DYNAMICALLY EVOLVING COORDINATION SOFTWARE ARCHITECTURES FOR PRODUCTION

Rob L.W. van de Weg and Rolf Engmann

Computer Science Department, University of Twente
P.O. Box 217, NL-7500 AE, Enschede, The Netherlands
e-mail: vandeweg@cs.utwente.nl

Abstract: To improve flexibility in the coordination of activities of a computer controlled production system the solution is often sought in developing autonomous software components. However, this solution may lack integration in coordination yielding undesirable consequences such as conflicts between components and unpredictable behavior in the cooperation of components. This paper aims at the development of coordination architectures preserving integration by starting with a proper hierarchical coordination architecture. It can be evolved to an architecture allowing more or complete autonomy of components by delegating coordination (sub)tasks to subordinated components. The evolution can be made dynamic by delegating or revoking authorization to execute a task where and when needed. *Copyright © 2001 IFAC*

Keywords: coordination, architectures, flexible automation, adaptive control, integration

1. INTRODUCTION

It is generally recognized that computer-integrated production has become more and more a factor of survival in the fierce international competition. In particular coordination (inclusive planning and scheduling) and control of the activities of the production system must become more flexible and integrated than it is nowadays (Giebels, 2000). This is due to the need for a higher variety of fashionable or even customer specific products with relatively short life cycles, varying product volumes, and shorter delivery times. On the other hand production coordination must allow for unexpected events or exceptions, such as breakdowns.

The required flexibility can be achieved by defining cooperating, autonomous components of the production system (e.g. agents). The relations between these components define a coordination architecture. In literature several types of coordination architectures for production are proposed (Arentsen, 1995). Our research investigates the properties (inclusive advantages and disadvantages) of these coordination software architecture types. The aim of our research is to be able to point out which software coordination architecture is the most appropriate in a given situation requiring a certain strategy in production.

An architecture can be made even more flexible by the possibility of dynamic adaptation of the architecture to changes in the required production strategy. In this paper we describe our approach to develop such flexible (dynamic) software architectures for production coordination. At the same time a complete overview of the production process is required (integration), e.g. to meet global goals of the company with respect to supply and demand chain management, and order tracking and tracing.

2. COORDINATION ARCHITECTURE TYPES AND COORDINATION STRATEGIES IN PRODUCTION

Defining cooperating, autonomous, communicating software components (e.g. agents) for the coordination of the activities of a production system should improve flexibility in coordination of activities in the production process. The activities of autonomous components itself are coordinated locally, and thus the control of the activities of those components is decoupled from other components. Decoupling leads to the possibility to rearrange relations between components in the system. The advantages of autonomous components are:

- improvement of reactivity due to locally adaptive behavior and exploitation of concurrent components;
- fault tolerance, because in case of exceptions (e.g. breakdowns) of short duration it is not necessary to interrupt the schedules of other components.

On the other hand integration is needed in order to achieve:

- a completely predictable deterministic behavior of the production system;
- meeting global enterprise goals by horizontal integration: supply chain and demand chain management;
- vertical integration of coordination and control, e.g. to obtain order tracing and tracking facilities.

The demands for local autonomy and at the same time for integration of coordination are contradictory. For instance, conflicting goals of different autonomous components may lead to conflicts in cooperation. Also, due to a possible lack of tuning and of anticipation, in the cooperation of autonomous components there may occur emergent behavior. This is in conflict with the required deterministic behavior. Moreover, the required level of autonomy and integration may be time dependent, because the strategy in coordination can change over time.

The operations of hardware components (production units) can be decoupled physically by introducing stores between hardware components. This offers the possibility to decouple the coordination between hardware components because stores serve as buffers or *decoupling points* (DPs) between these hardware components. A DP diminishes the need for synchronization and thus coordination of the operations. A DP makes hardware components less vulnerable to exceptions (e.g. breakdowns) occurring in a decoupled hardware component and supports graceful degradation of the production process in case of longer duration exceptions. Moreover, DPs offer the possibility to use different strategies in production coordination, such as make to order, make to stock, and assemble to order (Zijm, 2000).

The arrangement of relations between (organization of) components in the system defines a coordination architecture. An architecture consists of software components coordinating hardware components, and the relations between the software coordinator components are realized by communication channels. Arentsen (Arentsen, 1995) gives an overview of coordination architectures for production systems:

- The *centralized coordination architecture* consists of just one software coordinator component for a number of controlled hardware components. The coordinator component performs all coordination tasks.
- A *proper hierarchy* consists of a master-slave relationship between software coordinator components and subordinated software coordinator components or hardware components.
- A *modified hierarchy* extends the proper hierarchy with communication channels between subordinated coordinator components at the same level.
- In *heterarchical coordination,* there is no control hierarchy, and all coordinator components can communicate directly.
- A *holarchical architecture* consists of holons. A holon is an autonomous, cooperative agent that may consist of interacting subholons. Holons have goals and may negotiate about shared or conflicting goals. The holarchical architecture is a dynamic (self-organizing) hierarchy.

These latter three architectures should improve the autonomy and the reactivity of the production process components. Arentsen suggests that the order of the architectures above represents an evolution to distributed control.

3. DEVELOPING COORDINATION ARCHITECTURES FOR PRODUCTION

The goal of our approach is to develop flexible software architectures for coordination, such that *autonomous (decoupled) components behave as if they are coordinated*. In our approach it is very important that an autonomous component can anticipate the behavior of other components. Flexibility requires that coordination can be adapted continuously. Therefore, in our approach in a coordination hierarchy the authorization to execute a coordination task and related decision taking can be delegated dynamically to lower levels in the coordination hierarchy, and when integration is needed this authorization can be revoked.

To illustrate the basic concepts in our approach we will describe a small example. Two robots are moving in a narrow straight corridor in a warehouse approaching from opposite directions. The corridor is too narrow for the robots to pass each other. Both robots are provided with detectors used to avoid collisions. They will stop in front of each other and a deadlock situation occurs. To solve this deadlock situation we introduce an arbiter object capable to detect both robots and capable to reverse the direction of the movement of one of the robots for a while.

The decision rule used by the arbiter object to decide which robot will move backwards can be delegated to both robots. To be able to evaluate this rule for each robot, data on the direction in which the robot itself is moving must be available. The possibilities to execute the required operation of a robot are already available to the robots. So, delegation of decision rules implies that the data to evaluate the decision rule and the ability to execute the operations to apply the decision must be available by replication, sharing, or exchange (i.e. communication or transfer). This may result in redundancy in data and evaluation of rules, though controllable. The important advantage in the described situation is that a robot can always anticipate the reaction of other robots. No deliberation is needed, because the same decision rules are shared. One problem in delegation may be that operations are to be operationalised sometimes, because in general lower in the coordination hierarchy the time-scale to react is shorter and coordination tasks are more basic.

Our approach to develop a coordination architecture starts by developing a proper hierarchical architecture in which conflicts are solved and all exceptions that may occur are handled (e.g. caused by failures of controlled hardware). Such a method, a CASE-tool, and

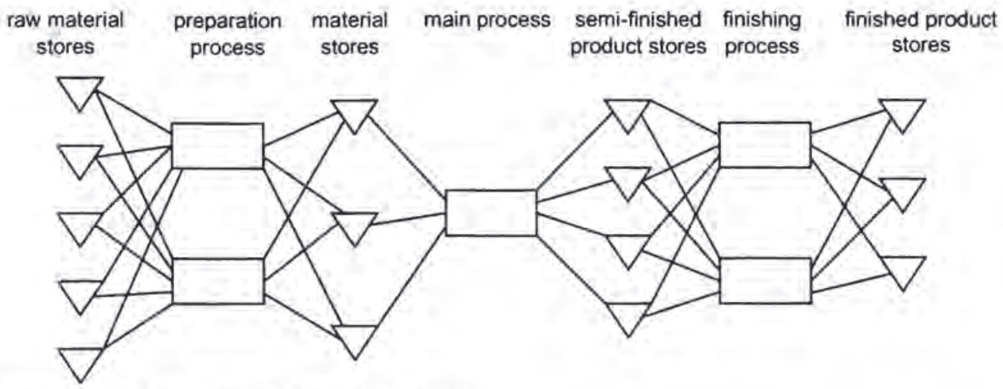

Figure 1. Simplified layout of the example production process

PLC-code generator are described in (Van de Weg, 1997). Arentsen suggests an evolution starting from this proper hierarchical architecture to the other types of coordination architectures offering more autonomy to components. We will use the delegation concept to construct other architecture types. Revocation of delegated responsibilities is used to (re-)construct the appropriate architecture dynamically. In this way the coordination architecture can be adapted dynamically by delegating authorizations when autonomy is needed and revoking authorizations when integrated coordination is desired.

However, first we have to explore what exactly is to be delegated. Based on systems theory we distinguish four subtasks (monitoring, diagnosis, decision taking, scheduling) in coordination and four related data stores (process state, goals, rules, schedule). This distinction is quite obvious. Therefore, it is not very surprising that in (Arentsen, 1995) and (McPherson, 1995) and in the literature on BDI-architecture based agents, e.g. (Ingrand, 1992; Rao, 1992) very similar subtasks and data sets are proposed. For the implementation of these subtasks very specific software techniques are developed (e.g. requiring specific ways of reasoning).

4. PROPER HIERARCHICAL COORDINATION ARCHITECTURE

In Figure 1 the simplified layout of an example production process is depicted. The production process consists of three production steps: preparation process, main process, and finishing process. The preparation process has as input the raw material stores (rm-stores) and as output material stores (m-stores). The main process uses m-stores as input and semi-finished product stores (sfp-stores) as output. Finally, the finishing process uses the sfp-stores as input and the finished product stores (fp-stores) for output. The sets of rm-, m-, sfp-, and fp-stores are decoupling points in the production process. These input and output stores are alternatives or may all be used in an allocated production step. For the preparation and finishing process there exist two production units, which may be used as alternatives or concurrently for the same allocated production step. Data on recipe options, schedules and states (including exceptions and results) for hardware components and stores are to be monitored and logged. Of course tracking and tracing are mandatory.

PROPERTIES OF A PROPER HIERARCHICAL COORDINATION ARCHITECTURE. The traditional structure of a production coordination software architecture is a multi-level control hierarchy. For the decomposition of production coordination into such a hierarchy four principles are used (derived and extended from (Albus, 1981; Jones, 1990):

For the decomposition into levels:

- Separation of concerns (i.e. separation of responsibility and authorization) with respect to different time scales or time horizons;
- Separation of concerns with respect to different levels of abstraction.

For the decomposition into functions at each level:

- As imposed by the business organization (function separation);
- As imposed by the production process due to the distribution of production steps in time, and of hardware components over locations.

The difference between levels is also caused by a further more operational separation of concerns:

- Higher in the hierarchy more aggregated data are needed to study longer-term trends.
- Another consequence of the difference in time scales is that at higher levels in the hierarchy actions performed at lower levels may be considered to be atomic. That means that at a higher level in the hierarchy the effects of shorter-term disturbances of the production process can be neglected and hidden, unless there are longer-term consequences. Also the more operational details are to be hidden.
- Implementation details relevant at a lower level are hidden for a higher level. This introduces software and hardware platforms offering standard interfaces for higher level functions.

At each level the production coordination architecture is to be decomposed into subordinated coordination components reflecting a separation of concerns and functions. The decomposition is strongly based on the existence of DPs. This makes the production process less vulnerable to influences of the business environment and makes a hardware component less depend-

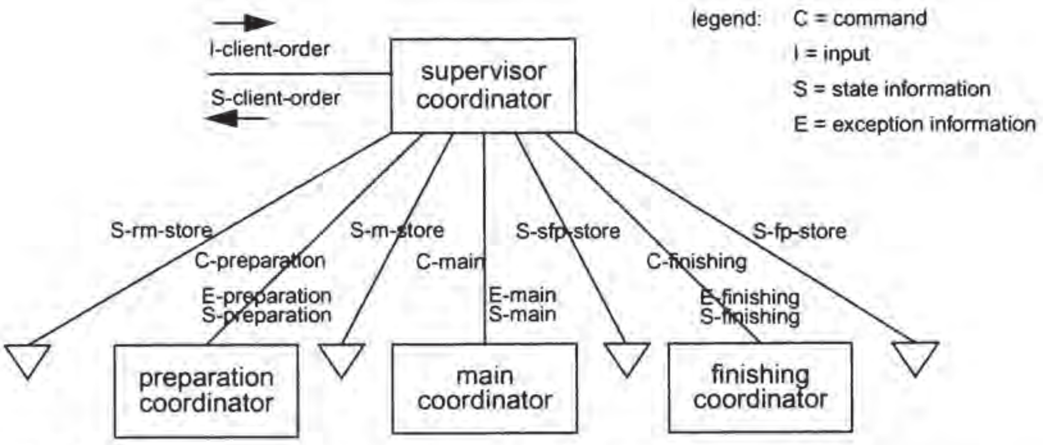

Figure 2. Organization diagram for production coordination according to the make-to-order strategy

able on the activities of the other hardware components. Also function separation is an important reason for decomposition to avoid entangling and even conflicting concerns. Hardware components not sharing the same resources (production units and stores) can be operated concurrently. However, if hardware components are not decoupled the operations must be synchronized.

A very important design heuristic is that the scope of effect of a coordination component must be within the scope of control of that component. The scope of control is defined by all components subordinate to the coordinator component. The scope of effect is defined by the components affected by decisions taken by the coordinator component. If this heuristic does not hold the coordination architecture is not hierarchical (i.e. a two or more bosses problem may occur). For this reason exceptions are always to be dealt with at the lowest possible level.

PROPER HIERARCHICAL COORDINATION ARCHITECTURE FOR MAKE-TO-ORDER STRATEGY. In Figure 2 the organizational model is shown for the proper hierarchical coordination architecture applying the make-to-order strategy. The organizational model specifies in detail how software components cooperate to coordinate the activities of controlled hardware components. The nodes of the diagram show software coordination components; lines represent the communication channels between the coordinator components. The nodes at the lowest level are so-called software base components (not shown in Figure 2), which directly coordinate and control the activities of corresponding hardware components.

In this context the relevant hardware components can independently receive and handle command signals and usually they transmit their status (as perceived by detectors) and exceptions to the coordination system via feedback signals. A base component controls just one specific hardware component; the base component initiates operations to be executed by the corresponding hardware component by sending commands, and monitors via feedback signals received from sensors whether the desired operation is carried out correctly and timely. The actions of base objects are coordinated and controlled by a hierarchy of supervisory software components, the so-called coordinator components or coordinators. Coordinators trigger the execution of base components or other coordinators. Exceptions should be solved at the lowest possible level in the hierarchy.

The make-to-order strategy as applied leads to the coordination task to synchronize operations of the preparation, main, and finishing processes at the same time and to monitor the presence of required (raw) material and semi-finished products in stores. The coordination tasks related to just one production step can be delegated to the preparation, main, or finishing coordinators.

For a make-to-order strategy two substrategies may be applicable. The first one is the exclusive control strategy when a client order has to satisfy very specific quality standards or has the highest priority. In this substrategy the allocation is aiming at the synchronized, consecutive execution of the production steps for the order. When orders do not require exclusivity and thus resources can be shared, batch forming and concurrency are allowed to serve these orders.

The preparation, main, and finishing coordinators each have to control and to log locally the operations, the state and operation results of the subordinated hardware components for tracking and tracing purposes. The supervisor coordinator maintains data restricted to the state of the cooperation of the processes, and of the state of all stores. To meet the tracking and tracing obligation the supervisor coordinator logs at the required level of aggregation the state of the coordinated processes and stores, and the state of client orders.

PROPER HIERARCHICAL COORDINATION ARCHITECTURE FOR MAKE-TO-STOCK, ASSEMBLE-TO-ORDER. Now the original goal to handle client orders in an efficient way is to be split into two sub-goals because of decoupling. The first sub-goal (according to the make-to-stock strategy) is the efficient and optimal usage of resources involved in the cooperation of the preparation and main production steps. The responsibility to maintain the make-to-stock strategy is delegated to a make-to-stock coordinator coordinating the operations of the preparation and main coordinators (see Figure 3). The make-to-stock coordinator has also the responsibility for the rm and m-stores. The input for

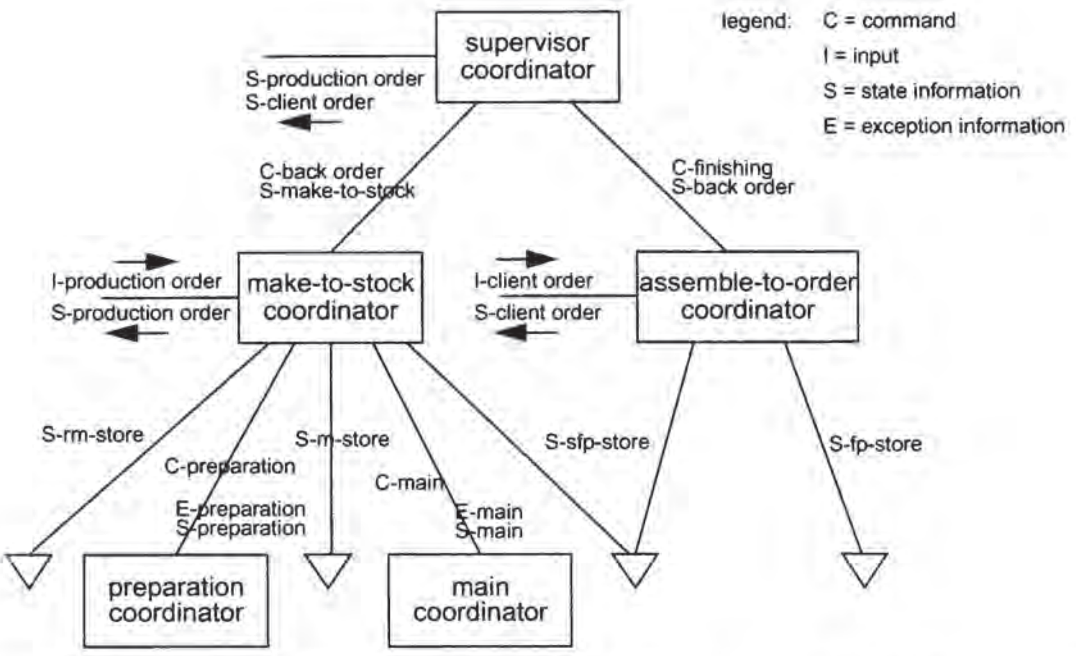

Figure 3. A proper hierarchical coordination architecture combining the make-to-(sfp)-stock with the assemble-to-order production strategies

the make-to-stock coordinator are production orders. The assemble-to-order strategy is to be maintained by the finishing (i.e. assemble-to-order) coordinator. The input for the assemble-to-order coordinator are client orders. Because the responsibility for the sfp-stores should to be shared with the make-to-stock coordinator this responsibility better stays with the supervisor coordinator in order to avoid conflicts.

Batch forming has to be independent for the make-to-stock coordinator and the assemble-to-order coordinator. No tuning is possible. However, tuning to the make-to-stock coordinator may be needed. For instance, in case shortage of semi-finished products threatens the execution of the assemble-to-order production step, such that a number of client orders can not be served in time, a production order (back order) must be generated to prevent the foreseen shortage. Another solution may be the sharing, replication, or exchange of information on the actual and future state of sfp-stores via the supervisor coordinator.

If we decide that back orders from the assemble-to-order coordinator can be communicated directly to the make-to-stock coordinator we have transformed the proper hierarchical coordination architecture to a modified one. This will be at the cost of redundancy in data because the supervisor still needs data, e.g. for tracking and tracing. A drawback of solutions based on a proper hierarchical coordination architecture is that, because they are static, only one strategy is applicable. Also information on stores serving as decoupling points is to be shared somehow. An exclusive control (lock, and unlock mechanism) of stores may be needed. The existence of decoupling points may form an obstacle for integration in case that coordination sub-tasks are delegated to sub-coordinators.

5. EVOLUTION TO MORE AUTONOMY IN COORDINATION

A heterarchical coordination architecture (see Figure 4) offers more autonomy of coordinators. We will start with a proper hierarchical coordination architecture as constructed for the make-to-order strategy. All coordination tasks of the supervisor coordinator are to be delegated to the preparation, main, and finishing coordinators, and to be operationalised for each coordinator. However, the coordination tasks to tune cooperation of hardware components require sharing of stores, and it may be necessary to have a locking mechanism for the use of stores. In particular there may arise conflicts when several coordinators want control of an intermediate store, for instance when exclusive control is needed or when the operations of consecutive production steps are not synchronized.

Scheduling and in particular batch forming may be complicated because a lot of deliberation is needed between coordinators unless decision rules are relatively simple. Also complex exception handling and related decision taking may cause a lot of deliberation when consecutive production step coordinators are involved. An advantage is that it is possible to adapt to different production strategies required for different orders. Possibly the easiest way is that production or client orders are input for the first production step coordinator involved in its production. Forward scheduling can be an appropriate approach in particular when an order has to have input for the first coordinator (make-to-order and make-to-stock). If the contents of input stores are sufficient for processing at the scheduled time the operations of the next coordinator in the sequence can be scheduled, otherwise a back order is to be generated. Another approach can be backward scheduling, which can be more appropriate in case the

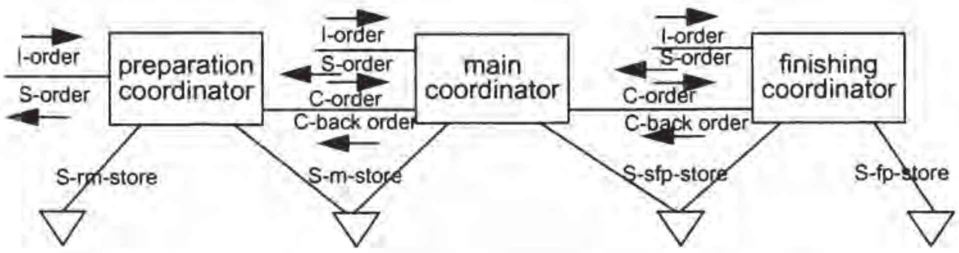

Figure 4. Heterarchical coordination architecture

production strategy is e.g. assemble-to-order. The advantages of a heterarchical coordination architecture applying the delegation mechanism are:

- Push and pull of orders is possible;
- Flexibility because it is possible to cope with a mix of orders requiring different production strategies.

Disadvantages are:

- Deliberation or negotiation is needed to schedule, to form batches and to tune consecutive activities (the operator may prescribe time slots for scheduling orders to be produced according to some strategy);
- Complicated control of intermediate stores;
- Lack of integration with respect to processing as well as data;
- Avoids conflicts in goals when derived from a proper hierarchical coordination architecture, however in the future maintenance may cause conflicts and unpredictable behavior.

6. DYNAMIC EVOLUTION OF COORDINATION ARCHITECTURES

From the previous chapters it is clear that the proper hierarchical coordination architecture usually is the most advantageous architecture when the delegation mechanism is applied, because it avoids redundancy, deliberation and negotiation. However, it is not flexible with respect to different production strategies. Therefore, the most flexible solution would be a temporary proper coordination hierarchy when required for a production strategy. It can be made dynamic because instead of delegating (sub) tasks and related data and operations forever, authorizations can be revoked when needed. That means that (sub-) coordinators can be created when needed. Due to the choice for a specific optimized operationalisation of a coordination task the implementation of the temporary delegation of a task to a subordinated coordinator will be implemented as the activation of the delegated and operationalised task. The related implementation of the revocation is to make the task inactive.

7. CONCLUSIONS

The delegation mechanism offers an improved insight in the relations between software coordination architecture types existing in production automation. The make-to-order strategy is a good starting point to construct a (centralized) proper hierarchical coordination architecture in which all kinds of conflicts are anticipated, as a make-to-order strategy requires an integrated approach. Applying the delegation mechanism to this architecture allows to derive in a simple way proper hierarchical coordination architectures for other production strategies. A problem may be that the operationalisation of coordination (sub) tasks the delegation can be irreversible. A heterarchical coordination architecture can also be derived from a proper hierarchical coordination architecture by delegation. However, additional deliberation and redundancy in data can be expected. A holarchical coordination architecture can be constructed from proper hierarchical coordination architectures dedicated to the strategies needed. Delegation and revocation of tasks can be implemented as activation and deactivation of tasks.

REFERENCES

Albus, J.S., A.J. Barbera and R.N. Nagel (1981). 'Theory and practice of hierarchical control', in: *Proc. of the 23rd IEEE Computer Society International Conference*, Washington, 18-39.

Arentsen, A.L. (1995). 'A generic Architecture for Factory Activity Control', Ph.D. thesis, Faculty of Mechanical Engineering, University of Twente, Enschede.

Giebels, M.M.T. (2000). 'EtoPlan: a Concept for Concurrent Manufacturing Planning and Control - Building holarchies for manufacture-to-order environments', Ph.D. thesis, Faculty of Mechanical Engineering, University of Twente, Enschede.

Ingrand, F. F., M.P. Georgeff and A.S. Rao (1992). 'An architecture for real-time reasoning and system control', *IEEE Expert*, **7**, 34-44.

Jones, A. and A. Saleh (1990). 'A multi-level/multi-layer architecture for intelligent shop floor control', *Int. Journal of Computer Integrated Manufacturing*, **Vol. 3**, No. 1, 60-70.

McPherson, R.F., and K. Preston White Jr. (1995). 'Dynamical issues in the planning and control of integrated manufacturing hierarchies', *Production Planning & Control*, **Vol. 6**, No. 6, 554-554.

Rao, A. S. and M.P. Georgeff, (1992). 'An abstract architecture for rational agents', In Rich, C., Swartout, W., and Nebel, B., editors, *Proceedings of the Third Conference on Knowledge Representation and Reasoning*, KR'92, San Mateo, CA. Morgan Kaufmann, 439-449.

Van de Weg, R., R. Engmann, R. van de Hoef and V. ten Thij (1997). 'An Environment for Object-Oriented Real-Time Systems Design', in: J. Ebert and C. Lewerentz, editors, *8th Conference on Software Engineering Environments*, IEEE Computer Science Press, 23-33.

Zijm, W.H.M. (2000). 'Towards intelligent manufacturing planning and control systems', *OR Spektrum*, **22**, Springer-Verlag 313-345.