification. Since re-use approaches will be separately discussed in section 3.6, the discussion of their re-

use approach is also postponed to subsection 3.6.2.

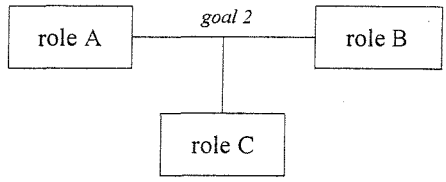

Figure 3.12: Goals implying interactions between roles in *Agent Interaction Analysis*.

3.5.5.2 Evaluation

The Agent Interaction Analysis methodology requires that the system goals can be refined into a goal hierarchy specifying exactly what is supposed to be achieved by the agent system. This goal hierarchy must be sufficiently operational so that a set of interaction mechanisms can be chosen whose combined effects achieve the overall system goals. Given the currently available interaction mechanisms (see subsection 2.2.2), this implies that the goal hierarchy for any complex system must be elaborated to a significant degree of detail. For the design of production control, this will require in particular that the goal hierarchy must be extended down to the actions of the local controllers because they will eventually achieve the system goals through their actions (without considering their potential actions it is not possible to understand the production process and thus the system goals). However, once the designer has succeeded in creating the link between the system goals and the physical actions of the local controllers, he has actually solved the control problem. For control design, an interaction analysis which starts at the system goals and moves only top-down is thus anticipating the solution. Consequently, an analysis of the control problem must move both top-down and bottom-up (see also the discussions in subsections 3.5.1 and 3.5.3).

3.5.6 Behaviour-oriented methodologies

Some methodologies have focused on developing agents which are primarily defined by their behaviour (as opposed to their cognitive capabilities – see subsection 2.2.1). Many of these methodologies, though, are very architecture-dependent, such as for instance the design approaches in (Bryson 2002) or (Bernon 2002). This subsection therefore reviews only the methodology Cassiopeia and the synthetic ecosystems approach which are both generally applicable.

3.5.6.1 The design methodology Cassiopeia

Collinot, Drogoul, and Benhamou propose Cassiopeia as a design methodology for deriving the individual behaviours of agents from the global specification of a collective task (Collinot 1996, Drogoul 1998). Cassiopeia therefore concentrates on the organisation of a multi-agent system in order to link individual and group behaviour.

Cassiopeia distinguishes three levels of behaviours: (i) elementary, (ii) relational, and (iii) organisational. Each level is designed in a distinct step (Collinot 1996, pp. 43).

1. *Identify elementary behaviours.* Elementary behaviours are those behaviours that are required for the achievement of the collective task.
2. *Specify relational behaviours.* Analyse the organisational structure with respect to the dependencies between elementary behaviours. Choose those that are most relevant and specify relational behaviours which enable agents to identify and handle these dependencies.
3. *Specify organisational behaviours.* Organisational behaviours are those that enable agents to manage the formation, durability, or dissolution of groups. Accordingly, this step identifies three types of organisational behaviours:
   - group formation behaviours
   - joining behaviours
   - group dissolution behaviours

Cassiopeia thus starts from the elementary behaviours and builds up the agent-based system by creating more complex behaviours.

3.5.6.2 The synthetic ecosystems approach

Parunak, Sauter, and Clark (1998) take a behaviour-based approach that is based on an agent model different to most of the agent models used in agent-oriented design methodologies. They do not assume the presence of explicit mechanisms for individual rationality in every agent, but rather view a multi-agent system as consisting of many very simple, interacting agents which nevertheless exhibit social coherence. Parunak et al. call these systems "synthetic ecosystems" (see (Parunak 1997) for a discussion). To design synthetic ecosystems, Parunak et al. propose four phases: (i) conceptual analysis, (ii) role playing, (iii) computer simulation, and (iv) implementation design. The following discussion concentrates on the first two phases which are concerned with the analysis and design of the agent-based system.

The conceptual analysis develops an initial design of what the system as a whole will do, which agents will be necessary, and how they will behave. The first step of the conceptual analysis therefore identifies the requirements on the system behaviour and

clarifies five aspects (Parunak 1998, p. 48):

- What is the desired overall system behaviour?
- What can be varied in an effort to achieve this behaviour?
- What must not be touched?
- What approach is currently taken to solving the problem?
- Why is a new solution being contemplated?

Given the overall system behaviour, the system is decomposed into agents. Agents are identified with the help of a linguistic case analysis. Each noun in a description of the system behaviour is a potential agent and the verbs describe the relationships between the agents. The identification is guided by a set of pre-defined agents (for manufacturing applications): unit, resource, manager, part, customer, and supplier agents. In addition, Parunak et al. provide a list of principles for validating the potential agents identified (pp. 50):

- Identify things rather than functions, i.e., identify agents that are responsible for components of the production system rather than specific functionality, like planning or scheduling.
- Identify small agents, i.e., prefer small and simple agents over large and complex ones.
- Constrain diversity and generalise with the help of inheritance, e.g., by creating a hierarchy of agents inheriting certain behaviours.

Given the agents of the system, the individual behaviours and classes of messages for interaction are defined. In this step, the methodology looks at the individual decision making of an agent, but focuses on the resulting interaction dynamics of the decisions. Again, Parunak et al. provide several principles for this step (pp. 53):

- Perform planning and execution concurrently in agents.
- Use virtual currency, like money, to guide the course of the agent system. A virtual currency makes it possible to introduce global parameters, for example prices, without introducing central components (see also subsection 2.2.2.2).
- Use local communication wherever possible.
- Make the agents learn from each other through the exchange of information.

The second phase of the methodology uses (human) role playing to validate and refine the initial design. The phase selects scenarios and scripts for the role playing, assigns agents to people, records actions of agents, and evaluates the interactions of each scenario. Since role playing certainly has limits concerning the number and the

complexity of scenarios played, the next phase further validates and refines the design with the help of a computer simulation.

3.5.6.3 Evaluation

The Cassiopeia design methodology is obviously strong in developing agents that form and dissolve groups. The methodology though does not elaborate how to identify the elementary behaviours, nor does it explain how to identify the agents in a problem domain. Cassiopeia is therefore a design *method* for the special purpose of developing coherent group behaviours in a multi-agent system that is already given.

The synthetic ecosystems approach, in contrast, covers the whole design process of an agent-based system. In particular, it identifies the necessary agents with the help of a linguistic case analysis of the problem description. However, as pointed out in subsection 3.3.5, a linguistic case analysis may identify agents which have no decisions to make, such as conveyor belts or lifts. Even the abstraction level of the agents identified is pre-determined by the requirements specification. If the description speaks of spindle, machining space, positioning, and tools to describe the processing of work pieces, a machine agent cannot be identified, even though such a level of abstraction is more appropriate in many cases. Similar to the PROSA framework (see subsection 3.5.2.3), Parunak et al. therefore try to reduce the risk of identifying inappropriate agents by providing a set of pre-defined agent types. However, it is not clear whether this pre-defined set is appropriate for any manufacturing application or which subtypes should be identified in one category. An agent-based production control system will certainly have resource agents, but the pre-defined set of agent types does not prescribe how the different resources should be assigned to agents. Finally, Parunak et al. discuss principles for validating candidate agents which are useful and relevant, but not sufficiently prescriptive to guide a designer in identifying agents. Basically, the same critique applies to the design of interactions in the synthetic ecosystems approach. Even though the methodology identifies the need for interaction by analysing dependencies of any kind between agents, the design of interactions is only guided by a set of principles which are neither comprehensive nor sufficiently prescriptive.

It should be noted though that Parunak et al. only intended to guide the designer in identifying agents (or designing interactions) by a body of "good practice" based on design experiences (Parunak 1998, p. 48). In this respect, the purpose of their methodology is already in conflict with the goal of this work (which tries to make a methodology sufficiently prescriptive so that also an inexperienced designer is able to build an agent system). The approach of Parunak et al. thus takes the risk that – if the body of good practice is not complete or does not apply – an inexperienced designer will fail to design an (appropriate) agent-based system.

3.5.7 Summarising the evaluation of agent-oriented methodologies

Agent-oriented methodologies are obviously more appropriate for designing agent-based systems than the previously developed methodologies because they explicitly model agent-oriented concepts. Existing agent-oriented methodologies, however, fail to adequately model agent-based production control systems because they focus on different concepts than those which are relevant for production control design. Existing agent-oriented methodologies model concepts like roles, tasks, responsibilities, organisations, and so on, but ignore the actual decision making of an agent or an application. In production control, however, the decision making is central to the design process (see sections 2.1 and 3.4). Without modelling the choices arising at the local controllers during the production process, it is simply not possible to achieve and, in particular, enact the production goals. An agent-oriented methodology for designing production control systems must therefore explicitly model the decision making of the agent-based system. This is done only by the methodologies of Kendall et al. and Colombo et al. (see subsection 3.5.2). Kendall et al., though, have based their methodology on object-oriented methodologies and the IDEF0 methodology which both provide insufficient support for identifying the decision making. Colombo et al. provide a model for decision making, namely Petri nets, but do not provide any criteria for mapping the Petri nets onto agents.

Furthermore, the existing agent-oriented methodologies are not sufficiently prescriptive to design agent-based production control systems. Most of the methodologies do provide criteria for agent identification, but, in all cases, these criteria lead to an inappropriate set of agents for production control. Object-oriented methods for agent identification, such as activeness or linguistic case analysis, may identify – as discussed in subsection 3.3.5 – objects which are not agents or even inappropriate agents. Likewise, knowledge-based, role-based, and system-oriented methodologies are difficult to apply to the identification of production agents because neither task hierarchies nor roles can be found in the initial description of production systems. Only the methodology of Parunak et al. provides some guidelines for the identification of agents in production control. These guidelines, though, are still too vague to be sufficiently prescriptive for an inexperienced engineer to identify control agents. However, if these guidelines are made more precise by redefining them in terms of the production models used by manufacturing control methodologies (see section 3.4) and re-organising them in a more structured design process, such as the grouping process of Prometheus (see subsection 3.5.4.5), then the agent identification step may become more appropriate as well as sufficiently prescriptive for designing production control agents. Achieving this will be the task of the methodology proposed in chapter 4.

Concerning interactions, most agent-oriented methodologies have incorporated guidelines for interaction design (e.g., CoMoMAS, MAS-CommonKADS, Kinny et al.,

Gaia methodology, MASSIVE) or protocol models (e.g., Burmeister, Gaia methodology, MaSE, MASSIVE). But only a few have elaborated how the need for interaction can be identified and no methodology has explained how interactions can be designed. The two notable exceptions are the methodology by Elammari and Lalonde and the *Agent Interaction Analysis* methodology by Miles, Joy, and Luck. Both methodologies, however, have taken a re-use-oriented approach to designing interaction protocols. To adequately evaluate their approaches, it is therefore necessary to first look at the basic concepts in software re-use and then to review to what extent these concepts have already been used in agent-oriented design approaches.

3.6 Re-use

The design methodologies presented in the previous sections have almost all focussed on designing a software system entirely from scratch. Designing new software systems, however, is a time-consuming and demanding process which must be conducted by specially trained software engineers.[13] Most of the time, though, these software engineers are not charged with completely new design problems, but rather face recurring design tasks such as determining necessary operations, choosing transportation paths, or allocating resources in manufacturing control design. Instead of solving these recurring design tasks over and over again, it would be beneficial to solve these tasks only once, and to re-use the design solutions every time a similar design problem is encountered. The advantages of re-using existing design solutions are clearly a higher productivity of the software development process (because copying existing solutions is faster than developing them from scratch); a higher reliability of the software artefacts (because re-used code has been repeatedly tested by more than one designer and application); and a higher maintainability of the software artefacts (because re-usable components have been developed without a particular application in mind) (Coulange 1998). This section therefore briefly discusses the basic concepts of software re-use (see subsection 3.6.1) and reviews those agent-oriented methodologies that have used a re-use approach to designing agent-based systems (see subsection 3.6.2).

3.6.1 General concepts in software re-use

Software re-use is the process of "using existing software artefacts during the construction of a new software system" (Krueger 1992, p. 133). During this construction process basically any kind of tangible result from previous development processes can be reused, in particular any kind of analysis, design, or code components (Coulange 1998). Consequently, there is a great diversity of different software

[13] The existence of the diverse range of design methodologies is already a proof of this.

engineering technologies that promote re-use (Krueger 1992, Coulange 1998). According to Krueger (1992), though, any kind of re-use involves the following four basic steps:

- ***Abstraction***
  The software artefact to be re-used is given an abstract representation in order to hide technical details of the artefact and thus to reduce the cognitive effort necessary to re-use the artefact. This step often involves a generalisation of the artefact in order to cover a wide range of applications.
- ***Selection***
  Given a design problem, the set of existing artefacts must be searched for a suitable component that solves the given design task.
- ***Specialisation***
  The selected re-usable component is specialised through parameters, transformations, constraints, or some other form of refinement in order to make the component fit into the specific context of the application that wants to use the component.
- ***Integration***
  Finally, the artefact to be used must somehow be integrated into the application. A typical example of an integration framework is a module interconnection language with which public functions of modules can be called in order to define the overall flow of control.

Even though all of the above steps are essential to re-use, the second step is arguably the most crucial step, as without it no re-use is possible. To be truly effective, re-use must take place across individual designers. Some designers create re-usable artefacts, while many others should use these artefacts in order to speed up their development process. This implies, though, that the designers trying to re-use artefacts, do not know these at first (because they have not created them) and must therefore search for them (e.g., in a library of re-usable components). Since re-use becomes more effective the more re-usable artefacts are available, it will be impossible for a designer to look at all existing artefacts. Consequently, to find a suitable component, there must be a search method that allows the designer to identify a suitable artefact without going through the whole library.[14] This need for effective search methods has been identified early on in software engineering research and a wide range of retrieval methods for re-usable components has been developed to date (Mili 1998). According to the state-of-the-art review in Mili et al. (1998), these methods can be divided into six types of approaches:

[14] This is supported by a survey on success factors for re-use (Morisio 2002) which has shown that setting up a library of re-usable assets is not enough to guarantee successful re-use, but that, among other aspects, non-re-use processes must also be changed in order to promote the actual re-use of existing assets.

- Information retrieval methods
- Descriptive methods
- Operational semantics methods
- Denotational semantics methods
- Topological methods
- Structural methods

Information retrieval methods view software assets, such as analysis descriptions, design specifications, or source code, as documents containing information. These documents can therefore be searched for keywords that are taken from a query for a re-usable artefact. To do so, these methods draw from a large set of techniques developed for general information storage and retrieval (Frakes 1992). Despite this large background, however, these methods suffer from two severe disadvantages which are due to the underlying information retrieval approach. First of all, information retrieval methods are dependent on a consistent use of terminology across all designers and can therefore only be used within small organisations which agree on a standard vocabulary. Secondly, because of the relatively informal description of assets (and the query), information retrieval methods are unable to assess the extent to which an asset really solves the design problem. With information retrieval, it can only be determined whether the descriptions of the asset and the query use similar terms.

Descriptive methods are a subclass of information retrieval methods which match a set of keywords against a list of keywords characterising each software asset. The list of keywords characterising an asset must be assigned to the asset by the designer or a librarian responsible for ensuring the consistent use of the keywords. An extension of the simple keyword-based approach is the faceted approach. In this approach, a multi-dimensional search space is created by specifying for each dimension (also called facet) a set of pre-defined keywords. Assets are classified according to this pre-defined search space, and a designer searching for an asset must specify the set of keywords that best characterises his design problem. The search method then retrieves all assets that match these keywords. The advantages of descriptive methods are clearly their simplicity and ease of implementation. A disadvantage is that the classification scheme is pre-defined and thus difficult to change or extend (because this would lead to inconsistencies in the long run). Another disadvantage is that the keyword match, as for all information retrieval methods, does not guarantee that the assets identified really solve the design task.

Operational semantics methods exploit the fact that (executable) software components can be characterised through their behaviour. An asset is retrieved by these methods if it creates the required output (specified by the query) on a set of sample input values.

These methods are obviously only applicable to assets that are given in executable form, as for example source or object code. Furthermore, they require that the designer is able to exactly specify the behaviour of the required asset at least for a sample input set. Finally, these methods may still retrieve assets that are not suitable because an asset may behave correctly on the sample input set, but differently otherwise.

Some methods use the *denotational semantics* of a software asset, usually a function, to match it against a query. In the simplest case, these methods only use the signature of a function for the match. This obviously leads to a very imprecise retrieval result because functions with the same signature may compute different results and functions with different signatures may still compute the same result if the input is just passed to the function in a different format. Methods that use the actual semantics of a function to perform the match, in turn, suffer from the inefficiencies of theorem proving which is necessary to show that the semantics of a function implies the required behaviour specified by the query. To date, denotational semantics methods are therefore not used in practice.

Instead of trying to find assets that match a query perfectly, *topological methods* search for assets whose features come closest to the required solution. Closeness is usually defined by a measure that determines the functional or structural distance between an asset and the query.[15] A topological method then searches the library for the asset that minimises this distance measure. Topological methods may thus provide the designer with a result even if there is no asset in the library matching the query perfectly. The main disadvantage of topological methods, in turn, is that the property of being closest to a query is a property of the library, not of a single asset. An asset is only closest to a query if there is no other asset in the library which is closer. In particular, an asset may be closest even though it is completely irrelevant to the query, just because there is no asset in the library that is relevant to the query at all.

Structural methods search for software assets that have the same structure (vs. the same function) as the solution specified by the query. These methods are particularly suited for software assets that have no clearly defined function, such as architectures or design patterns (Gamma 1995), but are rather characterised by a specific structure which can be instantiated in different ways to provide a solution. Design patterns, for instance, consist of participants or roles whose behaviour is only partially constrained by the pattern (Gamma 1995). The complete behaviour of a pattern thus depends on the actual participants eventually executing the pattern. To find a suitable design pattern, however, most existing structural methods match the query to the name or the natural language description of an asset. In the worst case, this implies that the designer must already know the assets in order to specify the correct name; in the best case, in which the query is matched against the textual description of the pattern, this approach suffers

[15] Topological methods can also be combined with other approaches, e.g., by defining a functional distance measure on the facet representation of an asset.

from the same problems as the information retrieval methods. Similarly, methods that parse the structure of the assets in order to compare it to the (structure of the) query, require the designer to specify the required structure already in the query, although the objective of the query is actually to find a suitable structure.

Mili et al. (1998) conclude their state-of-the-art survey on retrieval methods for software re-use libraries with the result that none of the existing methods satisfies all reasonable requirements for asset retrieval. With the current state-of-the-art, retrieval must therefore be tailored to the assets to be re-used and potentially to the applications to use the assets.

3.6.2 Agent-oriented re-use

Despite the large number of techniques developed, re-use has received very little attention in agent-oriented software engineering research to date, and most research conducted has focussed on transforming existing design knowledge into patterns. Following the approach in (Gamma 1995), for instance, Kendall and Malkoun (1997), Kendall et al. (1998) as well as Aridor and Lange (1998) propose a set of typical agent-oriented design patterns, such as the *Broker* pattern, the *Facilitator* pattern, or the *Meeting* pattern for mobile agents. These patterns, though, are all described in plain text that is only structured into headings such as *intent*, *motivation*, *applicability*, *participants*, *collaborations*, and *consequences*. Since these headings are even the same headings which are used for object-oriented patterns (Gamma 1995), Lind (2003) has proposed a pattern description scheme that is tailored towards the description of agent-oriented patterns. This pattern scheme consists of headings such as *problem*, *forces*, *entities*, *dynamics*, *dependencies*, and *consequences*. For specific types of patterns, the scheme can be extended by more specialised headings. An agent architecture, for instance, can also be characterised by *control flow*, *resource limitations*, *knowledge handling*, *reasoning capabilities*, and so on. But apart from the introduction of agent-oriented headings, the patterns in (Lind 2003) are still described in plain natural language, which has been shown in the previous subsection to provide comparatively little support for re-use because it requires the designer to know the design patterns in advance in order to select them. Specifying patterns is thus only a small step towards systematic re-use (covering just the abstraction of re-usable assets).

Brazier, Jonker, and Treur (2002) have developed a compositional framework for multi-agent design that promotes re-use. In their framework, called DESIRE, agents are created by incrementally combining previously designed components into larger ones. In the terminology of Krueger (1992), DESIRE is thus an integration framework that provides a mechanism for incorporating existing components into an application. However, how the components to be integrated can be identified is not elaborated in the corresponding DESIRE design method. Brazier et al. provide only examples of generic

components.

Bartolini, Preist, and Jennings (2003) looked at existing interaction techniques and developed a general framework for automated negotiation. They observed that negotiation processes can take many different forms and that the negotiation process is only fully specified if the negotiation protocol *and* the negotiation rules are fixed. Bartolini et al. therefore defined an abstract negotiation process, a generic negotiation protocol, and a taxonomy of negotiation rules which specify the admissible behaviour of negotiators engaging in the negotiation protocol. With the help of this taxonomy of negotiation rules, negotiation processes can now be classified with respect to their behaviour and thus become accessible to systematic re-use. The taxonomy alone, though, only provides an abstraction of the negotiation process and does not address the issue of selecting the right negotiation process in a particular design situation (or deciding whether a negotiation process is helpful at all). Furthermore, the taxonomy only concerns negotiation techniques and not interaction or even agent-oriented techniques in general.

General methods for selecting interaction techniques during the design phase were proposed by Elammari and Lalonde (see subsection 3.5.4.3) and by Miles, Joy, and Luck (2002). As described in subsection 3.5.4.3, the methodology of Elammari and Lalonde identifies different types of dependencies between agents and proposes a pre-defined interaction pattern for each type of dependency. Furthermore, the methodology allows to add new interaction patterns to a dependency type, but does not explain how the designer may choose the right interaction pattern in a particular design situation. Re-use is thus possible for only a few interaction patterns. In the *Agent Interaction Analysis* methodology of Miles et al. (2002), a co-ordination mechanism is selected for each goal that is identified during the analysis phase (see subsection 3.5.5). To select an appropriate co-ordination mechanism, the designer must browse through a list of co-ordination mechanisms that are defined in a standard pattern language (such as the ones described above) and choose the co-ordination mechanism that is most suitable for matching the goal's preferences. To ensure that the chosen mechanism really does satisfy the preferences, the methodology provides an assurance analysis method, in which the designer must acknowledge for four phases of an interaction (information acquisition, updating information, information analysis, and enactment) that the mechanism satisfies all preferences. This assurance analysis method, however, requires that the designer understands the co-ordination mechanism under consideration in detail and can therefore only be applied to a small set of mechanisms (see also subsection 3.6.1). Due to the limitations of the pattern-oriented approach discussed above, the designer is likewise not able to browse through a large set of co-ordination patterns. The approach proposed by the *Agent Interaction Analysis* methodology is therefore restricted to a small set of co-ordination patterns and is consequently not able to promote re-use for a large set of mechanisms across many designers.

In general, it can be concluded that agent-oriented software engineering research has not yet integrated satisfactorily existing re-use mechanisms into the agent-oriented design process. There have been several proposals for representing agent-oriented design solutions as patterns or frameworks, but only a few attempts have been made to select the right pattern given a design problem. Moreover, the selection mechanisms proposed fail to promote a wide spread use of the design patterns because the designer must read through each pattern. The selection mechanism proposed must therefore be significantly extended in order to reduce the efforts required by the designer willing to re-use a pattern. At this point, it could be argued that there is no need for agent-specific re-use techniques and that the standard techniques discussed in the previous subsection are also sufficient to promote re-use for agent-based systems. But even if this is true, it is still necessary to show how existing techniques will be effectively applied to agent-based systems since the above discussion has shown that this is not obvious.

3.7 Conclusions

This chapter has reviewed existing methodologies with respect to their suitability for wide spread use in the design of agent-based production control systems. Such wide spread use requires that the design methodology can be applied by an engineer with a standard qualification (in control engineering) and only minimal training in agent technology. In terms of the definition of a methodology, this translates into two main requirements (see section 3.1.1):

- **Model appropriateness.** The models of a methodology should be clearly related to the relevant concepts of the problem domain and should allow a straight and comprehensible transition from the domain to the agent-oriented concepts.
- **Method prescriptiveness.** The methods of the methodology should include sufficient criteria to perform each agent-related design step.

Against this background, the review has shown that no existing methodology sufficiently fulfils the above requirements such that it could be applied by an engineer without experience in agent design.

First of all, all methodologies employ a computational model that is either unable to model agent-based systems (see structured, data-centred and object-oriented methodologies) or does not allow a straight transition from domain to agent-oriented concepts (see existing manufacturing and agent-oriented methodologies). Structured and data-centred methodologies are obviously unsuitable for modelling autonomous and co-operative distributed computing. Object-oriented approaches equally fail to model agent-based systems because agents are more than objects (Jennings 2000,

Jennings 2001). In particular, objects are unable to model goal-based decision making and flexible interaction – two key aspects of agent-based production control. Existing manufacturing methodologies, in turn, are able to model the manufacturing control process, but do not support agent-oriented concepts, such as distributed autonomous and co-operative decision making.

Agent-oriented methodologies do provide suitable models for agent-based systems. But by and large these methodologies still fail to provide a straight transition from the domain of production control to agent-oriented concepts. This is partly due to the fact that most agent-oriented methodologies have been designed with a different type of application in mind. Role-based methodologies, for instance, have been developed for domains in which roles are given or can be easily identified – which is not true for production control. Other methodologies equally fail because they are based on concepts, such as roles, tasks, or services, that are unable to model the relevant aspects of production control.

Secondly, nearly all methodologies provide criteria for agent identification which lead to an inappropriate set of production agents. Object-oriented methods for agent identification, such as activeness or linguistic case analysis, may identify – as discussed in subsection 3.3.5 – objects which are not agents or even – in extreme cases – the wrong agents. Likewise, knowledge-based, role-based, and system-oriented methodologies are difficult to apply to the identification of production agents because neither task hierarchies nor roles can be found in the initial description of production systems. Some methodologies do identify manufacturing agents, but do so with strategies which are either intuitive or based on models that are unable to capture the decision making necessary in production control.

Concerning interactions, some agent-oriented methodologies have incorporated guidelines for interaction design (e.g., Kinny et al., Gaia methodology) or protocol models (e.g., Burmeister, Gaia methodology). But only a few have elaborated how the need for interaction can be identified, and only one methodology has explained how interactions can be designed. The re-use-oriented approach of Elammari and Lalonde as well as Miles et al., though, is based on the selection of suitable design patterns which with the given current state-of-the-art in re-use is still intuitive and in particular not scalable.

To conclude, there is currently no methodology for the design of an agent-based production control system which satisfies the requirements stated in subsection 3.1.1. In particular, existing methodologies fail to fulfil the requirements in three respects:

1. No existing methodology is able to adequately model the agent-oriented decision making in control applications, and thus to provide a straight and comprehensible transition from domain to agent-oriented concepts (see requirement III in subsection 3.1.1).

2. No existing methodology provides sufficient criteria for identifying suitable production control agents, and is thus sufficiently prescriptive with respect to the identification of production agents (see requirement IV in subsection 3.1.1).
3. Very few methodologies have looked at designing interactions and in particular none have provided a method for re-using interaction protocols on a large scale. No methodology is thus sufficiently prescriptive with respect to interaction design (see requirement IV in subsection 3.1.1).

It is therefore necessary to develop a comprehensive design methodology that captures goal-based decision-making in its models and provides a new list of criteria for agent identification and interaction design. This methodology can be built on several achievements and insights of existing methodologies. First of all, the methodology should clearly separate analysis and design activities (see subsection 3.2.3). Secondly, it should start by identifying the entities of the domain and transforming these into analysis and design objects (as is done by object-oriented methodologies (see subsection 3.3.5)). This will create a link between domain and solution concepts, and thus improve the appropriateness of the methodology. Thirdly, the methodology should clearly separate the design of agents from the design of the interactions (a second achievement of object-oriented methodologies (see subsection 3.3.5, also (Ciancarini 2000))). Fourthly, the methodology should make use of several aspects already proposed in existing agent-oriented methodologies. With respect to agent identification, the methodology should redefine existing criteria for identifying agents, such as those of Parunak et al. (see subsection 3.5.6.2), in terms of a decision-oriented model of manufacturing control and re-organise them in a more structured design process, such as the grouping process of Prometheus (see subsection 3.5.4.5). And with respect to interaction design, the methodology should build upon the approach of Elammari and Lalonde (see subsection 3.5.4.3) to identify dependencies between agents and associate protocols to these dependencies, as well as extend the re-use-oriented approach of Miles et al. (see subsection 3.6.2) in order to make it applicable to a wide range of interaction protocols. To incorporate the above achievements and insights into a new design methodology for agent-based production control systems that is both appropriate and sufficiently prescriptive with respect to the identification of the control agents and the design of the interactions between these agents is the goal of the next chapter.

Chapter 4

The DACS Design Methodology for Production Control

The review of existing design methodologies in the previous chapter has shown that conventional design methodologies, such as for instance object-oriented or manufacturing control methodologies, are insufficient for the design of agent-based production control systems. This is because the development of agent-based systems places different requirements on the software models and methods to be used. The review also demonstrated that the agent-oriented design methodologies developed to date lack either appropriate models for modelling control decisions or are not sufficiently prescriptive to enable a control engineer without prior experience in agent technology to design an agent-based production control system. Chapter 2, on the other hand, has shown that agent technology provides indispensable features for future control systems and that a wide spread industrial exploitation of this technology will require a design methodology for agent-based production control systems that can be applied by industrial engineers. To fill this gap between the state-of-the-art and industrial needs, this chapter presents the *DACS methodology for designing agent-based production control systems* that is both appropriate for control design and sufficiently prescriptive for a control engineer with only minimal training and no prior experience in agent technology.

The DACS methodology will follow the commonly used approach to first identify the agents of the application and then to design the interactions between them (see section 3.3). In so doing, the methodology will also clearly distinguish between an analysis and a design phase (see section 3.2). Second, the methodology will draw from the models of decision theory and manufacturing control theory to create a model of control decisions that can be used to identify the control agents (see section 3.4). And, finally, the methodology will build upon the well-proven concepts of coupling and cohesion (see subsection 3.2.2) and specific agent-oriented concepts, such as dependencies (see subsection 3.5.4.3) and "good design practice" (see subsection 3.5.6.2), in order to

create a set of prescriptive design rules for shaping the actual agents or interactions of an application. The methodology, in turn, will go significantly beyond the state-of-the-art in three respects:

- It will incorporate the above aspects into a new agent identification method which is both appropriate for production control because it is based on a decision-oriented model of the domain and sufficiently prescriptive because it employs elaborated design rules to group decisions into agents.
- It will provide a method for re-using existing interaction protocols that identifies and adapts the most appropriate interaction protocols for each interaction situation that arises during the execution of the agents.
- It will integrate these methods into a guided set of design steps which can be easily followed by the designer in order to go from the specification of a production process to the design of an agent-based control system that is able to operate the production process.

The DACS methodology is thus complete in the sense that it provides all tools necessary to address the agent-oriented aspects in the design of agent-based production control systems. To achieve this, the methodology will consist of three major steps:

1. *Analysis of decision making* – The control decisions that are necessary to operate the production process are identified and analysed. Furthermore, dependencies between the decisions that may require interaction during the execution are also identified and incorporated into a decision model.
2. *Identification of agents* – The system architecture of the agent-based control system is designed. In particular, this step identifies the agents of the control system, the decisions they are responsible for, and the interaction requirements of the agents. That is, when do agents need to interact and what does the interaction need to achieve.
3. *Selection of interaction protocols* – Each interaction requirement between different agents is classified and matched against a library of existing (agent-oriented) interaction protocols. The most suitable interaction protocol is selected and adapted to the specific needs of the particular interaction situation.

The result of the three steps of the methodology is a design that specifies the agents of the control system and for each agent how it interacts with the production process and how, if necessary, it interacts with other control agents. That is, the design is sufficiently modular to allow the independent design and implementation of each control agent. An overview of the DACS methodology is shown in figure 4.1.

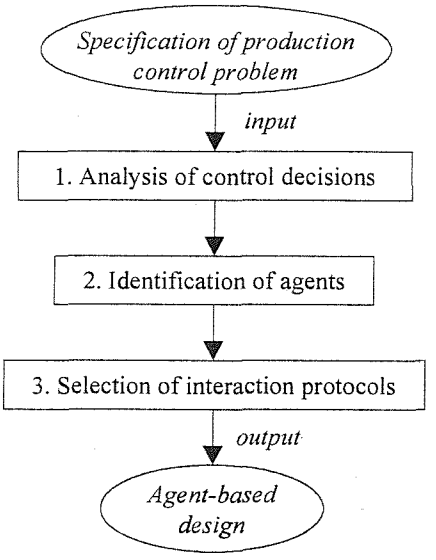

Figure 4.1: Major steps of the DACS methodology.

The remainder of this chapter will present input and output of the methodology as well as each step of the methodology in detail. Section 4.1 will outline the input to the design methodology, i.e., the *specification of a production process* and its associated control problem. Section 4.2 will then describe the first step of the methodology: the *analysis of the control decisions* necessary to operate the production process. The *identification of the agents* capable of co-ordinating these control decisions is discussed in section 4.3. Finally, section 4.4 presents the *selection of appropriate interaction protocols* for the co-ordination of the agents. A summary of the design steps is given in section 4.5. To illustrate the design methodology, each section discusses a running example which is a small, but realistic control problem. Chapter 5 will then evaluate the methodology with respect to the requirements specified in subsection 3.1.1. A complete industrial case study is given in appendix B.

4.1 Specification of the production control problem

The input to the design methodology is a requirements specification of the production process and its associated control problem. It must consist of three parts (see also (Groover 1987, Hitomi 1994)):

(i) a specification of the (physical) *production process* to be controlled, including the available control interfaces

(ii) a specification of the *production operation conditions*

(iii) a specification of the *production goals and requirements*

The first specification describes the (mechanical) components of the production system and their arrangement on the factory floor (Castillo 2002, Fanti 1996, Caselli 1992). The specification furthermore defines for each component its physical behaviour and its control interface (if existing). The physical behaviour can be defined with the help of various modelling approaches, such as for instance discrete event systems (Zeigler 1984, Fanti 1996), Petri nets (Murata 1989, Zhou 1992), or IDEF0 (Bravoco 1985). The control interface specifies which kind of sensory information about the status of the production component is provided and what kind of (physical) actions can be initiated at the component. Examples of mechanical components in discrete manufacturing are (Groover 1987, pp. 20):

- resources (such as machines or robots for processing; conveyor belts, lifts, switches for transporting of material; buffers for storing material);
- material (such as cylinder heads, car bodies, or assembled engines); and
- auxiliary material (such as pallets or processing tools).

Example. Throughout this chapter, the following small production system will be used to illustrate the design methodology. The small production system consists of a loading station, an unloading station, three flexible machining stations (M_1 to M_3), several transportation switches (S_1 to S_{12}), and several conveyor belts. The factory layout of this production system is shown in figure 4.2. Equipped with the right machine types and processing tools, such a production system would be able, for instance, to produce cylinder blocks for car engines.

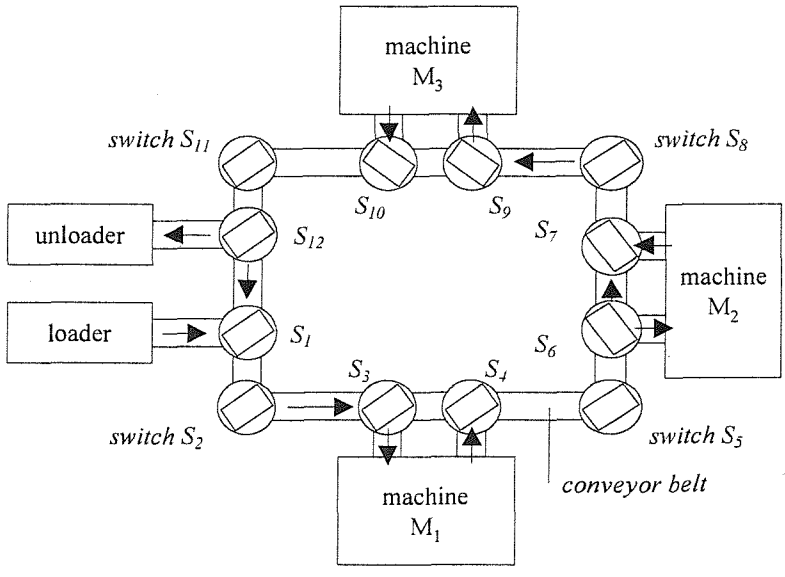

Figure 4.2: Example production system.

The production process of such a production system is as follows. At the loading station, workpieces are put onto a conveyor belt and fed into the production system. Each workpiece then circles around the production system until a transportation switch moves the workpiece to a machine. The machine takes the workpiece off the inbound conveyor belt and processes it. After processing, the workpiece is put back onto the outbound conveyor belt and the switch feeds the workpiece back into the transportation circle. The processing of the workpiece is repeated until the workpiece is finished. The workpiece is then moved to the unloading station, where it is manually taken off the conveyor belt.

The ***operation conditions*** specify the input and the (required) output of the production process as well as the spectrum of possible changes and disturbances that may occur during the operation of the production system (Fanti 1996, pp. 193) (see figure 4.3). The input to the production process is specified in terms of the (allowed) order mix that may be fed into the production system and the availability of the raw material that is necessary to execute the orders. The output of the production process is then simply the set of products specified by the order stream.

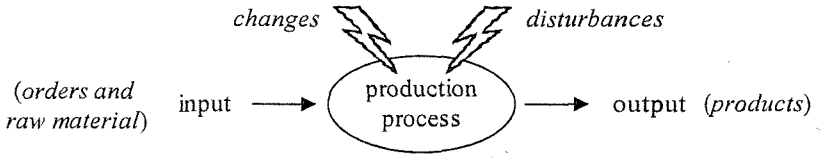

Figure 4.3: Operation conditions.

Disturbances are events during the production process that temporarily reduce the performance of a resource or a control unit. Most disturbances are out of human control and may occur at any time. Changes, on the other hand, are induced by the production management or the environment of the production process and may affect the production system or the input to the production system. The specification cannot identify all changes and disturbances that may occur, but it should at least identify those which are likely to occur and which should be dealt with automatically by the control system.

Example. The input stream for the example production system is an arbitrary mix of different products to be produced (in small volumes). Changes to the production process are not expected and the only possible disturbances are sudden breakdowns of machines.

The specification of the ***production goals*** determines the performance criteria for the

production system, i.e., how the production process should be optimised. Examples of production goals are high productivity or throughput; minimal lead times; or maximal capacity usage (Hitomi 1994, Sarin 1994). *Production requirements* place further constraints on the system functionality that are important for the operation of the system. Examples are flexibility with respect to resource or order changes; robustness with respect to mechanical or control failures; volume scalability; reconfigurability of components; quality assurance; maintainability; the possibility of human intervention; and the existence of fallback strategies in case of severe failures (Sarin 1994).

Example. The goal of the above production system is simply to maximise the throughput. Furthermore, the production process should be robust against machine failures and easily scalable through the addition of more resources.

A ***solution to the production control problem*** is a control system

(i) which is able to control the production system such that the system creates the specified products under the specified operation conditions, and

(ii) which optimises system performance with respect to the production goals and requirements.

In the following sections, it is assumed that a production control problem is given, i.e., the specific production process to be controlled, the operation conditions under which it is supposed to be controlled, and the production goals and requirements are known.

4.2 Analysis of control decisions

The very first step of the DACS methodology is to develop an initial understanding of the production control problem. Production control is the process of choosing, initiating, and monitoring actions in a production system in order to achieve specific production goals (see subsection 2.1.2). For an initial understanding of the control problem, the designer therefore needs to collect and characterise those situations arising during the production process in which control actions are necessary. In doing so, however, the designer should not anticipate any design decisions which might exclude alternative solutions from the design process. For instance, the designer should not assume that scheduling actions will be necessary even though it may be possible to control the process without assigning start dates to specific actions.[16] To avoid implicit assumptions about the control solution, the designer should therefore focus at the beginning of the design phase on identifying those decisions any solution to the given control problem must make. Since a controller can only interact with the physical production system through its control interface, any control system must eventually

[16] See subsection 2.3.3 for an example of a control system where scheduling is not required.

decide which physical actions to initiate at the components of the production system – no matter how these actions were actually determined. The methodology therefore starts by looking at the physical actions that are necessary to run the production process and identifies all those situations in which the controller has alternative actions to choose from (see step 1.1 in figure 4.4). These situations represent the decision tasks which the controller has to solve in order to achieve the production goals. These decision tasks – which will be called *effectoric decision tasks* – are characterised without specifying how these decisions will be made or what the (exact) result of each decision task should be.

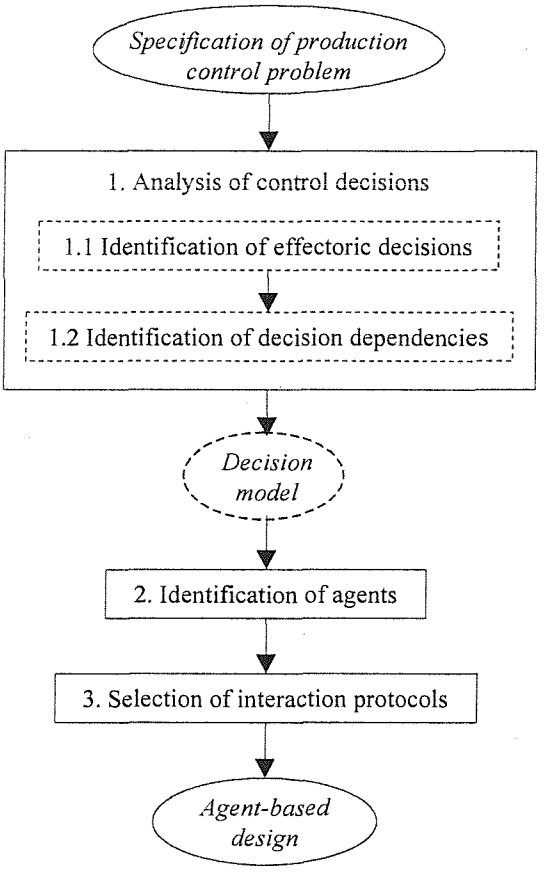

Figure 4.4: Steps for the analysis of control decisions.

Looking at the individual decision tasks, however, is not enough to fully understand the control problem to be solved (see also subsection 3.5.4.3). Physical actions are not executed in isolation, but in a (physical) system in which their effects add up and create a specific overall process behaviour. In a manufacturing system, for instance, the workload and thus the degree of capacity utilisation at each machine is determined by the individual assignments of workpieces to machines. To achieve an optimal distribution of the workload, a single workpiece may not simply be assigned to any appropriate machine, but this decision must be co-ordinated with the machine

assignments of other workpieces in the production system. In order to identify these kinds of dependencies between decision tasks, it is necessary to analyse the effects the choices of one decision have on the utility of other decisions. The second analysis step of the methodology is consequently to identify all relevant dependencies between decision tasks by analysing the effects of decision tasks on the physical production process, and in particular on other decision tasks (see step 1.2 in figure 4.4).

The remainder of this section is organised as follows. Subsection 4.2.1 describes the identification of the effectoric decision tasks, and subsection 4.2.2 discusses the identification of decision dependencies between the effectoric decision tasks. Subsection 4.2.3 then summarises the results of the two analysis steps. The decision model resulting from these two analysis steps will then serve as a basis for subsequent design steps (see figure 4.4).

4.2.1 Identification of effectoric decisions

The objective of the first analysis step is to collect and characterise all effectoric decision tasks that are necessary to run the production process. But before this can be done, it is necessary to first clarify what (effectoric) decisions are.

4.2.1.1 Modelling control decisions

According to classical decision theory, a *decision* is the choice of the "best" alternative out of a set of possible alternatives (Laux 1998, Coyle 1972). These alternatives must be different and exclusive in order to allow the application of classical decision theory. They must be different in the sense that at least two alternatives must have different effects (otherwise there is no decision problem). And they must be exclusive in the sense that the alternatives cannot be combined, i.e., only one can be chosen. To make a decision, the alternatives must be (totally) ordered according to their utility and the alternative with the highest utility is chosen (i.e., the utility function must identify at least one maximal element that is better than all other alternatives).

Classical decision theory can be applied to most production control problems in discrete manufacturing because for most production systems each decision task consists of a set of different and exclusive alternatives (for instance, move to machine A or B). The standard description of decision tasks in classical decision theory, though, must be extended because control decisions are embedded into a physical world and must be resolved continuously, not only once (see also (Dean 1991)). A ***control decision*** should therefore be characterised by a trigger, a decision space, a decision rule, and a control interface (see figure 4.5). The *trigger* specifies when the control decision is required, i.e., it specifies the situations occurring during the production process that require the decision. The *decision space* is defined by the set of possible actions the component can

perform in these situations. The *decision rule* is a preference function on the decision space that identifies the "best" alternative. Even though a total preference function is (theoretically) sufficient to resolve a decision task, the preference function is often complemented by a set of constraints forbidding certain actions in specific situations. And finally, the control interface that can be used to enact the decision is specified.

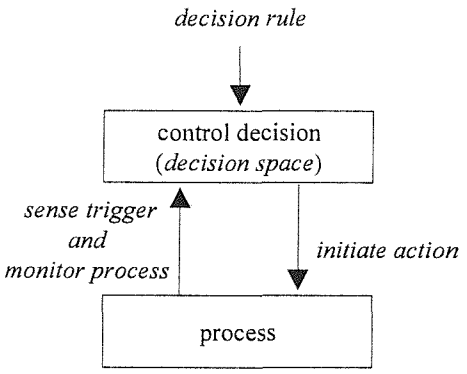

Figure 4.5: Abstract model of control decisions.

Given this model of control decisions, it is now possible to define what an effectoric (control) decision is. An ***effectoric decision task*** is a decision task whose decision space consists of at least one physical action.[17] As argued at the beginning of this section, an effectoric decision task is thus the only way for the controller to influence the course of the production process since only physical actions actually change the state of the production process. Consequently, if a designer is able to find a small set of effectoric decision tasks that covers all possible physical actions which are required to operate the production process, then this set of effectoric decision tasks is a complete representation of the control problem. That is, this small set of effectoric decision tasks includes all control decisions a control system can and must make in order to solve the production control problem. The methodology thus starts with collecting such a complete set of effectoric decision tasks.

4.2.1.2 Identifying effectoric decisions

An effectoric decision task was defined as a decision task that includes at least one physical action. To identify a set of effectoric decision tasks covering all possible decision situations of the production process, it is therefore straightforward to look at all production components and to collect all those situations in which at least two alternative actions are possible.

[17] Assigning a start date to a manufacturing operation is an example of a non-physical action because it only changes the state of the controller, not of the manufacturing process itself. Note that assigning a start date to an operation does not imply that the operation is really started at that date. For this, a controller has to initiate the operation at the specified date.

Example. In the example production system, transportation switch S_3 has two alternatives for any workpiece coming from switch S_2 (see figure 4.2). It can either move the workpiece to machine M_1 or to the switch S_4. Any control system controlling this production system must make this control decision at some point. Therefore, there must be an effectoric decision task covering this control decision and the possible physical actions in this situation.

Switch S_4 has no choice, but to move any arriving workpiece onto the conveyor belt leading to switch S_5. A decision situation thus arises only if two workpieces arrive at the switch (nearly) simultaneously. In such a case, the switch S_4 has the alternatives of moving one of the workpieces first. Even though occurring rarely, this decision may be relevant to the overall performance of the production system because it has an influence on the time it takes each workpiece to reach its destination. The decision task must therefore be included in the initial set of effectoric decision tasks.

In contrast to switch S_3 and S_4, transportation switch S_5 has no choice at all because it has only one exit and one entry. Theoretically, the switch could delay the transportation for any length of time. But there is no reason to do so. So practically, switch S_5 has no choice, but to immediately move the workpiece to its exit. Consequently, switch S_5 does not require an effectoric decision task (but can be controlled by a local controller without any interaction with other controllers).

The identification of effectoric decisions can be facilitated through the analysis of typical scenarios occurring during the production process. In this approach, the designer tries to list as many different production scenarios as possible and identifies all effectoric decisions appearing in these scenarios. To list typical scenarios, the designer can employ the use-case approach which identifies typical scenarios in which a system will be used (Jacobson 1992, Buhr 1998, Sutcliffe 1998). To identify the effectoric decisions required in one of these scenarios, the designer must analyse the different events and actions that should or could take place in such a scenario. This can be done with any of the modelling approaches for manufacturing processes (see section 3.4). An example of a typical scenario is the processing of a workpiece. In order to become a product, a workpiece must run through many steps: load workpiece into the system, transport it to the next machine, load it into the machine, process the workpiece, release it from the machine, transport it to the next machine, ..., transport it to the exit, and unload it from the system (see figure 4.6). If the designer mentally follows the possible processing (and transportation) steps the workpiece has to go through, many effectoric decisions become obvious. Another example of a typical scenario, which is orthogonal to the scenario of a workpiece, is that of a resource which is continuously processing

different workpieces. Even though this scenario partly overlaps with that of the workpiece, it may contain additional aspects, such as tool changes the machine has to perform between the processing of different workpieces. The generation of (typical) scenarios, however, is application dependent since the notion of a typical scenario depends on the nature of production process. Nevertheless, workpiece and resource scenarios are good candidates for identifying (nearly) all effectoric decision tasks because production processes are always designed to be minimal (i.e., they are equipped only with those capabilities which are required to create the desired products).

Example. Looking at the scenario of a workpiece in the example production system, it becomes clear that the control will involve two additional decision tasks. The first decision task will be responsible for loading the workpiece (at the loading station). A second decision task will be necessary, whenever the machines have overlapping capabilities, i.e., a workpiece can receive an operation at different machines. The second decision task will then be responsible for deciding which operations a workpiece will receive at a machine.

To find all effectoric decisions, it is often necessary to analyse many different scenarios in order to get a complete picture of all necessary effectoric decisions. To ensure that the list of effectoric decisions is complete, a valuable heuristic is to check the specification of the control interfaces in the production system for possible actions which have not been covered yet. Another approach could be to model the manufacturing process as a Petri net and use the associated analysis tools to identify any decision conflicts arising during the production process (see subsection 3.4.3). However, there is usually no guarantee that the model is complete, and the analysis might have to be repeated if missing decision tasks are discovered at later steps.

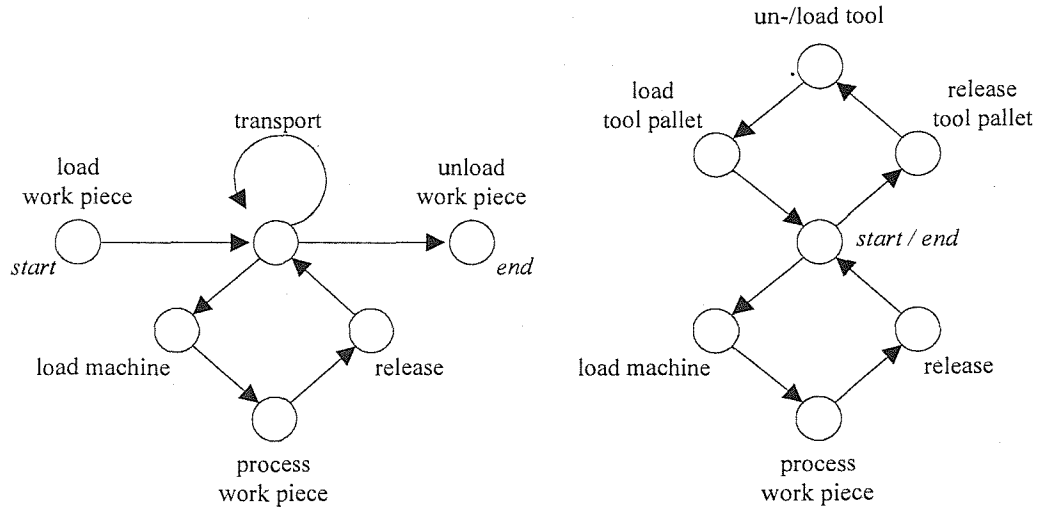

Figure 4.6: Typical scenarios for a workpiece and a machine.

Example. The four decision tasks identified in the example production system are the only decision tasks necessary to control the production process. Note that no decision task is necessary for the unloading of a workpiece because it is unloaded as soon as possible. Note also that there are no decision tasks for managing tools at a machine because for this simple production system it is assumed that the tools are handled manually.

4.2.1.3 Characterising effectoric decisions

Each effectoric decision task identified is characterised according to the pre-defined schema shown in table 4.1, which is derived from the model of a control decision discussed at the beginning of this subsection (see figure 4.5). The attributes *control interface*, *trigger*, *decision space*, and *local decision rule* are the core aspects of this definition (the attributes *id*, *title*, and *parameters* are only introduced for ease of reference and to facilitate the characterisation).

Attribute	Description	
id	unique identifier	mandatory
title	short description of the decision task	optional
parameters	Optional parameters that function as variables in the specification of the following attributes. With the help of parameters, similar decision tasks can be generalised into decision types covering different situations during the production process.	optional
control interface	The control interface that is necessary to enact the physical actions of the decision space.	mandatory
trigger	Specification of situations in the production process in which the effectoric decision becomes necessary.	mandatory
decision space	The set of decision alternatives available in this decision situation. At least one of the decision alternatives must be a physical action.	mandatory
local decision rule	Any constraints and preferences on the decision space that are induced by local aspects of the decision situation.	optional

Table 4.1: Schema for effectoric decision tasks.

An effectoric decision becomes necessary whenever a specific situation occurs during the production process. This situation (or set of situations) is specified by the trigger

attribute. To take an appropriate action in this situation, the controller has several options which are specified by the decision space. Due to the very general definition of an effectoric decision task, it is only required that the decision space has at least one physical action. The other alternatives may be computational actions, such as the commitment to a certain behaviour in the future, the acquisition of some relevant knowledge, or even the null action. In case the null action is included in the decision space, the controller may or may not choose to act. The attribute of the local decision rule, in turn, specifies which decision alternative should be chosen when considering only local aspects of the decision situation. In particular, a local decision rule may use only information that is physically available at the components involved in the decision situation. The specification of the decision situation in the trigger attribute thus also defines what constitutes local and non-local aspects. Finally, the control interface attribute must list all those control interfaces that are necessary to enact the decision alternatives requiring physical actions.

The restriction of the decision rule attribute to only local information facilitates the understanding of the control problem. If the decision rule was not restricted to local information, it could involve information or even decision tasks at other components if their results were relevant to the selection of the local decision alternatives. This could lead to arbitrarily complex decision rules which would be difficult to analyse in the first step. It is easier to start with the local decision rules and then to add any interrelations or dependencies between the decision tasks and their rules at later stages. Obviously, this implies that the *local* decision rules may be sub-optimal with respect to the production goals if possible dependencies are ignored at this stage. Any sub-optimality, though, will be remedied in the subsequent analysis step (see subsection 4.2.2).

The above definition of a decision task is kept very general in order to cover all possible kinds of effectoric decision tasks. Most effectoric decision tasks, though, will satisfy a more restricted definition which is given in table 4.2. These decision tasks are characterised by a decision situation which arises at a single production component and the decision alternatives are all physical actions that are available to the production component in this situation. The physical actions, however, may include the "null" action, which is assumed to have no effect. The necessary control interface is obviously the control interface of the production component. The following examples of effectoric decision tasks all satisfy this narrower definition.

Attribute	Description
id	(as in table 4.1)
title	(as in table 4.1)
parameters	(as in table 4.1)
control interface	the control interface of a production component
trigger	Specification of a situation at the production component in which the effectoric decision becomes necessary.
decision space	The set of physical actions provided by the control interface.
local decision rule	(as in table 4.1)

Table 4.2: A typical specification of an effectoric decision task.

Example. In the case of the transportation switch S_3, a decision is required every time a workpiece reaches the switch (see table 4.3). The switch must then choose one of the two possible exits and transfer the workpiece to this exit. The control interface for this decision task is therefore the control interface of the switch, and the decision space consists of the two possible exits of the switch. Since the decision should be made immediately in order not to block the entry, the decision space does not contain any timing aspects. The local decision rule for choosing the right exit is simply to (randomly) choose any free exit because any information necessary to correctly route the workpiece is non-local information. The decision task is not able to access any information about the capabilities and workload of any machine in the production system. This information, however, would be necessary to choose an appropriate next machine for the workpiece.

For switch S_4, a decision is required every time two workpieces reach the switch nearly simultaneously (see table 4.4). The switch must then choose one of the workpieces to move first. This problem is identical to choosing an entry. The decision space of this decision task consists therefore of the two entries of the switch. Since the decision should be made immediately in order not to block the entry, the decision space does not contain any timing aspects. The only local criteria available in this situation is to choose the workpiece that is closest to its deadline (assuming such information about the workpiece is attached to it).

Attribute	Description
id	D_2
title	Choose an exit at switch S_3
parameters	workpiece W_i
control interface	control interface to S_3
trigger	W_i arrives at entry
decision space	$\{exit_1, exit_2\}$
local decision rule	Choose any free exit.

Table 4.3: Example effectoric decision task at switch S_3.

Note that both effectoric decision tasks characterised above have already been generalised in order to be applicable to any decision situation arising at the switches. This has been achieved by introducing parameters for the workpieces. The decision tasks can be further generalised by replacing the reference to switch S_3 (or switch S_4 respectively) by a parameter; for example S_k. The decision tasks are then applicable to any switch in the example production system with the same number of entries and exits.

Attribute	Description
id	D_3
title	Choose an entry at switch S_4
parameters	workpieces W_i and W_j
control interface	control interface to S_4
trigger	W_i at $entry_1$ and W_j arrive at $entry_2$
decision space	$\{entry_1, entry_2\}$
local decision rule	Choose the workpiece closest to a deadline.

Table 4.4: Example effectoric decision task at switch S_4.

For completeness, the remaining two decision tasks of the example production system are characterised as well. First of all, a decision task for loading workpieces must be defined (see table 4.5). This decision task must decide which workpiece to load next once the exit of the loading station is free, i.e., once it is physically possible to load a new workpiece. It can choose any workpiece available in the loading stock.

Attribute	Description
id	D_1
title	Load workpiece from stock
parameters	(none)
control interface	control interface to loading station
trigger	exit is free
decision space	set of available workpieces
local decision rule	Choose any workpiece available in the stock.

Table 4.5: Example effectoric decision task for the loading station.

The last decision task to be defined for the example production system is responsible for processing a workpiece at a machine (see table 4.6). This decision must be made once a workpiece arrives at the machine. The machine may choose any set of operations that is required by the workpiece in its current processing state and that the machine is currently able to provide.

Attribute	Description
id	D_4
title	Process at machine
parameters	machine M_i, workpiece W_j
control interface	control interface to M_i
trigger	W_j arrives at M_i
decision space	set of operations of M_i
local decision rule	Choose the maximal set of operations that W_j requires in its current processing state and that is currently available at M_i.

Table 4.6: Example effectoric decision task for the machines.

Note that at this stage of the analysis, the above schema only prescribes the identification of local constraints and preferences for a decision task. Any constraints and preferences involving anything other than local aspects are added when the decision dependencies are identified (see subsection 4.2.2).

The set of decision tasks and their temporal order of occurrence can be represented in a ***trigger diagram***. In this diagram, decision tasks are indicated by nodes (annotated either with their id or with the title of the decision task) and the temporal order of their occurrence is specified through arrows. More precisely, an arrow expresses the fact that the physical action enacted because of the first decision may or must lead to a situation

triggering the second decision. The arrows thus identify all possible causal relationships between decision tasks. To improve readability, the trigger diagram may be extended by ***events*** in the production process that explain how the enaction of one decision task can lead to a situation triggering the other decision task. The following example illustrates the concept of the trigger diagram.

Example. The very first decision to be made in the example production system is to load a specific workpiece (effectoric decision task D_1). Once loaded, the workpiece continuously moves along the transportation circle until it is moved to a machine that processes it. That is, the workpiece enters the transportation circle at switch S_1 which requires decision task D_3. On the transportation circle, it either passes a machine (D_2 and D_3) or is moved into a machine (D_2) where the next effectoric decision is to choose the set of operations to be applied to the workpiece (D_4). After processing, the workpiece enters the transportation circle again over the second switch (D_3). Once the workpiece has received all processing specified by its associated order, the workpiece is moved to switch S_{12} and then to the unloader (D_2). Finally, the workpiece is unloaded which is represented only as an event in the production system, since it does not involve any decision – the workpiece is unloaded as soon as possible. The resulting trigger diagram is shown in the figure 4.7.

Since the trigger diagram illustrates the temporal order of decisions (as they are triggered by the physical process), it can be used as a visual aid in the following analysis and design steps.

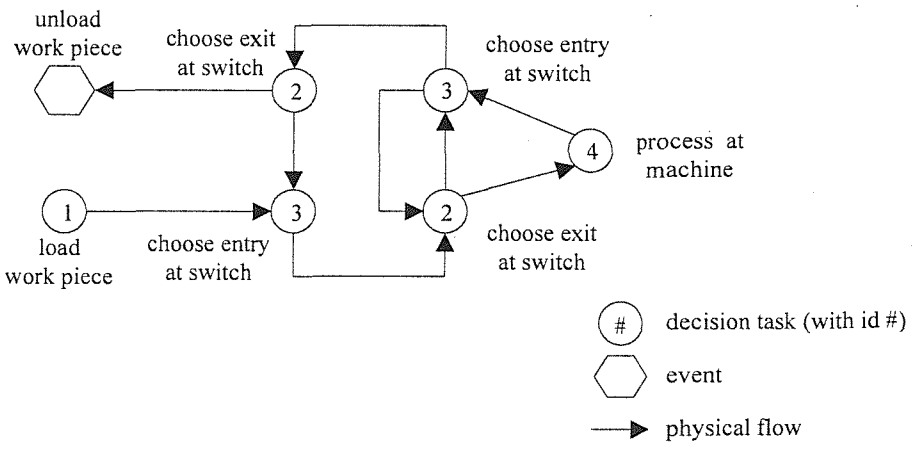

Figure 4.7: The trigger diagram for the example production system.

4.2.2 Identification of decision dependencies

The decision model developed in the previous subsection only covers purely local aspects of the control problem. For each effectoric decision task, it specifies only the local decision rule, i.e., the local constraints and preferences on the decision. But control decisions are not executed in isolation. The effects of their actions may interfere with each other and create a complex overall system behaviour. Purely local decisions in such a system may lead to undesirable situations, such as deadlocks or starvation. Or the system may simply show poor performance with respect to the production goals.

Example. The transportation switch S_3 of the example production system has to choose one of the exits for each workpiece arriving at its entry. Which exit it chooses is irrelevant to the switch. It can move a workpiece equally well to any of the exits (as long as both exits are free). With respect to the system performance, however, it is by no means irrelevant onto which exit a workpiece is moved. First of all, a workpiece should only be moved to machine M_1 if the machine is able to process the workpiece. And second, through its decision the switch determines the workload of machine M_1 and thus its capacity utilisation.

To achieve the production goals and avoid undesirable behaviour, it is therefore necessary to look also at the combined effects of the effectoric decision tasks. Assume there are two or more decision tasks whose effects are somehow linked. These decision tasks are called ***dependent (on each other)*** if the decision choices proposed by the local decision rules lead to either undesirable or poor performance of the production system (see subsections 2.2.2.1 and 3.5.4.3). In such a case, the decision tasks should choose a different combination of actions than prescribed by their local constraints and preferences. The local decision rules may either propose a combination of actions that should be avoided or that is simply inferior to other combinations. In the first case, the combination of actions can be avoided by imposing a constraint on the product of the decision spaces of the different decision tasks. In the second case, the inferiority of a combination can be expressed by a preference function on the product of decision spaces. Constraints and preferences on the product of different decision spaces will be called ***non-local constraints and preferences***.

Example. In the above example, the switch S_3 should not send a workpiece to machine M_1 if the machine is unable to process the workpiece. This can be expressed as a non-local constraint stating that the switch may only choose the exit leading to the machine M_1 if the set of operations that M_1 can apply to the workpiece in its current processing state is non-empty, otherwise the workpiece should be passed on to switch S_4. Furthermore, switch S_3 should pass the workpiece on if other machines that are also able to process the

workpiece currently have a smaller workload. This can be expressed as a preference function over the allowable exits. Note, however, that these constraints and preferences not only involve decision task D_2, but also D_4 (which determines the operations to be applied to a workpiece at a machine). These constraints and preferences are thus non-local.

The literature provides several different kinds of classifications for dependencies (see subsections 2.2.2.1 and 3.5.4.3). Some of these classifications are difficult to apply to the decision model developed in the previous analysis step because the classification concepts, such as goals, are not part of the decision model, while others, such as the plan relationships of Decker (see subsection 2.2.2.1), can provide valuable hints at situations in which decision tasks may enable, facilitate, hinder, or even cancel each others' effects. At this stage of the analysis, though, it is sufficient to determine whether there exists a dependency between decision tasks or not. A more detailed analysis of the dependencies is deferred until the selection of appropriate interaction protocols in step 3 of the methodology (see section 4.4). One special kind of dependency, however, deserves explicit attention at this stage because it may introduce a component not covered yet by the decision model. This kind of dependency is given if a decision task is dependent on the provision of non-local information, i.e., the decision task should choose different alternatives depending on this non-local information. For the first part of the above example, switch S_3 is actually only dependent on the information as to whether machine M_1 can process the workpiece at all, whereas for balancing the workload of the machines switch S_3 also needs to know which operations will actually be applied to the workpiece and thus how the decision tasks will be resolved. The non-local information a decision task is dependent on may be provided by another decision task, as in the above example, or by another source, be it a component of the production system or a data source external to the production system. In case the information is not available through another decision task, the corresponding information source must be included in the decision model (see for example figure 4.8).

To summarise, a dependency thus expresses that there are additional constraints and preferences that must be taken into account when choosing the best alternative. Since these constraints and preferences, however, are non-local, the local controllers must interact to resolve the dependencies (or acquire the non-local information). The task of the analysis phase is therefore to identify and characterise these dependencies.

4.2.2.1 Identifying dependencies

The identification of dependencies is usually obvious and straightforward (as in the previous example). Many dependencies can be identified simply by studying the trigger diagram which represents (most of) the effects decisions create during the production process. Other dependencies can be identified by studying related decision parameters

of decision tasks. If two tasks affect the same components, it is likely that their decisions will be dependent. In the previous example, the transportation switch (routing the workpiece) and the machine (applying operations to the workpiece) both make decisions about the same workpiece and are consequently linked somehow. In some cases, however, it can be quite difficult to identify the dependency between decision tasks. For a job-shop environment, for instance, it has been shown that the scheduling problem is NP-hard, i.e., solving the scheduling problem is computationally expensive (Garey 1979). The reason for this is that changing a resource allocation at one machine may create a ripple effect through the whole schedule and may thus cause a change of an allocation at a different resource for a quite different time slot. Moreover, it may not be obvious from the start that such an effect is possible.

In the following, it is assumed that the designer is able (with acceptable effort) to identify all *relevant* dependencies in the production system considered. This assumption is made because it is not the goal of this methodology to provide techniques to analyse production systems, but to develop a methodology for the design of agent-based production control systems. Furthermore, it is reasonable to assume that a control engineer is able to analyse the behaviour of the production system if he intends to build and operate it. Tools for doing so are for instance Petri nets and discrete event systems (see section 4.1). If, nevertheless, a relevant dependency is not identified at this stage, but detected at subsequent design steps, the designer has to return to this stage and include the dependency in the decision model. Depending on the impact of this dependency on the overall system design, all subsequent design steps may have to be (partially) redone. Likewise, the designer may decide at this stage that a dependency is not relevant to the achievement of the production goals and may omit it right from the beginning in order to reduce the necessary effort at later stages of the design. Such a design decision, though, should be documented.

4.2.2.2 Characterising decision dependencies

The decision dependencies identified are characterised with the help of the pre-defined schema shown in table 4.7. The core aspects of the schema are the attributes *decision tasks*, *constraints*, and *preferences*. The attribute *decision tasks* list the ids of those decision tasks that are affected by the decision dependency or characterise the information sources able to provide the non-local information needed. The attribute *constraints* lists any combinations of decision alternatives (of the different decision tasks) which should not be chosen. Alternatively, the attribute may list the set of decision combinations which are allowed. Finally, the attribute *preferences* defines a preference function on the allowed set of decision combinations. The attributes for *constraints* and *preferences* are both optional, but at least one must be non-empty (otherwise there is no dependency).

Attribute	Description	
id	unique identifier	mandatory
title	short description of the dependency	optional
decision tasks	decision tasks or information sources involved in the dependency	mandatory
constraints	Any combination of decision alternatives that should not be chosen.	optional
preferences	Preference function on the combination of decision alternatives that are not forbidden by the constraints.	optional

Table 4.7: Schema for decision dependencies.

Example. In the example production system, the simplest, but also most important, dependency between decision tasks is the requirement that a workpiece eventually arrives at a machine that is able to process it, or reaches the unloader if the processing is done. The routing of a workpiece is determined by the switches S_3, S_6, S_9, and S_{12}. The decision making of these switches is captured by a single decision task (namely D_2). Consequently, the dependency exists between instances of D_2 (see table 4.8).

Attribute	Description
id	DP_1
title	Route workpieces to the next machine
decision tasks	D_2 and machine capabilities
constraints	Eventually direct a workpiece to a machine that is able to process it, or to the unloader if the workpiece is finished.
preferences	Route the workpiece on the shortest path (if possible).

Table 4.8: Example dependency for routing workpieces.

The second important dependency in the example production system is to co-ordinate routing and processing of a workpiece such that the workpiece receives all the operations it requires and the number of stations visited is minimised (see table 4.9). Minimising the number of stations visited by each workpiece is important because otherwise a workpiece takes too long to run through the production system and thus consumes too much processing capacity.

Attribute	Description
id	DP_2
title	Minimise the number of stations visited
decision tasks	D_2 and D_4
constraints	Choose a sequence such that the workpiece receives all required operations (in the right order).
preferences	Choose a sequence of machines that is minimal.

Table 4.9: Example dependency for choosing machines.

The third important dependency is to limit the work-in-process in the production system. If the loading station introduces workpieces faster than these can be processed by the machines, then the transportation system will eventually overflow with workpieces and, as a consequence, run into a deadlock or a congestion. To prevent a transportation deadlock, the decision task D_1 must take into account the current work-in-process and delay the next loading operation if necessary. However, D_1 can only estimate how the current work-in-process will use the capacity of the machines and the transportation system if it can anticipate how the workpieces will be routed and processed. It is thus necessary for D_1 to co-ordinate its decision with the routing and processing decisions D_2 and D_4 (see table 4.10).

Attribute	Description
id	DP_3
title	Limit work-in-process
decision tasks	D_1, D_2 and D_4
constraints	Introduce only as many workpieces as the production system is able to handle.
preferences	

Table 4.10: Example dependency for loading workpieces.

The set of dependencies can also be represented in the trigger diagram by simply adding a new arrow type for dependencies. A dependency arrow connects two decision tasks if there exists a dependency between these decision tasks. Dependencies between more than two decision tasks are represented by an arrow with more than two ends (see figure 4.8). To indicate the dependency on an information source, the source is explicitly shown in the diagram by an information node.

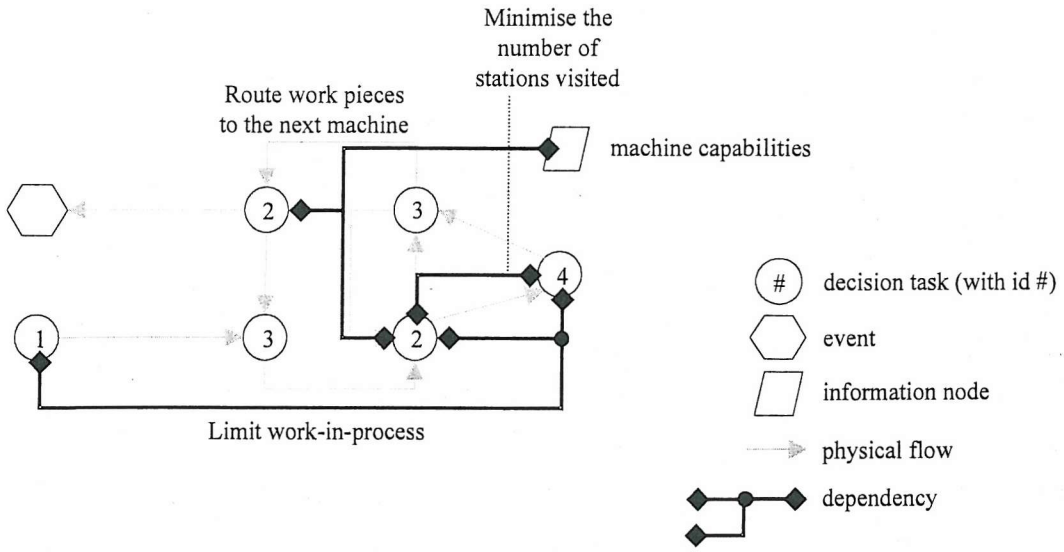

Figure 4.8: The dependency diagram for the example production system.

4.2.3 Result of the analysis steps

The result of the analysis steps is a decision model representing the production control problem. The decision model consists of three parts:

- a list and characterisation of all effectoric decision tasks necessary to run the production process;
- a list and characterisation of all relevant decision dependencies between the effectoric decision tasks; and
- a trigger diagram including all dependency relations.

This decision model contains all the decisions any control system must make in order to solve the control problem. It also contains any local or non-local constraints and preferences the system must take into account when choosing physical actions. In this sense, the decision model is complete. Any control system implementing the decision model thus solves the production control problem.

The decision model, however, is not yet executable. First of all, there may be conflicts between local and non-local decision rules, and the model does not yet say how to resolve these conflicts. Secondly, it is not yet clear how the non-local decision rule (and the conflict resolution strategy) can be computed; for this, the decision tasks will have to interact somehow. And thirdly, it is still not obvious which (effectoric) decision task should be assigned to which local controller and what kind of controller this should be.

All of the above questions will be answered in the following design steps. First, the

next section will explain how decision tasks can be assigned to controller agents. Then, section 4.4 will select appropriate interaction protocols that will enable the controllers to resolve any dependencies arising at run-time.

4.3 Identification of agents

With the initial decision model as a basis, it is now possible to start the design process by identifying the agents of the control system. The agents have to be identified first as they are the basic building blocks of an agent-based control system; they define the overall architecture of the system. Interactions can only be defined by specifying which agent (type) is interacting with which other agent (type). At the same time, however, the system architecture also restricts the set of possible interactions, since it specifies the set of agents existing in the control system. It is therefore crucial to identify a set of agents which supports the task of achieving the production goals best.

An agent is a decision maker which is able to pro-actively achieve its goals while it is continuously adapting to its dynamic environment (see section 2.2). With respect to production control, this definition requires that a control agent performs some kind of decision making (see section 2.1). Conversely, this definition also implies that any entity of the control system performing one of the decision tasks identified in the previous section *is* an agent (see also (Kendall 1996, Colombo 2002)). The first step of this design phase is therefore to introduce control agents by clustering decision tasks and assigning an agent to each cluster of decision tasks (see step 2.1 in figure 4.9). However, not every assignment of decision tasks to agents is equally suitable for the implementation of the control system. For instance, the designer could assign all decision tasks to a single agent. Although such an assignment is allowed by the above definition, it may result in an agent which is too complex to be implemented if the control problem consists of too many decision tasks. This design step therefore includes a set of rules for assigning decision tasks to agents which leads to a more modularised design and thus reduces the complexity of the remaining design tasks.

Furthermore, not every decision network is equally well suited to agent identification, irrespective of the assignment rules used to cluster the decision tasks. The analysis phase described in the previous section focused on control aspects and deliberately did not take into account any criteria for structuring an agent-based system. In case the structure of the decision model is contrary to the criteria for identifying agents, the rules for assigning decision tasks to agents may fail to increase the modularization of the design. It must therefore be possible to reorganise the decision model such that it becomes more suitable for agent identification. Such a reorganisation, though, must not change the functionality of the control system. That is, from the point of view of the controlled process, the reorganised control system should send the same control

commands as the original design. This requirement ensures that the reorganised decision model still solves the production control problem. Design step 2.2 therefore describes allowable modifications of the decision model that improve the decision model with respect to agent identification without changing the control functionality. These modifications must be applied to the decision model until either a satisfying set of agents has been identified, or it becomes clear that it is not possible to identify a suitable set of agents (see figure 4.9). In the latter case, the designer has to abandon the agent-based approach. Following the design steps for agent identification, the designer thus either identifies a suitable set of control agents or realises that an agent-based approach is not appropriate for the given control problem.

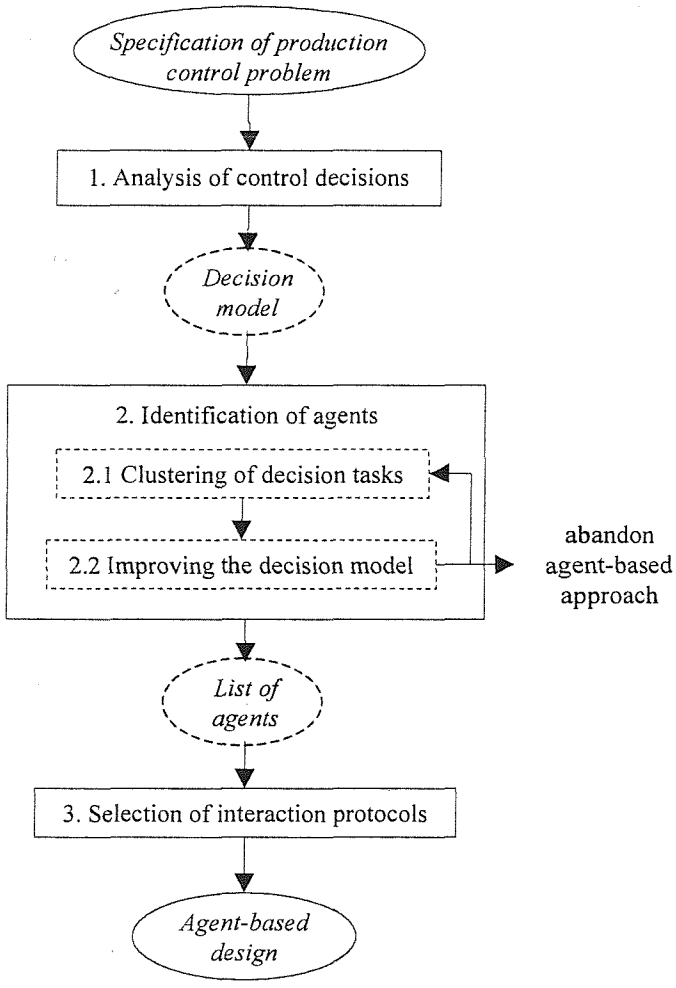

Figure 4.9: Steps for the identification of agents.

The presentation of the agent identification step is organised as follows. The first subsection describes the rules for assigning decision tasks to agents. Subsection 4.3.2 presents possible operations on the decision model that can be used to improve the decision model with respect to the identification of agents. The last subsection finally

summarises the overall process and the results of this design phase.

4.3.1 Clustering of decision tasks

In the decision model, a control agent is identified by assigning it to a set of decision tasks for which it is solely responsible. Since a decision task should be assigned to at most one agent (otherwise the decision task must be split into several decision tasks) and every decision task should be assigned to some agent (otherwise a decision situation is not served), the identification of the control agents is essentially a partitioning of the set of decision tasks. The objective of this design step is therefore to find a partition of the decision tasks that represents a good design for the given control problem.

4.3.1.1 Clustering algorithm

In principle, any partition of the set of decision tasks defines a possible set of agents for the control system. The set of all partitions, however, has exponential size in the number of decision tasks. For realistic design problems, it is therefore nearly impossible to manually consider all possible partitions. On the other hand, some partitions are obviously of lower value than others. If, for example, it is possible to assign all decision tasks to one agent, but also to distribute the decision tasks to several agents, the latter solution should be preferred because it creates a more modular design. Consequently, the clustering process should first consider those partitions that promise to provide a greater design value.

It is therefore proposed to construct the partitions by successively clustering decision tasks that should be performed by the same agent (see figure 4.10). Which decision tasks to cluster is determined by clustering rules (which are defined below). These rules state under which conditions two or more decision tasks should be assigned to the same agent. The clustering rules are applied according to the following (simple) algorithm:

1. Start with the set of clusters in which each cluster consists of a single decision task.
2. For each subset of clusters, apply every clustering rule to every combination of decision tasks from the different clusters.
3. Whenever a clustering rule applies, replace the clusters by a new cluster consisting of all the decision tasks from these clusters.
4. Repeat step 2 and 3 until no more clusters can be combined.

Note that step 2 considers the powerset of all decision tasks which is exponential in size. However, step 2 can be performed more efficiently if domain knowledge is

available (see subsection 4.3.1.2). It will then be obvious to which decision tasks the clustering rules may apply.

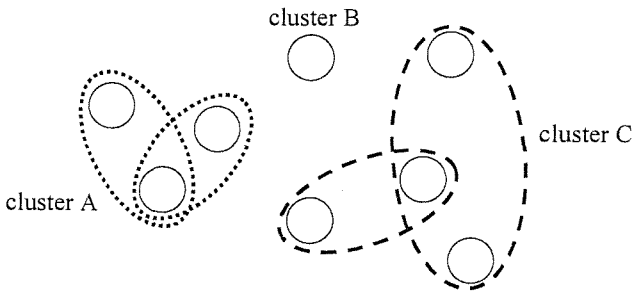

Figure 4.10: Example clustering.

4.3.1.2 Clustering rules

The clustering algorithm described above uses clustering rules to perform the actual clustering of decision tasks. This subsection presents three clustering rules for identifying agents.

Obviously, the most important design criterion for developing an agent-based control system is to find a set of agents that is able to achieve the production goals, and thus to solve the production control problem. However, every control system implementing the decision model achieves the production goals because the decision model specifies all effectoric decisions that are necessary to solve the control problem. All partitions are thus equivalent with respect to the goal achievement. Different partitions, however, are certainly not equivalent with respect to how they implement the goal achievement. Because of the different assignments of decision tasks to agents, different partitions require different kinds of interactions. For instance, in one partition a dependency may hold between decision tasks within one agent and thus requires no interaction, whereas in another it may hold between decision tasks of different agents which consequently must interact to resolve the dependency. The objective of the clustering rules should therefore be to find a partition that facilitates the implementation of the decision model, i.e., the corresponding set of agents supports the implementation of the necessary control functionality better than other sets of agents.

To derive a set of clustering rules that facilitates an agent-based implementation, it is necessary to examine whether the separation of two (or more) decision tasks is impossible or too difficult to implement, or whether such an implementation would be too inefficient. In these cases, the decision tasks should be assigned to the same agent. The following rules thus specify whether the coupling of two or more decision tasks is high (see also subsection 3.2.2).

Clustering rule #1: ***interface cohesion***

Decision tasks using the same control interface should be assigned to the same agent.

The control interface of a production component provides unrestricted access to the functionality of the component. That is, any control command issued through the control interface will be executed immediately. If two agents both had unrestricted access to the control interface, they would be able to issue contradictory commands which – in the worst case – could damage the component (or humans working near the component). The control over a physical component should therefore be assigned to only one agent. Naturally, if another agent also needs to use the component, it could always request the agent controlling the component to perform the required operations.

Note that the above clustering rule (as well as all of the following rules) should be used *minimally*. Assume for example that the production system consists of only two control interfaces and every decision task of the decision model accesses only one of the control interfaces, in particular none of the decision tasks use both. Then assigning all decision tasks to a single agent would be conform to the above clustering rule. However, separating the decision tasks into two partitions, each accessing only one of the control interfaces, would also be conform. Clearly, the latter alternative is to be preferred over the former because it preserves as much structure of the decision model as possible.[18]

Example. In the example production system, all instances of decision task D_2 responsible for the same switch should be assigned to the same agent. That is, at most one agent is responsible for all workpieces arriving at switch S_3.

A more general version of the first clustering rule is given by the next rule.

Clustering rule #2: ***state cohesion***

Decision tasks changing or affecting the state of the same production component should be assigned to the same agent.

The second clustering rule extends the first rule because an agent issuing commands through a control interface is automatically changing the state of the production component executing these commands. The second rule, though, generalises the first rule because it additionally applies to decision tasks that change the state of a production component without having direct access to the component through a control interface. A typical example is the state of a workpiece. The state of a workpiece is changed whenever the workpiece is processed. The decision task deciding to process the workpiece, though, enacts its decision not through a control interface to the

[18] This design strategy is in the same spirit as the design rule *identify rather small agents* of Parunak et al. (1998). Small agents are easier to construct and understand, but still provide a large space of possible (combinations of) behaviours (Parunak 1998, p. 51).

workpiece, but through a machine interface.

The motivation for this rule is to avoid redundant computation and interaction. If several decision tasks make decisions about the same component, they will have to perform similar computations and will require interactions with the other agents in order to make their individual decisions and to resolve any existing dependencies. This redundant computation and interaction can be avoided by clustering these decision tasks. Clustering these decision tasks will also avoid any intensive co-ordination between the associated agents which would be necessary because the decision tasks are affecting the same component. This clustering rule thus reduces computational and communicational requirements of the implementation, and increases the coherence of the design (see subsection 3.2.2) because control aspects that are logically related and dependent, namely decisions about the state of a single component, are clustered.

Example. Consider again the example production system. A workpiece moves along the transportation cycle until a switch diverts it to a machine. At each switch, decision task D_2 must decide for this switch whether the workpiece is moved to the machine or passed on to the next switch. The decision tasks D_2 share a dependency for the same workpiece in that they must eventually route the workpiece to a machine that is able to process the workpiece (or to the unloader) (see table 4.8).

The above rule for state cohesion proposes to cluster these decision tasks because they all access and alter the state of the workpiece. Assume these decision tasks were not clustered. Then each decision task would have to access the state of the workpiece, access the state of each machine of the production system in order to determine whether the machine is able to process the workpiece, and finally choose the machine which should process the workpiece next. Each decision task consequently has to perform a lot of redundant computation and interaction. Assigning the decision tasks to one agent avoids this redundancy. Clustering these decision tasks also avoids any co-ordination these decision tasks would have to do in order to ensure that the workpiece is not send to different machines.

The above example also shows that the clustering rule for state cohesion is not imperative. It is possible to co-ordinate the switches such that the workpiece is sent to a machine which is able to process the workpiece. However, the example also shows that the clustering rule can greatly facilitate the implementation of the control system. Generally speaking, the application of a clustering rule is thus not a deterministic design step which can be completely automated by a computer. It still leaves room for creative design decisions.

Clustering rule #3: *high interactive coupling*

Decision tasks that are *always* strongly dependent on each other should be assigned to the same agent.

This rule improves the implementability of the control system by reducing the interaction between agents. The rule basically proposes to cluster decision tasks whenever they are so strongly coupled that the agents are not able to solve their own decision tasks without solving simultaneously the other decision tasks. The following example illustrates this rule.

Example. In large-series production, the spot welding of car bodies is usually performed by a set of welding robots which are installed along a production line. That is, each car body has to pass through several welding stations in order to receive the necessary welding spots (see figure 4.11). Since the welding robots are quite flexible, one robot could take over the welding spots of a robot that has broken down – provided that it has the necessary tools to do so. The advantage of this flexibility is that the line does not have to stop just because a single robot has broken down; it can continue to weld car bodies, though with a reduced cycle time.

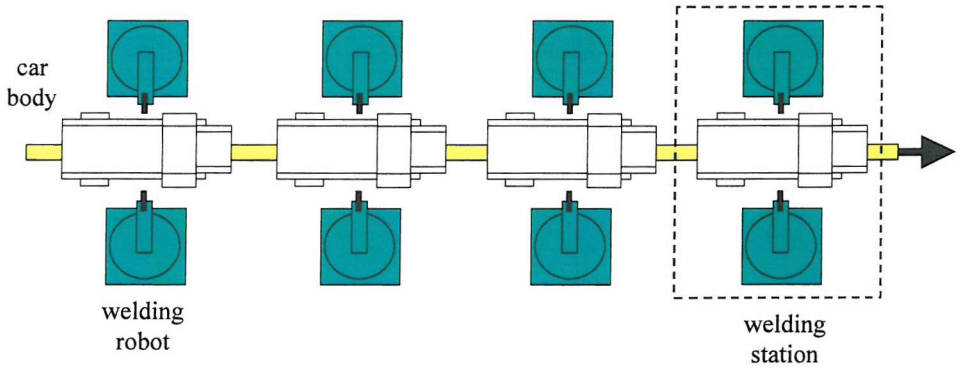

Figure 4.11: Spot welding line.

Analysing such a production system according to the methodology would yield a decision model that consists of a decision task for each robot. This decision task would decide for each car body arriving at the station which welding spots to apply to it. The decision task, however, would be dependent on the other decision tasks (for the same car body) because at the end of the line the car body must have received all welding spots it requires. That is, a decision task choosing to omit a spot at a particular station must ensure that this spot can still be welded by another robot down the line which has not yet broken down. Additionally, all welding robots should take nearly the same time to perform the welding operations in order to balance the production line and to reduce the idle times of the robots. This

means that robot A should choose a set of welding spots for its car body whose total processing time is nearly as long as that chosen by robot B for its car body. As a consequence, the decision tasks for different car bodies simultaneously welded are also dependent. It turns out that choosing a set of welding spots for the car body at the first welding station requires choosing an appropriate set of welding spots for the same car body at the other stations (in order to ensure the completeness of the welding spots), and to co-ordinate these decisions with the choices for the other car bodies already in the line (in order to balance the production line). The different decision tasks are consequently so dependent on each other that it makes more sense to implement a central optimiser, which computes the distribution of welding spots over the welding line and then distributes the results to the robots, than designing an agent-based solution.

The clustering rule for high interactive coupling is very restrictive. It only clusters decision tasks if these are logically inseparable. In principle, the rule could also be applied if a dependency is too strong or would require too much interaction. However, the judgement as to whether a dependency is too strong and cannot be resolved with the existing interaction techniques depends on the interaction techniques available at design time. Such a judgement can thus be made only after trying different interaction techniques. If, during the interaction design (see section 4.4), it becomes clear that the resolution of the dependency is difficult or infeasible, the designer may still return to this stage and cluster the decision tasks.

The application of the above clustering rules is demonstrated in the following with the help of the example production system.

Example. For the decision model shown in figure 4.8, the clustering rule for interface cohesion applies to every set of instances of the decision tasks D_1, D_2, D_3, and D_4 that are responsible for the same production component. For example, there exists an instance of decision task D_1 for every workpiece that is being loaded. However, each instance refers to the same production component, namely the loader. Consequently, there will be only one agent responsible for the loader. Likewise, there will be only one agent responsible for the switches S_1, S_3, S_4, S_6, S_7, S_9, S_{10}, and S_{12}, as well as for the machines M_1, M_2, and M_3.

Furthermore, the clustering rule for state cohesion applies to all instances of decision tasks which refer to the same workpiece. Since every decision task refers to a workpiece (even decision task D_1), the rule proposes to cluster those instances of decision tasks D_1, D_2, D_3, and D_4 which are referring to the same workpiece. That is, the rule forms a cluster for each workpiece.

The clustering rule for high interactive coupling, in contrast, does not apply to the decision model in figure 4.8 because none of the decision tasks are so tightly coupled that their decisions must be made as one.

As a consequence of the clustering process, all instances of the decision tasks are somehow clustered, either because they refer to the same control interface or to the same workpiece. The resulting partition therefore consists of only one agent.

The above example shows that a decision model may still collapse into a single agent, even if the clustering rules are used minimally. The reason for this is that the clustering rules only combine decision tasks: they are not able to separate different decision aspects of a decision task, or reorganise the decision model according to agent-oriented criteria. The following subsection therefore presents operations on the decision model that make the decision model more suitable for agent identification.

4.3.2 Improving the decision model

During the analysis phase, the decision model is deliberately developed without any consideration of criteria for structuring or even implementing an agent-based system. It may therefore be necessary to modify the decision model and make it more suitable for agent identification, in particular to avoid the collapse of the decision model during the clustering process. There are three major motivations for modifying the decision model (see also subsection 3.5.6.2):

- A decision rule is too complex to be computed in one step – it should be divided into subtasks which, in turn, may have to be assigned to different agents (cf. the strategy *identify rather small agents* in subsection 3.5.6.2).
- A decision task involves several independent production components – the decision should be divided into the different decision aspects concerning each production component in order to preserve their autonomy (cf. the strategy *identify things rather than function* in subsection 3.5.6.2).
- Decision tasks are too dependent on each other – the decision process must be re-arranged to reduce the dependencies, and thus the communication requirements.

This subsection presents two basic operations on the decision model which improve the structure of the decision process, but leave its functionality unchanged. That is, the new decision model executes the same control commands as the original decision model and consequently achieves the same goal satisfaction as the first. In this regard, the new and the original decision model are equivalent. The two operations are *distributing decision tasks* and *introducing new decision tasks*.

4.3.2.1 Distributing decision tasks

To reduce the complexity of a decision task, or to separate different decision aspects, a decision task can be distributed by replacing the original decision with several new decision tasks (see figure 4.12). Each new decision task is a partial copy of the original decision task. That is, the `distribute` operation may restrict either the decision space, the decision rule, the control interface, or several of them. The set of newly introduced decision tasks, however, must cover every aspect of the replaced decision task, and there must be at least one decision task that receives the trigger for this decision task and one that initiates the chosen action once the decision has been made, i.e., one must inherit the control interface.

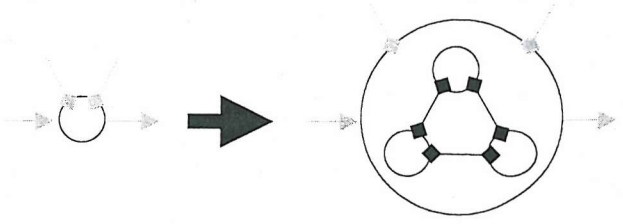

Figure 4.12: The `distribute` operation.

Note that distributing a decision task may have quite different effects depending on which decision aspects are distributed. The two most basic effects are discussed in the following.

distributing the decision space – The decision space is partitioned into non-overlapping subsets, and every newly introduced decision task inherits one partition of the original decision space.

In this case, at most one of the new decision tasks should decide to act in the end because the original decision task was supposed to select only one action out of the whole decision space. As a consequence, the decision space of each new decision task should include the null action (if it is not already included in the decision space).

distributing the decision rule, but not the decision space – The set of constraints and preferences is partitioned and every newly introduced decision task inherits only a partition of the original constraints and preferences, while only one decision task inherits the decision space.

In this case, at least one decision task has constraints or preferences about the actions of other decision tasks (for example, because it will be affected by their decisions) without enacting the decision itself (i.e., in all cases it will enact the "null" action). Formally, these constraints or preferences are

actually non-local constraints between the decision tasks and should therefore be represented as dependencies.

To distribute a decision task, the original decision task is replaced by at least two new decision tasks which are created according to the following rules:

- The decision space is partitioned into sub-spaces and every new decision task inherits a sub-space. A decision task may inherit an empty sub-space, i.e., its decision space consists only of a null action. However, the union of decision spaces of the newly introduced decision tasks must equal the original decision space.
- The decision rule is partitioned and every new decision task inherits a partition of the decision rule. That is, each decision task inherits only some of the constraints and preferences of the original decision task, possibly no constraints and preferences. However, the constraints and preferences of the newly introduced decision tasks must add up to the original decision rule.
- At least one new decision task receives the trigger.
- Each control interface of the original decision task is inherited by exactly one of the new decision tasks.

After distributing the decision task, the dependencies must be updated. First of all, the new decision tasks must inherit every dependency of the original decision task, such that each dependency is inherited by at least one of the new decision tasks. And second, any possible dependencies between the new decision tasks must be identified as prescribed in subsection 4.2.2.

Example. Several decision tasks of the example production system involve different production components: decision tasks D_2 and D_3 route a workpiece at a switch, and decision task D_4 chooses the processing operations for a workpiece at a machine (see figure 4.8). All three decision tasks are candidates for distribution.

Decision task D_2 is not distributed because the distribution does not provide any improvement of the decision model. Distributing decision task D_2 into two decision aspects, one for the switch (e.g., D_{2_SW}) and one for the workpiece (e.g., D_{2_WP}), would create an empty decision task D_{2_WP} because all attributes of decision task D_2 are inherited by D_{2_SW}. In decision task D_2, the only control interface belongs to the switch, the trigger arises at the switch, the decision space consists of actions available only to the switch, and finally the local decision rule simply chooses an exit randomly. To incorporate the decision aspect of the workpiece that it should be moved to a machine that is able to process it – which is already represented as a

dependency (DP_1) – it is necessary to introduce a new decision task covering exactly this decision problem. How this can be done is discussed in subsection 4.3.2.2.

Decision task D_3 is split into one decision aspect for the switch (D_{3_SW}) and one for each workpiece arriving at the switch (D_{3_WP}). Similarly as for decision task D_2, decision aspect D_{3_SW} inherits most attributes from the original decision task D_3. However, the decision aspect D_{3_WP} is responsible for determining the urgency of the workpiece, i.e., how close the workpiece is to the deadline of its associated order.

Finally, decision task D_4 is split into one decision aspect for the machine (D_{4_MA}) and one for the workpiece arriving at the machine (D_{4_WP}). Again, decision aspect D_{4_MA} inherits the attributes control interface, trigger, and decision space from decision task D_4 because they refer only to aspects of the machine. The local decision rule of D_4, however, is split into the aspects concerning the machine and those concerning the workpiece. The machine can only apply operations to the workpiece which are currently available at the machine, while the workpiece wants to receive the maximal set of operations that are still missing in its current processing state.

In the following, only the new decision tasks D_{4_MA} and D_{4_WP} are shown (in tables 4.11 and 4.12).

Attribute	Description
id	$D_{4\text{-}MA}$
title	Process at machine
parameters	machine M_i, workpiece W_j
control interface	control interface to M_i
trigger	W_j arrives at M_i
decision space	set of operations of M_i
local decision rule	Choose any set of operations currently available to M_i

Table 4.11: Effectoric decision task $D_{4\text{-}MA}$ of the example production system.

Attribute	Description
id	$D_{4\text{-}WP}$
title	Process at machine
parameters	machine M_i, workpiece W_j
control interface	
trigger	
decision space	
local decision rule	Choose the maximal set of operations that W_j requires in its current processing state.

Table 4.12: Effectoric decision task $D_{4\text{-}WP}$ of the example production system.

Figure 4.13 shows the new trigger diagram after distributing the decision tasks. Note that the distribution of the decision tasks D_3 and D_4 does not create any additional dependencies other than those that the distributed decision tasks need to agree on a decision.

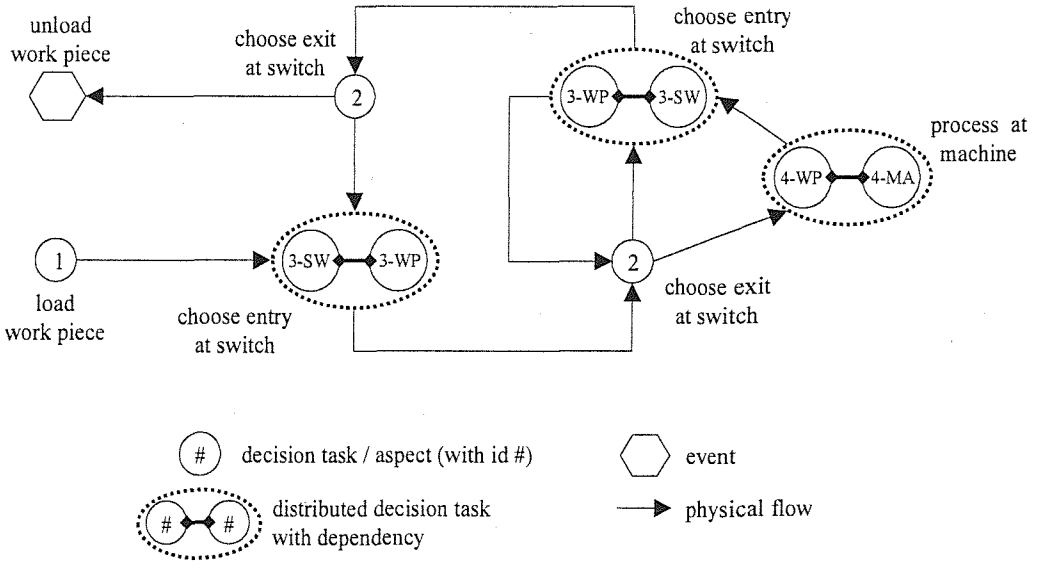

Figure 4.13: Trigger diagram after distributed decision tasks in the example production system.

4.3.2.2 Introducing new decision tasks

The distribution of decision tasks usually creates ***joint decisions***, i.e., the distributed decision tasks must jointly choose a single alternative from their common decision space. Alternatively, the introduction of new decision tasks makes it possible to introduce additional decision tasks which prepare other decisions, i.e., the new decision tasks reduce the decision complexity of the decision process by providing constraints that further restrict the decision space of other decision tasks. This may be necessary whenever the decision rule of a decision task is too complex to be computed in one step, several decision tasks have an overlapping decision problem, or the decision tasks share a strong dependency (see also (Jennings 1996)). A new decision task may then be introduced to reduce the complexity of the decision tasks or of the dependency by solving part of the decision problem beforehand.

The introduction of decision tasks increases the overall modularity of the decision process because common decision tasks are extracted, or decision tasks are divided into several decision steps. The `introduce` operation thus allows the designer to structure the decision process. To distinguish the additional decision tasks from effectoric decision tasks, the newly introduced decision tasks are called ***abstract decision tasks*** because these decisions do not directly determine the action of a component, but only influence effectoric decisions. Abstract decisions can themselves use other abstract decisions to simplify their own decisions, and may thus create an arbitrary hierarchy of decisions. The depth of this hierarchy depends on the complexity of the decision process.

To introduce an abstract decision task, the designer must create a new decision task whose decision outcome is provided as input to at least one existing decision task (see figure 4.14). The new decision task may have an arbitrary decision space and decision rule. It must, however, receive the same trigger (directly or indirectly) as one of the other already existing decision tasks and may have no control interface since it will not enact any physical actions. Furthermore, there must be at least one existing decision task which is changed such that its decision rule refers to the result of the newly introduced decision task. In turn, the decision task referring to the decision outcome of the new decision task may simplify its decision rule by relying on the abstract decision (even though it does not have to). After the introduction of a new decision task, the dependency diagram must be updated since the new decision task may also introduce new dependencies or eliminate existing dependencies.

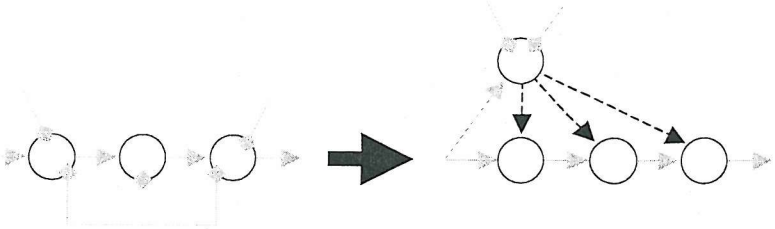

Figure 4.14: The `introduce` operation.

Example. In the example production system, a new decision task is introduced to reduce the complexity of the dependency DP_1 which exists between instances of decision tasks D_2. Dependency DP_1 states that the decision tasks D_2 should route a workpiece to a machine that is able to process it next. To determine a machine that could process the workpiece next, it is necessary to access the state of the workpiece and the state of the machines in the production system. With access to these states, it would be possible to find a machine currently providing the processing operations that are still missing in the current state of the workpiece. The information about the state of the machines, however, is not local to any of the switches. Instead of having the different switches access this information in parallel, it is easier to introduce a new decision task that provides this access and simply makes the decision which is then forwarded to the switches.

The new decision task D_5 chooses the next machine that should process the workpiece. The decision task is triggered either at the beginning, when the workpiece is loaded, or every time the processing of the workpiece at a machine has just been completed. In case the workpiece is not finished yet, it must choose a new machine in order to receive the remaining operations. In case it is finished, the workpiece chooses the unloader. In principle, the workpiece may choose any machine of the production system. However, the workpiece should choose only a machine that provides at least one of the operations that the workpiece requires next, and the workpiece should prefer the machine with the maximal set of operations that can be applied to the workpiece in its current state. As an abstract decision task, D_5 does not have a control interface.

Attribute	Description
id	D_5
title	Choose next machine
parameters	workpiece W_i
control interface	
trigger	W_i just left the loader or a machine.
decision space	set of machines M_j
local decision rule	Choose a machine with the maximal set of operations that can be applied to the workpiece in its current state.

Table 4.13: Abstract decision task D_5
of the example production system.

The result of the abstract decision task D_5 is communicated to the decision tasks of D_2. As a consequence, these may simplify their local decision rule in that they simply route the workpiece in the direction of the chosen machine. Likewise, in order to recognise the decision situation that triggers the decision task, a signal is sent by the decision tasks D_1 and D_4 to D_5 whenever the workpiece either leaves the loader or the machine at which it was processed last.

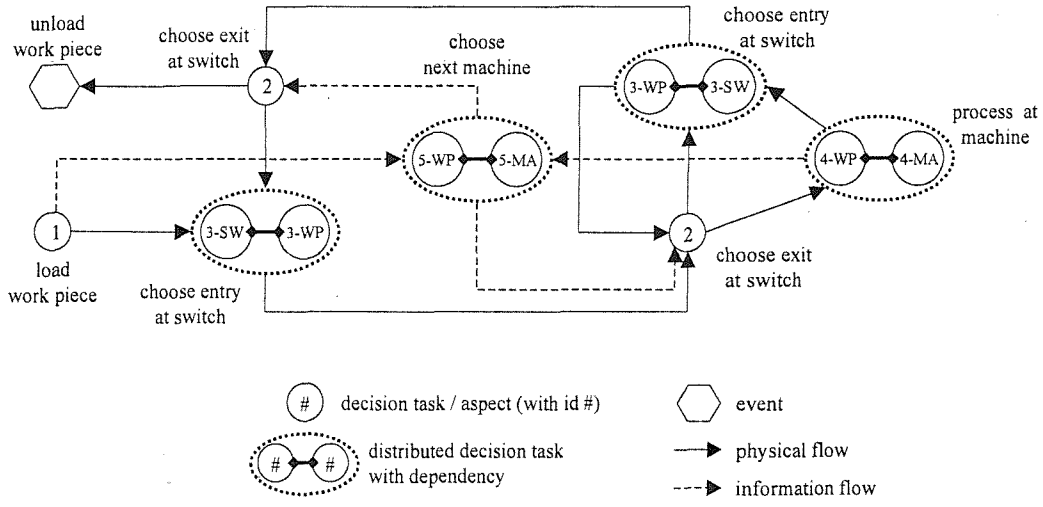

Figure 4.15: Trigger diagram after introducing
the abstract decision *Choose next machine* in the example production system.

Figure 4.15 shows the new trigger diagram. Since the decision task D_5 involves several production components, namely the workpiece and all machines, the decision task has already been distributed into the two decision aspects $D_{5\text{-WP}}$ and $D_{5\text{-MA}}$.

The dependencies are adapted accordingly (see figure 4.16). Dependency DP_1 is made obsolete because of the introduction of decision task D_5. Dependency DP_2 now holds between different instances of decision task D_5 (for the same workpiece) because these decision tasks are now responsible for choosing the stations to be visited. For the same reason, dependency DP_3 is extended to also include decision task D_5. In contrast to dependency DP_2, though, DP_3 still includes the decision task D_2 because the actual routing of a workpiece is also relevant to the work-in-process of a production system.

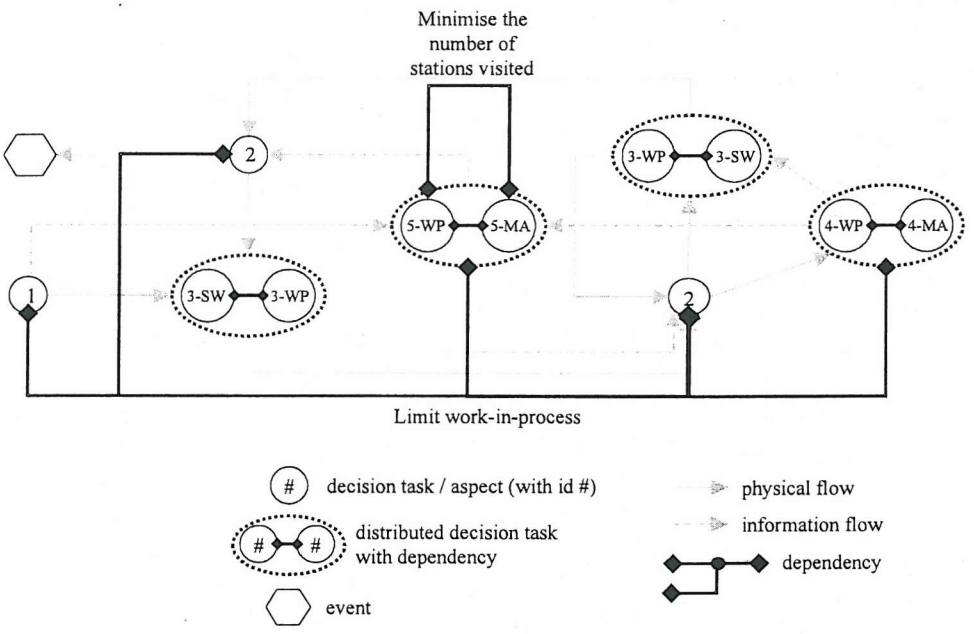

Figure 4.16: Dependency diagram after introducing the abstract decision *Choose next machine* in the example production system.

With the above changes, the decision model of the example production system no longer collapses into a single agent during the clustering process. The clustering rule for interface cohesion clusters all instances of decision tasks D_2 and $D_{3\text{-}SW}$ referring to the same switch, and all instances of decision tasks $D_{4\text{-}MA}$ referring to the same machine. The clustering rule for state cohesion, on the other hand, clusters only the instances of the decision tasks $D_{3\text{-}WP}$, $D_{4\text{-}WP}$, and $D_{5\text{-}WP}$ which are referring to the same workpiece. The resulting set of agents for the example production system is shown in table 4.14.

Agent name	Control interface	Decision tasks
machine agent	machine M_i	choose next machine ($D_{5\text{-}MA}$); process at machine ($D_{4\text{-}MA}$)
switch agent	switch S_i	choose exit at switch (D_2); choose entry at switch ($D_{3\text{-}SW}$)
loader agent	loader	load workpiece (D_1)
workpiece agent	workpiece W_i	choose next machine ($D_{5\text{-}WP}$); process at machine ($D_{4\text{-}WP}$); choose entry at switch ($D_{3\text{-}WP}$)

Table 4.14: Agents identified for the example production system (based on figure 4.15).

4.3.3 Summary of agent identification step

The agent identification step consists of two design steps – the clustering process and the improvement of the decision model – which are repeated until either a suitable set of agents has been found, or the agent-based approach is abandoned (see figure 4.9). The clustering process tries to assign decision tasks to agents through clustering decision tasks according to a set of clustering rules. The clustering process fails whenever all decision tasks are assigned to the same agent – the control system then consists of only one agent. In such a case, the decision model must be reorganised in order to avoid its collapse. For this, the improvement step provides a set of operations that reorganise the decision model without changing its control functionality. After the reorganisation of the decision model, the clustering process is repeated with the improved decision model. In case the decision model collapses again into a single agent, the agent identification loop of clustering and improving the decision model may be repeated until either a set of agents has been found, or it becomes clear that it is impossible to avoid the collapse of the decision model. In the latter case, the control system should consist of only one agent (rather than a collection). In such a case, the techniques provided by multi-agent systems research, in particular the large set of interaction protocols, cannot be used and an agent-based approach should therefore be abandoned. In such situations, the designer should look for other techniques that enable him to develop a (central) control system responsible for all the decision tasks identified.

If the clustering process is successful, a set of agents has been identified and each agent is associated with a set of decision tasks. The agents are solely responsible for the execution of their decision tasks, but depend on other agents whenever these decision tasks share dependencies with decision tasks which were assigned to different agents. To resolve these dependencies, the agents must interact before making their decision.

How the agents should interact is determined in the next section.

4.4 Selection of interaction protocols

The previous design step has identified the control agents necessary to solve the production control problem and has assigned a set of decision tasks to each agent for which the agent is solely responsible. Some of the decision tasks, though, cannot be executed in isolation – they are dependent on other decision tasks. For those dependencies which involve decision tasks assigned to different agents, the corresponding agents must interact in order to find decision alternatives which not only satisfy the local, but also the non-local constraints and preferences.

An interaction between a group of agents consists of a set of messages these agents exchange during the course of the interaction. Interactions are usually specified (and programmed) in the form of interaction protocols. Interaction protocols fix the types of messages that may be exchanged and restrict the possible sequences of messages that are allowed in a protocol-conformant interaction (Burmeister 1995). To complete the overall design of the agent-based control system, the designer of the production control system must specify an interaction protocol for each inter-agent dependency such that the decisions chosen at the end of the interaction resolve the corresponding dependency.

A wide variety of different interaction protocols have been proposed in the multi-agent systems literature and more are continually being developed. Given this rich set of interaction techniques, the designer should first try to reuse the existing protocols before designing a new interaction protocol from scratch (which can be a very challenging task, sometimes worth a PhD on its own). Reusing existing techniques has the clear advantage that it reduces the time and effort to perform the design task as well as the risk of failure because the designer can rely on proven techniques (see section 3.6). Obviously, the designer may still develop a new interaction protocol if all existing techniques turn out to be unsuitable.

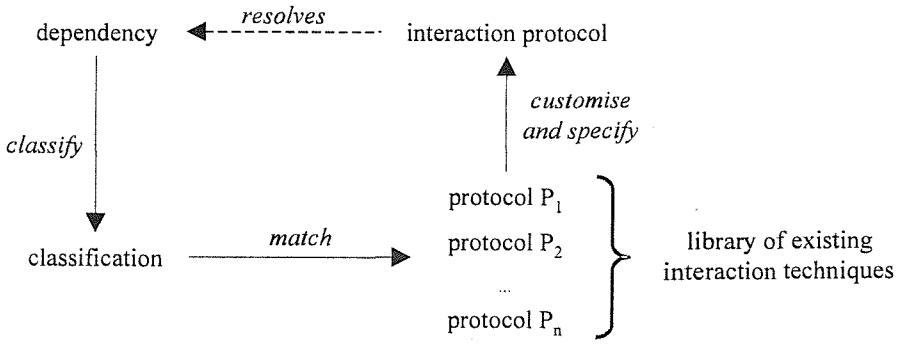

Figure 4.17: The process for selecting interaction protocols.

To reuse existing interaction protocols, there must be a design method that enables the designer to select a suitable protocol given the description of a decision dependency. Such a design method must provide a set of criteria such that the interaction protocol which matches a dependency best – according to these criteria – is also the best interaction protocol to resolve the dependency. Given such a set of criteria, the designer only needs to classify a dependency according to these criteria and then search through a library of existing interaction techniques to find the interaction protocol that matches the classification best (see figure 4.17). In case this library is computer-based, the search process may even be done automatically.

The proposed process for selecting interaction protocols is a heuristic classification (Clancey 1985) because the selection mechanism is based on an abstract description of the interaction situation and the existing protocols (see also the facet approach in subsection 3.6.1). Such an abstraction is necessary if there is no direct matching between problem and solution (see (Clancey 1985) for a discussion). This direct link does not exist because a dependency may be solved by several different interaction protocols.

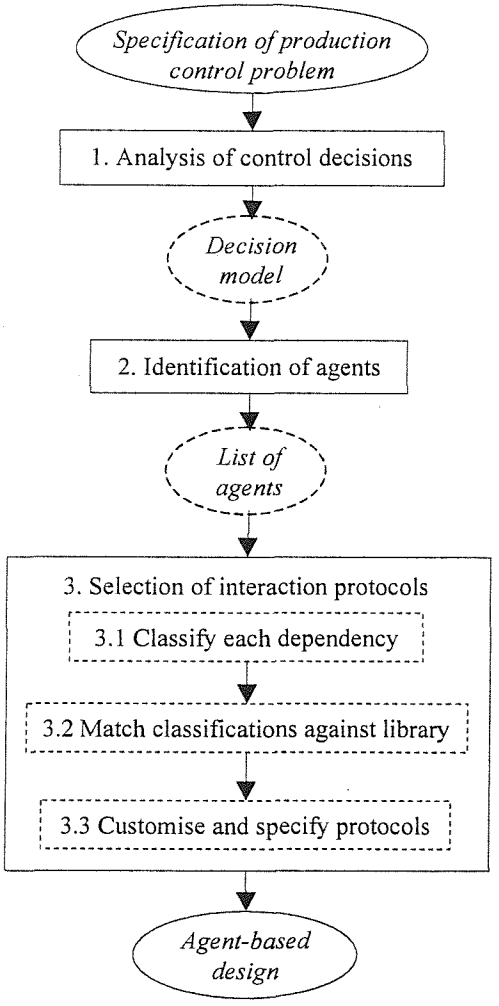

Figure 4.18: Steps for the selection of interaction protocols.

To select an interaction protocol for a given dependency, the designer must thus perform three steps (see figure 4.18). The first step is to classify the dependency according to a pre-defined set of criteria. This pre-defined set of criteria will be called the ***classification scheme*** in the following. The second step of the selection process is to match the classification of the dependency against a library of existing interaction protocols. A ***matching procedure*** specifies how the matching is performed and how, based on the results of the matching, the interaction protocol best suited to resolve the dependency is identified. To make such a matching possible, the existing interaction protocols must be classified according to the same criteria as the dependency. This process, which needs to be done only once for each interaction protocol, will be called ***protocol characterisation*** in the following. Once a suitable interaction protocol has been identified, the last step of the selection process is to specify the selected interaction protocol in terms of the application and, if necessary, to adapt it to the specific requirements of the dependency situation. This final step will be referred to as the ***protocol customisation***.

The following subsections describe each aspect of the selection process in detail. The first subsection develops the classification scheme for dependencies. Subsection 4.4.2 discusses how existing interaction protocols must be characterised in order to match the classification scheme and gives some example characterisations of existing interaction protocols. Subsection 4.4.3 then presents the procedure for matching dependencies to existing protocols, and explains the extent to which a chosen protocol can be customised to fit an actual dependency situation. Finally, the last section summarises the selection process.

4.4.1 Classification scheme

The classification scheme is intended to classify dependencies such that the interaction protocol which matches the classification of a dependency best is also the best interaction protocol to resolve the dependency. To achieve this aim, the classification scheme must consist of classification criteria that put dependencies into different classes if they require different (kinds of) interaction protocols. To identify such a set of criteria, it is necessary to look at the requirements a dependency may impose on the interaction process, and collect those aspects which differentiate dependencies most with respect to the required interaction process. This is the objective of this subsection.

A dependency consists of a set of decision tasks and a set of non-local constraints and preferences these decision tasks must fulfil (see subsection 4.2.2.2). Each decision task specifies a set of possible ***start situations*** in which the decision problem arises; and the decision tasks in combination with the non-local constraints and preferences specify what ***goal state*** must be achieved in the end. Any interaction protocol intended to resolve the dependency must be able to reach the goal state from any possible start situation. Start situations and goal state of a dependency thus delineate the functionality of the required interaction protocol. Both start situations and goal state are therefore analysed below in order to identify classification criteria distinguishing interaction protocols (see subsections 4.4.1.1 and 4.4.1.2). In principle, the interaction protocol may choose any approach to resolve a dependency as long as it is guaranteed that the goal state is reached. However, the context of a dependency may impose restrictions on the way the dependency can be resolved. Since these restrictions may rule out some interaction protocols, it is worthwhile considering what kind of restrictions might be present. This will be done in subsection 4.4.1.3.

4.4.1.1 Start situations

The start situations of a single decision task are defined by the set of situations at the associated production components that trigger the decision task, and the actual decision problem that needs to be solved in these situations (see subsection 4.2). For a

dependency, the start situations are the different combinations of the start situations of the decision tasks involved in the dependency. The start situations of a dependency therefore consist of the following aspects:

(i) all combinations of situations at the production components that trigger the dependency

(ii) the decision tasks to be performed in these situations, i.e., the decision tasks involved in the dependency

(iii) the non-local constraints and preferences describing the actual dependency between the decision tasks

Not all of these aspects, though, are relevant to the classification of a dependency. In particular, the situations that trigger the dependency only describe when the interaction becomes necessary. The actual problem to be solved by the interaction is specified by the decision tasks and the non-local constraints and preferences. The following analysis of the start situation will therefore focus on the decision tasks involved in the dependency and the non-local constraints and preferences between these decision tasks.

Decision tasks involved in the dependency

The first relevant criterion for the selection of a suitable interaction protocol is certainly the number of decision tasks that need to be co-ordinated. Is there, for instance, a small and fixed number of decision tasks that need to interact? Or does the set of decision tasks change over time? The number of decision tasks to be co-ordinated, however, is not identical to the number of decision tasks involved in the dependency because some decision tasks may be assigned to the same agent. Any dependencies between decision tasks of the same agent can be resolved internally by the agent and do not require any interaction. The first relevant criterion for selecting interaction protocols is therefore the number of agents involved in the dependency.

Criterion #1: ***Number of agents involved***

How many agents are involved in the dependency right from the start? May other agents join later?

The possible answers to the above questions are classified according to the requirements they impose on the required interaction process.

<n> The number of agents involved in the dependency is fixed and already known at design time. This case is indicated by the actual number of agents involved (for example, two for a workpiece and a machine agent negotiating the set of operations to be applied to the workpiece).

fixed The number of agents involved in the dependency is fixed, but not known at design time. That is, the number of agents is only known once the interaction starts, but does not change afterwards.

changing The number of agents involved may change during the interaction, i.e., agents may join the interaction process after it has been started. Agents may join later, for example, because they have been introduced to the control system after the beginning of the interaction.

The above classes represent an increasing set of requirements on the interaction process. If the number of agents involved in the dependency is fixed and already known at design time, it is sufficient to use an interaction protocol that is only able to deal with the number of agents indicated. Some interaction protocols, for instance, are only able to co-ordinate two agents. If the number of agents involved is fixed, but not known at design time, the interaction protocol chosen must be able to deal with a fixed, but arbitrary number of agents. Finally, in case the number of agents is not fixed, the protocol must additionally be able to integrate new agents into the interaction process after it has been initiated. Obviously, such an interaction protocol should also be able to handle a fixed set of agents if no agents are introduced during the interaction.

Another possible criterion for selecting a suitable interaction protocol could be the types of decision tasks that are involved in the dependency. Such a criterion would be useful if different types of decision tasks required different kinds of interaction protocols. The majority of interaction protocols, however, do not make any assumptions about the actual decision tasks to be co-ordinated. They are applicable to almost any kind of decision task (obviously, not to any kind of non-local constraints and preferences, as will be shown below). Consequently, such a criterion would hardly distinguish any interaction protocols and is therefore of low value to the selection of interaction protocols.

Non-local constraints and preferences

The other important aspect of the start situation is how the decision tasks involved in the dependency are related to each other. Each agent has its local decision tasks, but is not able to execute them alone because of the non-local constraints and preferences that restrict the local decision making. As a consequence, the agents need to interact. The nature of the restrictions on the local decision making, however, has an influence on the kind of interaction required to deal with these restrictions. Agents that have completely opposing interests will have to interact more than agents that just want to avoid some damaging effects. The second relevant criterion for the selection of a suitable interaction protocol is therefore the relation of local and non-local constraints and

preferences.

Criterion #2: ***Compatibility of constraints and preferences***

How compatible are the local and non-local constraints and preferences involved in a dependency?

The compatibility is classified according to the kinds of restrictions that create the dependency:

only constraints

There are only constraints. These constraints – by definition – only rule out certain combinations of decision alternatives. Any combination of decision alternatives that is not ruled out is a solution resolving the dependency. However, there may exist no solution satisfying all constraints.

compatible preferences

There exists at least one (local or non-local) preference function on the outcome of the interaction (and possibly additional constraints). In case of more than one preference function, there are solutions that are to the mutual benefit of all agents, i.e., there exists a solution such that all preference functions are maximally satisfied.

opposing preferences

There are at least two agents that have preferences on the outcome of the interaction and these preferences are opposing, i.e., any combination of decision alternatives that is better for one agent is worse for the other. (Constraints may or may not be present.)

The above classes also subsume each other. First of all, an interaction protocol able to reconcile opposing interests can also find a solution for compatible preferences. And, secondly, both cases for preferences also allow constraints to be present. Obviously, this does not imply that an interaction protocol for opposing preferences is equally efficient for solving a dependency situation consisting purely of constraints.

Another important aspect of the constraints and preferences linking the decision tasks is to what extent these constraints and preferences are global, i.e., encompass all decision tasks of a dependency. By definition, the non-local constraints and preferences involve at least two decision tasks. However, if there are more than two agents, the non-local constraints and preferences may involve all agents, and thus be global, or only link subsets of the agents. This distinction is particularly relevant if there are many agents. In such a case, it may be far easier to co-ordinate small subsets of these agents than to

make sure that all agents satisfy a global constraint or maximise a global preference function. Therefore, the start situation is also classified according to the existence of global constraints and preferences.

Criterion #3: ***Global constraints and preferences***

If there are more than two agents, does there exist a global constraint or preference that involves all agents?

Cases in which there are more than two agents and a global constraint or preference exists are indicated by ***global***, all other cases are marked as ***non-local***.

The above classes are obviously mutually exclusive.

Table 4.15 summarises the possible classifications of a start situation. If a dependency arises in start situations with different classifications, the designer must choose the worst classification, i.e., the classification that puts the most requirements on the interaction protocol. In case the classifications of the different start situations are not comparable, the chosen interaction protocol must either satisfy all classifications or different interaction protocols must be chosen for each start situation.

	Classification criteria	Possible properties
#1	**Number of agents involved**	*<n>*
		fixed
		changing
#2	**Compatibility of constraints and preferences**	*only constraints*
		compatible preferences
		opposing preferences
#3	**Global constraints and preferences**	*global*
		non-local

Table 4.15: Classification of a start situation.

4.4.1.2 Goal state

To resolve a dependency, the agents involved in the dependency need to choose an action for each decision task such that the local and non-local constraints and preferences are satisfied in the best way possible. The goal state of a dependency is thus specified by a list of actions – one for each decision task. At least something about this goal state must be initially unknown in order to represent a decision problem. Thus, it will either be unclear which actions are to be taken by each agent or, if the decision spaces include the null action, which agents will be taking an action at all. If nothing is

unclear, the agents do not have a decision task. Consequently, an interaction protocol has to answer at least one of the following questions in order to reach the goal state:

1. Which actions should be executed?
2. Which agent should commit to which action(s)?

How much is unknown about the goal state is certainly relevant to the selection of an interaction protocol. The more is unknown about the goal state, the more the interaction protocol must determine during the interaction process. The extent to which the above questions are unanswered at the beginning of the interaction is therefore a good candidate for distinguishing interaction situations, and thus to select a suitable interaction protocol. The first question – which action should be executed –will however be unanswered in most cases, and will therefore hardly distinguish interaction situations. On the other hand, the second question – which agent should commit to an action – may or may not be clear at the beginning. The second question is thus not common to all interaction situations and may consequently be used to distinguish dependencies with respect to the requirements they impose on the interaction protocol.[19] This will be done in subsection *role variability*.

Another important aspect of the goal state is how the actual decisions made relate to each other. Obviously, there must be some relations between the decisions, because otherwise there is no need to interact. Furthermore, it is the main purpose of the interaction process to handle decision relations because otherwise the agents might as well make their decisions independently. The decision relations to be handled during the interaction are induced by the non-local constraints and preferences, and the actual decisions made must somehow satisfy these constraints and preferences. To find a set of decisions satisfying these relations, the agents consequently interact. However, it is not sufficient to identify the appropriate decisions, the agents must also execute the decisions agreed, i.e., they must commit themselves to the corresponding actions. Not every agent though may have to commit itself in front of everybody else. Maybe some agents form a subgroup that is independent in their execution of the rest of the agents involved in the dependency. The number and size of the required joint commitments, though, is relevant to the selection of a suitable interaction protocol. Bilateral joint commitments are easier to achieve than a joint commitment encompassing all agents. The required joint commitments are therefore analysed first (in subsection *joint commitments*).

Joint commitments

In the context of this work, a set of commitments is called a ***joint commitment*** if the

[19] The case that the action and the agent performing it is known does not represent a decision problem at all. The case that the action is known, but the agent performing it is not known, is covered by the question which agent should commit to an action.

failure to fulfil one of the commitments jeopardises the success of the other commitments (cf. (Jennings 1996)). That is, the set of commitments only makes sense if all commitments are fulfilled. If one agent de-commits, all other agents should de-commit, too.

Example. In the simple production system, several workpieces enter the production system and need to be processed. Each workpiece (agent) chooses a machine for its processing and engages in a joint commitment with that machine. If either the machine or the workpiece de-commits, there is no point for the other agents to maintain their commitment any longer. However, if a machine for example breaks down and de-commits from all its engagements, the commitments of the other machines are not affected. They can still process the workpieces assigned to them.

Formally, joint commitments are represented by subsets of the agents involved in a dependency. If one agent of such a subset de-commits, all other agents in this subset should also de-commit. The joint commitments required by a dependency may thus have quite diverse structures: any subset of the agents is theoretically a possible joint commitment. However, to make a comparison of joint commitments feasible and efficient, the classification of the required joint commitments is reduced to three criteria: the number of (independent) joint commitments, the size of the commitments, and how the commitments relate to each other. Just as for most of the other classification criteria, the above properties are either known at design time, at the beginning of the interaction, or must be determined by the interaction protocol.

Criterion #4: ***Number of joint commitments***

Is the number of required joint commitments already known at the beginning of the interaction, or must it be determined by the interaction protocol?

The possible answers to the above question are indicated as follows:

<n>	In the special case that the number of joint commitments is already known at design time, the actual number of joint commitments may be used to indicate this case (for example, one for a dependency involving only two agents committing to perform a joint action).
fixed	The number of required joint commitments is known at the beginning of the interaction.
variable	The number of required joint commitments must be determined by the interaction protocol.

Criterion #5: ***Size of joint commitments***

How many agents are involved in a joint commitment? Do all joint commitments have the same size (i.e., the same number of agents involved)?

The possible answers to the above question are indicated as follows:

<n>	All joint commitments have the same size and this size is already known at design time. In this case, the actual size of the joint commitments is used to indicate this case (for example, two for a workpiece and a machine agent agreeing to process the workpiece at the machine).
fixed	All joint commitments have the same size, but the size is only known at the beginning of the interaction.
variable	The size of each joint commitment must be determined by the interaction protocol.

Criterion #6: ***Relation of commitments***

How do the commitments relate to each other and to the whole set of agents? Do commitments overlap, i.e., may an agent be involved in more than one joint commitment? Do commitments completely cover the set of agents, i.e., is every agent engaged in at least one joint commitment?

There are four possible answers to the above questions (only the first and the last two exclude each other):

overlapping	An agent may be involved in more than one joint commitment.
non-overlapping	An agent may be involved in at most one joint commitment.
complete coverage	Each agent must be involved in at least one joint commitment.
incomplete coverage	Not every agent must be involved in a joint commitment.

Criteria #4 and #5 obviously indicate how many joint commitments must be created by the interaction process, and thus characterises the different requirements that may be imposed on the interaction process. Criteria #6 is also relevant to the selection of a suitable interaction protocol because it characterises global properties of the joint commitment set. When matching agents, for instance, non-overlapping joint commitments are more difficult to achieve than overlapping commitments because

agents may be left unmatched if the wrong agents are assigned during the interaction process. Likewise, engaging each agent in a joint commitment is more difficult if some agents are difficult to match. Criteria #6 thus provides additional information about the joint commitment set.

The classification criteria for the joint commitments are summarised in table 4.16.

	Classification criteria	Possible properties
#4	**Number of joint commitments**	*<n>* *fixed* *variable*
#5	**Size of joint commitments**	*<n>* *fixed* *variable*
#6	**Relation of joint commitments**	*overlapping / non-overlapping* *complete / incomplete coverage*

Table 4.16: Classification of the required joint commitments.

Role variability

The goal state is described by a set of agent-action pairs, specifying which agent is executing which action. As discussed above, it may be unclear which of the agents available in the interaction situation will actually perform an action, and thus will be a member of one of the agent-action pairs. To capture this potential uncertainty, the goal state is characterised with the aid of roles (see subsection 3.5.3). A role describes a specific behaviour without specifying which agent will actually perform this behaviour.[20] In this view, the goal state thus consists of a set of roles, each specifying an action, and one task of the interaction protocol – apart from identifying these actions – is to assign these roles to agents. To classify this assignment problem for a given dependency, it is necessary to identify which roles are already assigned to agents at the beginning of the interaction and which must be assigned during the interaction process.

Criterion #7: *Role assignment*

Is an agent role already assigned to an agent, or must the role assignment be determined by the interaction protocol?

For each role, there are two possible answers:

fixed The role is already assigned to an agent at the beginning of the interaction.

[20] There is more to the concept of roles, but in this context this simple view of a role is sufficient.

variable	The role must be assigned to an agent by the interaction protocol.

Since each role is either fixed or variable, the classification of the agent roles can be summarised by stating how many roles are variable (all others must then be fixed). Three cases are distinguished:

none	None of the agent roles are variable.
subset	A subset of the agent roles is variable.
<n>	In case the number of variable agent roles is already known at design time, the number itself can be used to indicate this case.
all	All agent roles are variable.

Table 4.17 summarises the possible classifications of a dependency according to the role variability. Alternatively, the classification could also state the number of fixed agent roles, or compare both.

	Classification criteria	**Possible properties**
#7	**Role assignment (variable roles)**	*none* *subset* *<n>* *all*

Table 4.17: Classification according to the agent roles of a dependency.

4.4.1.3 Process requirements

An interaction protocol solves a dependency if it is able to reach the goal state from any possible start situation. How the goal state is reached is generally left to the interaction protocol. The interaction situation, however, may impose restrictions on how this may be done. Potentially, the interaction situation may impose quite diverse restrictions on the interaction process, such as incompatible communication languages, hard real-time constraints, communication bandwidth restrictions, untruthful or even deceiving agents, and so on. This subsection, though, focuses on the two most common restrictions for technical (production) systems.

First of all, an agent may be unable or unwilling to provide certain information to other agents. In the case of multiple firms negotiating contracts for instance a firm is certainly unwilling to lay open all cost information about its processes to other agents. Secondly, agents need to search the decision space and choose a set of actions. This process may be simplified if at least part of the search process can be done locally by a single agent.

The delegation of the search process, however, would require that the agents also delegate part of their decision autonomy to other agents.[21] This may not be acceptable to any agent.

Criterion #8: ***Availability of information***

How much local information can be made available to other agents?

Is an agent able and willing to communicate its decision alternatives, the local constraints and preferences, or other information about its local situation? In one extreme, an agent provides everything it knows to the other agents. In the other extreme, the agent makes public only some of the actions contained in the decision space.[22]

The rationale behind this question is that the more information is made available, the easier it may be to resolve the dependency. The following two main levels of information availability are distinguished.

alternatives only	Propose and discuss only decision alternatives. The decision space may or may not be made available. Constraints and preferences are not disclosed.
preferences	Lay open (immediately or successively) local constraints and preferences.

The second level of information availability – preferences – makes it easier to find a solution to the decision problem. The agents may then estimate the valuation of an agent before making a proposal. The second level, however, also discloses (possibly) private information which may not be acceptable in every situation, as for example in a negotiation setting.

Criterion #9: ***Delegation of decision autonomy***

How far can decision autonomy be delegated?

Does an agent accept a decision if other agents believe it to be the best decision for all? Under what circumstances does an agent accept a group decision? Or must an agent explicitly accept any decision proposed? In case the agent accepts a decision under certain conditions, it would be sufficient to inform the agent that these conditions hold for the final solution found. In case the agent did not delegate any decision autonomy, there must be at least a final commitment round in which each agent is asked to accept the proposed action. In particular, if one agent rejects the proposed action (for whatever reason), another solution must be

[21] The agent searching the decision determines the order in which possible choices will be considered during the decision process and consequently has a strong influence on the decision process.
[22] Note that this is the other extreme because if each agent does not publish any alternatives it has, there is nothing to co-ordinate.

found.

There are basically two levels of delegation with respect to decision autonomy. The levels describe an increasing loss of autonomy.

no delegation	Retain the right to reject any proposal.
partial delegation	Accept a proposal if it meets certain conditions required by the agent. These conditions may concern the properties of the solution or the process for determining the solution, i.e., the interaction protocol. In general, the conditions may be of quite different nature.

This criterion is related to criterion #8. It is only possible to delegate decision autonomy, if the agent also makes some of its local information available. If an agent does not announce any of its decision alternatives, it is impossible for another agent to choose an alternative for this agent. However, an agent could communicate nearly all of its information to the other agents without delegating any decision autonomy. Both criteria are therefore not identical.

Table 4.18 summarises the possible classifications of a dependency according to the process requirements.

	Classification criteria	**Possible properties**
#8	**Availability of information**	*alternatives only* *preferences*
#9	**Delegation of decision competence**	*no delegation* *partial delegation*

Table 4.18: Classification according to the process requirements.

4.4.1.4 Summary

This section has identified nine classification criteria that characterise decision dependencies with respect to the interaction process they require. These criteria define 2304 possible classifications – namely the product of the possible classifications for each criterion.[23] Due to the diverse criteria used and the large number of possible classes, the classification scheme is obviously able to distinguish a wide range of different dependency situations. This becomes obvious when classifying existing interaction protocols in the next subsection: most interaction protocols fall into

[23] Actually, the number of classifications is indefinite because criteria #1, #4, and #5 allow the selection of a natural number as a classification.

different classes.

Note, however, that the classification scheme is not complete in that there may exist additional criteria which could distinguish dependencies (and interaction protocols) further. However, the classification scheme was deliberately restricted to above criteria for the following reasons. First of all, the number of criteria (and also the number of possible attributes) should be limited. Too many criteria, or significantly more attributes, would increase the effort the designer would have to go through when classifying a dependency and would thus decrease the acceptance of the methodology. Second, the objective of the classification scheme is to reduce, for each dependency, the large set of existing interaction protocols to a much smaller set of potentially applicable protocols, rather than to reduce the set of protocols to the one and only protocol applicable in a dependency situation. It is questionable whether a classification scheme with single class instances exists because such a classification scheme would have to anticipate any new protocol developments in the future. Consequently, it seems reasonable to create a classification scheme that is efficient and extensible to future developments.

In this context, the classification scheme is extensible in two ways. First of all, the list of classification criteria can be extended by adding new criteria, such as the possibility of deception or the necessity of mobility for the agents. Secondly, a single class can be refined by a specialised set of classification criteria if a classification returns too many interaction protocols. Assume, for example, a classification returns a large set of negotiation protocols, then a sub-classification similar to the taxonomy of Bartolini et al. can be used to discriminate the negotiation protocols (see subsection 3.6.2). For any sub-classification, however, it should verified whether the classification can be done solely on the basis of information about the dependency, or whether the designer has to evaluate the interaction mechanisms in order to perform the final selection. In the latter case, the classification process can no longer be done by only analysing the dependency situation.

4.4.2 Characterising interaction protocols

The previous subsection has presented a classification scheme that characterises a decision dependency according to the interaction process that is required to resolve the dependency. In order to efficiently identify an existing interaction protocol that is able to create and control the required interaction process, it is necessary to characterise the existing interaction protocols according to the same criteria which were used to classify the decision dependency. Once such a characterisation of the interaction protocols is given (and it needs to be done only once), the most suitable interaction protocol can be identified by matching the classification of the dependency against the characterisations of existing interaction protocols. Given a computer-based library of existing interaction

protocols, this match might even be done automatically, i.e., the designer just enters the classification of a dependency and the computer presents the set of interaction protocols that match the classification best.

A first step towards this library is done in appendix A by characterising a selected set of existing interaction protocols. This set of interaction protocols is not supposed to be either a complete or representative set of existing interaction protocols. After more than twenty years of research into multi-agent systems, even the discussion of a representative set of interaction protocols would be beyond the scope of this work. A complete, and thus closed library is also not appropriate because new interaction protocols are continuously developed. The library should therefore be treated as an open repository of knowledge into which new interaction protocols can be easily incorporated. The interaction protocols discussed in appendix A have thus only been chosen in order to provide a wide range of examples with quite different characterisations.

Table 4.19 summarises the characterisations of some interaction protocols discussed in appendix A. The table shows that different interaction protocols are characterised quite differently if they address different interaction situations. The classification scheme is therefore able to distinguish different interaction protocols and thus reduces the number of interaction protocols a designer has to look at.

	Plurality voting	**English auction**	**Continuous double auction**	**DCS-ABS**	**Coalition formation**	**PGP**
#1	fixed	fixed	changing	fixed	fixed	changing
#2	opposing	opposing	opposing	constraints	compatible	compatible
#3	non-local	non-local	non-local	global	non-local	global
#4	1	1	variable	variable	variable	variable
#5	fixed	2	2	variable	variable	variable
#6	-	-	non-overlapping incomplete	any	overlapping incomplete	any
#7	all variable	1/1	all variable	all variable	all variable	all fixed
#8	alternatives	alternatives	alternatives	alternatives	alternatives	preferences
#9	partial	partial	no	partial	partial	partial

Table 4.19: Example characterisations.

4.4.3 Matching and protocol customisation

The two preceding subsections have presented the classification scheme for dependencies and the characterisation of some example interaction protocols according

to this classification scheme. This subsection now describes how the designer should actually use this classification scheme in order to identify the most suitable interaction protocols for his application. This is done in subsection 4.4.3.1. This subsection also discusses possible adaptations of the protocols if the protocols identified do not exactly match the dependency situations (see subsection 4.4.3.2).

4.4.3.1 Matching dependencies with interaction protocols

To identify the most suitable interaction protocols for a given decision dependency, the designer should go through the following steps:

1. Collect all decision tasks involved in a dependency.
2. Identify all possible start situations in which this dependency may arise.
3. Perform the classification of the dependency according to the classification scheme described in subsection 4.4.1.
4. Given a library of characterised interaction protocols, search for the interaction protocol that best matches the classification of the dependency. An interaction protocol matches a classification best if the characterisation of the interaction protocol has the most properties in common with the classification.[24]
5. For each protocol identified, verify whether it is able to reach the goal state from all possible start situations. If this is not the case for a protocol, try to modify the protocol accordingly (see subsection 4.4.3.2).
6. From the set of interaction protocols that effectively resolve the dependency, choose the protocol that is best from the point of view of the application. Specify the (possibly adapted) interaction protocol (see for example (Burmeister 1993, Parunak 1996, Odell 2001)).

If all six steps of the above method are successfully completed, the designer has found an interaction protocol that resolves the dependency in all possible start situations. If the designer is able to do so for all dependencies, then the system design for the agent-based control system is completed (cf. subsection 4.1).

Example. In the simple production system, there is a strong dependency between the decision aspects $D_{5\text{-}WP}$ and $D_{5\text{-}MA}$ in that these decision tasks must agree on a machine that is supposed to process the workpiece next (see figure 4.16). In the following, it is shortly discussed how an appropriate interaction protocol is found for this dependency (more examples will be discussed in appendix B).

[24] Note that there may be several interaction protocols that have a maximum number of properties in common with a specific classification.

1. The dependency involves the different decision aspects of the decision task *choose next machine*. Affected by the dependency are thus the workpiece agent ($D_{5\text{-}WP}$) and several machine agents ($D_{5\text{-}MA}$).
2. The start situation of the dependency is either that the workpiece has been loaded or that it has just left a machine.
3. The classification of the dependency is as follows (see table 4.20). The number of agents is fixed (the workpiece agent and several machine agents which could possibly process the workpiece in its current state). The workpiece agent is looking for a single commitment with a (single) machine agent such that the machine agent processes the workpiece. The role of the workpiece agent is fixed, the role of the machine agent engaging in the processing is not.

   The preferences of the workpiece and the machine agents are compatible. The former tries to be processed as fast as possible, the latter tries to maximise its utilisation. Whenever in conflict, the preference of the workpiece has priority.[25] It is further assumed that the agents would be willing to make their preferences available and to delegate decision autonomy, since both agents are supposed to optimise the overall production performance (see section 4.1). Finally, there are no global constraints and preferences to be taken into account (for this dependency).

	Classification criteria	Classification
#1	Number of agents involved	**fixed**
#2	Compatibility of preferences	**compatible**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**2**
#6	Relation of commitments	**-**
#7	Role assignment	**1/1**
#8	Availability of information	**preferences**
#9	Delegation of decision autonomy	**partial**

Table 4.20: Classsification of dependency between the decision aspects $D_{5\text{-}WP}$ and $D_{5\text{-}MA}$.

4. Of the interaction protocols characterised in appendix A, the plurality

[25] If the two preferences were not prioritised, the preferences of the agents would be (potentially) opposing.

voting, the English auction, and the contract net protocol match the above classification (i.e., could possibly resolve the dependency). The contract net protocol, though, matches the classification best because it has only one property (criterion #8) which differs from the above classification (but which nevertheless subsumes the above). A short analysis also shows that the contract net protocol is sufficient to resolve the dependency. It is therefore not necessary to consider the other interaction protocols.

5. The contract net protocol reaches the goal state (choosing the next machine) from the start situation. Consequently, there is no need to modify it.

In general, however, the above method may fail to identify a suitable interaction protocol for a dependency. There may be two reasons for this:

1. It is not possible to resolve the dependency without resolving simultaneously other dependencies the decision tasks are involved in.

   Example. Assume that the workpieces in the example production system have deadlines. With deadlines for workpieces, dependencies between instances of $D_{5\text{-}WP}$ and $D_{5\text{-}MA}$ can no longer be resolved independently (see figure 4.16). If a single workpiece chooses the next machine without co-ordinating itself with the other workpieces, it may block a time slot that is required by another workpiece in order to finish before the deadline. Instead the workpieces should allocate time slots at the machines and schedule these slots such that the average deadline violation is minimised. Consequently, the dependencies between all instances of $D_{5\text{-}WP}$ and $D_{5\text{-}MA}$ must be resolved through a joint scheduling mechanism.

   In such a case, the above method must be repeated with an enlarged scope. That is, in step 1 of the method all decision tasks involved in the set of (potentially) linked dependencies are collected.

2. It is possible to resolve the dependency, but there is no suitable interaction protocol in the library available to the designer. In this case, a new interaction protocol must be designed (or the decision model must be changed).

4.4.3.2 Customising interaction protocols

For each interaction protocol that matches the dependency classification, it must be verified whether this protocol is able to reach the goal state from all possible start situations. An interaction protocol may fail to do so either because it is not applicable to

one of the start situations, or because it does not reach the desired goal state. In the latter case, the designer must either redesign the protocol or choose a different protocol. This case is not considered in the following discussion, as it is the goal of this work to provide a mechanism to re-use interaction protocols instead of designing new or re-designing existing protocols. In the former case, it may be possible – either at design or at run time – to transform the actual start situation into a situation to which the protocol can be applied. In the following, two basic techniques for adapting a protocol to a start situation are discussed.

In order to execute an interaction protocol, the agents participating in the interaction must be equipped with the right capabilities:

- Each agent supposed to initiate an interaction receives an appropriate trigger.
- Each agent has sufficient knowledge to process received messages, and to choose and compute the right response messages.

Each aspect will be discussed in the following. Note that at this stage, the methodology does not consider aspects like computational and communicational capabilities of the agents since these capabilities depend on the implementation.

Providing the right triggers

The description of each decision task specifies which trigger(s) this decision task receives from the production process. On the other hand, every agent supposed to initiate an interaction requires some kind of trigger telling it when to initiate the interaction. The requirements of an interaction protocol and the actual triggers available from the production process may differ in two respects.

First, the wrong agents may be triggered. In such a case, the designer must ensure that either an additional trigger is sent to the right agent (and the triggered agent ignores the trigger), or the triggered agent sends a message to the agent supposed to initiate the interaction.

Second, several agents receive a trigger and thus initiate the interaction in parallel. In case the protocol is not able to handle parallel initiations by itself, the designer must introduce additional steps at the beginning of the interaction to ensure that the actual interaction protocol is initiated only once (e.g., by performing a voting protocol at the beginning of the interaction).

Providing sufficient knowledge

To fulfil a role in an interaction, an agent must have sufficient knowledge to decide when to react to a message and how to react, i.e., which type of message to send and what to put in the content of the message. It can be safely assumed that the agent knows its decision space and its local constraints and preferences. However, many other

aspects may be unknown to the agent – these may include:

- Who are the other agents involved in this dependency, i.e., with whom should the agent interact?
- What are the non-local constraints and preferences? The knowledge about the non-local constraints and preferences must somehow be distributed among the agents.
- What is the knowledge of the other agents? Is their knowledge consistent with the agent's own knowledge?
- Is there really a dependency in this particular situation?

To remedy these information deficits the designer basically has three options:

- equip the agent with *more sensors*
  Sensors can help to provide more local data.
- equip the agent with interaction protocols for *requesting more information*[26]
  Communication with other agents provides the agent with non-local information available at the other agents.
- equip the agent with *more processing capabilities*
  The processing capabilities can be used, for example, to analyse history data, or to infer implicit knowledge.

From the point of view of the design methodology presented here, only the second approach is of relevance to the design because the other two aspects – more sensors and more processing capabilities – only concern the internals of an agent. This is because additional interaction protocols require that the agents from which the information will be requested are also equipped with the same interaction protocols. Their design must consequently be changed too.

To summarise, the designer is either able – with the help of the above methods – to adapt the selected interaction protocol to the specific needs of his application, or fails to do so and must therefore choose a different protocol.

4.4.4 Result of interaction protocol selection

The result of this design step is the specification of an interaction protocol for each dependency or subset of dependencies between different agents such that the interaction protocol resolves these dependencies in all situations. The protocol specification then only needs to be projected onto the different agents participating in the interaction in order to specify for each agent exactly how it should behave during the interaction.

[26] Many interaction protocols and mechanisms for requesting information have been developed in the context of information agents (see for instance (Kandzia 1997, Klusch 2001)).

With the list of agents and their decision responsibilities, the protocol specifications complete the design of the agent-based control system such that each agent can be implemented separately and the implemented agents solve the production control problem.

4.5 Summary

This chapter has presented the DACS methodology for designing agent-based production control systems that is appropriate for control design and sufficiently prescriptive for a control engineer with only minimal training and no prior experience in agent technology to design an agent-based production control system (cf. subsection 3.1.1). To ensure the appropriateness, the methodology was successively derived from a generic model of control processes in order to achieve a straight and comprehensible transition from domain to agent-oriented concepts during the design process. Second, to achieve a sufficient degree of prescriptiveness, each design step was accompanied with explicit design rules for the agent-oriented aspects of the design step. This included, in particular, a rule specifying when to abandon an agent-oriented design approach for a given application.

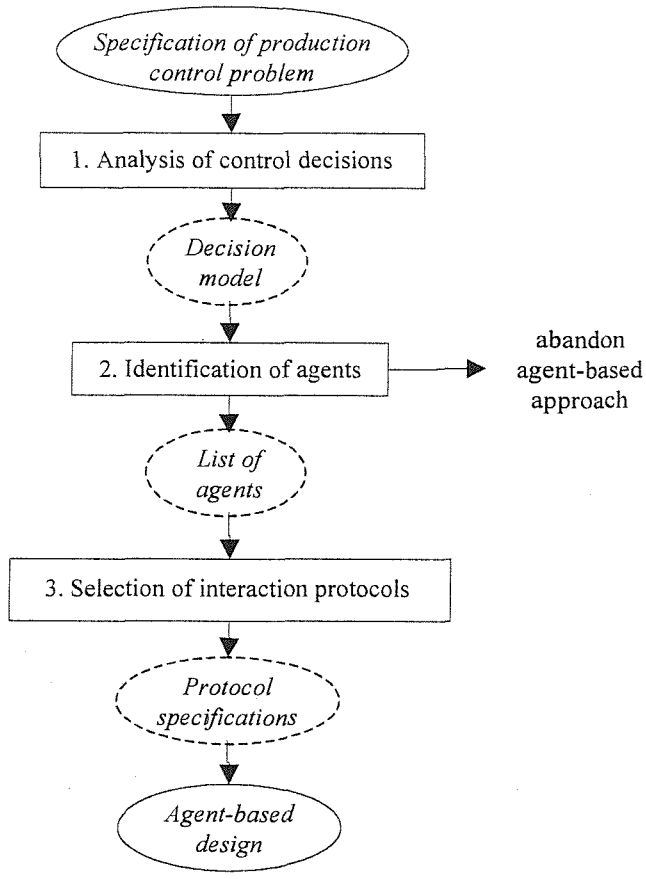

Figure 4.19: Results of the DACS methodology.

The DACS methodology requires as input the specification of a production control problem, defining the production process to be controlled, the production conditions under which to operate the process, and the production goals to be achieved by the control system. The methodology itself consists of three methods (see figure 4.19):

- a method for analysing the decisions necessary to control the production process under the restrictions defined;
- a method for identifying agents with their control responsibilities and their dependencies on other agents; and
- a method for selecting interaction protocols which are able to resolve the above dependencies.

The result of the methodology is a specification of the control design including the agents of the control system, and for each agent how it interacts with the production process and how, if necessary, it interacts with other control agents (see figure 4.19). In particular, the resulting design is sufficiently modular in order to enable the independent implementation of each agent.

The following chapter will evaluate this methodology with respect to the requirements put forward in subsection 3.1.1. In particular, the following chapter will provide initial evidence that the methodology fulfils these requirements. A full example application of the methodology to an industrial test case will be given in appendix B.

Chapter 5

Evaluation of the DACS Methodology

Chapter 4 presented the DACS methodology for designing agent-based production control systems. For this methodology, it is claimed that it is both appropriate for the domain of production control and sufficiently prescriptive for a control engineer with only minimal training in agent technology and no prior experience in developing agent-based systems to successfully design an agent-oriented production control system (cf. subsection 3.1.1). To substantiate this claim, it would be necessary to perform a large series of industrial case studies with a significant number of control engineers. However, this is clearly impossible within the scope of this thesis because for a realistic control problem the design process from the problem specification to the agent-based control design requires a significant investment in terms of human resources. A single case study would thus incur significant costs, a large field test prohibitively high costs.

Nevertheless, to provide support for the above claims, this chapter will discuss the results of several case studies that have been conducted using the methodology. In particular, this chapter will make three assertions. First of all, this chapter will demonstrate the applicability of the design methodology to production control problems by presenting and discussing two industrial case studies. Second, this chapter will show that the methodology is more appropriate for production control than other existing (agent-oriented) design methodologies by highlighting in what respect the methodology overcomes the limitations of the methodologies discussed in chapter 3. And finally, this chapter will present four reviews of the design methodology performed by students or control engineers with no or only minimal prior experience in agent development. Thus, this chapter will provide initial evidence that the DACS methodology is appropriate and sufficiently prescriptive for a control engineer with only minimal experience in agent development to design agent-oriented production control systems, and that consequently DACS achieves the goal of this work set forth in the introduction.

The chapter is organised as follows. Section 5.1 examines the applicability of the DACS design methodology. Section 5.2 compares the methodology with the state-of-the-art discussed in chapter 3, and section 5.3 discusses the results of the third-party

reviews evaluating the suitability of the methodology. Finally, section 5.4 summarises the evaluation and concludes the chapter.

5.1 Applicability to production control problems

The first and most important assertion to be made is that a designer, following the methodology is able to create an agent-based production control system. To show this, the author has designed two real-world production control systems according to the rules of the methodology: the *Kowest* control system and the *Wörth* control system. Both case studies are briefly discussed in the following.

The Kowest control system is an agent-based system that was developed for a flexible cylinder head manufacturing system which is now in operation at a DaimlerChrysler plant in Germany (see subsection 2.3.3). The design of the agent-based control system was originally created in 1997 without the help of the DACS methodology. Actually, the development of the Kowest system initially triggered the development of a design methodology for agent-based production control systems, because at that time the development of such systems was still a very ad hoc process. The DACS methodology was therefore applied *ex-post* to the Kowest control problem in order to verify whether the methodology is able to support design processes for which it was originally developed. That is, the author redesigned the Kowest control system according to the rules of the methodology given only the specification of the Kowest control problem (and obviously the experience from the initial design project). The process itself and the design result were documented and then evaluated by Klaus Schild, a member of the former design team for Kowest. The evaluation can be summarised as follows:

- The agent identification method creates the same kind of agents with the same responsibilities (namely workpiece, machine, transportation, and loader agents), each responsible for their (production) component.
- The protocol selection method proposed the same interaction protocols as were employed in the original design, although this is not surprising as for example the Kowest protocol which was selected from the library was originally developed for the Kowest control problem.
- The methodology, in particular the methods for analysing the control problem and identifying the agents, were regarded as intuitively appropriate for designing agent-based control systems, and it was felt that the methodology would have facilitated the original design process if it had been available at that time.

The methodology was thus able to reproduce the Kowest design and the original designers regarded the methodology as appropriate.

In the second case study, a control system was developed for an existing car body shop at the DaimlerChrysler truck plant at Wörth, Germany (see appendix B). For this shop, there already existed a conventionally developed non-agent-based control system. The goal of the case study was therefore to develop an agent-based control solution, compare its performance to that of the conventional control system, and show that the agent-based solution is able to run a real car body shop with the same or better performance. To this end, the author – without prior knowledge of the existing solution – applied the design methodology to the control problem of this shop and implemented the agent-based control system in a software prototype. The actual control problem specification and each design step are presented in appendix B.

This case study showed that the methodology was able to successfully support the design task. In particular, three observations can be made:

- The method for analysing the decision making captures the relevant aspects of the control problem, namely the necessary resource allocation and timing decisions.
- The method for identifying the agents produces a set of agents that reflects the production components (namely, AGVs, cells, and workers), and the different control tasks (such as assigning resources and scheduling tasks) are distributed among these agents.
- The method for selecting the interaction protocols identifies suitable interaction protocols, and in case there does not exist a suitable interaction protocol in the library, either points to similar protocols, or changes the interaction problem by considering combined dependencies (as in the case of the timing dependencies in subsection B.4.6).

The agent-based control solution described in appendix B was implemented as a software prototype and applied to a simulation of the shop (provided by the plant). The agent-based solution was tested with about four job databases taken from the plant and the prototype showed a performance comparable to that of the existing control solution. The case study was therefore regarded as successful (for more details see section B.5).

To summarise, the methodology was applied successfully to two industrial control problems. In the first case study, the methodology was able to reproduce the design results of a completed agent-based design project. In the second case study, the methodology was able to develop a completely new agent-based solution that performed as well as the existing and optimised solution.

5.2 Comparison to the state-of-the-art

The second assertion to be made about the methodology is that it is not only applicable, but also more suitable for the design of agent-based production control systems than other existing design methodologies. To demonstrate this, this section qualitatively compares DACS to the state-of-the-art.

The state-of-the-art review in chapter 3 has shown that existing design methodologies are either not appropriate or not sufficiently prescriptive for designing agent-based production control system. In particular, the review has identified three main limitations of the existing methodologies which are briefly repeated below (see section 3.7).

First of all, no existing methodology is able to adequately model the agent-oriented decision making which is necessary in control applications. Most agent-oriented design methodologies use modelling concepts, such as roles or goals, which have no direct counterpart in the domain of production control (see subsection 3.5.7). Manufacturing control methodologies, in turn, do use modelling frameworks, such as Petri nets, which are appropriate for modelling the actual production process. These methodologies, however, either do not explicitly model the control decisions, or do so in a centralised or hierarchical manner, which is inappropriate for an agent-based design (see section 3.4 and subsection 3.5.2). The state-of-the-art consequently lacks a modelling framework that supports a smooth transition from the manufacturing domain to agent-based decision making.

Second, no existing methodology provides sufficient criteria for identifying suitable production control agents. Most methodologies provide either no criteria, only heuristics (see for example PROSA in subsection 3.5.2.3 and the synthetic ecosystems approach in subsection 3.5.6.2), or criteria which – due to the underlying concepts – lead to an inappropriate set of agents (see subsections 3.3.5, 3.5.1.3, and 3.5.3.6). In particular, the few methodologies intended for the identification of production agents are not sufficiently precise about the rules of their identification method, i.e., when and where to apply the identification or aggregation of agents (see subsection 3.5.2.4). The state-of-the-art thus also lacks a prescriptive method for identifying production control agents.

Third, very few methodologies have looked at designing interactions and in particular none have provided a method for designing the necessary interaction protocols. One noteworthy exception is the methodology of Elammari and Lalonde (see subsection 3.5.4.3) which identifies different types of dependencies between agents in order to guide the interaction design (but matches these dependencies only to pre-defined interaction protocols, such as an execute message (see subsection 3.5.4.3)). The second noteworthy exception is the agent interaction analysis methodology of Miles et al. (see subsection 3.5.5.1), which proposes to re-use existing interaction protocols (but does

not provide a mechanism for (automatically) matching interaction situations and protocols, so that the number of protocols that can be considered during the design phase must be small (see subsection 3.6.2)). The state-of-the-art thus still lacks a method for designing as well as re-using interaction protocols on a large scale.

The DACS methodology extends the state-of-the-art in that it provides a set of appropriate and sufficiently prescriptive methods for each of the above design aspects. Thus it improves the methodological support for designing agent-based control systems. The methodology starts with an analysis method that is appropriate for modelling control tasks, and thus bridges the gap between the domain of production control and agent-based systems. This method first identifies and characterises the local decision tasks arising at the production components and then adds any dependencies between these decisions. By identifying *local* decision tasks first, the designer starts with control aspects with which he is familiar, namely the control of a single production component, and which are described in the control problem specification. Likewise, the characterisation step for the local decision tasks requires only local aspects to be specified and the exact information to be provided for a local decision task is defined by a schema. A control engineer should therefore be able to easily perform the first analysis step.

Non-local decision relations are then added in a second step by identifying the dependencies between the local decision tasks. That is, the designer identifies the dependencies only once he is familiar with all the local aspects of the control problem. Furthermore, identifying the dependencies requires only domain reasoning, of which the designer (as a control engineer) should be capable, and for each dependency, the designer is only required to provide constraints and preferences characterising each dependency without specifying at this stage how it should be resolved. With the help of the analysis method, the designer is thus able to create a model of the necessary decision making by reasoning only about the domain of production control. The method thus provides a smooth transition between the problem and the solution domain.

The second method of the methodology, the agent identification method, takes the decision model created by the first method and identifies the control agents by grouping decision tasks according to a set of rules. More precisely, this method defines a set of operations the designer may perform on the decision model, and a set of rules prescribing or recommending to perform specific operations in certain situations. Thus both operations and rules guide the design process in that the operations delineate what the designer may do in order to identify the control agents, and in that the rules tell the designer what he should do in order to arrive at an appropriate set of control agents. In particular, the design rules include a rule that specifies when to abandon the agent-oriented approach (namely when the decision model collapses into a single agent). In contrast to the literature, the method proposed consequently provides a much higher

degree of methodological support for identifying control agents because it specifies allowable identification steps and captures the relevant agent-oriented design criteria for these steps in a set of rules which can also rule out an agent-oriented approach. Obviously, the method does not specify a deterministic set of design rules prescribing exactly which operations to perform in which order, because otherwise the method would become an algorithm always resulting in a specific design. The method thus leaves room for creativity, while specifying all agent-oriented criteria which must be fulfilled in order to arrive at a truly agent-oriented design.

Finally, the third method of the methodology selects an interaction protocol for each dependency between different agents. It does so by first classifying the dependency according to a pre-defined classification scheme and then searching in a library of existing interaction protocols for one that best matches the classification of the dependency. The first step for the designer is thus to analyse the dependency in question with respect to each criterion of the classification scheme. To do this, the designer is not required to have any knowledge about the interaction protocols because the classification scheme is defined only in terms of the decision model. And, once the classification is produced, the matching process itself can be performed automatically if all interaction protocols have also been characterised according to the classification scheme (and this needs to be done only once). The designer is involved again only after a set of suitable interaction protocols has been found, and only for those he has to verify that at least one does resolve the considered dependency. In other words, the designer is not required to know all the available interaction protocols, but only to look at those which promise to resolve the dependency. In contrast to previous work, the re-use mechanism is consequently scalable to a large set of interaction protocols and can thus draw from the large body of interaction techniques already developed in agent technology. Obviously, not covered by the methodology is the design of new interaction protocols. This challenging task is left to future work (see section 6.3).

In summary, the methodology provides a greater degree of methodological support for designing agent-based production control systems than existing approaches because the models employed are more appropriate for capturing control decisions and the methods provide more criteria for performing the agent-related design steps (i.e., they are more prescriptive). This, however, does not yet imply that a designer with only basic training in agent technology and no prior experience in agent development is also able to successfully apply the methodology. This final assertion is made in the following section.

5.3 Third-party reviews

To provide initial evidence that a designer with no prior experience in agent

development can apply the methodology, two third-party case studies and two reviews by control engineers were performed. For each case study, a student with no prior experience in agent development was asked to independently apply the methodology to a realistic production control problem. Additionally, the methodology was presented to several control engineers developing control systems, and the engineers were asked to evaluate the methodology.

5.3.1 First third-party case study

The first case study was performed by Ilka Lehweß-Litzmann, a computer science student with only basic training in agent concepts and no prior experience in agent development. For the purpose of this case study, it was sufficient to ask a computer science student because, with respect to computing, a computer scientist and a control engineering have broadly comparable backgrounds (this obviously changes once a computer science student specialises in a specific topic, which had not yet been the case for Ilka at the time of the case study).

For the case study, Ilka was asked to apply a preliminary version of the DACS methodology to the design of a control system for a flexible engine assembly process. The process to be controlled was a flexible assembly system for the production of car engines in small to medium volumes. The production system consists of a main assembly line and several additional component assembly lines which supply parts to the main assembly line (see figure 5.1). Each line consists of a series of assembly stations, some of these performing the same operation, and the transportation between the stations is realised with the help of AGVs. The critical aspect about the assembly process is the co-ordination of the parts to be assembled at the intersection of the main and the component lines. In particular, it has to be ensured that the parts to be assembled into one product arrive at the same time at the same station.

Ilka applied the methodology to the above control problem and ran through each step of the methodology. Each design step and the resulting agent-oriented control system were documented by her. The design consists of an agent for each AGV, station, worker, job, and in particular each component part. As interaction protocols, the contract net was chosen for assigning various resources, and a distributed constraint satisfaction algorithm for scheduling and thus co-ordinating main and components lines. After finishing the design, Ilka was asked to review her experience with the preliminary version of the methodology. In particular, she was asked to fill out a questionnaire with the following types of questions (see also section C.1):

- Is the description of the methodology comprehensive? What is easy / what is difficult to understand? What should be described in more detail / with more examples?

- Does the approach make sense? Is it easy to apply? Which parts are difficult to apply? Is the methodology sufficiently prescriptive?
- What is the general impression of the methodology?

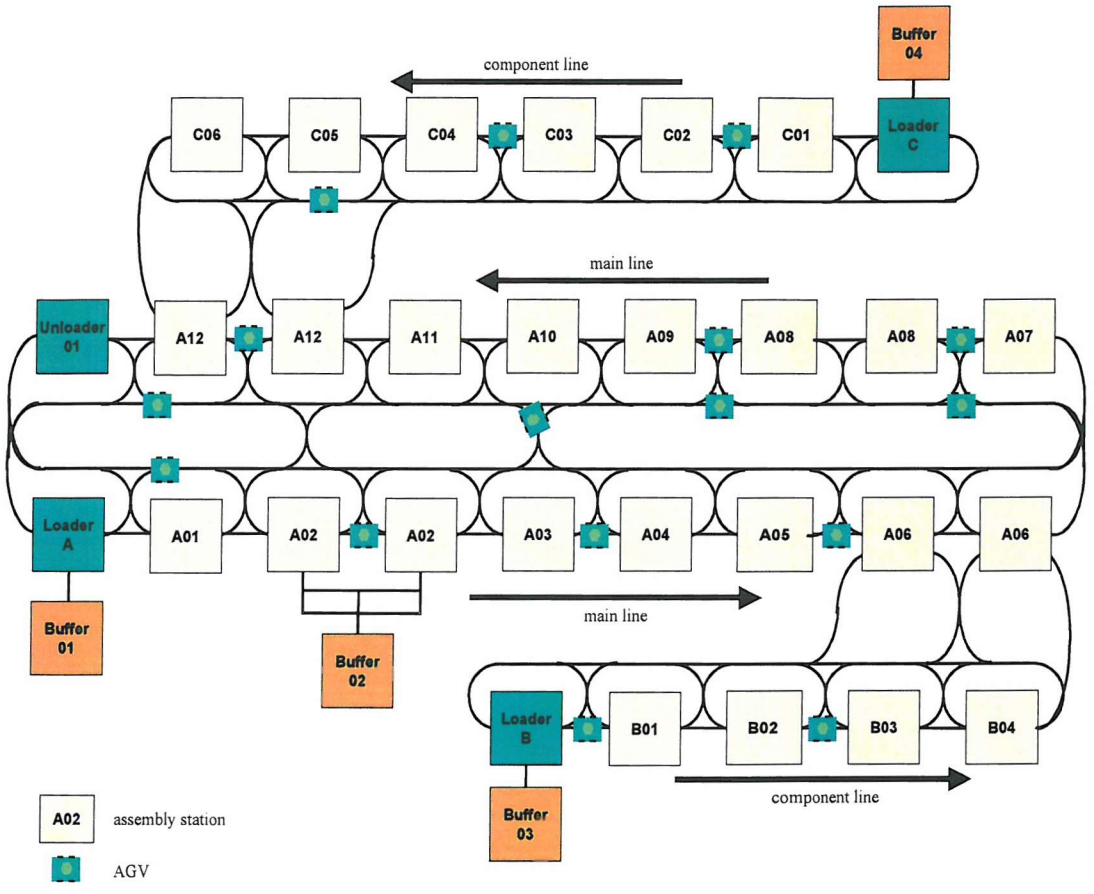

Figure 5.1: Example assembly layout.

Most points raised in the feedback are concerned with the description of the methodology because some parts were unclear or did not provide sufficient examples. Concerning the concepts of the methodology, Ilka raised only four points (see also section C.1):

- Decision tasks should be characterised during the analysis phase and not during the identification of the agents (as it was in the preliminary version).
- The methodology missed a schema for specifying decision tasks.
- The characterisation of dependencies (in the preliminary version by *intensity* and *importance*) was difficult to understand and apply.
- The operations to modify the decision model (in the preliminary version *divide* and *expand*) were not sufficiently specified, which led to misunderstandings.

All in all, Ilka found that the methodology is appropriate for designing agent-based

systems and really helps to perform the design. As a general improvement, she proposed to use more diagrams, in particular with UML notation. As a consequence of the case study, the methodology was significantly revised with respect to the points raised in the feedback. The revised version of the methodology was then used as a basis for the second case study.

5.3.2 Second third-party case study

The second case study was performed by Laura Obretin, also a computer science student with only basic training in agent concepts and no prior experience in agent development. Like Ilka, Laura had not yet specialised in computing at the time of the case study.

Laura was also asked to apply the DACS methodology to the flexible engine assembly control problem in order to verify whether the revised methodology was an improvement over the original version. She thus ran through each step of the revised methodology and documented the design steps as well as the design result. Her design consists of an agent for each AGV, station, component, loader, and transportation crossing.[27] Her design thus differs from the design of Ilka because she decided to handle jobs and the transportation differently and therefore re-organised the decision model according to a different strategy. This is possible and desirable because, despite the agent-oriented design rules, the identification of the agents remains essentially a creative process. As interaction protocols, the contract net protocol, a modification of the Kowest protocol, and a distributed constraint satisfaction algorithm were chosen.

After completing the design, Laura was also asked to fill out a questionnaire in order to document her experience with the revised version of the methodology. The questionnaire contained essentially the same questions as the one filled out by Ilka. The feedback of Laura in the questionnaire was only positive (see also section C.2). She found the methodology comprehensive and appropriate. An analysis of her documentation, however, revealed that the method for selecting interaction protocols offered too many protocols. The selection process was thus not sufficiently precise. This method was therefore significantly revised by detailing the classification scheme. Laura was then asked again to read the description of the revised methodology in order to test whether she found that the revised version was an improvement. She confirmed this. The revised version resulted thus in the final version of the methodology which was presented in chapter 4.

[27] A worker agent was not identified because workers were omitted in the specification of the second case study. After the first case study, it was realised that workers are not a scarce resource which needs to be controlled.

5.3.3 Third-party reviews

In a final evaluation step, several control engineers, experienced with designing control systems, were asked to review the design methodology. That is, the methodology in its final version was presented in two workshops and the engineers were asked to fill out a questionnaire with the following questions:

- Is the methodology comprehensible?
- Does the approach make sense?
- Is the approach sufficient to develop the agent-oriented aspects of a control system? What is missing? What should be improved?
- Is it possible to apply the methodology in your area?

The results of the reviews are discussed in the remainder of this subsection. The reviewers were Armando Walter Colombo, Ralf Neubert, and Boris Süssmann from Schneider Electric, Germany, as well as Dirk Hofmann and Raimund Krieg from the DaimlerChrysler plant at Wörth, Germany. All reviewers were familiar with basic agent-oriented concepts, but had not designed an agent-based control system yet. The results of each workshop are discussed in the following subsections.

5.3.3.1 Review by Schneider Electric

The feedback from Schneider Electric concerning the methodology itself was very positive (see also section C.3). It was confirmed that the methodology is comprehensive and well justified. In particular, it provides a good transition from the specification of the production process to agent-oriented concepts. Only three aspects concerning the methodology itself were raised:

- The methodology tends to identify reactive agents, but it was felt that this is probably sufficient for control systems.[28]
- It should be specified what kind of competencies or knowledge, for example about production processes, is required by the designer to apply the methodology.
- The methodology should use standard notation, as for example UML or SDL, as much as possible.

All in all, it was confirmed that the methodology is well suited for control applications and could be applied at Schneider Electric, even though it was also stressed that more validation needs to be done before the methodology can be promoted in industrial

[28] The methodology only starts with a reactive point of view in order to avoid any implicit design decisions (see section 4.2). The designer may identify more deliberative agents by grouping local decision tasks or introducing abstract decisions.

projects.

During the discussion, the engineers also raised many issues relating to possible extensions of the methodology or the integration of the methodology into the different development phases for a control system.

- How is it possible to assess the implementability of a design, for example whether it is (soft) real-time capable? What is the time and space complexity of a protocol? What is the communication requirement of a protocol? How can a worst-case analysis be made?
- How does the methodology integrate with existing agent architectures or tools? How can components of agents be standardised?
- How is it possible to more extensively use design (or implementation) experience from other projects? How can maintenance aspects be integrated into the design?
- How can an agent-oriented control system interface with other existing enterprise software systems, such as enterprise resource planning (ERP) systems?
- How can human workers be integrated into the design (other than simply regarding them as just another resource)?

Obviously, the above aspects remain open issues and need to be addressed in future work on the methodology or on agent technology in general.

5.3.3.2 Review by DaimlerChrysler

The feedback from engineers of the DaimlerChrysler plant at Wörth was also very positive (see also section C.4). It was confirmed that the methodology is comprehensive and straightforward. In particular, it was regarded as an advantage that each step of the methodology provides a set of rules for performing the design step. Only two aspects concerning the methodology itself were raised:

- How can an iterative design process be supported? Iterative design processes are very common in practice.
- How can an interaction protocol be assessed with respect to the real situation on the shop floor? Is the interaction protocol implementable with respect to the conditions at the shop floor?

All in all, it was confirmed that the methodology is well suited for control applications and that it could be applied to the development of future control system, even though two aspects must be addressed to enable the use of agent technology at the plant.

- There must be appropriate tools for the development of agent-based control systems that interface well with existing tools for control design and implementation (i.e., simulation, PLC programming tools, etc.).
- There must be reliable tools and hardware for implementing the agent-based control systems at the shop floor.

Obviously, the above aspects must be addressed by control vendors promoting agent technology.

5.3.3.3 Summary

To summarise, the methodology was regarded by all reviewers as a comprehensive and appropriate approach to designing agent-based control systems that improves the development process in practice. The methodology must now be integrated into the different development phases for control systems and further issues, such as the analysis of computational and communicational requirements, standardisation of agent components, and human integration, must be addressed in agent research.

5.4 Summary

For the DACS methodology proposed in chapter 4, it is claimed that it is both appropriate for the domain of production control and sufficiently prescriptive for a control engineer with no prior experience in agent development to successfully design an agent-based production control system. To prove this claim, this chapter has provided initial evidence by comparing the methodology to the state-of-the-art and providing several industrial case studies. More precisely, this chapter has made three assertions:

1. The methodology is applicable to the design of agent-based production control systems.

   This was shown by presenting two case studies (performed by the author) in which agent-based control systems were developed for industrial production systems.

2. The methodology is more appropriate and prescriptive than other existing methodologies.

   This was shown by highlighting in what respect the methodology overcomes the limitations of the state-of-the-art identified in chapter 3.

3. The methodology can be successfully applied by engineers with no prior experience in agent development.

This was documented by two case studies in which computer science students applied the methodology to an industrial control problem specification, and two reviews by engineers responsible for designing control systems.

These assertions are certainly not sufficient to prove the above claims. In particular, the third-party case studies have only been performed by computer science students. It is therefore necessary to do more field tests with control engineers in the future. Nevertheless, the above assertions do provide initial evidence that the DACS methodology is both appropriate and sufficiently prescriptive for the design of agent-based production control systems.

Chapter 6

Conclusions

The goal of this thesis was to develop a methodology for the design of agent-based production control systems which can be successfully applied by a control engineer with only minimal training in agent technology and no prior experience in agent development. This chapter concludes the thesis by reviewing how this has been achieved, putting this work into context, and pointing to possible future work.

6.1 Review

First, chapter 2 set the scene of this thesis by reviewing the state-of-the-art in agent-based production control. It began by defining the terms "production" and "production control", and motivating why production control is of economic importance. It furthermore analysed the limitations of existing control approaches and derived future requirements on production control. Chapter 2 then briefly reviewed agent technology by summarising the state-of-the-art in agent modelling and interaction techniques. Finally, chapter 2 turned to the intersection of agent technology and production control, namely agent-based production control systems. Here, it was argued that agent-based systems are the right technology to meet many of the new requirements on production control and that this is consistent with current trends in production research. Furthermore, it presented several industrial test cases, and showed that there is not just one universal agent-based solution for all control problems, but that the control solution must be tailored to the specific needs of an application. The chapter therefore concluded that a wide spread use of agent technology in production control requires a recurrent design effort and thus a design methodology for agent-based production control systems. It furthermore concluded that such a design methodology must be applicable by a control engineer with no significant experience in agent development because the agent-oriented design – no matter how important to the overall performance of a system – is only one aspect of a control implementation.

Motivated by the discussion of agent-based production control, chapter 3 then looked at the state-of-the-art in design methodologies. It first defined the term methodology and analysed the requirements on a design methodology for production control. In particular, it derived the requirements of model appropriateness and method prescriptiveness from the overall goal of the thesis. Without the fulfilment of both requirements, most control engineers will fail to develop an agent-based control system. The chapter then continued by reviewing existing design methodologies with respect to their suitability for the task at hand. Object-oriented methodologies were found to already violate the first requirement (namely that of model appropriateness). Unable to model agent-based systems in general, they equally fail to model the decision making necessary in agent-based production control systems. As a consequence, for example, the use of object-oriented methodologies leads to a set of agents that are inappropriate for implementing control systems. Manufacturing control methodologies, in turn, provide sufficient means to model production processes as well as the associated control decisions, but either ignored any dependencies between the decisions or resolved these in a centralised manner. Manufacturing control methodologies are thus equally inappropriate to model agent-based production control systems.

Most of the limitations of object-oriented and manufacturing control methodologies were remedied by the introduction of agent-oriented methodologies. These methodologies are obviously appropriate for modelling agent-based systems. However, despite the wide variety of different approaches, none of these methodologies are either appropriate or sufficiently prescriptive for production control. Most agent-oriented methodologies are inappropriate because they were developed with other applications in mind, and therefore build upon concepts (such as roles or tasks), which have no direct counterpart in production control. In addition to those methodologies that do model decision making remain vague with respect to how these decisions should be identified or assigned to agents. With respect to interaction design, the methodologies are even less developed. Only two methodologies consider the re-use of existing interaction protocols. Chapter 3 therefore reviewed also the state-of-the-art in re-use and found that none of the proposed approaches are sufficiently prescriptive to enable the re-use of a large set of interaction protocols. With these methodologies, the designer must explicitly consider each existing protocol in detail in order to decide which protocol can be used in his application. In sum, chapter 3 concluded that none of the existing methodologies are appropriate or sufficiently prescriptive for designing agent-based production control systems. In particular, the state-of-the-art exhibits limitations with respect to modelling control decisions, identifying (control) agents, and re-using / designing interaction protocols.

To overcome the limitations of existing methodologies, chapter 4 presented the main contribution of this thesis: the DACS design methodology for agent-based production control systems. This methodology starts from the specification of the production

control problem, which, among other aspects, includes a specification of the production components to be controlled and their physical behaviour. The first method of the methodology, the *analysis of decision making*, then analyses the decision making necessary to operate the production system by looking at the local decision tasks arising at each production component and identifying any decision dependencies that may exist between the local decision tasks. The second method of the methodology, the *identification of agents*, builds upon the decision model derived by the first method and clusters decision tasks in order to assign each cluster to an agent. For the clustering process, the method provides three rules that were derived from the concepts of coupling and cohesion and adapted to the concepts of agent-oriented design. These rules make it possible to cluster, but not to separate decision tasks, and may thus result in clustering all decision tasks into a single agent. The method therefore provides a set of operations that modify the decision network, either by separating decision tasks or by introducing new decision tasks. These operations allow the designer to re-arrange the decision model and thus to improve the clustering process. If, however, the re-arrangement fails to avoid the collapse of the decision model into one agent, then an agent-oriented approach is not suitable for the given control problem. The method for agent identification thus also provides a rule for verifying the appropriateness of an agent-oriented approach for a specific application. Finally, the third method of the methodology, the *selection of interaction protocols*, provides a mechanism for re-using existing interaction protocols in order to resolve any decision dependencies between different agents. This method provides a scheme for classifying decision dependencies that is based solely on criteria characterising the dependency itself. The classification can therefore be performed by a designer with minimal knowledge of the existing interaction protocols. On the other hand, as has been shown in chapter 4, the classification scheme is sufficiently abstract to be applied to generally defined interaction protocols and sufficiently precise to separate existing interaction protocols into different classes. The designer will thus be prompted with only a small set of interaction protocols once he has classified the dependency. The third method of the methodology thus enables the efficient re-use of a large set of interaction protocols. In sum, the successive application of all three methods of the methodology results in an agent-based design solving the given control problem (presuming that there exists at least one suitable interaction protocol for each decision dependency). The resulting design consists of a list of agents with their decision responsibilities (in terms of a set of decision tasks), and a set of interaction protocols for each dependency between different agents. The resulting design is thus sufficiently modular to allow the independent implementation of each agent.

Chapter 5 then provided an initial evaluation of the claim that the methodology is applicable to the design of real production control systems and, in particular, is

applicable by a control engineer with no prior experience in agent development. To this end, the chapter made three assertions:

1. The methodology is applicable to production control problems.
2. The methodology provides more design support than other existing methodologies.
3. The methodology can be applied by students and/or engineers with little or no experience in agent development.

For the first assertion, the chapter reviewed two test cases performed by the author. For the second assertion, the chapter compared the methodology to the state-of-the-art in design methodologies and argued qualitatively that DACS is more appropriate and prescriptive than existing methodologies. In particular, it showed that the designer may create the decision model by only analysing domain-specific concepts (of which he is assumed to be capable as a control engineer) and that all agent-oriented design criteria necessary to perform the design are captured in a set of design rules. To further support this argument, chapter 5 reported on two case studies in which a student with no prior experience in agent development applied the DACS methodology to an industrial control problem and two reviews performed by engineers developing control systems. Apart from critiquing earlier versions of the methodology, the feedback from the case studies and the reviews was unequivocally that the methodology is suitable for production control and can be applied by a control engineer with no prior experience in agent development. It can therefore be concluded from this initial evidence that the thesis has achieved its goal, even though, as also pointed out by the control engineers during the review, more validation needs to be done in the future. With the evaluation and the successive improvements performed, however, the DACS methodology is ready to be used in industrial pilot applications.

6.2 The work in context

The DACS design methodology presented in this thesis is a novel contribution to the state-of-the-art. One the one hand, this methodology builds upon several achievements from different research fields. It uses, for instance, a model of decisions from decision theory, the basic concept of dependencies from agent theory, the notions of coupling and cohesion from general software engineering, and the idea of heuristic classification from artificial intelligence. On the other hand, the methodology significantly extends the state-of-the-art in design methodologies in at least three important respects:

- The methodology provides a method for analysing the production control problem that bridges the gap between the domain of production control and agent-based systems. In particular, the method provides an agent-oriented

model of control which consists of domain concepts – namely decision tasks and decision dependencies – that the designer should be familiar with and which can be derived by only looking at the control problem specification. Furthermore, the method introduces the concept of trigger diagrams which express the decision tasks and their dependencies graphically. The designer is thus able to create the analysis model without any knowledge of agent technology. Agent-oriented concepts are only introduced later in the design process.

- The methodology provides a set of operations and criteria for identifying agents which either lead to an appropriate set of control agents or suggest abandoning an agent-oriented approach. These operations and criteria are defined only in terms of the decision model developed during the analysis and can thus be applied by a control engineer with no prior experience in agent development.
- The methodology provides a classification scheme for re-using interaction protocols that is based only on criteria characterising the decision situation arising during the production process. That is, for selecting an interaction protocol, the designer is only required to analyse the production situation in which the dependency to be resolved arises. The re-use mechanism for selecting interaction protocols is thus scalable to a large set of existing protocols.

The DACS methodology thus extends the state-of-the-art by providing appropriate and sufficiently prescriptive methods for designing agent-based production control systems. The applicability of the methodology, though, is not restricted to production control alone. The methodology is also able to design other kind of control systems. The analysis method of the methodology only assumes that there is a set of physical components that need to make decisions as the process evolves. This view also applies to other control problems, such as the underground transportation of baggage in an airport, the movement of containers in a port, or the co-ordination of planes approaching an airport. The methodology is thus applicable to any application domain in which a *physical process can be controlled by discrete decisions.* The methodology, however, is not applicable to any application domain for which an agent-oriented approach is suitable. For instance, modelling an organisation and its business processes requires concepts like roles, services, or tasks (see section 3.5). For such an application, role-based or task-based methodologies are much more appropriate. The DACS methodology will thus never replace all methodologies and become the one and only agent-oriented methodology, but rather it will belong to a set of agent-oriented methodologies which all together cover a wide range of application domains.

Nevertheless, some aspects of the methodology are more generally applicable (i.e., applicable to domains other than control). The approach to first create a semi-formal

domain model through an analysis and then to operate on this model in order to derive an agent-based design may also be helpful to other methodologies in order to increase their prescriptiveness. For DACS, this approach facilitated the definition of design rules and in particular the definition of a rule stating when to abandon an agent-oriented approach. Likewise, the heuristic classification of interaction situations could be used in other methodologies. The classification scheme presented in section 4.4 mostly considers aspects applicable to any interaction situation, such as the number of agents involved in an interaction, or the number of joint commitments to be achieved by the interaction. The classification scheme, though, may have to be extended for other application domains if in these domains other aspects, such as truthfulness, interoperability, adaptability, security, and so on, play a more dominant role.

6.3 Future work

Despite the achievements of this thesis, many issues are still left to future work. This section points to the most important ones.

First of all, more validation needs to be done in the form of industrial case studies or field tests in order to provide more evidence for the applicability of the methodology, but also to disseminate the methodology to more practising control designers. To further promote the use of the methodology, there is also a need for tools that assist the designer in performing the different steps of the methodology. These tools could include, for example, editors for specifying the decision tasks and dependencies, graphical editors for modifying and clustering the decision tasks, and libraries for automatically searching interaction protocols given a classification of an interaction protocol.

Secondly, several open issues were raised by the engineers during the review of the methodology that need to be addressed in order to integrate the methodology with industrial control development. The most important of these are listed below:

- How can the methodology be integrated with existing agent architectures, tools, or standards?
- How is it possible to assess the implementability of a design in its early stages, in particular with respect to its (soft) real-time capabilities?
- How can the designed agent-based control systems interface with other existing enterprise software systems, such as enterprise resource planning systems or quality monitoring systems?
- How can human workers (and in particular their flexible and intelligent capabilities) be integrated into the design?

Thirdly, there are several conceptual issues that are worth considering in future research. The most important among these is certainly the design of new interaction protocols if no existing interaction protocol is applicable. This is obviously a very challenging task and it is questionable whether it will be possible to provide a methodology for designing new interaction protocols that can be used by a designer with no prior experience in agent development. Here, probably, the goal of the research needs to somehow be reduced, for example to only aim at designing interaction protocols from standard components or messages.

Another important research issues is to integrate the methodology with other methodologies covering other aspects of manufacturing design, such as the design of the physical material flow or PLC controllers. It is reasonable to expect that the overall design will be improved if the different design phases are co-ordinated because a control system can only create a certain system behaviour if the physical process is able to perform the required operations, and, vice versa, a physical process can only show a certain behaviour if there is a control system that determines which operations must be executed in order to create the desired behaviour (cf. figure 2.5). Physical process and control system, therefore, must be designed simultaneously.

Finally, it seems worthwhile to try to integrate the DACS methodology with other agent-oriented methodologies covering other application domains. Currently the designer must choose one methodology for his application and perform the design according to this methodology. In applications which span several different domains, however, this may be a disadvantage. For instance, an agent-based system supposed to model or simulate a manufacturing company will need to model the production process including its control as well as the business processes of the company. For the latter, a role-based modelling approach is more appropriate. The designer may therefore need to use a combination of different methodologies. A first step towards such a combination of methodologies was done in (Juan 2003), but more work on a truly comprehensive set of methodologies is required in the future.

Appendix A

Characterisation of Example Interaction Protocols

This appendix characterises a set of example interaction protocols with respect to the classification scheme presented in subsection 4.4.1. The characterisation of an interaction protocol is not simply a classification according to the classification scheme. Instead of assigning it to a specific class of dependencies, an interaction protocol should be assigned to all those classes which it can efficiently solve. The task of the characterisation is therefore to analyse the interaction protocol with respect to the classes of dependencies it could possibly address. To express the fact that an interaction protocol addresses several classes of dependencies, the characterisation must either specify a class that subsumes all other classes (as for example for criterion #1) or else list all exclusive classes of a criterion that the interaction protocol addresses (as for example for criterion #3). Note furthermore that when discussing an interaction protocol, only the protocol is characterised, not the underlying interaction technique. An interaction technique may provide the basis for quite diverse interaction protocols that will address different classes of interaction situations. An interaction technique may thus address many classes of decision dependencies. Interaction protocols, on the other hand, are usually much more focussed on specific interaction situations. The following characterisation therefore concentrates only on interaction protocols. Finally, note that the following characterisation will not discuss in detail the motivation or background of an interaction protocol, nor will the interaction protocols be changed in any way to be applicable to other classes of dependencies. Interaction protocols are treated as given, and the sole purpose of this appendix is to characterise these protocols according to the classification scheme presented in subsection 4.4.1.

In the following, each section describes an interaction protocol or a set of similar interaction protocols and characterises each protocol according to the classification scheme. The interaction protocols discussed are categorised into the following broad areas:

- Voting (subsection A.1)
- Negotiation (subsection A.2)
- Auctions (subsection A.3)
- Distributed constraint satisfaction (subsection A.4)
- Coalition formation (subsection A.5)
- Co-ordination of multi-agent plans (subsection A.6)
- Application-specific interaction protocols (subsection A.7)
- Simple interaction protocols (subsection A.8)
- Social laws (subsection A.9)

A.1 Voting

The interaction of autonomous agents often includes the reconciliation of conflicting interests. The agents of a control system might for instance disagree about the next action to be taken, or about the distribution of available resources. To resolve conflicts, the agents must somehow find a compromise that is acceptable to all agents. Probably the simplest approach to find a compromise is to vote over possible alternatives. Many voting procedures have been proposed with quite different properties (Sandholm 1999). A simple example of such a voting procedure is the *plurality voting* protocol, which will be characterised below. To contrast this simple protocol with more sophisticated voting protocols, the Clarke tax protocol is also discussed and its characterisation is compared to that of the plurality voting protocol.

A.1.1 The plurality voting protocol

In the plurality voting protocol, the set of possible solutions is announced and each agent votes for the solution that is best for it. The solution that receives the most votes is then chosen as the compromise that must be accepted by all agents. This protocol is obviously simple and efficient, but it also has several drawbacks. For instance, the protocol is not stable with respect to irrelevant alternatives. An additional, but irrelevant alternative may split the votes for the most preferred alternative and may thus lead to the selection of a different alternative whose votes were not affected by the irrelevant alternative (see (Sandholm 1999) for a more detailed discussion). Despite the drawbacks, however, the plurality protocol may be sufficient in some cases.

The plurality voting protocol is characterised as follows (see table A.1). An arbitrary number of agents may participate in the voting process. Agents may not join once the

voting process has started (even though the protocol could be changed to accommodate late votes). The number of agents involved is thus fixed. Furthermore, the voting mechanism is able to resolve conflicts due to opposing interests (by simply choosing the solution preferred by most agents). Of course, due to the simplicity of the selection mechanism, the protocol might not really reconcile possible conflicts between the agents. The preferences may nevertheless be opposing. The protocol is not able to consider global constraints and preferences because the votes are made only on the basis of the individual constraints and preferences.

	Classification criteria	Classification
#1	Number of agents involved	**fixed**
#2	Compatibility of preferences	**opposing**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**fix**
#6	Relation of commitments	**-**
#7	Role assignment	**all variable**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**partial**

Table A.1: Characterisation of the *plurality voting protocol*.

The protocol determines a single solution to which all agents must commit themselves (see also figure A.1). Since there is only one joint commitment to be achieved, criterion #6 – the relation of commitments – does not apply. The protocol puts no restrictions on the agent roles in the goal state. However, to make the roles variable, every possible role assignments must be defined as a possible solution over which the agents vote.

The protocol allows the voting over a set of possible solutions. Since the agents only declare their most favourite choice, they do not have to reveal their preferences. With respect to criterion #8, the protocol is thus classified as "alternatives only". Finally, the agents have to accept the solution which receives the most votes. The agents thus delegate their decision autonomy to the group.

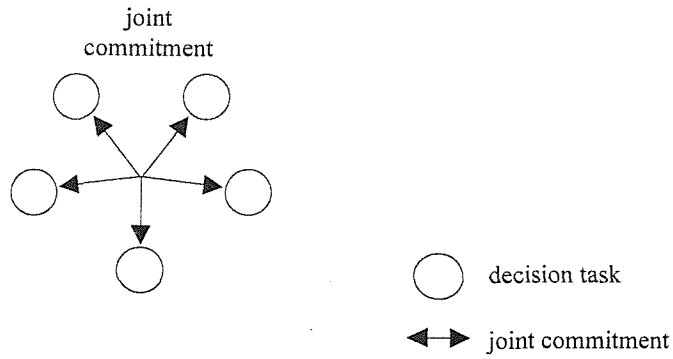

Figure A.1: The decision structure of the plurality voting protocol.

A.1.2 The Clarke tax protocol

An important limitation of the plurality voting protocol is that it considers only the most preferred choice of each agent – the agents are not able to express how much they prefer a solution over another, or with respect to another agent. Several protocols therefore allow agents to quantify their preferences. The difficulty with quantitative preferences though is that agents may strategically declare false preferences in order to manipulate the voting process. An agent may for example declare a very high preference in order to have his preferred solution win. To avoid insincere declarations of preferences, the Clarke tax protocol levies a tax on those agents that made the solution win. That is, the agents declare the utility they expect from each alternative, and the alternative with the largest sum of associated utility values is chosen. In addition to the plurality protocol, though, the agents that made the chosen alternative win due to their declared utility values have to pay a tax. The tax amount is equal to the amount of utility that caused the alternative to win over the other alternatives. It has been proven that this tax forces the agents to declare their true utilities. That is, due to the tax the agents have no incentive to declare lower utility – because then their most preferred choice may not be selected – or to declare higher utility – because then the tax amount is higher than their benefit (see (Clarke 1971)). With the Clarke tax protocol, agents thus declare their preferences sincerely.[29]

The Clarke tax protocol has basically the same characterisation as the plurality voting protocol. The characterisation only differs with respect to criterion #8 in that the agents have to reveal their preferences in order to compute the tax (see table A.2).

[29] An important requirement for the applicability of the Clarke tax protocol is of course that there exists a "monetary system" which can be used to levy the tax.

	Classification criteria	Classification
#8	Availability of information	**preferences**

Table A.2: Characterisation of the *Clarke tax protocol.*

A.2 Negotiation

In many situations, voting over a pre-defined set of alternatives is not sufficient to resolve conflicts between autonomous agents. In particular, in the case of opposing interests, the agents need to either relax their demands or search for new alternatives during the interaction, in order to reach a mutually acceptable compromise. This process of continuous concession and search for alternatives, which is more powerful, but also more time-consuming than voting, is called *negotiation* (see subsection 2.2.2.2). Several approaches to automated negotiation and even more protocols have been proposed in the multi-agent systems literature (see (Jennings 2001b) for an overview). In this section, only three examples of negotiation protocols will be discussed and contrasted:

- Service-oriented negotiation
- The monotonic concession protocol
- The DECIDE conflict resolution protocol

A.2.1 Service-oriented negotiation

Faratin has developed a sophisticated interaction model for bilateral negotiation of services (see subsection 2.2.2.2). In this model, the two negotiation agents basically exchange new proposals until either both agents accept the last proposal or one of the agents withdraws from the negotiation. This interaction process is complemented with a set of negotiation tactics and strategies for generating a new proposal. Faratin has defined tactics for reacting to time or resource constraints, for imitating the opponent's behaviour, for trading off different aspects, or for manipulating the set of negotiation issues. These tactics are combined into strategies by weighting the influence of a tactic on the generation of the next proposal.

The characterisation of the corresponding interaction protocol is as follows (see table A.3). First of all, the protocol defines a bilateral negotiation process and may thus involve only two agents. The agents may have opposing interests. The two negotiating agents are trying to find a mutually acceptable compromise, to which they both commit themselves. There is thus only one joint commitment and this commitment is of size

two. The goal state of the negotiation is that the two agents agree on a deal. Each agent then will have to execute its part of the deal. All roles are thus fixed. The agents only need to exchange proposals and therefore interact on the level of alternatives only. And finally, each agent must explicitly accept any deal. There is thus no delegation of decision autonomy.

	Classification criteria	Classification
#1	Number of agents involved	**2**
#2	Compatibility of preferences	**opposing**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**2**
#6	Relation of commitments	**-**
#7	Role assignment	**all fix**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**no delegation**

Table A.3: Characterisation of the *Service-oriented negotiation.*

A.2.2 The monotonic concession protocol

Rosenschein and Zlotkin (Rosenschein 1994) have proposed a monotonic concession protocol for bilateral negotiations. In this protocol, each agent starts from the deal that is best for it and either concedes (monotonically) or stands still in each interaction round. To concede monotonically means that an agent proposes a new deal that is better for the other agent (regarding its evaluation function). The negotiation thus continues until either the agents agree on a compromise deal or both agents no longer concede. In the latter case, the negotiation fails and the default deal is enacted (which is assumed to be worse for both agents). As an optimal behaviour for the monotonic concession protocol, Rosenschein and Zlotkin have identified the Zeuthen strategy. With this strategy, only the agent that has most to loose concedes.

The monotonic concession protocol basically addresses the same interaction situations as the service-oriented negotiation, even though the associated Zeuthen strategy is much simpler than the negotiation tactics proposed by Faratin. However, there is one essential difference. The monotonic concession protocol requires the agents to know

each other's preference functions (i.e., the cost function for evaluating the deals) in order to determine which agent should concede next (see table A.4).

	Classification criteria	Classification
#8	Availability of information	**preferences**

Table A.4: Characterisation of the *Monotonic concession protocol.*

A.2.3 The DECIDE conflict resolution protocol

Hollmann et al. proposed an interaction protocol for resolving conflicts between agents evaluating proposals (see subsection 2.2.2.2). The agents are supposed to search for a proposal which satisfies all agents. An agent is "satisfied" with a proposal if the evaluation result of the proposal passes a given threshold. The protocol proposed by Hollmann et al. differs significantly from the above negotiation protocols in that it allows several agents to participate in the conflict resolution process. On the other hand, the protocol is not able to reconcile opposing interests. The agents make new proposals, but there is no explicit process of concession (obviously, the agents could always concede by themselves). The resulting characterisation of this protocol is summarised in table A.5.

	Classification criteria	Classification
#1	Number of agents involved	**fixed**
#2	Compatibility of preferences	**constraints**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**fixed**
#6	Relation of commitments	**-**
#7	Role assignment	**all fixed**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**no delegation**

Table A.5: Characterisation of the *DECIDE conflict resolution protocol.*

A.3 Auctions

Auctions are trade mechanisms for exchanging commodities, for example for trading goods for money (see subsection 2.2.2.2). There are *one-sided* and *two-sided* auctions. One-sided auctions allow only bids or only asks, whereas two-sided auctions allow both. Furthermore, an auction is called *continuous* if it allows bids and asks to arrive over time (and matches both over time). There are many types of auctions (and probably many more will be developed in the future). This subsection characterises three prominent examples: the *English auction*, the *contract net protocol*, and the *continuous double auction*.

A.3.1 The English auction

The English auction is a one-sided auction in which only bids are allowed (see subsection 2.2.2.2). In this auction, the auctioneer wants to sell a (specific) good for the highest price possible and asks for bids by announcing the good to be sold. The bidders respond with bids and increase their bids until no bidder bids a higher price. The auctioneer then sells the good to the bidder with the highest bid for the price of the highest bid.

The characterisation of the English auction is as follows (see table A.6). The English auction involves several agents, namely one auctioneer and at least two bidders. The number of bidders must be fixed at the beginning of the interaction because the auctioneer announces the good to be auctioned off at start and does not repeat this information afterwards. The English auction in the form described above is therefore not able to cope with a changing set of agents.[30] The English auction (as almost all auctions) was designed to reconcile opposing interests: the auctioneer wants to achieve the highest price, the bidder wants to pay the lowest price possible. However, no global constraints or preferences are taken into account by the English auction.

There is obviously only one joint commitment in the goal state, namely that of seller and buyer, and the size of this joint commitment is two (see figure A.2). In the goal state, there are two agents enacting a decision: the auctioneer selling the good, and the bidder with the highest bid buying the good. The role of the auctioneer is obviously fixed, the agent becoming the buyer is to be determined, i.e., this role is variable.

[30] Naturally, the English auction could be adapted to cope with a changing set of agents, but it is not the purpose of this work to modify the existing interaction protocols.

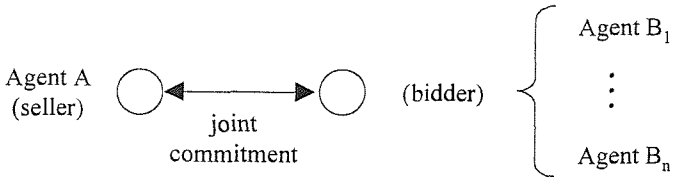

Figure A.2: The decision structure of the English auction.

Since the protocol of the English auction only allows to issue bids containing the price the bidder is currently willing to pay for the good, the agents are interacting on the basis of alternatives only. In particular, the agents do not exchange their preferences, for example how much they are willing to pay. In the original English auction protocol, the auctioneer delegates part of its decision autonomy because it must accept any bid that is the highest bid. In many cases, however, the auctioneer may specify a reservation price, i.e., a lower limit for the bids. But also in this case the auctioneer must accept any bid above this reservation price. The bidders obviously retain their full decision autonomy because they issue the bids.

	Classification criteria	Classification
#1	Number of agents involved	**fixed**
#2	Compatibility of preferences	**opposing**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**2**
#6	Relation of commitments	**-**
#7	Role assignment	**1/1**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**partial**

Table A.6: Characterisation of the *English auction.*

A.3.2 The contract net protocol

The *contract net protocol* (CNP) is a simple, but efficient protocol for assigning tasks to individual nodes in a network (see subsection 2.2.2.2). It assumes that one node has a task that needs to be executed (by another node) and that there are (potentially) several nodes that are able to execute this task. The node with the task is called the *manager*

and the other nodes are (potential) *contractors*. The manager initiates the protocol and proceeds as follows. First, it announces the task to the potential contractors. The contractors answer with a bid. The necessary information provided in the bid was specified by the manager in the announcement message. The manager compares the bids and chooses the best bid according to its preferences. The node which has sent the best bid then receives an award message and is said to have a contract with the manager about the execution of the task. The other nodes may or may not receive a reject message.

Even though the CNP was designed for task distribution, it is actually a one-sided first-price sealed-bid auction. First of all, it is irrelevant to the protocol whether the agents exchange a task or some other kind of good.[31] Second, the CNP simply stops after the auctioneer received the first bid from each bidder (first-price), instead of repeating the auction until no bidder changes its bid. And thirdly, the bid is sealed, i.e., not made known to the other bidders. In the English auction, it is a fundamental prerequisite that the bids are made open-cry, i.e., are broadcasted to all agents, because otherwise the other bidders do not know whether they need to raise their bid in order to win the auction.

The characterisation of the CNP as a one-sided first-price sealed-bid auction is nearly identical to the characterisation of the English auction. The only difference is with respect to criterion #2. The constraints and preferences of the different agents must be at least compatible (see table A.7). If the preference were opposing, it would not be possible to find a mutually acceptable compromise with the first bid.

	Classification criteria	**Classification**
#2	Compatibility of preferences	**compatible**

Table A.7: Characterisation of the *Contract net protocol.*

A.3.3 The continuous double auction

In the *continuous double auction* (CDA), participants may pose bids or asks (see subsection 2.2.2.2). The bids and asks are continuously matched by a neutral auctioneer according to a set of market rules. The rules basically state that bids and asks are matched if the price of the bid is higher than the price of the ask.[32]

The characterisation of the CDA is as follows (see table A.8). The CDA is able to handle a changing set of agents: Buyers and sellers may join the auction at any time,

[31] The only minor difference to an auction is that the CNP does not require that actually two goods are exchanged, i.e., the auctioneer does not request a price in the CNP, but that the task is done. This difference, however, is irrelevant to the protocol itself.

[32] There are actually many different versions of the CDA in use (Friedman 1991). In this subsection, the standard CDA described in (Friedman 1991) is used as a reference.

and buyers and sellers that have been matched leave the auction. Just as the English auction, the interests of buyers and sellers may be opposing, even though only those agents are matched for which a complementary bid or ask exists. The CDA does not consider any global constraints or preferences.

	Classification criteria	Classification
#1	Number of agents involved	**changing**
#2	Compatibility of preferences	**opposing**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**variable**
#5	Size of joint commitments	**2**
#6	Relation of commitments	**non-overlapping** **incomplete**
#7	Role assignment	**all variable**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**no delegation**

Table A.8: Characterisation of the *Continuous double auction.*

The number of joint commitments produced by the CDA is obviously variable because it continuously matches buyers and sellers. The size of the joint commitments, though, is always two: The CDA only matches a single buyer with a single seller. The relation of the agent roles and the joint commitments in the CDA is shown in figure A.3. The joint commitments are non-overlapping because an ask or bid can only be matched once. The CDA, however, does not guarantee that all asks or bids are served (incomplete). For example, if only one ask and one bid are remaining, but the bid is lower than the ask, the two are not matched.

Since agents are continuously matched and it is not clear at the beginning which agent will be matched with which other agent, all roles are variable. The only exception may be a central auctioneer that runs the auction. Just as for the English auction and the CNP, the CDA operates with single actions only. In contrast to the English auction, the CDA allows only one action per agent. Finally, no agent delegates decision autonomy because all agents are matched only according to the specified or a better price.

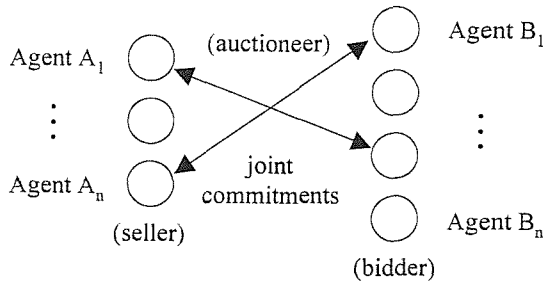

Figure A.3: The decision structure of the continuous double auction.

A.4 Distributed constraint satisfaction

A constraint satisfaction problem consists of variables and constraints between the variables, and the task is to find values for each variable such that these values satisfy the constraints. In *distributed constraint satisfaction* (DCS) problems, variables and constraints are distributed among different agents (Yokoo 2000). Several techniques have been proposed to solve DCS problems. Two examples of DCS algorithms are the *asynchronous backtracking search* (ABS) and the *asynchronous weak-commitment search* (AWS) (Yokoo 2000). In ABS, agents have a unique priority and assign their variables in the order of their (pre-defined) priority. If an inconsistency occurs, the variables with the lower priority are relaxed first. In AWS, the priorities of the agents are dynamically changed according to a min-conflict heuristic, i.e., an agent increases its priority value such that the conflicts with other agents are minimised. Both ABS and AWS are complete in that they eventually search the whole solution space. More efficient, but incomplete algorithms are for example the *distributed breakout algorithm* (Yokoo 2000) and the *constraint partition and co-ordinated reaction* algorithm (Liu 1994).

DCS can be regarded as a distributed decision making approach in which each agent has decision variables and there are constraints (i.e., dependencies) between these variables. The decision space of a decision task is then represented by a variable, and the decision alternatives of this decision task are the values the variable may accept. Dependencies, though, may only be expressed as constraints that rule out certain combinations of alternatives. Preferences cannot be expressed in a DCS problem. In the following, only the ABS protocol is characterised. The characterisation of the AWS protocol is identical.

A.4.1 Asynchronous backtracking search

The ABS protocol is characterised as follows (see table A.9). The ABS protocol

performs a complete search through the space defined by all the variables. Variables, i.e., agents, may thus not be added (or removed) after the search has started. The ABS protocol is not able to handle any preferences. The ABS protocol only searches for a solution satisfying all constraints and stops after the first has been found. However, global constraints can be defined (global preferences cannot – see criterion #2).

Since the ABS protocol is able to solve any decision problem if the solution is only restricted by constraints, the protocol is consequently able to create any number of joint commitments of any size. Likewise, the joint commitments may create any kind of commitment structure (overlapping or not; complete or not). A DCS problem may assign an arbitrary number of roles since each decision variable may include the null action.

The ABS protocol only requires the agents to announce a new value for a decision variable if the constraint net is otherwise unsatisfiable. Agents are therefore only required to provide single alternatives. Since the ABS protocol runs a pre-defined search algorithm, the agents have no control over the overall search process. They may only choose the order in which they announce their decision alternatives.

	Classification criteria	Classification
#1	Number of agents involved	**fixed**
#2	Compatibility of preferences	**constraints**
#3	Global constraints and preferences	**global**
#4	Number of joint commitments	**variable**
#5	Size of joint commitments	**variable**
#6	Relation of commitments	**any**
#7	Role assignment	**all variable**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**partial**

Table A.9: Characterisation of the *Asynchronous backtracking search.*

A.5 Coalition formation

Coalitions in multi-agent systems are groups of agents that co-operate in order to achieve goals they would not be able to achieve individually. The formation of optimal coalitions, however, is very time-consuming because for n agents there are 2^n possible

coalitions. Shehory and Kraus (Shehory 1996) have therefore proposed an approximation algorithm for coalition formation that is distributed among the agents. The algorithm produces an approximation in that it only considers possible coalitions with a limited number of participating agents.

The proposed algorithm consists of three steps. The agents first distribute the responsibility for evaluating a coalition. In a second step, the agents calculate the added benefit for each coalition. Then, in a third step, the best coalition (providing the greatest benefit) is chosen. Since agents may participate in more than one coalition, the agents determine to what extent the newly formed coalition consumes their resources and then repeat the process of coalition formation with an updated evaluation of the remaining coalitions until no more beneficial coalitions can be found.

	Classification criteria	Classification
#1	Number of agents involved	**fixed**
#2	Compatibility of preferences	**compatible**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**variable**
#5	Size of joint commitments	**variable**
#6	Relation of commitments	**overlapping** **incomplete**
#7	Role assignment	**all variable**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**partial**

Table A.10: Characterisation of the Shehory and Kraus *Coalition formation algorithm.*

The characterisation of the corresponding interaction protocol is as follows (see table A.10). The protocol assumes that the set of agents does not change during the coalition formation process. Agents only form a coalition if the coalition is to their mutual benefit. Therefore, the preferences of the agents cannot be opposing. Moreover, the coalition formation algorithm is not able to handle global constraints and preferences. Agents join a coalition solely on the individual basis of their valuation of the coalition. There may be any number of joint commitments (i.e., coalitions). The size of the joint commitments may even differ from coalition to coalition. Furthermore, agents may participate in more than one coalition (i.e., the coalitions may overlap). Since an agent only participates in a coalition if it is to its benefit, the set of joint commitments does

not need to be complete. The roles of the agents depend on the coalition they join. The agents exchange only possible coalitions and their evaluation of them, i.e., interact only on the level of alternatives. Finally, the agents need to delegate their decision autonomy. This is because the protocol prefers (and forms) the coalitions which achieve the best overall benefit, not necessarily the best individual benefit. However, an agent only joins a coalition that is beneficial to it.

Note that the coalition formation approach may be only a partial solution to a decision problem. The coalition formation does assign a set of agents to a decision problem, but it does not specify which agent is going to play which part in the coalition. The coalition formation algorithm of Shehory and Kraus does consider the required resources to fulfil a goal and determines the value of a coalition on the basis of the resources the agents can contribute to a coalition. This, however, may not be sufficient to determine the precise role of each agent in a coalition, in particular if the agents have overlapping capabilities. It may therefore be necessary to complement the coalition formation algorithm by a second assignment step for each coalition.

A.6 Co-ordination of multi-agent plans

All of the interaction protocols discussed in the previous subsections are concerned with co-ordinating the next action of each agent. In some cases, however, it is necessary to look further into the future and to co-ordinate the plans of the agents about their future actions. Several mechanisms for the co-ordination of multi-agent plans have been proposed (see subsection 2.2.2.1). In this subsection, three examples are presented and discussed:

- Partial global planning
- Generalised partial global planning
- Consensus-based distributed planning

A.6.1 Partial global planning

Partial global planning (PGP) was developed to co-ordinate distributed planners for sensory interpretation, each executing its own local plan for how to perform the interpretation of the distributed data (see subsection 2.2.2.1). To achieve the co-ordination of the distributed planners, the agents abstract from their plans and exchange these abstractions. Given the different local plan abstractions, each agent is then able to identify common goals to which the local goals of the agents contribute. Since these common goals may be only partially known to the agents, they are called *partial global goals*. Once a partial global goal has been identified, the local plans can be integrated

into partial global plans. PGP in its original description provides two mechanisms to perform this integration: redundant tasks are avoided, and tasks are performed earlier if this facilitates the work of other agents. In contrast to many other interaction protocols, PGP is therefore an on-going mechanism for global co-ordination.

PGP is characterised as follows (see table A.11). Since the planning process is on-going and intertwined with the execution, agents may join the planning process at any time. However, only co-operative agents may join the process because there is no mechanism in PGP to reconcile opposing interests, whereas the agents are able to construct (partial) global plans and thus optimise the overall system behaviour.

	Classification criteria	Classification
#1	Number of agents involved	**changing**
#2	Compatibility of preferences	**compatible**
#3	Global constraints and preferences	**global**
#4	Number of joint commitments	**variable**
#5	Size of joint commitments	**variable**
#6	Relation of commitments	**any**
#7	Role assignment	**all fixed**
#8	Availability of information	**preferences**
#9	Delegation of decision autonomy	**partial**

Table A.11: Characterisation of the *Partial global planning algorithm.*

The number of joint commitments and their size depends on the global goals (i.e., the dependencies) that are identified. PGP is (theoretically) able to achieve any number of joint commitments. Furthermore, each agent has its own local plans and thus its own role to perform. The plans, however, may be simplified if redundant tasks are eliminated.

Since in PGP the agents also exchange goal representations, they also disclose their preferences about future actions. In so doing, the agents are expected to be co-operative and interested in the overall good of the agent society. Changes are thus not explicitly agreed. It is assumed that an agent agrees to a change if that change improves the overall behaviour.

A.6.2 Generalised partial global planning

Decker (see subsection 2.2.2.1) generalised the partial global planning approach in order for it to be applicable to a wider range of applications (rather than looking only at distributed sensory networks). In comparison to PGP, generalised partial global planning (GPGP) mainly provides (i) general co-ordination relationships, (ii) an approach for detecting these relationships, and (iii) mechanisms to deal with them. Examples of these mechanisms are to communicate results either anytime or only when requested; or to schedule a task earlier if this is required by another agent. The generalisations made, though, do not change the general characterisation of the PGP approach, i.e., GPGP is characterised analogously.

A.6.3 Consensus-based distributed planning

Ephrati and Rosenschein (Ephrati 1996) have designed a distributed planning mechanism for constructing a joint plan through iterative voting about the next state to be achieved. That is, the possible next states are announced and then voted upon by the agents until a full multi-agent plan has been constructed. As a voting protocol for each step, the mechanism uses the Clarke tax voting protocol (see subsection 6.3) in order to ensure that the agents vote sincerely. The characterisation of the planning protocol is therefore identical to the characterisation of the Clarke tax voting protocol. This is due to the fact that the classification scheme does not classify the goal state with respect to the number of actions it consists of.

A.7 Application-specific interaction protocols

Besides the general interaction protocols, also many application-specific interaction protocols have been developed. Even though customised to the specific needs of an application, many of these application-specific interaction protocols may be usable in other applications with similar properties. It is therefore worthwhile to also characterise these interaction protocols even if they are not generally defined. As an example for an application-specific interaction protocol, this subsection characterises the KoWest protocol for work-in-process control which has been developed for a manufacturing control application (see subsection 2.3.3).

A.7.1 Kowest work-in-process control protocol

The Kowest protocol was designed to control the work-in-process of a manufacturing system, i.e., the number of workpieces simultaneously being processed in the system (see subsection 2.3.3). Its main task is to optimise the distribution of workpieces to

machines and to prevent an overflow of material in the manufacturing system. To this end, the protocol manages three aspects of the manufacturing process. First of all, the protocol assigns workpieces to machines through an auction protocol, similar to the contract net protocol described in subsection A.3.2. Secondly, every workpiece is assigned to the virtual buffer of at least one machine. That is, each machine manages a 'virtual buffer' in which each workpiece which has been assigned to the machine and has not yet found a new machine is listed. Workpieces must inform their last machine if they want to leave the virtual buffer (for example, because they have found a new machine). And thirdly, the assignment of workpieces to machines is modified such that a workpiece is not able to move from one machine to the next if the virtual buffer of the next machine is full. The protocol thus stops the flow of material if the buffer capacity is exceeded.

The characterisation of this application-specific protocol is as follows (see table A.12). New workpiece agents are created and join the interaction process as new workpieces enter the manufacturing system. The number of agents involved is thus changing. These agents, however, are all designed to optimise the overall system performance. For instance, the workpiece agents are programmed to wait until one of the possible next machines has an empty buffer space. The work-in-process control limits the overall number of workpieces simultaneously in the manufacturing system, i.e., primarily optimises a global performance measure.

The interaction protocol continuously creates joint commitments of size two between a workpiece agent and a machine agent. Since a machine agent may simultaneously have several joint commitments with different workpieces, the set of commitments is overlapping. The roles of the workpieces are fixed because a workpiece must be processed. However, the machine that is able to and will process the workpiece is not fixed and must be determined by the protocol.

In some cases, the agents may communicate their complete decision space. A workpiece agent searching for a suitable next machine, for example, sends all operations to be performed next on the workpiece to each machine agent (for the machine agents to return the maximal number of operations they can currently perform). Workpiece and machine agents act according to the pre-defined set of rules given by the interaction protocol, and thus delegate their decision autonomy.

	Classification criteria	Classification
#1	Number of agents involved	**changing**
#2	Compatibility of preferences	**compatible**
#3	Global constraints and preferences	**global**
#4	Number of joint commitments	**variable**
#5	Size of joint commitments	**2**
#6	Relation of commitments	**overlapping**
#7	Role assignment	**some fixed / some variable**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**partial**

Table A.12: Characterisation of the *Kowest work-in-process control protocol.*

A.8 Simple interaction protocols

The interaction protocols characterised in the previous subsections have all been designed for interaction situations in which a solution to the decision dependencies is not straightforward, but must be determined through the interaction of the agents. In some cases, however, an interaction situation can be resolved in a much simpler way. If, for instance, only one agent is able to perform a specific action and this agent is designed to be helpful because all agents are working towards a common goal, without further ado the required action may simply be requested from this agent. This can be done by a very simple interaction protocol (see below). The advantage of these simple interaction protocols is that they are easier to implement and require less communication than more sophisticated protocols. The simple interaction protocols should thus be preferred if they fulfil all interaction requirements. Consequently, simple interaction protocols should also be characterised, and an example of such a characterisation is given in the following.

A.8.1 Requesting action

Assume that one agent – supposed to perform an action – realises that it requires an auxiliary action from another agent and that it is obvious which agent this will be. Furthermore assume that it is also reasonable to believe that this agent will honour the request in most cases. The agent requiring the auxiliary action may thus employ the

following simple protocol: The agent requiring the action sends a message to the other agent specifying the requested action. The requested agent either answers with an accept message and performs the requested action, or sends a reject. This simple protocol is characterised as follows (see table A.13).

One agent requests an action from a second agent. There are consequently only two agents involved. And these agents must have compatible preferences, otherwise the requested agent will not honour the request. There is only one commitment between the two agents, and both agents have a fixed role (one performing an action that requires an auxiliary action and the other performing the auxiliary action). Since the first agent is only requesting a single action from the second agent, the agents interact on the level of alternatives (execute action or not). And finally, the requested agent is able to reject the request (for whatever reason).

	Classification criteria	Classification
#1	Number of agents involved	**2**
#2	Compatibility of preferences	**compatible**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**2**
#6	Relation of commitments	**-**
#7	Role assignment	**all fixed**
#8	Availability of information	**alternatives only**
#9	Delegation of decision autonomy	**no delegation**

Table A.13: Characterisation of the *Requesting action protocol.*

A.9 Social laws

Shoham and Tennenholtz (Shoham 1995) have considered the possibility of defining *social laws* for the interaction of autonomous agents. A social law is a set of behavioural rules that every agent must obey when it interacts with other agents. The basic idea of social laws is that if all agents follow a certain set of rules, the agent system shows a more coherent behaviour without the need for a lot of communication (ideally, without any communication). Imagine for instance two robots moving along a hallway in opposite direction. If both robots follow the rule to move only on the right

side of a hallway (and assuming the hallway is wide enough), the robots do not collide and they achieve this without having exchanged a single message. Naturally, the social laws necessary to create coherent behaviour are different for each application – there is no universal set of social laws. This is mainly due to the fact that coherence has a different meaning for each application.

With respect to the goal of this section, i.e., the selection of an interaction protocol for a given decision dependency, the idea of social laws is to avoid the need for interaction in the first place. To avoid interaction, the designer must find local decision rules that, when applied in any start situation, always fulfil the non-local constraints and preferences of the decision dependency (cf. subsection 4.2.2.2). The social laws approach can thus be employed as a pre-step to the selection of an interaction protocol. The designer first tries to avoid the interaction by adapting the local decision rules of the agents, and only if this is not possible or cannot be done efficiently, the designer looks for a suitable interaction protocol. Due to the complexity of finding effective social laws (see (Shoham 1995)), however, it can only be expected to find such a set of local decision rules for rather simple situations. Interaction will be more powerful in most cases.

Appendix B

Application of the DACS Methodology to an Industrial Test Case

This appendix presents the application of the DACS design methodology to an industrial test case. The purpose of this test case was twofold. First of all, the test case was supposed to show that the methodology developed in chapter 4 can be successfully applied to an industrial design problem (see also section 5.1). Secondly, the resulting agent-based control design was supposed to be compared with an existing control system in order to assess whether the agent-based solution performs as well as the existing solution. To this end, the agent-based solution was implemented and evaluated with the help of a simulation. To document this test case, this appendix describes the application of the methodology to the corresponding design problem and discusses the results of the evaluation. The test case is an existing car body shop of the DaimlerChrysler truck plant at Wörth, Germany. This shop, called the *backboard welding shop* (BWS), assembles the backboard of the truck driver's cab through the welding of metal sheets.

This appendix is organised as follows. First, section B.1 specifies the control problem associated with the test case. The next three sections then describe the application of each step of the methodology: Section B.2 develops the decision model, section B.3 identifies the agents, and section B.4 selects the necessary interaction protocols. Section B.5 then discusses the results of the test case. Finally, section B.6 summarises the design and evaluation of the test case.

B.1 Specification of the production control problem

This section specifies the production control problem of the BWS test case (according to subsection 4.1). Subsection B.1.1 describes the actual backboard welding shop and its production process. Subsection B.1.2 defines the conditions under which the production process of the BWS should operate. Subsection B.1.3 sketches the control

interfaces of the BWS components. And finally, subsection B.1.4 presents the production goals and requirements for the BWS.

B.1.1 Production system

The *BWS* consists of a *loading station*[33] (station 01 with cells 0101 to 0110), a *check station* (station 02 with cell 0201), a *robot welding station* (station 03 with cells 0301, 0302, 0304, and 0305), and an *unloading station* (station 04 with cell 0401). The transportation of workpieces between stations is conducted by automated guided vehicles (AGVs), each equipped with a pallet for a specific workpiece type. The AGVs move on fixed routes through the shop and may park at any of the cells or at specific parking places (cells 0070 to 0077 belonging to station 00). The overall organisation of the shop with its cells and traffic routes is shown in figure B.1.

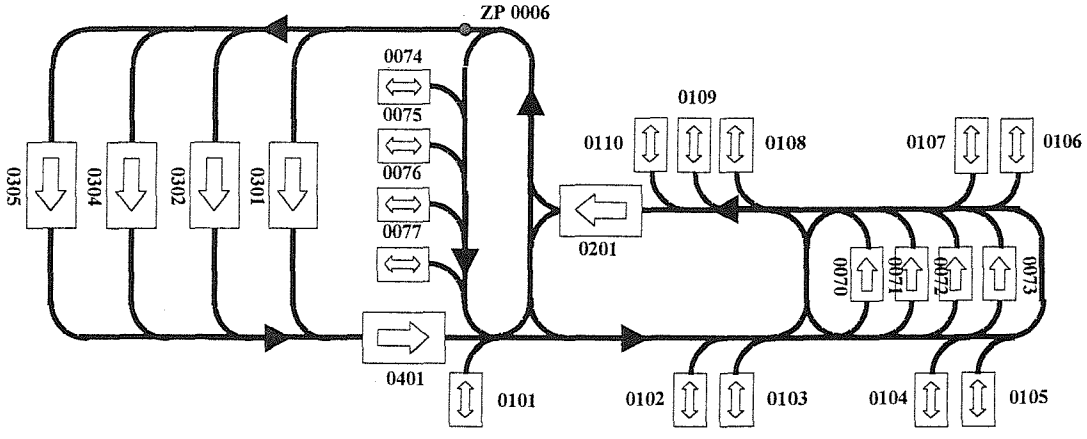

Figure B.1: Layout of the backboard welding shop.

An AGV receives a new job as soon as it has finished its last job, i.e., once the current workpiece has been unloaded at the unloading station. The new job assigned to the AGV must be a job whose workpiece type is compatible with the pallet type of the AGV. To execute the job, the AGV first has to choose a loading cell at which the necessary metal sheets are available. At such a cell, the loading is performed manually by workers which must be requested from a worker pool (see figure B.2). After the loading step, the AGV moves to the check station which verifies that the parts are correctly clamped. Then the AGV moves to the routing point ZP0006 from which it goes to one of the welding cells. After the welding process the AGV leaves the welding cell in order to be unloaded at the unloading station and to receive a new job.

[33] The term *station* is used here as a (physical) collection of (production) cells.

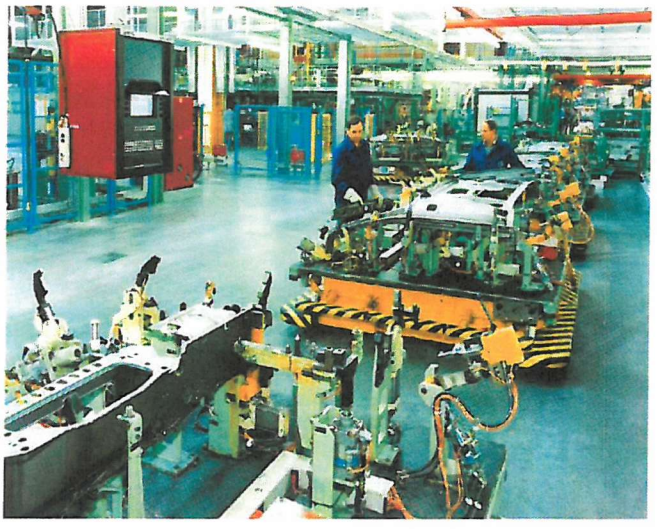

Figure B.2: Manual loading of AGVs.

During the execution of a job, the AGVs may park at various places. Whenever the AGV is waiting for a loading cell or the check cell, it may park at any free loading or parking cell. After the check station it must move directly to the welding station. If that is not possible, the AGV must go through the check station again before it can move to the welding station. At the welding station, an AGV can park before and after a welding cell without blocking other routes.

B.1.2 Production operation conditions

The jobs to be executed by the production system are provided in a job database. The job database specifies, for each job:

- the workpiece type;
- the allowed loading, check, welding, and unloading cells; and
- the required processing time at each cell.

The job database also defines a temporal sequence in which the workpieces should be delivered to the unloading station (see also subsection B.1.4).

The execution of welding jobs may be interrupted because of machine failures or delayed loading. Failures may occur at any automatic cell (i.e., at check, welding, or unloading cells). The mean time to repair is at most 10 minutes, and the overall machine availability at least 90%. The manual loading process at the loading cells may be subject to unforeseen delays. For various reasons, workers may take up to 15% longer than planned to load the necessary parts for a job. AGVs are not subject to failures or malfunctions (more precisely, the system does not need to be robust with respect to failures of the transportation system).

B.1.3 Control interfaces

The backboard welding shop provides the following control interfaces to the physical components of the production system.

- Job database

  The control system can access the job database and retrieve jobs.

- Automatic cells

  The automatic cells (cells 0201, 0301, 0302, 0304, 0305, and 0401) provide sensory information about the state of the cell (whether it is processing or idle, available or disturbed; whether there is an AGV in the station, entering or leaving the cell; and so on). The automatic cells may also be commanded to execute cell-specific operations, such as welding operations at the robot cells.

- Loading and parking cells

  The loading and parking cells only provide sensory information. They report whenever AGVs enter or leave the cell.

- Workers

  Workers report their activities through an on-line reporting system. The system signals whenever the workers start or finish loading an AGV. Furthermore, the workers can be asked to perform a loading operation at a specific cell.

- AGVs

  AGVs report their position as well as entering and leaving cells. AGVs can also be commanded to move to a new destination. The AGVs automatically compute the necessary route and avoid collisions with other AGVs.

B.1.4 Production goals and requirements

The production goals are

- to ***maximise the throughput*** of the shop and
- to ***minimise the deviation*** of the actual delivery sequence from the planned delivery sequence.

The deviation of a job from its planned delivery slot is computed as follows. The job database defines a temporal order in which the workpieces should be delivered to the unloading station. This sequence defines a position for each job. The difference between the actual and the planned position is the deviation of a job. This is an important measure because jobs that deviate from their scheduled position must be

resorted in later production steps in order to reach the final assembly station in the right order.

B.2 Analysis of decision making

This section describes the analysis of the decision making necessary to control the backboard welding shop. According to the analysis step of the methodology (see section 4.2), it is necessary to first identify and characterise the effectoric decision tasks, and then to identify and characterise any dependencies between the decision tasks.

B.2.1 Identification of effectoric decisions

The basic production process of the BWS is as follows (cf. the strategies for identifying effectoric decision tasks in subsection 4.2.1.2):

1. An empty AGV is assigned to a welding job such that the pallet type of the AGV is compatible with the type of the job (i.e., the pallet is able to hold the necessary welding components).
2. The AGV moves to a loading cell at which the necessary components are manually put onto the pallet. The AGV may only move to loading cells which have the components in their local stock.
3. The AGV moves to the check cell at which the components are clamped and the completeness of the components is checked.
4. The AGV moves to a welding cell at which the components are welded to form the backside of the driver's cab, the end product of the job. The AGV may only move to a welding cell which is equipped with the tools to perform the welding operations.
5. The AGV moves to the unloading cell where the part is taken off the AGV via a lift and transported to the final welding shop (assembling all sides of the driver's cab).

In-between steps 1, 2 and 3, the AGV may enter a parking space. If the AGV enters a parking space after step 3, it must redo step 3 before beginning with step 4.

The difficulty about controlling the backboard welding shop is to decide when to go through the production steps for each job such that the end product of the job is delivered on time, but at the same time the throughput is maximised. This is particularly difficult because different jobs may compete for resources and resources may fail at any time.

To control the above production process, the controller has to make the following decisions during the production process:

- D_1 Assign a job to an AGV
- D_2 Choose the next goal cell and departure time for an AGV[34]
- D_3 Assign a worker to a loading cell
- D_4 Select an AGV to cross a junction first

Decision D_1 is necessary to eventually process jobs (see step 1 above). An AGV must be assigned to a job before it can be loaded with components at a loading cell. This decision implicitly decides when to start processing a job (i.e., when to take it from the job data base).

Decision D_2 is the main decision type that enables an AGV to move through the production system. It can be used to choose an appropriate cell for the next production step or to choose a parking place. It thus decides about the physical path that is taken through the production system and when each step is taken. In so doing, it must also decide about which of the alternative cells or parking places to use.

For loading stations, decision D_2 is supplemented by decision D_3, which assigns a worker to a loading cell. Once sufficient workers have arrived at the loading cell, the loading step starts automatically. Note that it is also not necessary to choose a start time at the check cell, the welding cells, or the unloading cell, as these start processing the AGV immediately.

Finally, decision D_4 determines which AGV may go first if two AGVs want to use the same exit at a junction. In such a case, deciding which AGV may go first also determines in which order the AGVs arrive at the next cell (if they have the same cell as the next goal).

Note that there are also three cycles in the transportation system: two around the parking places and one around parking places and check cell. Because of these cycles, an AGV has potentially infinitely many ways of reaching a destination. However, going into an additional cycle does not yield any benefit (if an AGV needs to be buffered or delayed it can be moved to a parking place). Consequently, there is no need to choose between "alternative" routes (for the same destination). Rather, an AGV should always prefer the shortest path.

In the following, the above decisions are specified as decision tasks in tables B.1 to B.4 (according to the scheme outlined in subsection 4.2.1.3).

[34] Note that D_2 is already a generalisation abstracting the actual AGV, the actual cell, and whether the AGV should move to the next cell or to a parking place.

Attribute	Description
id	D_1
title	Assign job to AGV
params	AGV V_i
control interface	job data base, AGV V_i
trigger	V_i is free.
decision space	set of jobs x time
local decision rule	Choose oldest job matching the AGV type and start processing immediately.

Table B.1: Effectoric decision D_1.

Attribute	Description
id	D_2
title	Choose next goal cell and departure time for AGV
params	AGV V_i, cells C_j
control interface	AGV V_i, cells C_j
trigger	Current processing step for job of V_i is finished.
decision space	set of possible cells x time
local decision rule	Choose cell that is able to process the job and is able to do so at the earliest time possible.

Table B.2: Effectoric decision D_2.

Attribute	Description
id	D_3
title	Assign worker to a loading cell
params	AGV V_i, Worker W_j
control interface	AGV V_i, Worker W_j
trigger	AGV entered a loading cell.
decision space	set of workers x time
local decision rule	Choose a worker that is able to load the AGV at the earliest time possible.

Table B.3: Effectoric decision D_3.

Attribute	Description
id	D_4
title	Select AGV at junction
params	AGVs V_1 and V_2
control interface	AGVs V_1 and V_2
trigger	V_1 and V_2 are both heading for the same junction and will choose the same exit.
decision space	$\{V_1, V_2\}$
local decision rule	Choose an AGV randomly.

Table B.4: Effectoric decision D_4.

The trigger diagram of the BWS decision process is given in figure B.3 (see subsection 4.2.1.3). Note that theoretically, D_2 and D_3 are parallel decisions which must be co-ordinated. Practically, however, the workers are assigned after the cell has been chosen because workers are the scarcer resource. Decision tasks D_4, though, is a decision that may or may not be necessary while moving to the next cell. When necessary, it is thus performed in parallel to D_3. Note furthermore that a loading step may require several workers. Consequently, decision task D_3 may have to be performed several times for a single AGV.

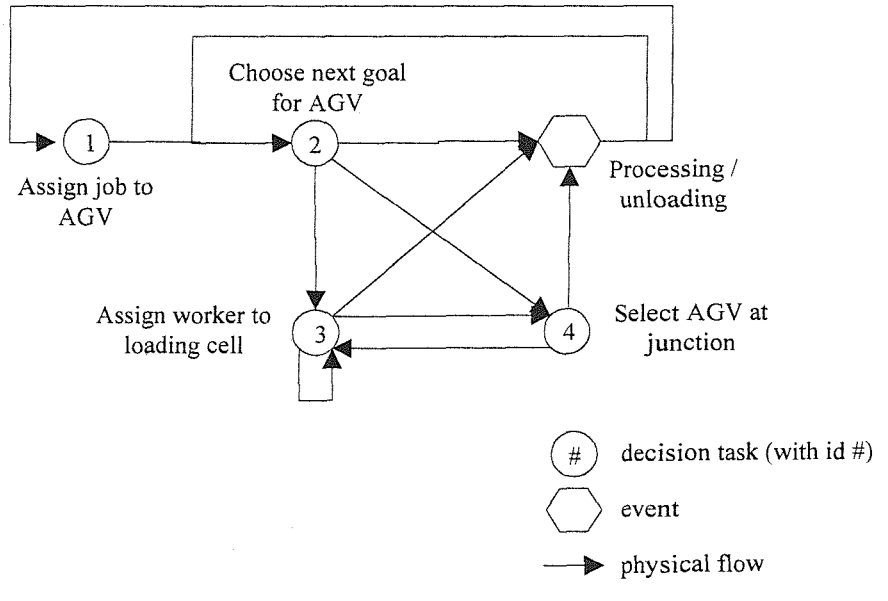

Figure B.3: The trigger diagram for the BWS decision process.

B.2.2 Identification of decision dependencies

An analysis of the production process according to the strategies specified in subsection 4.2.2.1 reveals two types of decision dependencies between the decision tasks of the BWS: the first concerns any resource conflicts, and the second the timing (more precisely, the final sequence of the jobs at the unloading station). Concerning the first type, there are three resources that the jobs are competing for. First, jobs need to be assigned to an AGV. Second, AGVs request processing at cells. And third, AGVs require workers at the loading cells. The resolution of each resource conflict influences the production performance, in particular the throughput. To see this, assume that AGV V_1 can be loaded at cell C_1 or C_2, while AGV V_2 can be loaded at cell C_2 only. If V_1 arrives first and chooses C_2 for the loading operation, V_2 has to wait until the loading of V_1 is finished, whereas if V_1 had chosen C_1, V_2 could have been loaded in parallel at C_2. It is therefore not irrelevant to the production performance how these resource conflicts are resolved. Note however that the consequence of not resolving the resource conflicts is only that a job receives the necessary resource later than it should. All jobs are still processed eventually.

The dependencies, including the corresponding constraints and preferences, are listed in tables B.5, B.6, and B.7 (according to the scheme defined in subsection 4.2.2.2).

Attribute	Description
id	DP_1
title	Jobs compete for AGVs
decision tasks	D_1
constraints	An AGV may only carry one job at a time.
preferences	Assign jobs such that the throughput is maximised.

Table B.5: Dependency DP_1.

Attribute	Description
id	DP_2
title	AGVs compete for loading, check, welding, or unloading cells (or parking places)
decision tasks	D_2
constraints	A cell may only process one AGV at a time.
preferences	Assign AGVs to cells such that the throughput is maximised.

Table B.6: Dependency DP_2.

Attribute	Description
id	DP_3
title	AGVs compete for workers to load AGVs
decision tasks	D_2 and D_3
constraints	Assign sufficient workers to a loading cell which is occupied by an AGV. A worker may only process one AGV at a time.
preferences	Assign workers such that the throughput is maximised.

Table B.7: Dependency DP_3.

Note that most of the non-local constraints and preferences listed above are only qualitatively defined – they are not directly executable. For instance, it is not clear yet how cells should be assigned to AGVs in order to maximise the throughput! Before implementation, these definitions must be transformed into an executable specification. This can be done at this stage or during the design. At this point, it is only important that all relevant dependencies are listed (cf. subsection 4.2.2.1).

The second type of dependency is concerned with the timing of the processing steps. Each job has a deadline for its delivery at the unloading station which is defined by the job data base (see subsection B.1.4). The processing steps must be scheduled such that the job is delivered on time. Since each decision includes a time (or sequence) aspect, this dependency exists between all decisions.

Attribute	Description
id	DP_4
title	Jobs have deadlines for their delivery
decision tasks	D_1, D_2, D_3, and D_4
constraints	-
preferences	Schedule the different steps of each job such that the jobs are delivered on time and in the pre-defined sequence at the unloading station.

Table B.8: Dependency DP_4.

Figure B.4 shows the dependencies between the control decisions of BWS.

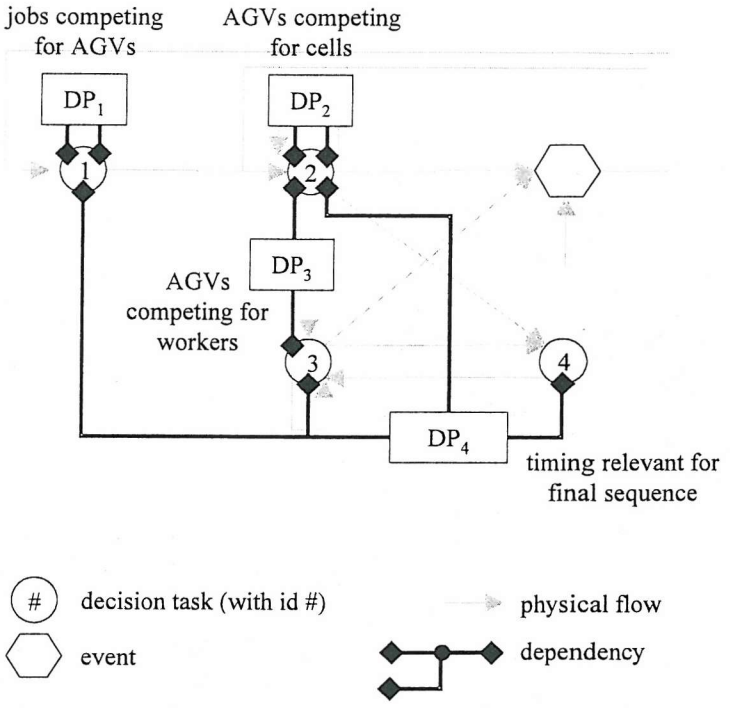

Figure B.4: Dependencies in the BWS decision process.

B.3 Identification of agents

To identify the agents of the control system, the decision tasks identified in the previous section must be clustered according to the clustering rules defined in subsection 4.3.1. However, when applying these clustering rules, the decision model shown in figure B.4 collapses into a single agent. For the same AGV, decision tasks D_1, D_2, and D_4 are assigned to a single agent because all three decision tasks access the control interface of the AGV (see clustering rule for interface cohesion). The resulting clusters (for different AGVs) and decision task D_3, however, are also assigned to the same agent because decision tasks D_2 and D_3 refer to the same cells (see again clustering rule for interface cohesion). Already because of the first clustering rule the decision model collapses into a single agent. It is therefore necessary to modify the decision model first.

B.3.1 Improving the decision model

To prevent the collapse of the decision model, interfaces and component states must be separated by splitting the decision tasks into the corresponding aspects (see subsection 4.3.2.1). The new trigger diagram resulting from the application of the `distribute` operation is shown in figure B.5.

Each of the decision tasks $D_{1\text{-AGV}}$, $D_{2\text{-AGV}}$, $D_{2\text{-CE}}$, $D_{3\text{-AGV}}$, $D_{3\text{-WO}}$, and $D_{4\text{-AGV}}$ are instantiated for each AGV, cell, or worker. Only $D_{1\text{-JOB}}$ exists just once because there is only one job data base. Note that it would be possible to further split decision task $D_{1\text{-JOB}}$ into several sub-decisions for each job in the job data base. Introducing an agent for each job, however, would create many agents without a significant benefit to solving decision task D_1. All job agents would have to co-ordinate their actions whenever an AGV requests a new job (i.e., the job agents would share a very strong interactive coupling). And for all other decisions tasks, the job aspect is identical to the AGV decision aspect. More precisely, the AGV always acts on behalf of the job.

For completeness, the definition of all newly introduced decision aspects is given below (see tables B.9 to B.15). For decision tasks D_2 and D_3, control interface, decision rule, and decision space are distributed; for decision task D_1, control interface and decision rule are distributed; and finally for decision task D_4, only the control interface is distributed (cf. subsection 4.3.2.1).

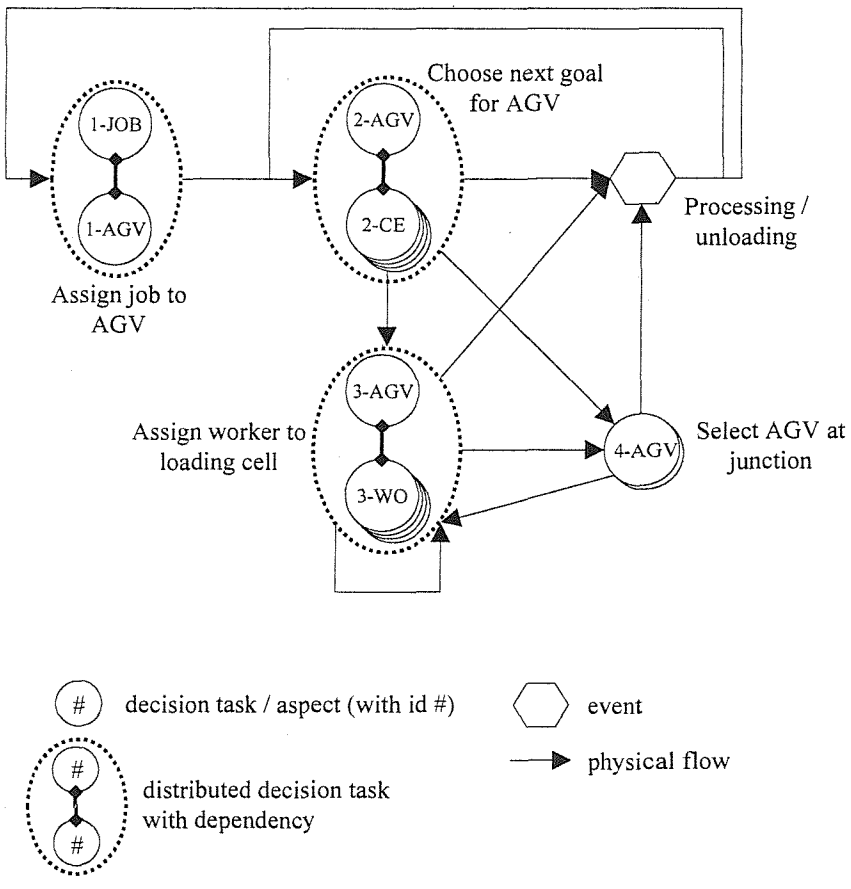

Figure B.5: The adapted decision model.

Attribute	Description
id	$D_{1\text{-JOB}}$
title	Assign job to AGV
params	job data base
control interface	job data base
trigger	Request from $D_{1\text{-AGV}}$.
decision space	set of jobs x time
local decision rule	Choose oldest job matching the AGV.

Table B.9: Effectoric decision $D_{1\text{-JOB}}$.

Attribute	Description
id	$D_{1\text{-AGV}}$
title	Assign job to AGV
params	AGV V_i
control interface	AGV V_i
trigger	V_i is free.
decision space	set of jobs x time
local decision rule	Choose *any* job matching the AGV type and start processing immediately.

Table B.10: Effectoric decision $D_{1\text{-AGV}}$.

Attribute	Description
id	$D_{2\text{-AGV}}$
title	Choose next goal cell and departure time for AGV
params	AGV V_i
control interface	AGV V_i
trigger	Current processing step for job of V_i is finished.
decision space	set of possible cells x time
local decision rule	Choose cell that is able to process the job and is able to do so at the earliest time possible.

Table B.11: Effectoric decision $D_{2\text{-AGV}}$.

Attribute	Description
id	$D_{2\text{-}CE}$
title	Accept AGV for processing
params	Cell C_j
control interface	Cell C_j
trigger	Request from $D_{2\text{-}AGV}$.
decision space	time | {null}
local decision rule	Accept only jobs that the cell is able to process and accept only time slots that are still free.

Table B.12: Effectoric decision $D_{2\text{-}CE}$.

Attribute	Description
id	$D_{3\text{-}AGV}$
title	Assign worker to an AGV at a loading cell
params	AGV V_j
control interface	AGV V_j
trigger	AGV entered a loading cell.
decision space	set of workers x time
local decision rule	Choose a worker that is able to load the AGV at the earliest time possible.

Table B.13: Effectoric decision $D_{3\text{-}AGV}$.

Attribute	Description
id	$D_{3\text{-}WO}$
title	Assign worker to an AGV at a loading cell
params	Worker W_j
control interface	Worker W_j
trigger	Request from $D_{3\text{-}CE}$.
decision space	time | {null}
local decision rule	Accept only time slots that are still free.

Table B.14: Effectoric decision $D_{3\text{-}WO}$.

Attribute	Description
id	$D_{4\text{-AGV}}$
title	Select AGV at junction
params	AGVs V_i
control interface	AGVs V_i
trigger	V_i and another AGV V_j are both heading for the same junction and will choose the same exit.
decision space	$\{V_i, V_j\}$
local decision rule	Choose an AGV randomly.

Table B.15: Effectoric decision $D_{4\text{-AGV}}$.

The distribution of the decision tasks leads to additional decision dependencies which are mainly concerned with ensuring that the distribution of each decision task leads to a mutually accepted control decision in the end (see subsection 4.3.2.1). The new decision dependencies are listed below (see tables B.16 to B.19); the other dependencies still apply.

Attribute	Description
id	DP_5
title	Choose an order for the pallet type of the AGV
decision tasks	$D_{1\text{-JOB}}$ and $D_{1\text{-AGV}}$
constraints	Choose a job that matches the pallet type.
preferences	-

Table B.16: Dependency DP_5.

Attribute	Description
id	DP_6
title	Choose a cell for processing the AGV
decision tasks	$D_{2\text{-AGV}}$ and $D_{2\text{-CE}}$
constraints	The AGV chooses a cell and time that is accepted by the chosen cell.
preferences	-

Table B.17: Dependency DP_6.

Attribute	Description
id	DP_7
title	Assign worker to an AGV at a loading cell
decision tasks	$D_{3\text{-}AGV}$ and $D_{3\text{-}WO}$
constraints	The AGV chooses worker and loading time, and the chosen worker accepts the choice.
preferences	-

Table B.18: Dependency DP_7.

Attribute	Description
id	DP_8
title	Choose the AGV to go first at a junction
decision tasks	$D_{4\text{-}AGV}$
constraints	Both AGVs accept the choice.
preferences	Choose the AGV with the earliest finish time.

Table B.19: Dependency DP_8.

The new decision dependency diagram is shown in figure B.6.

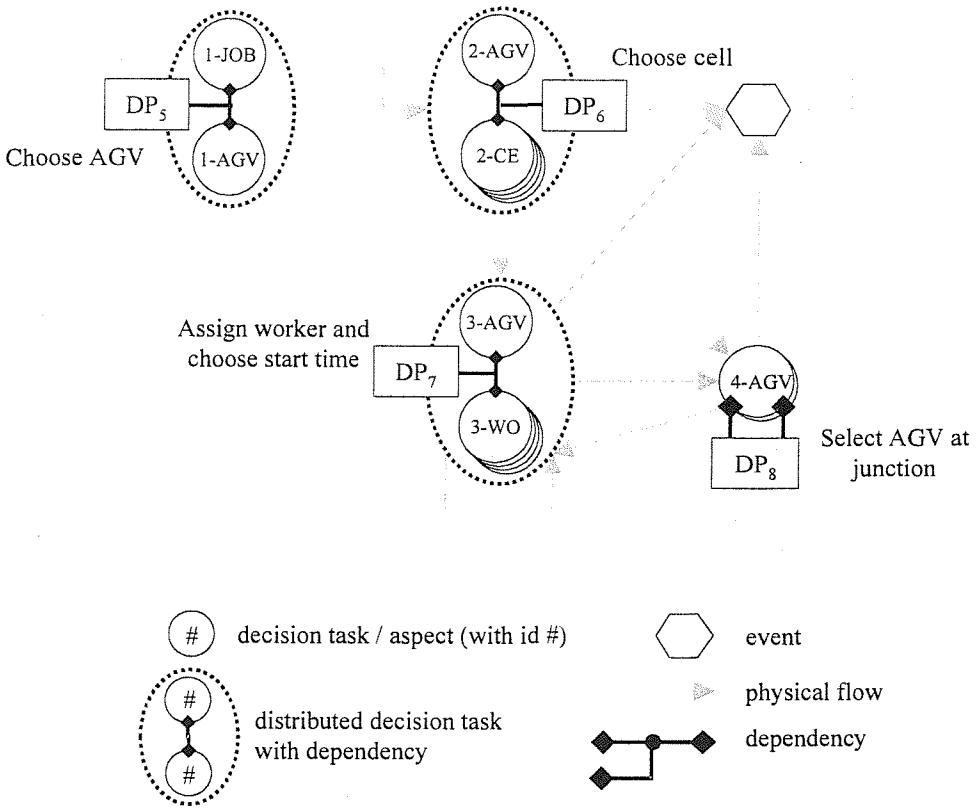

Figure B.6: Additional dependencies after splitting decision tasks.

B.3.2 Clustering of decision tasks

The distribution of decision tasks as shown in figure B.6 is sufficient to prevent the collapse of the decision model into a single agent when applying the clustering rules. Table B.20 lists the resulting agent types with their associated control responsibilities and control interfaces.

Agent type	Control responsibilities	Control interface
job agent	$D_{1\text{-JOB}}$	job data base
AGV agent	$D_{1\text{-AGV}}$, $D_{2\text{-AGV}}$, $D_{3\text{-AGV}}$, $D_{4\text{-AGV}}$	AGV
cell agent	$D_{2\text{-CE}}$	cell
worker agent	$D_{3\text{-WO}}$	(worker)

Table B.20: Agents and associated decision tasks.

Of these agent types, all are instantiated for each production component (i.e., for each AGV, each cell, and each worker). Only the job agent is instantiated just once (namely for the job data base).

B.4 Selection of interaction protocols

The previous section has identified the agents and the decision tasks the agents are responsible for. Some of these decision tasks, however, are dependent on decision tasks which are in the responsibility of other agents. To resolve these dependencies during the control process, the agents must interact. Following the method for selecting interaction protocols (see section 4.4), this section looks at each dependency identified in section B.2 or section B.3 and chooses an appropriate interaction protocol to deal with it.

To identify appropriate interaction protocols, each dependency is classified according to the classification scheme defined in subsection 4.4.1. This classification is then matched against the library of interaction techniques presented in appendix A in order to identify a suitable interaction protocol. Finally, it is verified whether the identified interaction protocol actually resolves the dependency, and the interaction protocol, if necessary, is customised to the particular situation of the dependency (see subsection 4.4.3).

The section starts with the dependencies DP_5 to DP_8 because these must be resolved in order to operate the production system (if dependencies DP_1 to DP_4 are not resolved the production performance degrades, but the production system can still operate).

B.4.1 Dependency DP_5

Attribute	Description
id	DP_5
title	Choose an order for the pallet type of the AGV
decision tasks	$D_{1\text{-}JOB}$ and $D_{1\text{-}AGV}$
constraints	Choose a job that matches the pallet type.
preferences	

Table B.21: Dependency DP_5.

The decision dependency can be classified as follows (see table B.22). The AGV has only constraints on the job to be chosen, whereas the job agent has constraints and preferences on the chosen job. Furthermore, there is a single commitment between the

job agent and the AGV agent, for which both roles are fixed. Finally, the AGV agent is willing to provide any useful information and to delegate the decision if its constraints are fulfilled.

	Classification criteria	Classification
#1	Number of agents involved	**2**
#2	Compatibility of preferences	**compatible**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**2**
#6	Relation of commitments	**-**
#7	Role assignment	**all fixed**
#8	Availability of information	**preferences**
#9	Delegation of decision autonomy	**partial**

Table B.22: Classification of dependency DP_5.

The classification matches best to one of the simple protocols, namely the requesting action protocol (see subsection A.8.1). This protocol is also easily applied to the given dependency: The AGV sends a request for a new job to the job agent, specifying its pallet type. The job agent chooses the oldest order matching the pallet type and sends this order to the AGV.

B.4.2 Dependency DP_6

Attribute	Description
id	DP_6
title	Choose a cell for processing the AGV
decision tasks	$D_{2\text{-}AGV}$ and $D_{2\text{-}CE}$
constraints	The AGV chooses a cell and time that is accepted by the chosen cell.
preferences	

Table B.23: Dependency DP_6.

The decision dependency can be classified as follows (see table B.24). The AGV has constraints and preferences on the cell to be chosen, whereas the cell agent has only

constraints. There is a single commitment between the AGV agent and one of the cell agents. For this commitment, one role is fixed and one is to be assigned. Again, the cell agents are willing to provide any useful information and to delegate the decision if their constraints are fulfilled.

	Classification criteria	Classification
#1	Number of agents involved	**fixed**
#2	Compatibility of preferences	**compatible**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**2**
#6	Relation of commitments	**-**
#7	Role assignment	**1/1**
#8	Availability of information	**preferences**
#9	Delegation of decision autonomy	**partial**

Table B.24: Classification of dependency DP_6.

The classification matches best to the contract net protocol (see subsection A.3.2). According to this protocol, the AGV agent announces the next task to be performed (i.e., loading, checking, welding, or unloading) to the cells that could possibly execute this task. The cells answer with a bid specifying when they could execute this task (offering the earliest start time possible). The AGV then chooses the best bid and informs the winning cell that it requests the offered time slot. The corresponding customisation of the contract net is omitted here since it is rather straightforward.

B.4.3 Dependency DP_7

Attribute	Description
id	DP_7
title	Assign worker to AGV at loading cell
decision tasks	$D_{3\text{-}AGV}$ and $D_{3\text{-}WO}$
constraints	The AGV chooses worker and loading time, and the chosen worker accepts the choice.
preferences	

Table B.25: Dependency DP_7.

Dependency DP_7 is classified analogously to dependency DP_6. Dependency DP_7 may thus also be resolved by a (customised) contract net protocol.

B.4.4 Dependency DP_8

Attribute	Description
id	DP_8
title	Choose the AGV to go first at a junction
decision tasks	$D_{4\text{-}AGV}$
constraints	Both AGVs accept the choice.
preferences	(Choose the AGV with the earliest finish time.)

Table B.26: Dependency DP_8.

The decision dependency can be classified as follows, taking into account that there may be more than two agents that meet at a crossing (see table B.27). The goals of the AGVs may be conflicting, for example if all have to move first in order to meet their deadline. The goals are therefore not compatible and must be classified opposing. There is a single commitment between all agents involved, and each agent has a fixed role (moving its AGV). Despite the conflicting interests, the agents are willing to accept any sequence of movements that optimises the overall production performance.

	Classification criteria	Classification
#1	Number of agents involved	**fixed**
#2	Compatibility of preferences	**opposing**
#3	Global constraints and preferences	**non-local**
#4	Number of joint commitments	**1**
#5	Size of joint commitments	**fixed**
#6	Relation of commitments	**-**
#7	Role assignment	**1/1**
#8	Availability of information	**preferences**
#9	Delegation of decision autonomy	**partial**

Table B.27: Classification of dependency DP_8.

This classification matches best the plurality voting protocol (see subsection A.1.1). Because of the adherence to the overall production goals, however, the agents can resolve the dependency even more simply than through a voting protocol by exchanging the priority of their order and selecting the most urgent order to move first. Since a global delivery sequence is defined for the set of orders, it is straightforward to determine which order has the highest priority.

Each AGV thus sends a detect conflict message (including its own priority) as soon as it detects a possible conflict with another AGV. An AGV receiving a detect message responds also with a detect message (if it has not send one yet). Each AGV then can compute the order of movement by comparing the priorities of the AGVs involved. The AGV with the highest priority may move first, all others must clear the path for this AGV.

B.4.5 Dependencies DP_1, DP_2 and DP_3

Because of their similarities, dependencies DP_1, DP_2 and DP_3 are considered jointly.

Attribute	Description
id	DP_1
title	Jobs compete for AGVs
Decision tasks	$D_{1\text{-JOB}}$ and $D_{1\text{-AGV}}$
Constraints	An AGV may only carry one job at a time.
Preferences	Assign jobs such that the throughput is maximised.

Table B.28: Dependency DP_1.

Attribute	Description
id	DP_2
title	AGVs compete for loading, check, welding, or unloading cells (or parking places)
decision tasks	$D_{2\text{-AGV}}$ and $D_{2\text{-CE}}$
constraints	A cell may only process one AGV at a time.
preferences	Assign AGVs to cells such that the throughput is maximised.

Table B.29: Dependency DP_2.

Attribute	Description
id	DP_3
title	AGVs compete for workers to load AGVs
decision tasks	$D_{2\text{-AGV}}$, $D_{2\text{-CE}}$, $D_{3\text{-AGV}}$, and $D_{3\text{-WO}}$
constraints	Assign sufficient workers to a loading cell which is occupied by an AGV. A worker may only process one AGV at a time.
preferences	Assign workers such that the throughput is maximised.

Table B.30: Dependency DP_3.

The decision dependency DP_1 captures the resource conflicts between orders competing for AGVs. The resolution of these resource conflicts is relevant for the final delivery sequence. If an order is assigned to an AGV too late, it is impossible to deliver the order on time. However, it only makes sense to resolve this dependency if all other dependencies (such as DP_2 and DP_3) are resolved in a similar fashion. Dependency DP_2, for example, can be simply resolved by assigning different slots to the AGVs. An AGV that is processed too late, however, may not be able to finish its order on time. Whether it will finish its order too late, though, depends also on the resource assignment at the following stations and in particular at the last station. The resource conflicts of DP_1, DP_2 and DP_3 should therefore be addressed when resolving DP_4. Formally, this means

that the above dependencies must be resolved with the enlarged scope of DP_1, DP_2, DP_3, and DP_4.

B.4.6 Dependency DP_4

Attribute	Description
id	DP_4
title	Jobs have deadlines for their delivery
decision tasks	$D_{1\text{-JOB}}$, $D_{1\text{-AGV}}$, $D_{2\text{-AGV}}$, $D_{2\text{-CE}}$, $D_{3\text{-AGV}}$, $D_{3\text{-WO}}$, and $D_{4\text{-AGV}}$
constraints	
preferences	Schedule the different steps of each job such that the jobs are delivered on time and in the pre-defined sequence at the unloading station.

Table B.31: Dependency DP_4.

This decision dependency captures the timing dependencies between the different decision tasks $D_{1\text{-JOB}}$ to $D_{4\text{-AGV}}$. The decision dependency can be classified as follows (see table B.32). The AGVs may have conflicts concerning the use of resources (see subsection B.4.5). In particular, it may not be possible that all AGVs meet their deadlines (e.g., in case of disturbances). A global preference is to minimise the deviation from the planned delivery sequence.

Continuously, AGVs (whose roles are fixed) must be assigned to resources (whose roles are not fixed). Some of these assignments also include the workers – the size of the joint commitments is therefore variable. Since the AGVs engage in several joint commitments (one for each processing step), the relation of the commitments is overlapping, but incomplete since a cell is not required to engage in the processing at all.

Despite the conflicting interests, the agents are willing to provide any useful information and to accept any resource assignments that optimises the overall production performance.

	Classification criteria	Classification
#1	Number of agents involved	**changing**
#2	Compatibility of preferences	**opposing**
#3	Global constraints and preferences	**global**
#4	Number of joint commitments	**variable**
#5	Size of joint commitments	**variable**
#6	Relation of commitments	**overlapping** **incomplete**
#7	Role assignment	**some variable**
#8	Availability of information	**preferences**
#9	Delegation of decision autonomy	**partial**

Table B.32: Classification of dependency DP_4.

The classification does not match any of the protocols listed by the methodology perfectly. However, it partly matches the continuous double auction, the asynchronous backtracking search, the (generalised) partial global planning approach, and the KoWest protocol. A thorough comparison of the dependency and the candidate interaction protocols shows that the continuous double auction and the KoWest protocol are not able to resolve such a kind of dependency. The continuous double auction is not able to resolve the dependency because it is not able to handle the global preference of minimising the delivery deviation. The Kowest protocol is not able to resolve the dependency for the same reason (it is able to optimise the work-in-process, but not the delivery sequence). This leaves the asynchronous backtracking search and the partial global planning approach. Because of the design and computational complexity of the asynchronous backtracking search, it is first tried to modify the generalised partial global planning approach.

B.4.6.1 Adaptation of the generalised partial global planning approach

To resolve the timing dependency DP_4 (and simultaneously the resource dependencies DP_1, DP_2, and DP_3), the agents will first of all have to plan ahead. An AGV can only detect that the scheduling of a single processing step will violate the delivery sequence if it also schedules all other, in particular the last processing step (whose finish time is by definition the delivery time). As a consequence, an AGV cannot wait until the last processing step starts in order to detect a violation of the delivery deadline because then it is too late to reschedule previous processing steps (which have already been

executed). The AGVs must therefore plan ahead (or at least estimate the start and finish times of the remaining processing steps).

Secondly, the timing dependencies may cause different AGVs to run into resource conflicts (cf. dependencies DP_1, DP_2, and DP_3 in tables B.28, B.29, and B.30). For example, if an AGV has to reschedule a processing step because the last processing step does not meet the delivery deadline, it must find earlier slots for its processing steps. In a fully utilised production system, however, moving a processing step to an earlier time slot will require that the processing step of another AGV which is occupying the earlier slot is rescheduled. The AGV freeing the processing slot then has to find a new slot for its processing step, potentially violating its own precedence constraints. If so, this AGV will also have to reschedule (some of) its processing steps. The result of one change may therefore be a cascade of changes affecting other AGVs. Consequently, there must be some kind of co-ordination mechanism that determines which AGV should reschedule and which should not, respectively which AGV has priority.

To incorporate the above aspects into an interaction protocol, the protocol for assigning resources, which has been chosen in subsection B.4.2, needs to be extended in two respects. First, the mechanism is used to successively schedule all processing steps of a job once it is assigned to an AGV. Second, co-ordination mechanisms, for example to determine which AGV should reschedule in case of a conflict, are added to the protocol. The resulting interaction protocol is thus an adaptation of the generalised partial global planning approach. For completeness, the actual protocol is sketched in the following.

Every AGV agent schedules its processing steps as soon as it wants to start with the first step. To schedule each processing step, the AGV agent uses the contract net protocol described in subsection A.3.2 (cf. subsection B.4.2). With this protocol, the AGV agent chooses for each processing step an appropriate resource and a time slot during which the processing is scheduled at this resource. When scheduling its processing steps, each AGV agent is responsible for ensuring that there is enough time between two processing steps for the AGV to move to the next cell.

To resolve any resource conflicts, each AGV agent is assigned a priority which corresponds to its due date, i.e., an AGV has a higher priority the earlier its due date is. Thus, when scheduling its processing steps, an AGV may use the processing slots of agents with lower priority. If a slot already assigned to an agent is occupied by an agent with a higher priority, the agent (with the lower priority) must reschedule its processing slot. An AGV agent also reschedules its processing steps whenever the actual execution of a processing step is delayed.

An AGV agent always tries to schedule its last processing step (directly) after the corresponding processing step of the job which is directly preceding itself in the global delivery sequence. If the processing step of this job is scheduled much later than its due

date, the AGV agent tries to find an earlier slot minimising the deviation from its position in the global delivery sequence.

B.4.7 Summary

With the selection of an interaction protocol for each dependency, the design of the BWS control system is now complete. That is, section B.3 identified a set of agent types for the BWS control system (namely job, AGV, cell, and worker agents), and assigned a set of decision tasks to each agent type (see table B.20). Section B.4 selected an interaction protocol for each dependency (see table B.33).

Dependencies	Interaction protocols
DP_5	Requesting action protocol
DP_6	Contract net protocol
DP_7	Contract net protocol
DP_8	Simple protocol
DP_1, DP_2, DP_3, and DP_4	Adaptation of the GPGP approach

Table B.33: Dependencies and interaction protocols resolving the dependencies.

It is now possible to realise each agent independently by implementing its decision tasks and the roles it plays in the interaction protocols resolving its dependencies. The design may even be implemented incrementally, by first implementing only those decision tasks and interaction protocols which are necessary to operate the production system, and then to successively add the remaining decision tasks and interaction protocols in order to improve the production performance. Decision tasks D_1, D_2, and D_3 and the interaction protocols resolving decision dependencies DP_5, DP_6, and DP_7 are absolutely essential to operate the production process, because without these resources are not assigned to jobs. Any decision tasks or dependencies concerning timing aspects can be added later.

B.5 Results from the test case

To evaluate the design developed in the previous sections, the agent-based control system was prototypically implemented in Java and applied to a simulation of the existing backboard welding shop (see figure B.7).

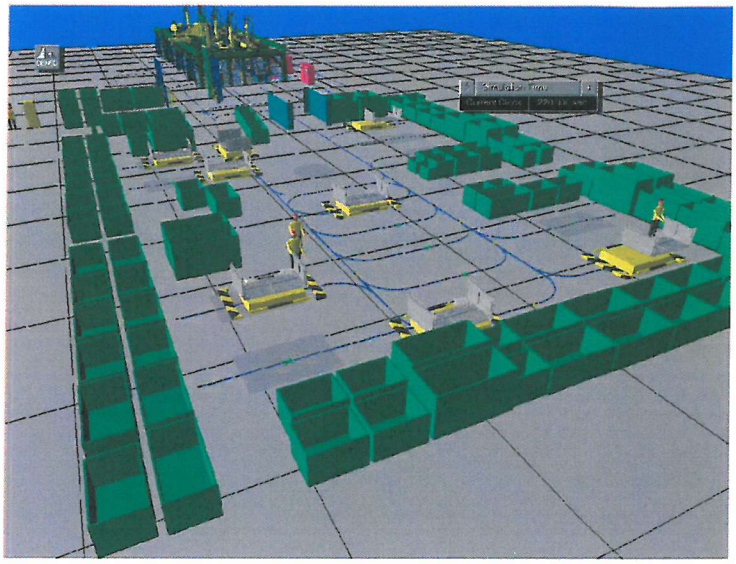

Figure B.7: Simulation screen shot of the BWS.

The prototype was then run with four different job databases provided by the plant, all containing 1700 jobs, and the performance of the control system was measured with respect to the throughput and deviation of the delivery sequence (see subsection B.1.4). With respect to the throughput, the agent-based control system achieved on average 93% of the throughput of the existing control system. Given the fact that the existing control system has been optimised and tailored to the current production process over the years, while the agent-based system was only developed within a feasibility study, a performance of 93% was considered satisfactory (by the plant). With respect to the deviation, the agent-based control system was able to keep most of the deviating jobs within a bound of +/- three positions (see figure B.8). Only 5% of the jobs on average left the BWS outside of this range. A range of +/- three positions was chosen as a goal because the succeeding production steps are able to resort jobs within this range immediately. Jobs outside this range have to go into a longer loop to be resorted.

All in all, it was confirmed by the plant engineers (who have developed the existing control system) that the agent-based solution achieves a performance comparative to the existing control system, but provides a more modular approach to designing control systems. It is therefore envisioned to employ an agent-based approach for the next control system to be developed within a few years. Against this background, it was also stressed that in order to use the technology it would be necessary to have tools to implement an agent-based system within the software systems they currently use to develop and implement control systems (i.e., simulation tools, PLC programming environments, and so on), and that it is indispensable to have a design methodology guiding the engineers in developing agent-based control systems. The DACS design methodology was therefore presented to the engineers and their feedback was already presented in section 5.3.3.

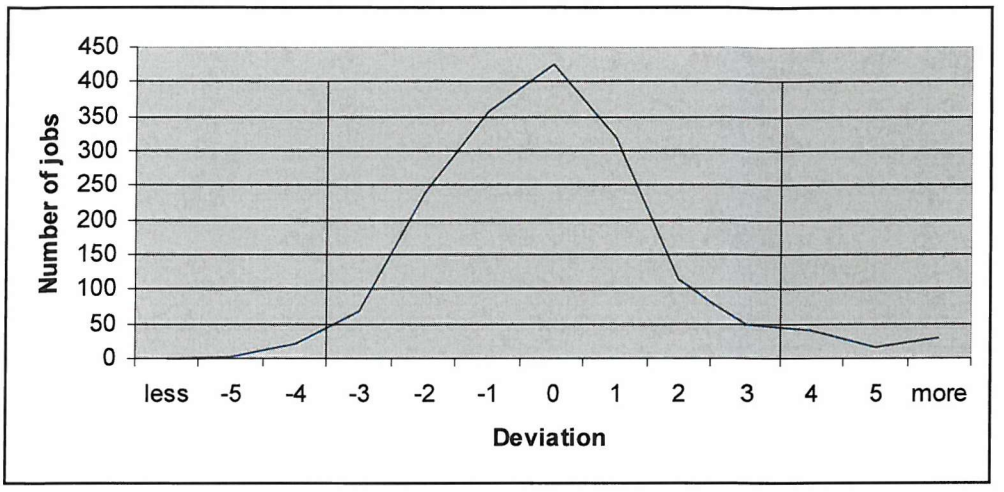

Figure B.8: Deviation in the BWS produced by the agent-based control.

B.6 Summary of the test case

This appendix has presented the successful application of the DACS design methodology proposed in the chapter 4 to an existing industrial production control problem. In particular, the appendix has sketched the specification of the industrial production control problem and has run through each step of the methodology in detail in order to demonstrate how the methodology supports the designer in performing the actual agent-oriented design of the control system. For this industrial application, the major benefits of using the methodology were threefold:

- The production control problem was transformed into a decision model which captures the relevant aspects for the agent-based design.
- The agent identification method prescribed how to modify the decision model in order to identify the control agents. The initial decision model collapsed into a single control agent.
- The method for selecting interaction protocols helped to identify suitable interaction protocols for resolving the dependencies between the control agents. For the timing dependency, there was no interaction protocol in the library which matched the classification of this dependency perfectly, but the method was nevertheless able to point to several similar interaction protocols, one of which was successfully adapted to resolve the dependency.

To summarise, this example application showed that the DACS design methodology can be applied to industrial design problems and that is does support the designer in performing the agent-oriented design steps. Furthermore, the comparison of the design

solution with the existing control system showed that the agent-based approach achieves satisfactory performance, and thus fulfils industrial performance requirements.

Appendix C

Third-Party Reviews

This appendix lists the questionnaires that were used for the third-party reviews and summarises the actual feedback that was given during the reviews. All reviews were held in German and were consequently translated into English.

C.1 Review by Ilka Lehweß-Litzmann

After applying a preliminary version of the DACS methodology to a case study (see subsection 5.3.1), the student Ilka Lehweß-Litzmann was asked to fill out the following questionnaire. The questionnaire refers to a document describing the design methodology.

1. Is the methodology comprehensible?
2. Which parts are difficult to understand? Which are easy to understand? Which parts should be described in more detail?
3. Which terms are unclear? Is it necessary to provide more background information?
4. Does the document provide sufficient examples?
5. Is the approach of the methodology straightforward? Does it make sense? Is the starting point of each design step clear? Is it clear what the result of each design step should be?
6. Are the methods of the methodology sufficiently clear? Is their application straightforward? Are the methods sufficient to perform the design?
7. What is the general impression of the methodology?

Ilka provided detailed answers to each point listed in the questionnaire. Nevertheless, the answer are not reproduced here because they refer to an early version of the methodology which has been significantly changed afterwards.

C.2 Review by Laura Obretin

After applying a preliminary version of the DACS methodology to a case study (see subsection 5.3.2), the student Laura Obretin was asked to fill out the following questionnaire. The answers were recorded by herself, and refer to a document describing the methodology.

1. Is the methodology comprehensible?

   Yes, the methodology is very comprehensible, in particular because many examples are given.

2. Is the approach of the methodology straightforward? Does it make sense?

   Yes. Each transition from one chapter to another is good. In particular, the definitions provided by each part of the description are very helpful.

3. Which aspects are not comprehensible?

   Because of the many examples, all aspects of the methodology are clear and there are no open questions.

4. Which aspects are missing?

   Nothing is missing. Every step is well described.

5. What should be improved?

   A few sentences are too long.

C.3 Review by Schneider Electric

The following feedback was given by Armando Walter Colombo, Ralf Neubert and Boris Süssmann from Schneider Electric after a presentation of the DACS design methodology in its final version (see also subsection 5.3.3.1). The answers were recorded during a discussion about the methodology.

1. Is the methodology comprehensible?

   Yes (three times).

The underlying agent concept is reactive. How can more intelligent agents be designed?

How is the behaviour of the agents described?

How is it possible to take into account experiences / constraints from the implementation? Or feedback from later development phases?

How about worst-case analysis (for example, strategies for breakdowns, control system failures, and so on)?

The concepts of the methodology represent a good transition from the description of a production process to agent concepts. In particular, the starting point of the methodology are the problems that may arise in control design (i.e., it is a problem-oriented methodology).

The consideration or integration of neighbouring or superior software systems, such as ERP systems, are missing.

2. Is the approach of the methodology straightforward? Does it make sense?

   Yes, the approach is comprehensible and well justified. It makes sense and represents a good starting point.

   The methodology is based on a special agent concept, which, however, is probably sufficient for control systems.

   In comparison to other methodologies or work on agent-based system this methodology is more convincing. It is also essential for the technology to advance.

3. Is the design approach sufficient to design all agent-oriented aspects of a control system? What is missing?

   More practical experience with the methodology is missing. There is to date no more validation of the models or an implementation of tools for the methodology (so far only theory, but convincing theory). How can be shown that the methodology works in practice? Validation is very important.

   The constraints on the realisation of a design should be taken into account by the methodology. Is it possible to realise the design? For example, consider real-time or soft real-time constraints.

   It would be helpful if the selection of interaction protocols would also provide information about the "complexity" of the protocols.

   The methodology considers only one aspect of the life-cycle of a production system (namely the design). How about maintenance? How can maintenance be improved? Extensibility is missing.

How does the resulting design fit with other systems (integration with other software or control systems)? There should also be a possibility to interface with other agent architectures / tools.

How can experience be used?

What are the competencies and knowledge (for example about production systems) a designer should have to apply the methodology?

How can system optimisation aspects be integrated (for example, load balancing)?

How can standards or standardised components be supported?

The integration of humans has not been considered yet.

Standard notations, such as UML or SDL, should be used. Standard graphical notations should be used.

4. Is it possible to apply the methodology in your design projects?

   Yes (three times). The methodology is applicable, well thought out.

   It can be imagined to use the methodology at Schneider.

C.4 Review by the DaimlerChrysler

The following feedback was given by Dirk Hofmann and Raimund Krieg from the DaimlerChrysler plant at Wörth after a presentation of the DACS design methodology in its final version (see also subsection 5.3.3.2). The answers were recorded during a discussion about the methodology.

1. Is the methodology comprehensible?

   Yes. The structuring of the design process proposed by the methodology is helpful.

2. Is the approach of the methodology straightforward? Does it make sense?

   The design steps are largely straightforward. The design steps make sense.

   But how about iterative design processes? What are the consequences of changes during the design? The structure proposed by the design methodology may help to reduce the consequences of these changes.

3. Is the design approach sufficient to design all agent-oriented aspects of a control system? What is missing?

   Positive: The design rules are necessary in order to perform the design steps.

The specification of the production control problem is realistic.

The methodology misses a way of assessing whether a chosen interaction protocol can be implemented on the shop floor, and a way to adapt the protocol if not (fine tuning).

4. Is it possible to apply the methodology in your design projects?

   Yes.

   An application would also require tools that are integrated into the existing IT systems.

   How will the agent-based design be transformed into a control system (running in the factory)?

References

AAMAS (2002). *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*. Bologna, Italy.

Agha, G. (1990). Concurrent object-oriented programming. In *Communications of the ACM*, Vol. 33, No. 9, pp. 125 – 141.

Aridor, Y., and Lange, D.B. (1998). Agent design patterns: elements of agent application design. In *Proceedings of the Second International Conference on Autonomous Agents (AA'98)*, pp. 108 – 115. ACM Press: N.Y., NY, USA

Ashworth, C.M. (1988). Structured systems analysis and design method (SSADM). In *Information and Software Technology*, Vol. 30, No. 3, pp. 153 – 163.

Baker, A.D. (1996). Metaphor or reality: A case study where agents bid with actual costs to schedule a factory. In S.H. Clearwater (Ed.), *Market-Based Control*, pp. 184 – 223. World Scientific: London, UK, 1996.

Bartolini, C., Preist, C., and Jennings, N.R. (2003). Architecting for reuse: a software framework for automated negotiation. In Giunchiglia, F., Odell, J., and Weiß, G. (Eds.), *Agent-Oriented Software Engineering III*, LNCS 2585, pp. 88 – 100. Springer-Verlag: Berlin, Germany.

Becker, S., & Parr, R. (1994). Scheduling manufacturing systems. In Dorf, R. C., & Kusiak, A. (Eds.), *Handbook of Design, Manufacturing and Automation*, pp. 529 – 551. John Wiley & Sons: New York, NY, USA.

Bernon, C., Gleizes, M.-P., Peyruqueou, S., & Picard, G. (2002). ADELFE: a methodology for adaptive multi-agent systems engineering. In *Proceedings of the Third International Workshop on Engineering Societies in the Agents World (ESAW'2002)*, Madrid, Spain.

Bond, A.H., & Gasser, L. (Eds.) (1988). *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publ.: San Mateo, CA, USA.

Booch, G. (1991). *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings: Redwood City, CA, USA.

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications.* Addison-Wesley:

Reading, MA, USA.

Booth, A.W. (1998). Object-oriented modeling for flexible manufacturing systems. In *International Journal of Flexible Manufacturing Systems*, Vol. 10, pp. 301 – 314.

Bradshaw, J.M. (1997). An introduction to software agents. In Bradshaw, J.M. (Ed.),*Software Agents*, pp. 3 – 46. AAAI Press/MIT Press: Menlo Park, CA, USA.

Bravoco, R.R., & Yadav, S.B. (1985). A methodology to model the functional structure of an organization. In *Computers in Industry*, Vol. 6, pp. 345 – 361.

Brazier, F.M.T., Jonker, C.M., and Treur, J. (2002). Principles of component-based design of intelligent agents. In *Data and Knowledge Engineering*, Vol. 41, pp. 1 – 28.

Brennan, R.W., & Norrie, D.H. (2003). From FMS to HMS. In Deen, S.M. (Ed.), *Agent-Based Manufacturing – Advances in the Holonic Approach*, pp. 31 – 49. Springer-Verlag: Berlin, Germany.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2001). A knowledge level software engineering methodology for agent-oriented programming. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 648 – 655. Montreal, Canada. ACM, N.Y., NY, USA.

Brooks, R.A. (1986). A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, Vol. 2, No. 1, pp. 14 – 23.

Bryson, J.J. (2002). The behavior-oriented design of modular agent intelligence. In *Workshop Proceedings of the Net.ObjectDAYS Conference*, pp. 142 – 156. Erfurt, Germany.

Budgen, D. (1994). *Software Design*. Addison-Wesley: Wokingham, UK.

Buhr, R.J.A. (1998). Use case maps as architectural entities for complex systems. In *IEEE Transactions on Software Engineering*, Vol. 24, No. 12, 1998.

Burbridge, J.L. (1978). *The Principles of Production Control*. McDonald & Evans, UK.

Burmeister, B., Haddadi, A., & Sundermeyer, K. (1995). Generic configurable cooperation protocols for multi-agent systems. In Castelfranchi, C., & Müller, J.-P. (Eds.), *From Cognition to Action (MAAMAW'93)*, LNAI 957, pp. 157 – 171. Springer-Verlag: Berlin, Germany.

Burmeister, B. (1996). Models and methodology for agent-oriented analysis and design. In Fischer, K. (Ed.), *Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems*, pp. 7 – 17. Document D-96-06. DFKI: Saarbrücken, Germany.

Bussmann, S. (1996). A multi-agent approach to dynamic, adaptive scheduling of material flow. In Perram, J.W., & Müller, J.-P. (Eds.), *Distributed Software Agents and Applications (MAAMAW'94)*, LNAI 1069, pp. 191 – 205. Springer-Verlag: Berlin, Germany.

Bussmann, S. (1998). An agent-oriented architecture for holonic manufacturing control. In *Proceedings of First International Workshop on Intelligent Manufacturing Systems (IMS-Europe 1998)*, pp. 1 – 12. EPFL: Lausanne, Switzerland.

Bussmann, S., & Schild, K. (2000). Self-organizing manufacturing control: an industrial application of agent technology. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*, pp. 87 – 94. Boston, MA, USA.

Bussmann, S., & Sieverding, J. (2001). Holonic control of an engine assembly plant – an industrial evaluation. In *Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference*. Tucson, AZ, USA.

Bussmann, S., & Schild, K. (2001b). An agent-based approach to the control of flexible production systems. In *Proceedings of the 8th IEEE International Conference on Emergent Technologies and Factory Automation (ETFA 2001)*, pp. 481 – 488. Antibes Juan-les-pins, France.

Bussmann, S., Jennings, N.R., & Wooldridge, M. (2001c). On the identification of agents in the design of production control systems. In Ciancarini, P., & Wooldridge, M.J. (Eds.), *Agent-Oriented Software Engineering*, LNCS 1957, pp. 141 – 162. Springer-Verlag: Berlin, Germany.

Bussmann, S., Jennings, N.R., & Wooldridge, M. (2002). Re-use of interaction protocols for agent-based control applications. In *Proceedings of the Third International Workshop on Agent-Oriented Software Engineering* held at the *First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy.

Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearny, P., Stark, J., Evans, R., & Massonet, P. (2002). Agent oriented analysis using Message/UML. In Wooldridge, M.J., Weiß, G., & Ciancarini, P. (Eds.), *Agent-Oriented Software Engineering II*, LNCS 2222, pp. 119 – 135. Springer-Verlag: Berlin, Germany.

Cameron, J.R. (1986). An overview of JSD. In *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, pp. 222 – 240.

Caselli, S., Papaconstantinou, C., Doty, K.L., & Navathe, S. (1992). A structure-function-control paradigm for knowledge-based modeling and design of manufacturing workcells. In *Journal of Intelligent Manufacturing*, Vol. 3, pp. 11 – 30.

Castelfranchi, C., Miceli, M., & Cesta, A. (1992). Dependence relations among autonomous agents. In Werner, E., & Demazeau, Y. (Eds.), *Decentralized AI 3*, pp. 215 – 227. Elsevier Science: The Netherlands.

Castillo, I., & Smith, J.S. (2002). Formal modeling methodologies for control of manufacturing cells: survey and comparison. In *Journal of Manufacturing Systems*, Vol. 21, No. 1, pp. 40 – 57.

Cernuzzi, L., & Rossi, G. (2002). On the evaluation of agent oriented modelling methods. In

Proceedings of the Workshop on Agent-Oriented Methodologies at the *17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 2002). Seattle, WA, USA.

Chokshi, N.N., & McFarlane, D.C. (2002). Rationales for holonic applications in chemical process industries. In Marik, V., Stepankova, O., Krautwurmova, H., & Luck, M. (Eds.), *Multi-Agent Systems and Applications II*, LNAI 2322, pp. 336 – 350. Springer-Verlag: Berlin, Germany.

Christensen, J. (1994). Holonic manufacturing systems: initial architecture and standards directions. In *First European Conference on Holonic Manufacturing Systems*, European HMS Consortium/University of Hannover, Hannover, Germany.

Ciancarini, P., Omicini, A., & Zambonelli, F. (2000). Multiagent system engineering: the coordination viewpoint. In Jennings, N.R., & Lespérance, Y. (Eds.), *Intelligent Agents VI*, LNAI 1757, pp. 250 – 259. Springer-Verlag: Berlin, Germany.

Ciancarini, P., & Wooldridge, M.J. (Eds.) (2001). *Agent-Oriented Software Engineering*, LNCS 1957. Springer-Verlag: Berlin, Germany.

Clancey, W.J. (1985). Heuristic classification. In *Artificial Intelligence*, Vol. 27, pp. 289 – 350.

Coad, P., & Yourdon, E. (1991). *Object-Oriented Analysis.* Prentice-Hall: Englewood Cliffs, NJ, USA.

Coad, P., & Yourdon, E. (1991b). *Object-Oriented Design.* Prentice-Hall: Englewood Cliffs, NJ, USA.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., & Jeremaes, P. (1994). *Object-Oriented Development. The Fusion Method.* Prentice-Hall: Englewood Cliffs, N.J., USA.

Collinot, A. Drogoul, A., & Benhamou, P. (1996). Agent-oriented design of a soccer robot team. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS'96)*, pp. 41 – 47. AAAI Press: Menlo Park, CA, USA.

Collis, J., & Ndumu, D. (1999). ZEUS methodology documentation - the role modelling guide. Technical Report, Release 1.01, *British Telecommunications* plc, UK.

Colombo, A.W., Neubert, R., & Süssmann, B. (2002). A coloured Petri net-based approach towards a formal specification of agent-controlled production systems. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.* Hammamet, Tunisia.

Cossentino, M., & Potts, C. (2002). A CASE tool supported methodology for the design of multi-agent systems. In *Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERO'02).* Las Vegas, NV, USA.

Coulange, B. (1998). *Software Reuse.* Springer-Verlag: London, UK.

Coyle, R.G. (1972). *Decision analysis.* Thomas Nelson and Sons: London, UK.

Crowston, K. (1994). *A Taxonomy of Organizational Dependencies and Coordination Mechanisms.* Technical Report #174, Center for Coordination Science, MIT: Boston, MA, USA.

Dam, K.H., & Winikoff, M. (2003). Comparing agent-oriented methodologies. In *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems* held at the *Second International Joint Conference on Autonomous Agents and Multi-Agent Systems.* Melbourne, Australia.

Dean, T.L., & Wellman, M.P. (1991). *Planning and Control.* Morgan Kaufmann Publ.: San Mateo, CA, USA.

Decker, K.S. (1992). *Environmental Centered Analysis and Design of Coordination Mechanisms.* Phd Thesis, University of Massachusetts, MA, USA.

Decker, K.S., & Lesser, V.R. (1992b). Generalizing the partial global planning algorithm. In *International Journal on Intelligent Cooperative Information Systems*, Vol. 1, No. 2, pp. 319 – 346.

Decker, K.S., & Lesser, V.R. (1995). Designing a family of coordination algorithms. In *Proceedings of the First International Conference on Multi-Agent Systems*, pp. 57 – 63. San Francisco, CA, USA.

Decker, K.S., & Li, J. (2000). Coordinating mutually exclusive resources using GPGP. In *Autonomous Agents and Multi-Agent Systems*, Vol. 3, pp. 133 – 157.

Deen, S.M. (Ed.) (2003). *Agent-Based Manufacturing – Advances in the Holonic Approach.* Springer-Verlag: Berlin, Germany.

DeLoach, S.A., Wood, M.F., & Sparkman C.H. (2001). Multiagent systems engineering. In *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, pp. 231 – 258.

DeMarco, T. (1978). *Structured Analysis and System Specification.* Yourdon Inc.: New York, NY, USA.

Dijkstra, E.W. (1968). The structure of "THE"-multiprogramming system. In *Communications of the ACM*, Vol. 11, pp. 341 – 346.

Dilts, D.M., Boyd, N.P., & Whorms, H.H. (1991). The evolution of control architectures for automated manufacturing systems. In *Journal of Manufacturing Systems*, Vol. 10, No. 1, pp. 79 – 93.

d'Inverno, M., Kinny, D., Luck, M., & Wooldridge, M. (1998). A formal specification of dMARS. In Singh, M., Rao, A., & Wooldridge, M. (Eds.), *Intelligent Agents IV*, LNAI 1365, pp. 155 – 176. Springer-Verlag: Berlin, Germany.

d'Inverno, M., & Luck, M. (2001). *Understanding Agent Systems.* Springer-Verlag: Berlin, Germany.

Dorf, R.C., & Kusiak, A. (Eds.) (1994). *Handbook of Design, Manufacturing and Automation.* John Wiley & Sons: New York, NY, USA.

Dorf, R.C., & Bishop, R.H. (1998). *Modern Control Systems.* Addison-Wesley: Menlo Park, CA, USA.

Dori, D. (2002). *Object-Process Methodology.* Springer-Verlag: Berlin, Germany.

Drogoul, A., & Collinot, A. (1998). Applying an agent-oriented methodology to the design of artificial organizations: A Case Study in Robotic Soccer. In *Autonomous Agents and Multi-Agent Systems*, Vol. 1, No. 1, pp. 113 – 129.

Duffie, N.A., & Piper, R.S. (1987). Non-hierarchical control of a flexible manufacturing cell. In *Robotics and Computer-Integrated Manufacturing*, Vol. 3, No. 2, pp. 175 – 179.

Duffie, N.A., Chitturi, R., & Mou, J.-I. (1988). Fault-tolerant heterarchical control of heterogeneous manufacturing system entities. In *Journal of Manufacturing Systems*, Vol. 7, No. 4, pp. 314 – 327.

Durfee, E.H. (1996). Planning in distributed artificial intelligence. In O'Hare, G.M.P., & Jennings, N.R. (Eds.), *Foundations of Distributed Artificial Intelligence*, pp. 231 – 245. John Wiley & Sons: New York, NY, USA.

Durfee, E.H. (1999). Distributed problem solving and planning. In Weiss, G. (Ed.), *Multi-Agent Systems*, pp. 121 – 164. MIT Press: Cambridge, MA, USA.

Elammari, M., & Lalonde, W. (1999). An agent-oriented methodology: high-level and intermediate models. In *Proceedings of First Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS'99)*, Heidelberg, Germany.

Eurostat Datashop Berlin (2002). *Email communication.* Berlin, Germany.

Fanti, M.P., Maione, B., Piscitelli, G., & Turchiano, B. (1996). System approach to design generic software for real-time control of flexible manufacturing systems. In *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, Vol. 26, No. 2, pp. 190 – 202.

Faratin, P., Sierra, C., Jennings, N.R. (1998). Negotiation decision functions for autonomous agents. In *International Journal of Robotics and Autonomous Systems*, Vol. 24, No. 3-4, pp. 159 – 182.

Faratin, P. (2000). *Automated Service Negotiation Between Autonomous Computational Agents*, Ph.D. Dissertation, University of London, 2000.

Federal Statistical Office of Germany (2001). *Statistical Yearbook 2001.* Wiesbaden, Germany.

Feldmann, K, Schnur, C, & Colombo, W. (1996). Modularised, distributed real-time control of

flexible production cells, Using Petri Nets. In *Control Engineering Practice*, Vol. 4, No. 8, pp. 1067 – 1078.

Feldmann, K., & Colombo, A.W. (1998). Material flow and control sequence specification of flexible production systems using coloured Petri nets. In *International Journal of Advanced Manufacturing Technology*, Vol. 14, pp. 760 – 774.

Ferrarini, L. (1992). An incremental approach to logic controller design with Petri nets. In *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22, No. 3, pp. 461 – 473.

Fichman, R.G., & Kemerer C.F. (1992). Object-oriented and conventional analysis and design methodologies – comparison and critique. In *IEEE Computer*, Vol. 25, No. 10, pp. 22 – 39.

Fischer, K. (1994). The design of an intelligent manufacturing system – a multi-agent system approach. In S.M. Deen (Ed.), *Proceedings of the Second International Working Conference on Cooperating Knowledge Based Systems (CKBS)*, pp. 83 – 99. Dake Centre, University of Keele: Keele, UK.

Fisher, M., Müller, J., Schroeder, M., Staniford G., & Wagner, G. (1997). Methodological foundations for agent-based systems. In *The Knowledge Engineering Review*, Vol. 12, No. 3, pp. 323 – 329.

Fitoussi, D., & Tennenholtz, M. (2000). Choosing social laws for multi-agent systems: minimality and simplicity. In *Artificial Intelligence*, Vol. 119, No. 1-2, pp. 61 – 101.

Frakes, W.B., & Baeza-Yates, R. (Eds.) (1992). *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall: Upper Saddle River, NJ, USA.

Franklin, S., & Graesser, A. (1997). Is it an agent, or just a program?: a taxonomy for autonomous agents. In Müller, J.P., Wooldridge, M.J., & Jennings, N.R. (Eds.), *Intelligent Agents III*, LNAI 1193, pp. 21 – 39. Springer-Verlag: Berlin, Germany.

Friedman, D. (1991). The double auction market institution: A Survey. In Friedman, D., & Rust, J. (Eds.), *The Double Auction Market – Institutions, Theories, and Evidence*, pp. 3 – 25. Addison-Wesley: Reading, MA, USA.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns*. Addison-Wesley: Reading, MA, USA.

Gane, C., & Sarsen, T. (1979). *Structured System Analysis: Tools and Techniques*. Prentice-Hall: Englewood Cliffs, NJ, USA.

Garey, M.R., & Johnson, D.S. (1979). *Computers and intractability*. Freeman: New York, NY, USA.

Gervais, M.-P. (2003). ODAC: An agent-oriented methodology based on ODP. In *Autonomous Agents and Multi-Agent Systems*, Vol. 7, pp. 199 – 228.

Glaser, N. (1997). The CoMoMAS methodology and environment for multi-agent system

development. In Zhang, C., & Lukose, D. (Eds.), *Multi-Agent Systems – Methodologies and Applications*, LNAI 1286, pp. 1 – 16. Springer-Verlag: Berlin, Germany.

Gomaa, H. (1984). A software design method for real-time systems. In *Communications of the ACM*, Vol. 27, No. 9, pp. 938 – 949.

Groover, M.P. (1987). *Automation, Production Systems, and Computer Integrated Manufacturing*. Prentice Hall: Englewood Cliffs, NJ, USA.

Haddadi, A. (1995). *Communication and Cooperation in Agent Systems*, LNAI 1056. Springer-Verlag: Berlin, Germany.

Hahndel, S., Fuchs, F., & Levi, P. (1996). Distributed negotiation-based task planning for a flexible manufacturing environment. In Perram, J.W., & Müller, J.-P. (Eds.), *Distributed Software Agents and Applications (MAAMAW'94)*, LNAI 1069, pp. 179 – 190. Springer-Verlag: Berlin, Germany.

Hartley, J. (1984). *FMS at Work*. IFS Publ.: Kempston, Bedford, UK.

Hatvany, J. (1985). Intelligence and cooperation in heterarchic manufacturing systems. In *Robotics and Computer-Integrated Manufacturing*, Vol. 2, No. 2, pp. 101 – 104.

Hayashi, H. (1993). The IMS international collaborative program. In *Proceedings of the 24th ISIR*, Japan Industrial Robot Association.

Hitomi, K. (1994). Analysis and design of manufacturing systems. In Dorf, R.C., & Kusiak, A. (Eds.), *Handbook of Design, Manufacturing and Automation*, pp. 405 – 433. John Wiley & Sons: New York, NY, USA.

Hoitsch, H.-J. (1993). *Produktionswirtschaft*. Verlag Franz Vahlen: München, Germany. (in German)

Hollmann, O., Ranze, K.C., Müller, H.J., & Herzog, O. (2000). Conflict resolution in distributed assessment situations. In Müller, H.J., & Dieng, R. (Eds.), *Computational Conflicts*, pp. 182 – 201. Springer-Verlag: Berlin, Germany.

Holonic Manufacturing Systems Consortium (2002). *Web site*. 'http://hms.ifw.uni-hannover.de/'.

Huhns, M.N., & Singh, M.P. (Eds.) (1998). *Readings in Agents*. Morgan Kaufman Publ.: San Francisco, CA, USA.

Huhns, M.N. (2001). Interaction-oriented programming. In Ciancarini, P., & Wooldridge, M.J. (Eds.), *Agent-Oriented Software Engineering*, LNAI 1957, pp. 29 – 44. Springer-Verlag: Berlin, Germany.

Hußmann, H. (1997). *Formal Foundations for Software Engineering Methods*. LNCS 1322, Springer-Verlag: Berlin, Germany.

IDEF (1993). *Integration Definition for Function Modeling (IDEF0)*. Draft Federal

Information Processing Standards Publication 183. At http://www.idef.com or as FIPS 183 at http://www.itl.nist.gov/fipspubs.

Iglesias, C.A., Garijo, M., Gonzalez, J.C., & Velasco, J.R. (1998). Analysis and design of multiagent systems using MAS-CommonKADS. In Singh, M.P., Rao, A., & Wooldridge, M.J. (Eds.), *Intelligent Agents IV (ATAL'97)*, LNAI 1365, pp. 314 – 327. Springer-Verlag: Berlin, Germany.

Iwata, K., & Onosato, M., & Koike, M. (1994). Random manufacturing systems: a new concept of manufacturing systems for production to order. In *Annals of the CIRP*, Vol. 43, No. 1, pp. 379 – 383.

Jackson, M.A. (1975). *Principles of Program Design.* Academic Press: London, UK.

Jackson, M.A. (1983). *System Development.* Prentice-Hall: London, UK.

Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach* Addison-Wesley.

Jennings, N.R. (1996). Coordination techniques for distributed artificial intelligence. In O 'Hare, G.M.P., & Jennings, N.R. (Eds.), *Foundations of Distributed Artificial Intelligence*, pp. 187 – 210. John Wiley & Sons: New York, NY, USA.

Jennings, N.R., & Wooldridge, M. (1998). Applications of intelligent agents. In Jennings, N.R., & Wooldridge, M.J. (Eds.), *Agent Technology – Foundations, Applications, and Markets*, pp. 3 – 28. Springer-Verlag, Berlin, Germany.

Jennings, N.R. (2000). On agent-based software engineering. In *Artificial Intelligence*, Vol. 117, pp. 277 – 296.

Jennings, N.R. (2001). An agent-based approach for building complex software systems. In *Communications of the ACM*, Vol. 44, No. 4, pp. 35 – 41.

Jennings, N.R., Faratin, P., Lomuscio, A.R., Parsons, S., Sierra, C., & Wooldridge, M. (2001b). Automated negotiation: prospects, methods and challenges. In *International Journal of Group Decision and Negotiation*, Vol. 10, No. 2, pp. 199 – 215.

Johnson, R.A. (2000). The ups and downs of object-oriented systems development. In *Communications of the ACM*, Vol. 43, No. 10, pp. 69 – 73.

Juan, T., Pearce, A., & Sterling, L. (2002). ROADMAP: extending the Gaia methodology for complex open systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pp. 3 – 10. Bologna, Italy.

Juan, T., Sterling, L., & Winikoff, M. (2003). Assembling agent oriented software engineering from methodologies. In Giunchiglia, F., Odell, J., and Weiß, G. (Eds.), *Agent-Oriented Software Engineering III*, LNCS 2585, pp. 198 – 209. Springer-Verlag: Berlin, Germany.

Kaindl, H. (1999). Difficulties in the transition from OO analysis to design. In *IEEE Software*,

Sept./Oct, pp. 94 – 102.

Kendall, E.A., Malkoun, M.T., and Jiang, C.H. (1997). Design patterns for the development of multiagent systems. In Zhang, C., & Lukose, D. (Eds.), *Multi-Agent Systems – Methodologies and Applications*, LNAI 1286, pp. 17 – 31. Springer-Verlag: Berlin, Germany.

Kendall, E.A., and Malkoun, M.T. (1997). A methodology for developing agent based systems. In Zhang, C., & Lukose, D. (Eds.), *Distributed Artificial Intelligence – Architecture and Modelling*, LNAI 1087, pp. 85 – 99. Springer-Verlag: Berlin, Germany.

Kendall, E.A. (1998). Agent roles and role models: new abstractions for multiagent system analysis and design. In *International Workshop on Intelligent Agents in Information and Process Management*, at the *German Conference on Artificial Intelligence*, Bremen, Germany.

Kendall, E.A., Krishna, P.V.M., Pathak, C.V., and Suresh, C.B. (1998). Patterns of intelligent and mobile agents. In *Proceedings of the Second International Conference on Autonomous Agents (AA'98)*, pp. 92 – 99. ACM Press: New York, NY, USA

Kendall, E.A. (2001). Agent software engineering with role modelling. In Ciancarini, P., & Wooldridge, M.J. (Eds.), *Agent-Oriented Software Engineering*, LNAI 1957, pp. 163 – 169. Springer-Verlag: Berlin, Germany.

Kinny, D., Georgeff, M., & Rao, A. (1996). A methodology and modelling technique for systems of BDI agents. In Van de Velde, W., & Perram, J.W. (Eds.), *Agents Breaking Away (MAAMAW'96)*, LNAI 1038, pp. 56 – 71. Springer-Verlag: Berlin, Germany.

Kinny, D., & Georgeff, M. (1997). Modelling and design of multi-agent systems. In Müller, J.P., Wooldridge, M.J., & Jennings, N.R. (Eds.), *Intelligent Agents III (ATAL'96)*, LNAI 1193, pp. 1 – 20. Springer-Verlag: Berlin, Germany.

Koestler, A. (1989). *The Ghost in the Machine.* Arkana Books.

Korson, T., & McGregor, J.D. (1990). Understanding object-oriented: a unifying paradigm. In *Communications of the ACM*, Vol. 33, No. 9, pp. 40 – 60.

Krueger, C.W. (1992). Software reuse. In *ACM Computing Surveys*, Vol. 24, No. 2, pp. 131 – 183.

Lind, J. (2001). *Iterative Software Engineering for Multiagent Systems – The MASSIVE Method.* Springer-Verlag: Berlin, Germany.

Lind, J. (2003). Patterns in agent-oriented software engineering. In Giunchiglia, F., Odell, J., and Weiß, G. (Eds.), *Agent-Oriented Software Engineering III*, LNCS 2585, pp. 47 – 58. Springer-Verlag: Berlin, Germany.

Maley, J. (1988). Managing the flow of intelligent parts. In *Robotics and Computer-Integrated Manufacturing*, Vol. 4, No. 3-4, pp. 525 – 530.

Malone, T.W., & Crowston, K. (1994). The interdisciplinary study of coordination. In *ACM Computing Surveys*, Vol. 26, No. 1, pp. 87 – 119.

Malone, T.W., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., Quimby, J., Osborne, C., Bernstein, A., Herman, G., Klein, M., & O'Donnell, E. (1999). Tools for inventing organizations: toward a handbook of organizational processes. In *Management Science*, Vol. 45, No. 3, pp. 425 – 443.

Marik, V., Fletcher, M., & Pechoucek, M. (2002). Holons & agents: recent developments and mutual impacts. In Marik, V., Stepankova, O., Krautwurmova, H., & Luck, M. (Eds.), *Multi-Agent Systems and Applications II*, LNAI 2322, pp. 233 – 267. Springer-Verlag: Berlin, Germany.

Marik, V., McFarlane, D., & Valckenaers, P. (Eds.) (2003). *Holonic and Multi-Agent Systems for Manufacturing*, LNAI 2744. Springer-Verlag: Berlin, Germany.

Markus, A., Kis, T., Vancza, J. & Monostori, L. (1996). A market approach to holonic manufacturing. In *Annals of the CIRP*, Vol. 45, No. 1, pp. 433 – 436.

Martin, J., & Leben, J. (1989). *Strategic Information Planning Methodologies.* Prentice-Hall: Englewood Cliffs, NJ, USA.

Martin, J., & McClure, C. (1985). *Diagramming Techniques for Analysts and Programmers.* Prentice-Hall: Englewood Cliffs, NJ, USA.

Maturana, F.P., & Norrie, D.H. (1996). Multi-agent mediator architecture for distributed manufacturing. In *Journal of Intelligent Manufacturing*, Vol. 7, pp. 257 - 270.

McFarlane, D.C. (1995). Holonic manufacturing systems in continuous processing: concepts and control requirements. In *Advanced Summer Institute '95 on Life Cycle Approaches to Production Systems*, pp. 273 – 282. Lisbon, Portugal.

McFarlane, D.C., & Bussmann, S. (2000). Developments in holonic production planning and control. In *International Journal of Production Planning and Control*, Vol. 11, No. 6, pp. 522 – 536.

McFarlance, D.C., & Bussmann, S. (2003). Holonic manufacturing control: rationales, developments and open issues. In Deen, S.M. (Ed.), *Agent-Based Manufacturing – Advances in the Holonic Approach*, pp. 303 – 326. Springer-Verlag: Berlin, Germany.

Meyer, J.-J.C., & Tambe, M. (2002). *Intelligent Agents VIII*, LNAI 2333. Springer-Verlag: Berlin, Germany.

Miles, S., Joy, M., & Luck, M. (2001). Designing agent-oriented systems by analysing agent interactions. In Ciancarini, P., & Wooldridge, M.J. (Eds.), *Agent-Oriented Software Engineering*, LNAI 1957, pp. 171 – 183. Springer-Verlag: Berlin, Germany.

Miles, S., Joy, M., & Luck, M. (2002). Towards a methodology for coordination mechanism selection in open systems. In *Proceedings of the Third International Workshop on*

Engineering Societies in the Agents World (ESAW'2002). Madrid, Spain.

Mili, A., Mili, R., & Mittermeir, R.T. (1998). A survey of software reuse libraries. In*Annals of Software Engineering*, Vol. 5, pp. 349 – 414.

Monden, Y. (1983). *Toyota Production System.* IIE Press, Norcross, GA, USA.

Moore, C.A. (Ed.) (1991). *Automation in the Food Industry.* Blackie and Son: London, UK.

Morisio, M., Ezran, M., & Tully, C. (2002). Success and failure factors in software reuse. In *IEEE Transactions on Software Engineering*, Vol. 28, No. 4, pp. 340 – 357.

Moulin, B., & Cloutier, L. (1994). Collaborative work based on multiagent architectures: a methodological perspective. In Aminzadeh, F., & Jamshidi, M. (Eds.), *Soft Computing: Fuzzy Logic, Neural Networks and Distributed Artificial Intelligence*, pp. 261 – 296. Prentice-Hall: Englewood Cliffs, NJ, USA.

Moulin, B., & Brassard, M. (1996). A scenario-based design method and an environment for the development of multiagent systems. In Zhang, C., & Lukose, D. (Eds.), *Distributed Artificial Intelligence – Architecture and Modelling*, LNAI 1087, pp. 216 – 232. Springer-Verlag: Berlin, Germany.

Müller, J.P. (1996). *The Design of Intelligent Agents - A Layered Approach*, LNAI 1177. Springer-Verlag: Berlin, Germany.

Murata, T. (1989). Petri nets: properties, analysis, and applications. In *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 541 – 580.

Narahari, Y., & Viswanadham, N. (1985). A Petri net apporach to the modeling and analysis of flexible manufacturing systems. In *Annals of Operations Research*, Vol. 3, pp. 449 – 472.

Nwana, H.S., & Ndumu, D.T. (1997). An introduction to agent technology. In Nwana, H.S., & Azarmi, N. (Eds.), *Software Agents and Soft Computing*, LNAI 1198, pp. 3 – 26. Springer-Verlag: Berlin, Germany.

Odell, J.J., Parunak, H.V.D., & Bauer, B. (2001). Representing agent interaction protocols in UML. In Ciancarini, P., & Wooldridge, M.J. (Eds.), *Agent-Oriented Software Engineering*, LNCS 1957, pp. 121 – 140. Springer-Verlag: Berlin, Germany.

O'Hare, G.M.P., & Jennings, N.R. (Eds.) (1996). *Foundations of Distributed Artificial Intelligence.* John Wiley & Sons: New York, NY, USA.

Okino, N. (1993). Bionic manufacturing system. In Peklenik, J. (Ed.), *Flexible Manufacturing Systems Past-Present-Future*, pp. 73 – 95. Faculty of Engineering: Ljubljana, Slowenia / CIRP.

O'Malley, S.A., & DeLoach, S.A. (2002). Determining when to use an agent-oriented software engineering paradigm. In Wooldridge, M.J, Weiß, G., & Ciancairni, P. (Eds.), *Agent-Oriented Software Engineering II*, LNCS 2222, pp. 188 – 205. Springer-Verlag: Berlin,

Germany.

Omicini, A. (2001). SODA: societies and infrastructures in the analysis and design of agent-based systems. In Ciancarini, P., & Wooldridge, M.J. (Eds.), *Agent-Oriented Software Engineering*, LNCS 1957, pp. 185 – 193. Springer-Verlag: Berlin, Germany.

Padgham, L., & Winikoff, M. (2002). Prometheus: A pragmatic methodology for engineering intelligent agents. In *Proceedings of the Workshop on Agent-Oriented Methodologies* at the *17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA 2002). Seattle, WA, USA.

Padgham, L., & Winikoff, M. (2003). Prometheus: a methodology for developing intelligent agents. In Giunchiglia, F., Odell, J., & Weiß, G. (Eds.), *Agent-Oriented Software Engineering III*, LNCS 2585, pp. 174 – 185. Springer-Verlag: Berlin, Germany.

Page-Jones, M. (1988). *The Practical Guide to Structured Systems Design.* Second Edition, Prentice-Hall.

Parnas, D. (1972). On the criteria to be used in decomposing systems into modules. In *Communications of the ACM*, Vol. 15, No. 12, pp. 1053 – 1058.

Parunak, H.V.D., Irish, B.W., Kindrick, J., & Lozo, P.W. (1985). Fractal actors for distributed manufacturing control. In *Proceedings of the Second Conference on AI Applications (CAIA'85)*, pp. 653 – 660. Miami, FL, USA.

Parunak, H.V.D, White, J.F., Lozo, P.W., Judd, R., Irish, B.W., & Kindrick, J. (1986). An architecture for heuristic factory control. In *Proceedings of the American Control Conference*, pp. 548 – 558. Seattle, WA, USA.

Parunak, H.V.D (1987). Manufacturing experience with the contract net. In Huhns, M.N. (Ed.), *Distributed Artificial Intelligence*, pp. 285 – 310. Pitman: London, UK.

Parunak, H.V.D. (1991). Characterizing the manufacturing scheduling problem. In *Journal of Manufacturing Systems*, Vol. 10, No. 3, pp. 241 – 259.

Parunak, H.V.D. (1996). Applications of distributed artificial intelligence in industry. In O'Hare, G.M.P., & Jennings, N.R. (Eds.), *Foundations of Distributed Artificial Intelligence*, pp. 139 – 164. John Wiley & Sons: New York, NY, USA.

Parunak, H.V.D. (1997). Go to the ant: engineering principles from natural multi-agent systems. In *Annals of Operations Research*, Vol. 75, pp. 69 – 101.

Parunak, V., Sauter, J., & Clark, S. (1998). Toward the specification and design of industrial synthetic ecosystems. In Singh, M.P., Rao, A., & Wooldridge, M.J. (Eds.), *Intelligent Agents IV (ATAL'97)*, LNAI 1365, pp. 45 – 59. Springer-Verlag: Berlin, Germany.

Parunak, H.V.D (1999). Industrial and practical applications of DAI. In Weiss, G. (Ed.), *Multi-Agent Systems*, pp. 377 – 421. MIT Press: Cambridge, MA, USA.

Parunak, H.V.D. (2000). A practitioners' review of industrial agent applications. In

Autonomous Agents and Multi-Agent Systems, Vol. 3, pp. 389 – 407.

Pruitt, D.G. (1981). *Negotiation Behaviour.* Academic Press: New York, NY, USA.

Rao, A.S., & Georgeff, M.P. (1992). An abstract architecture for rational agents. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pp. 439 – 449. Boston, MA, USA.

Ritter, A., Baum, W., Höpf, M., & Westkämper, E. (2002). Agentification for production systems. In *Second International Workshop on Integration of Specification Techniques for Applications in Engineering (INT 2002)* at the *European Joint Conference on Theory and Practice of Software (ETAPS 2002).* Grenoble, France.

Robinson, P.J. (1992). *Hierarchical Object-Oriented Design.* Prentice-Hall: Englewood Cliffs, NJ, USA.

Ross, D.T., & Schoman, K.E. (1977). Structured analysis (SA): a language for communicating ideas. In *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, pp. 16 – 34.

Ross, D.T., & Schoman, K.E. (1977b). Structured analysis for requirements definition. In *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, pp. 6 – 15.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object-Oriented Modeling and Design.* Prentice-Hall: Englewood Cliffs, NJ, USA.

Saad, A., Kawamura, K., Biswas, G. (1997). Performance evaluation of contract net-based heterarchical scheduling for flexible manufacturing systems. In *Intelligent Automation and Soft Computing*, Vol. 3, No. 3, pp. 229 – 248.

Sandholm, T.W. (1999). Distributed rational decision making. In Weiss, G. (Ed.), *Multi-Agent Systems*, pp. 201 – 258. MIT Press: Cambridge, MA, USA.

Sarin, S.C, & Das, S.K. (1994). Computer integrated production planning and control. In Dorf, R.C., & Kusiak, A. (Eds.), *Handbook of Design, Manufacturing and Automation*, pp. 483 – 508. John Wiley & Sons: New York, NY, USA.

Scherer, E. (1998). The reality of shop floor control – approaches to systems innovation. In Scherer, E. (Ed.), *Shop Floor Control – A Systems Perspective*, pp. 3 – 26. Springer-Verlag: Berlin, Germany.

Scherer, E. (1998b). Models, systems and reality: knowledge generation and strategies for systems design. In Scherer, E. (Ed.), *Shop Floor Control – A Systems Perspective*, pp. 93 – 125. Springer-Verlag: Berlin, Germany.

Schreiber, G., Wielinga, B.J., de Hoog, R., Akkermans, H., & Van de Velde, W. (1994). CommonKADS: a comprehensive methodology for KBS development. In *IEEE Expert*, Vol. 9, No. 6, pp. 28 – 37.

Shaw, M.J., & Whinston, A.B. (1985). Task bidding and distributed planning in flexible manufacturing. In *Proceedings of the Second Conference on AI Applications (CAIA'85)*,

pp. 184 – 189. Miami, FL, USA.

Shaw, M.J. (1987). A distributed scheduling method for computer integrated manufacturing: the use of local area networks in cellular systems. In *International Journal on Production Research*, Vol. 25, No. 9, pp. 1285 - 1303.

Shoham, Y. (1993). Agent oriented programming. In *Artificial Intelligence*, Vol. 60, No. 1, pp. 51 – 92.

Shoham, Y., & Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. In *Artificial Intelligence*, Vol. 73, No. 1-2, pp. 231 – 252.

Sichman, J.S., Conte, R., Castelfranchi, C., & Demazeau, Y. (1994). A social reasoning mechanism based on dependence networks. In *Proceedings of the 11th European Conference on Artificial Intelligence*, pp. 188 – 192. John Wiley & Sons: Chichester, UK.

Singh, M.P. (1996). Towards interaction-oriented programming. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS'96)*, p. 457. AAAI Press: Menlo Park, CA, USA.

Smith, R.G. (1980). The contract net protocol: high-level communication and control in distributed problem solving. In *IEEE Transactions on Computers*, Vol. C-29, No. 12, pp. 1104 – 1113.

Sommerville, I. (1995). *Software Engineering*, 5th Edition. Addison-Wesley: Harlow, UK.

Sousa, P., & Ramos, C. (1998). A dynamic scheduling holon for manufacturing orders. In *Journal of Intelligent Manufacturing*, Vol. 9, pp. 107 – 112.

Sturm, A., & Shehory, O. (2003). A framework for evaluating agent-oriented methodologies. In *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems* held at the *Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*. Melbourne, Australia.

Suda, H. (1989). Future factory systems formulated in Japan. In *Techno Japan*, Vol. 22, pp. 15 – 25.

Suda, H. (1990). Future factory systems formulated in Japan (2). In *Techno Japan*, Vol. 23, pp. 51 – 61.

Süssmann, B., Colombo, A.W., & Neubert, R. (2002). An agent-based approach towards the design of industrial holonic control systems. In Marik, V., Camarinha-Matos, L.M., & Afsarmanesh, H. (Eds.), *Knowledge and Technology Integration in Production and Services: Balancing Knowledge in Product and Service Life Cycle*, Fifth IFIP/IEEE International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services (BASYS'02), pp. 255 – 262. Cancun, Mexico. (IFIP Conference Proceedings 229, Kluwer 2002)

Sutcliffe, A. (1988). *Jackson System Development.* Prentice-Hall.

Sutcliffe, A.G., Maiden, N.A.M., Minocha, S., & Manuel, D. (1998). Supporting scenario-based requirements engineering. In *IEEE Transactions on Software Engineering*, Vol. 34, No. 12, pp. 1072 – 1088.

Tharumarajah, A. (2003). From fractals and bionics to holonics. In Deen, S.M. (Ed.), *Agent-Based Manufacturing*, pp. 11 – 30, Springer-Verlag: Berlin, Germany.

Tilley, K.J., & Williams, D.J. (1992). Modelling of communications and control in an auction-based manufacturing control system. In *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pp. 962 – 967. Nice, France.

Tilley, K.J. (1996). Machining task allocation in discrete manufacturing systems. In Clearwater, S.H. (Ed.), *Market-Based Control*, pp. 224 – 252. World Scientific: London, UK.

Ueda, K. (1993). A genetic approach toward future manufacturing systems. In Peklenik, J. (Ed.), *Flexible Manufacturing Systems Past-Present-Future*, pp. 211 – 228. CIRP.

Valckenaers, P., Van Brussel, H., Bongaerts, L., & Wyns, J. (1994). Results of the holonic control system benchmark at the KU Leuven. In *Proceedings of the CIM and Automation Technology (CIMAT) Conference*, Rensselaer Polytechnic Institute, New York, NY, USA.

Valckenaers, P., & Van Brussel, H. (1994). Theoretical foundations for preserving flexibility in manufacturing. In *Advanced Summer Institute '94 in Computer Integrated Manufacturing & Industrial Automation (CIMIA).* University of Patras, Greece.

Valckenaers, P., Heikkilä, T., Baumgärtel, H., McFarlane, D.C., & Courtois, J.-P. (1999). Towards a novel manufacturing control principle. In *Proceedings of Second International Workshop on Intelligent Manufacturing Systems (IMS-Europe 1999)*, pp. 871 – 875. Leuven, Belgium.

Van Brussel, H., Bongaerts, L., Wyns, J., Valckenaers, P., & Van Ginderachter, T. (1999). A conceptual framework for holonic manufacturing: Identification of manufacturing holons. In *Journal of Manufacturing Systems*, Vol. 18, No. 1, pp. 35 – 52.

Vancza, J., & Markus, A. (1998). Holonic manufacturing with economic rationality. In *Proceedings of the First International Workshop on Intelligent Manufacturing Systems*, pp. 383 – 394. EPFL: Lausanne, Switzerland.

Van Leeuwen, E.H., & Norrie, D. (1997). Holons and holarchies. In *Manufacturing Engineer*, pp. 86 – 88, April.

Veeramani, D. (1994). Control of manufacturing systems. In Dorf, R. C., & Kusiak, A. (Eds.), *Handbook of Design, Manufacturing and Automation*, pp. 553 – 566. John Wiley & Sons: New York, NY, USA.

Von Martial, F. (1992). *Coordinating Plans of Autonomous Agents*, LNAI 610. Springer-Verlag: Berlin, Germany.

Warnecke, H.J. (1993). *The Fractal Company*. Springer-Verlag: Berlin, Germany.

Weber, T. (1997). Wettbewerbsfähigkeit am Standort Deutschland durch ein innovatives Gesamtkonzept. In Gesellschaft für Fertigungstechnik (Ed.), *Stuttgarter Impulse: Innovation durch Technik und Organisation – FTK'97*, pp. 3 – 27. Springer-Verlag: Berlin, Germany. (in German)

Weiss G. (Ed.) (1999). *Multi-Agent Systems*. MIT Press: Cambridge, MA, USA.

Wirfs-Brock, R., Wilkerson, B., & Weiner, L. (1990). *Designing Object-Oriented Software*. Prentice-Hall: Englewood Cliffs, NJ, USA.

Wirth, N. (1971). Program development by stepwise refinement. In *Communications of the ACM*, Vol. 14, No. 4, pp. 221 – 227.

Wooldridge, M., & Jennings, N.R. (1995). Intelligent agents: theory and practice. In *Knowledge Engineering Review*, Vol. 10, No. 2, pp. 115 – 152.

Wooldridge, M. (1997). Agent-based software engineering. In *IEE Proceedings of Software Engineering*, Vol. 144, pp. 26 – 37.

Wooldridge, M. (1999). Intelligent agents. In Weiss, G. (Ed.), *Multi-Agent Systems*, pp. 27 – 77. MIT Press: Cambridge, MA, USA.

Wooldridge, M., Jennings, N.R., & Kinny, D. (1999b). A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (AA'99)*, pp. 69 – 76. ACM Press: N.Y., NY, USA.

Wooldridge, M., Jennings, N.R., & Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. In *Autonomous Agents and Multi-Agent Systems*, Vol. 3, No 3, pp. 285 – 312.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons: Chichester, UK.

Wooldridge, M., Weiß, G., & Ciancarini P. (Eds.) (2002). *Agent-Oriented Software Engineering II*, LNCS 2222. Springer-Verlag: Berlin, Germany.

Yokoo, M., & Ishida, T. (1999). Search algorithms for agents. In Weiss, G. (Ed.), *Multi-Agent Systems*, pp. 165 – 199. MIT Press: Cambridge, MA, USA.

Yourdon, E., & Constantine, L.L. (1979). *Structured Design*. Prentice Hall: Englewood Cliffs, NJ, USA.

Yourdon, E. (1989). *Modern Structured Analysis*. Yourdon Press.

Yu, E. (2002). Agent-oriented modelling: software versus the world. In Wooldridge, M.J., Weiß, G., & Ciancarini, P. (Eds.), *Agent-Oriented Software Engineering II*, LNCS 2222,

pp. 206 – 225. Springer-Verlag, Berlin: Germany.

Zambonelli, F., Jennings, N.R., & Wooldridge, M. (2001). Organisational rules as an abstraction for the analysis and design of multi-agent systems. In *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11, No. 3, pp. 303 – 328.

Zambonelli, F., Jennings, N.R., Omicini, A., & Wooldridge, M. (2001b). Agent-oriented software engineering for internet applications. In Omicini, A., Zambonelli, F., Klusch, M., & Tolksdorf, R. (Eds.), *Coordination of Internet Agents*, pp. 326 – 346. Springer-Verlag, Berlin: Germany.

Zambonelli, F., & Parunak, H.V.D (2003). Signs of a revolution in computer science and software engineering. In Petta, P., Tolksdorf, R., & Zambonelli, F. (Eds.), *Engineering Societies in the Agents World III*, LNCS 2577, pp. 13 – 28. Springer-Verlag: Berlin, Germany.

Zaytoon, J. (1996). Specification and design of logic controllers for automated manufacturing systems. In *Robotics & Computer-Integrated Manufacturing*, Vol. 12, No. 4, pp. 353 – 366.

Zeigler, B.P. (1984). *Multifaceted Modeling and Discrete Event Simulation.* Academic Press: New York, NY, USA.

Zhang, T.I., Kendall, E., & Jiang, H. (2002). An agent-oriented software engineering methodology with application of information gathering systems for LCC. In *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems* (AOIS-2002) held at the *Fourteenth International Conference on Advanced Information Systems Engineering.* Toronto, Canada.

Zhou, M.-C., DiCesare, F., & Desrochers, A.A. (1992). A hybrid methodology for synthesis of Petri net models for manufacturing systems. In *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 3, pp. 350 – 361.