Flexibility in manufacturing: An architectural point of view

H.J. Pels, J.C Wortmann, A.J.R. Zwegers *

Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, Netherlands

Abstract

The objective of this paper is to present the influence of some architectural choices on the flexibility of a manufacturing control system. Architectures play a variety of roles in the redesign of manufacturing systems. Evolving technology and an increasingly demanding manufacturing environment have been accompanied by an evolutionary growth in control architectures. We present a number of architectural paradigms and their application in a model factory, which is based on a real-life PCB assembly plant. Flexibility is determined, to a large extent, by the architectural design. However, its achievement may be limited by the technology that is used for implementation. Guidelines are needed that prescribe when certain architectural principles should be applied. © 1997 Elsevier Science B.V.

Keywords: Manufacturing architectures; Architectural paradigms; Flexibility in manufacturing

1. Introduction

Enterprises face adaptability problems due to necessary internal modifications to cope with severe competition in the global marketplace. In order to survive in the global market, efficient operation and innovative management of change are essential. Rigid manufacturing and information systems are of serious concern in improving the prevailing enterprise operation. In addition, since companies try to meet competition and exploit new opportunities, enterprise operations have to evolve continuously. Manufacturing systems have to provide the flexibility that enables this continuous change. Therefore, an important aim when designing manufacturing systems is to attain flexibility in such a way that necessary modifications in response to environmental changes or internal requirements, can be easily implemented.

Flexibility in manufacturing is defined as the ability to adjust the primary process according to new requirements of the environment [5]. Adjustments of the primary process might require adjustments in the structure of the shop floor control system. In this paper, the scope of flexibility is restricted to the flexibility of the structure of the shop floor control system. Examples of this type of structural flexibility are: flexibility to add a controller in the control architecture, to modify a controller, to replace two controllers with one new controller, and so on.

The objective of this paper is to present the influence of some architectural choices on control system flexibility. This paper also gives an overview of ongoing research on the design of shop floor control architectures at the Eindhoven University of Technology. The control architecture determines how control functions are distributed and co-ordinated. When new requirements demand changes in the distribution of these functions, the architecture has to

* Corresponding author. Tel.: 0031 40 2472671; fax: 0031 40 2436492; e-mail: azw@tm.tue.nl.
enable these modifications. Thereby, it determines the possibilities and limitations for changing the system, i.e. the structural flexibility.

In this paper, a model factory is used for illustration. This model factory assembles and tests pseudo Printed Circuit Boards (PCBs). It is a miniaturised model of a real life PCB assembly and test plant, and it emulates many of the operations which are performed on real PCBs during their manufacturing process [14].

This paper is organised as follows. Section 2 details the role of architectures. Section 3 presents the evolution of control architectures and a number of current architectural paradigms. Section 4 briefly describes the products, primary process, and the control system of a model factory. Section 5 illustrates the application of architectural principles in the model factory. Finally, we conclude this paper with a discussion of our findings.

2. Architectures

2.1. Definition

Although the term ‘architecture’ is widely used in publications, it is generally assumed that the reader knows what this term means [16]. In this section, we give a definition that is used in the remainder of the paper. A dictionary definition of the term architecture is “the art or science of building: especially designing and building habitat structures” or “a unifying or coherent form or structure” (Websters [17]). The former definition implies a domain and the latter suggests a product or outcome of the domain (‘the architecture of something’). Hence, the second definition is more suitable for our purposes.

In this paper, we consider an architecture of a system or product as the result of a design process. It is a set of specifications which express the functions of components and their interfaces. The abstract level of the specifications is higher than that of the specifications which are the result of the subsequent design activities. Drafting an architecture is concerned with the specification of components, their interfaces, interactions, and constraints. This specification serves as a basis for further design and implementation activities.

2.2. The role of architectures

An architecture allows one to present a complex system in a simple structure. An architecture is an abstract description of a system or its subsystems. It supports the analysis of relationships as an aid to understanding complexities in a design environment. In particular, an architecture is needed in complex, dynamic environments [16]. Zachman [19] states that the increased scope of design and levels of complexity of system implementations are forcing the use of architectures for defining and controlling the interfaces and the integration of all system components.

Other roles of an architecture involve its contribution to the effective re-design of a system. The architecture should reduce the impact of changes to the lower control levels, and to as few controllers as possible. Both for production control systems and for products, it is advantageous to use as many parts of the existing system or product design as possible. In a re-engineering trajectory, an architectural model of the system allows one to pinpoint and discuss the areas requiring major change, and to integrate the new specifications into the existing model. Furthermore, the casc of architectural change is determined by the system components, the interfaces between these components, the ease with which components can be modified, replaced or with which the architectural model can be extended by new components and the extent to which the interfaces of the new components match those of existing ones. Since system implementation is based on architectural specifications, the ease of changing an architecture influences the possibility to modify or extend an existing system. In addition, the use of standards in the architectural specification positively affects system modifiability and extendibility.

In the re-design of a system, an architecture may play the role of a means of communication during the design process. The architect can use it to visualise various aspects of the system to be designed, thus providing the various parties concerned with a basis for discussion and decision-making. Architectures help each party to clarify its perception of the problem area. By visualisation and explanation of the relevant aspects of the problem area, and the possible relationships between them, the attention of the various actors is focused on the essential elements, thus
providing a basis for discussion of the problems. In addition, the distinction in entities may serve as the managerial basis for cost estimation and project management.

3. Architectural paradigms

In this section, we give a brief overview of some well-known architectural paradigms. For more information, the reader is referred to [4].

3.1. Evolution of control architectures

Technological advances in computing and communication technology have made it possible to consider a wide range of possibilities in the design of control architectures. This rapid growth in technology in addition to an increasingly demanding manufacturing environment has been accompanied by an evolutionary growth in control architectures. Fig. 1 shows the evolution of architectural models, which is characterised by an increase in the autonomy of the controllers, a reduction of the use of aggregated information and a relaxation of master–slave relations.

The earliest control architectures employed a centralised approach. However, advances in manufacturing required that designers considered other options in the development of control architectures. Because of the apparent disadvantages of central control, other control architecture designs, such as hierarchical control, were considered. Hierarchical control architectures were developed primarily because designers had been conditioned through training and experience to approach complex system design from a hierarchical standpoint.

After all, many organisations, are designed according to the hierarchical paradigm. In addition, current computer technology encouraged designers to define control levels that corresponded to different types of computers, e.g. a VAX station was often used for control at the cell level, and a PC for control at the workstation level.

Technological advancement in the area of distributed computing and the rapidly declining price/performance ratio of hardware have led to the consideration of more decentralised approaches. The heterarchical control architecture applies the advantages achieved by distributed intelligence, creating a flat architecture that divides control responsibilities among co-operating controllers.

3.2. Hierarchy

A hierarchical architecture is characterised by the usage of control levels and contains several control
modules arranged in a pyramidal structure. These distinct levels have their own purpose and function. All activities of the subordinate level are dictated by the supervisor level, and the subordinates have no option but to obey. The control decisions are operated top-down, with status reporting operating bottom-up (i.e. from slave to master). Implementing the assorted levels of control will be varied with the use of a variety of computing technology, including mainframes at the highest levels, minicomputers in the intermediate levels, and microcomputers at the lowest level. The refinement process of breaking down aggregated decisions and the concept that sensory information at the higher levels is more abstract and requires the integration of data over longer time intervals, implies that aggregated databases are found at each level.

3.3. Hierarchy

Heterarchical control structures have distributed autonomous entities that communicate with other entities without the master/slave relationship found in a hierarchical architecture. The main concept of the heterarchical control architecture is the goal of full local autonomy and a co-operative approach to global decision making. Supervisory decision making is located locally at the point of information gathering rather than in a central location. Full local autonomy requires that controllers are intelligent and that global information is minimised or eliminated. This implies that only local databases will be maintained within the system. The cooperation between entities is arranged via a negotiation procedure. To ensure full local autonomy, a protocol must allow controllers to refuse the transfer or acceptance of an order based on its own status. By using such a protocol, controllers negotiate with each other to arrange scheduling and routing of parts. The common types of control operations that each peer controller must perform makes it more appropriate to make use of the same type of physical resource for each.

3.4. Modules

The concept of modules has been applied in several disciplines. The architect Le Corbusier introduced the term in the context of a system of standard measures for buildings to achieve harmonious designs. Later on, architects used the term for exchangeable standard building blocks that enabled the construction of flexible houses. The concept of modules played an important role in the development of structured programming. One of the first proposals to apply modules to master the increasing complexity of information systems stems from Parnas [11], who introduces 'information hiding' as the criterion that "every module is characterised by the design decisions it hides for other modules". More detailed criteria are proposed by Yourdon and Constantine [18], who elaborate the concept of information hiding into three criteria:

1. moderate complexity: the structure of a module should be understandable by its designer and its users;
2. minimal coupling: the amount of specifications that is known by other modules should be minimised (information hiding);
3. maximum cohesion: the complexity of a module should be large compared with the complexity of its view on its environment.

The principles of modular design have been formalised with respect to data structures in information systems in [12,13]. This study shows that the essential properties of a module are that its interfaces are precisely defined and that a clear distinction is made between the input interface and the output interface. The input interface, called the foreign domain of the module, contains all specifications of other modules that must be known to design, validate and operate the module. The output interface, called the public domain, contains all specifications of the module itself that are visible to other modules (the non-hidden information). Fig. 2 shows the foreign domain and the own domain of modules. The public domain of module A that is visible to module B, and vice versa, is in grey.

Modular design applies the concept of modules as the basic thread for building the system architecture. The steps for system design are:
1. Define the functions of the system.
2. Assign each function to a module.
3. Specify in detail the interfaces between the modules.
4. Design the non-interface part for each module separately.

The modules in a system architecture should have the following properties:
1. Modules have a clear and recognisable function.
2. Modules have explicitly defined interfaces, with clear distinctions between input and output.
3. Modules can be designed independently, without knowledge of external specifications unless explicitly stated in the foreign domain (input interface).
4. A module can be validated and tested as a stand-alone unit.
5. A module can be integrated (after validation) with other modules without further testing, as long as the interfaces match.
6. A module can be operated/controlled independently: without knowledge of specifications and state of the environment except for the input-interface.
7. A module can be exchanged when interfaces remain unchanged.

Properties 1 and 2 express that modules are basic building blocks in an architecture: function and interface. Properties 3–6 are referred to as ‘module independence’; they express that the interfaces should effectively decouple the modules. Property 7, exchangeability, is essential for flexibility.

It is clear that flexibility is proportional to the chance that the public domain (outgoing interface) remains untouched when a module is to be changed for a system modification. Therefore, there are two golden rules for modular design. First, make as little specifications as possible known to other modules in order to maximise cohesion. In other words, keep interfaces simple!

The modular approach has been applied to the design of shop floor management systems by Timmermans [14]. The role of modular design in the model factory is explained in Section 5.3.

3.5. Object oriented

The principles of object oriented design are beyond the scope of this paper. The reader is referred to [10] for an in-depth discussion. Rather, we focus on how object oriented design supports a designer in the definition of controllers.

The object approach with its combination of attributes and functionality in objects, inspires a designer to emphasise similar behaviour of controllers. Object oriented design adds to the modularity concept the possibility to exploit similar behaviour of controllers. The additional design criterion is the procedural similarity of controllers. Units with a similar (abstract) procedural description are recognised and considered as instances of the same class. In addition, derivation and polymorphism allow a designer to re-use a piece of code without changes. Adaptations needed for a special unit could be added in a derived class, which for instance enables the separation of device control and configuration dependent control.

3.6. Other paradigms

In addition to the aforementioned architectural paradigms, there are other paradigms used by researchers and system designers. Two currently popular approaches are based on agents and holons that originate from the fields of artificial intelligence and socio-economic systems, respectively. For more information on these approaches, the reader is referred to [3,9] (agents) and [7,15] (holons).

4. Model factory

4.1. The product

The final product of the model factory is a printed circuit board (PCB) with dummy electronic inte-
4.2. The primary process

The model factory is a miniaturised model of a real PCB production line. It contains the basic operations of 'screen printing', 'component placement', 'reflow and cleaning', and 'test and repair'. Blank boards and components are automatically supplied from a centralised raw material store or component store. The model factory is designed for batch production; batch size, which is three or less, can vary from batch to batch.

The layout of the model factory's primary process is depicted in Fig. 3. The first stock in line contains the two types of empty boards. All products pass the screen printer, but alternative routings are possible between the two component placement stations. After reflow and cleaning, the batches are stored in the in-process-store, which consists of three locations, one for each of the three products. Then, products are tested and—if necessary—repaired. The repair buffer can contain one batch. The finished-product-store is randomly accessible and can contain nine individual products. Note that there is a loop from the in-process-store to the screen printer. This loop is necessary to manufacture PCBs that have components on both sides and that have to pass through the process twice, since only one side can be finished in one pass.

4.3. The control system

All workstations in the model factory are fully automated, with the exception of the repair station, where a manual operator is required. Besides the actual operation, each workstation has to manage temporary storage and retrieval of PCBs, indexing of PCBs through the process, inventory of raw materials, etc. This necessitates many sensors in the model factory in addition to solenoid stops, motors, lights, conveyers, pneumatics, etc. Obviously, all logical I/O signals to and from these sensors and actuators are controlled by a PLC and its associated programs. There is also a higher level supervisory system to manage the overall production process. Hence, the implementation of the hardware consists of two levels, a PLC level and a VAX level. In summary, a considerable amount of equipment was needed to implement this I/O controller and supervisory computer system.

Like the primary process, the control system of the model factory is based on the control system of a real production plant. The different types of devices and the control hardware, i.e. PLCs and supervisory computers, are widely used for industrial control. Even the use of commercial software tools and related integration platforms, follows modern industrial systems closely.

5. Application in the model factory

5.1. Layers

A functional decomposition into layers has been defined for the model factory. For each layer, functions and interfaces with adjacent layers have been

---

Fig. 3. Primary process of the model factory.
defined, thereby accomplishing separation of concerns [6]. In addition, at the highest level, the application layer, interfaces between stations were specified. Fig. 4 shows the different layers and interfaces. This division in layers holds for all architectural paradigms in this section. In this paper, we mainly focus on the application layer, where the coordination between stations takes place.

According to a defined interface, units communicate with neighbouring units at the application layer. The communication consists of receiving and giving requests for batches, and coordinating the situation if a batch arrives at a unit or if the production on a batch finished.

The application layer gives commands to the control layer. Specific commands are defined to bring a unit into a particular state, or start the operation on a batch. The application layer is an abstraction of the production process of every unit, so general commands can be used towards the control layer. The specific station information resides in the lower layers of a station. Commands are given about the whole batch, no distinction is made among the individual boards in a batch [1].

5.2. Heterarchy

Fig. 5 illustrates the heterarchical control architecture that has been implemented in the model factory. The principal functionality of the heterarchy is as follows. Controllers request product batches from each other, resulting in a pull oriented control of the model factory. The last controller in the line, i.e. the final product store controller, receives a work order. This controller creates a batch and a request for that batch based on Statistical Inventory Control. The controller specifies in the batch definition what type of product has to be produced and how many (the batch size). The request for this batch is then sent to the preceding controller in line and test & repair controller. This controller in its turn creates its own request for the same batch and forwards it to its preceding controller. This process of receiving and forwarding requests continues until the beginning of the production line is reached. Here, the physical batch is created by releasing empty boards from the raw material store. This physical batch is sent to the requesting station, thereby eliminating the request for that batch. Hence, all requests for the batch will be eliminated when the physical batch arrives at the final product store. Here, the batch definition itself will also be removed from the information base.

Note that the division in controllers in Fig. 5 is identical to the division in modules as presented in the next subsection. Evidently, other divisions are possible as well. Above all, controllers request batches from each other, and send batches to the requesting controller. Although this principle is a
characteristic of the heterarchy, it can be applied together with other architectural paradigms, as is done with module design and object oriented design.

An advantage of the heterarchical model is its ease of reconfiguration and adaptability. Reduced software complexity is realised because of enhanced modularity, reduced coupling among modules, and implicit fault-tolerance. Enhanced modularity and reduced coupling between modules reduces complexity and simplifies development and future modifications and extensions.

5.3. Modules

Decomposition of the conceptual schema follows the structure of the organisation. Each module corresponds to an organisational unit, so that its own domain corresponds to the registration tasks of that unit and its view domain to the formal information needs [13]. In addition, Timmermans gives some criteria for the demarcation of a module in terms of modularity of the information system [14]. For the control system of the model factory, autonomous controllers are specified for each self-contained unit. Furthermore, an information system module is designed for each of the controllers in the control architecture. The specification consists of a functional description of the module, a conceptual schema, the constraints, and the domain definitions.

For example, the functional description of the component placement #1 controller is as follows [14]. The component placement #1 controller receives a request from the component placement #2 controller. This request is converted to a request for the screenprinter controller. When a batch is received from the screenprinter, the appropriate components are placed upon the boards. The batch is forwarded to component placement station #2 when all operations have been performed on all products in a batch. If the component placement #1 controller has a shortage of components the status of the component buffer is set to empty in order to trigger the material handler for replenishment.

The conceptual schema, the constraints, and the domains are defined as follows. A data structure diagram of the conceptual schema is given in Fig. 6. The central object classes in the diagram are request and batch. A request refers to the batch that is requested. Furthermore, a request refers to the station that will consume the batch related to the request and to the station that will produce the batch related to...
the request. The batch refers to the item_type it contains, and to the station that created the batch. From the station, there is an optional relation to the batch to indicate the batch-in-process. Finally, there is a component buffer which refers to the station and the item_type it contains.

Examples of only one class and one integrity constraint are given for brevity:

SCHEMA component-placement#1

CLASSES

class station

attributes

station_name: string;
produced_requests: SET OF request;
received_requests: SET OF request;
batch_available: (available, non-available);
ready_to_receive: (Yes, No);
batch_in_process: batch
end; — class station

INTEGRITY CONSTRAINTS

—for every information base state i must hold that the component_placement #1 may not produce more than one request per batch.

\[ C(i) = (\forall b: b \in i . \text{batch}:
\text{(#r: r \in i.request: r.consumer.station_name = 'component-placement#1' \land r.batch = b) } \leq 1) \]

DOMAIN RULES

—the own domain of this module consists of the objects of the object types request, station and component_buffer that have ‘component-placement#1’ as the name of the associated station.

own domain (i) = \{ t \in i . \text{request} | t . \text{producer.station_name} = 'component-placement#1' \} \cup
\{ t \in i . \text{station} | t . \text{station_name} = 'component-placement#1' \} \cup
\{ t \in i . \text{component_buffer} | t . \text{buffer_station.station_name} = 'component-placement#1' \}

—the foreign domain of this module consists of all objects of the object types item_type and batch, the objects of the object type request with ‘component-placement#1’ as the producer station name, and the objects of the object type station with ‘component-placement#2’ or ‘screenprinter’ as the station name.

foreign domain (i) = \{ t \in i . \text{item_type} \} \cup \{ t \in i . \text{batch} \} \cup
\{ t \in i . \text{request} | t . \text{producer.station_name} = 'component-placement#1' \} \cup
\{ t \in i . \text{station} | t . \text{station_name} = 'component-placement#2' \lor t . \text{station_name} = 'screenprinter' \}

END; schema component-placement#1

Often different functional decompositions are possible for the same system. In that case, the criterion of modularity is used to discriminate between alternative choices. Modularity is determined by three criteria: complexity, coupling, and cohesion. The modular decomposition technique provides a more formal method to demonstrate that a module is independent; a module is independent if all applicable constraints are visible.

The advantages of the described design for information systems are found in the achieved modularity. The reduced coupling between modules simplifies the interfaces. Modules consist of a generic part and a situation specific part. Evidently, the generic control part is re-usable in new stations. The modular design makes it possible to enhance the system by adding new modules without affecting the existing ones or to modify one of the modules without affecting the other modules. In addition, it enabled the one-by-one design and implementation of the controllers in the model factory.

Modularity of a shop floor control system is not solely determined by the modularity of the conceptual schema. Other aspects that have to be considered are the product, process and control structure. In the application of the model factory, the choice of modules was determined by the organisation of the primary process. Modules corresponded to the main functions that could be distinguished in the primary process.

Modularity of a shop floor control system is also determined by the implementation architecture of the system. The definition of the implementation architecture of a manufacturing system is a critical design choice, which determines the conditions for modular (re-)design of information systems at the conceptual level. Even more, the application of the principles of modular design is sometimes hindered due to technical limitations in the implementation of the information system. In the model factory, the PLC program contained too much intelligence. For example, the routing and the Bill of Materials is coded in the PLC program; when a component placement module is
provided with a product type, the PLC program knows the routing and the requested board type. This implies that changes in routings or products can only be made by changing the PLC program, which requires a lot of effort. Therefore, despite the fact that the application layer is modular, the system cannot be considered as flexible [8].

5.4. Object oriented

As stated in Section 3, the object approach considers in addition to modularity the similar behaviour of controllers. The main design criterion to distinguish controllers is procedural similarity, i.e. units are distinguished according to similar procedural descriptions. Every unit in the model factory which meets the description of accepting a batch, perform an operation, and release it, is called a station [2].

Three types of stations can be distinguished: the producer, the store, and the mover. The producer has one input and one output stream and performs independently a well defined task on the product. In addition, the producers are divided in three categories: standard units, placers and assembly units. A placer adds components to a main product. A secondary line is coupled to the main production line to deliver the components. The assembly unit assembles two parts into one product. A store has zero or one input and one output stream. Between the input and output, a direct relationship does not necessarily exist. A store can contain more than one batch of products, reorder the batches and initiate a new request. A mover consists of one or more input streams and one or more output streams, and makes an explicit decision about the routes products follow.

However, the number of different station types can be reduced by some design choices. For instance, by introducing a generalisation in the station class, the need to make a distinction between a mover and a producer is removed; both can be treated as an ordinary production unit. In addition, the store is modelled as a specialisation of the production unit, which overrules the default behaviour of forwarding the batch to the next station as a part of the production cycle.

In some cases, configuration dependent information is necessary to control a specific shop floor. A typical example is the deadlock prevention in case of a cycle in the production process. In that case, all stations might contain batches and wait for another station to accept the batch. The general solution to this problem is to limit the number of batches in this part of the production process. Positioning this extra control information in the particular station is not recommended from the flexibility point of view; a new configuration would mean that this device controller should be adapted, even if the function of the device is not changed at all.

The solution is to add a dedicated, logical station, which encapsulates configuration dependent details, but also has the interface of a normal station. This approach makes it possible to give every production unit (i.e. a station with a physical device attached) an own stable control mechanism and connect them together in the desired configuration. At the necessary places, a logical station can be inserted in a transparent way. Fig. 7 depicts an example of a production cycle with a logical station L added. The

---

**Fig. 7. Deadlock prevention.**
function of L is to limit the number of batches in this part of the system to two batches at a time. If a third batch is available for this subsystem, station L will not notify station A until one of the batches leaves the subsystem through station D.

After the basic building blocks have been defined, they are assigned to stations. For most stations, a direct mapping is possible. The next step is to define a configuration with these stations and to add logical stations where control problems occur (see Fig. 8). Some problems stem from savings on ironware and special configurations, such as cycles in the production process.

An example of a logical station that is added, is the Second Side Manager (SSM). In front of the tray with PCBs is space for three batches of double sided products waiting for the second cycle. However, a station (and ironware) to control this buffer does not exist. A logical station, named SSM, is used as buffer controller and accepts the batches by a positive reply on the first three requests. A fourth one will be held until the PCB tray station announces that a place is available again.

Finally, routing and processing information is added. A station must be able to identify the kind of product and determine which actions to take. As opposed to the previously discussed design where routing and processing information was stored in the PLC program, a data structure in the application program contains these data. A special class is created that is linked to the production unit class and the logical station class. This Process Operation Description class (POD) is a description dedicated to a station with which the latter determines which actions to perform. It describes which semi-finished product is needed for an applied product, and on which transport lines products come in and go out. On the basis of the POD, a station can forward an incoming request.

Modifiability and extendibility of modules is enlarged by using fixed, general interfaces in the design of controllers. Fig. 9 shows the interfaces in the current design. All stations have the same interface to control the devices (e.g. receive for accepting a batch, start for performing an operation and release for releasing the batch) and to interact with other stations. As a result, a generic controller can be defined.

Because all stations have the same, fixed interface, a station can be coupled to every other station. In addition, as long as a new station satisfies the common interfaces, a relatively small effort is needed to insert this station into the existing system. The actual working of this station is not of interest; the
station can for instance be a complete factory, using a hierarchical control mechanism. This also implies that a station can be modified without affecting other stations. The only constraint is that the common interface may not be violated.

6. Concluding remarks

The main focus of the first design based on modular decomposition is to make a modular design of the shop floor, in order to keep changes locally. The complex shop floor control system was divided into independent, autonomous modules, thereby considerably reducing the complexity of re-design. The exchange of modules influences only a small part of the whole system and could therefore be done with a relatively small effort.

The first design taught us that the modularity of an information system may be limited by the technology that is used for the implementation [14]. Only the application layer has been designed modularly, and flexibility problems are introduced in the underlying layers. Especially, the monolithic PLC program hinders the easy modification of modules; changes in the product definition or the routing information require changes in the PLC program.

The station based architecture is based on a modular design of the control layer as well as the logic layer. The hard coded product and routing parameters in the PLC program are replaced with a data structure in the application program. By defining a fixed interface between the autonomous components, based on messages instead of overlapping data structures, the factory layout is easier to change. The architecture strictly separates the device control and the configuration dependent control into production units and logical stations respectively, which results in small units that can be combined 'without' limitations.

If we ignore the increase of flexibility because of an improved control and logic layer, the difference in flexibility between the two designs is mainly caused by the re-usability features of the design based on stations. Both production units and logical stations exhibit the same control behaviour. In addition, the interfaces among these units and between the units and the control layer are deduced from the same general definition.

Furthermore, by using logical stations, specific control roles can be carried out. Control roles in the model factory all involve the prevention of deadlocks by blocking incoming requests when the number of outstanding requests is larger than a predefined number. These roles had to be defined because of the primary process. However, in some situations, it might be beneficial to add logical stations for simplification reasons (e.g. separation of device con-
control and configuration dependent control). An example is a situation where a client negotiates about an order with various servers. Instead, a logical station might be introduced that takes care of choosing the optimal server; the client gives the order to this broker, and the latter distributes the order to the optimal server based on its own knowledge of the situation. Both the client and the server do not need to have any negotiation capabilities.

A field for further research might be the development of a method and guidelines that indicate when to apply certain architectural principles. For instance, the method should state when certain building blocks such as agents, modules, etc. are appropriate, and when certain relations such as ‘master–slave’, ‘peer-to-peer’, etc. are suitable. First, however, more clarity about the concepts of agents and holons is highly desired. In addition, languages are needed to describe the building blocks and their relations.

For future research, we recommend that possibilities to make the architecture itself flexible are investigated. Building blocks have to be defined that can be used in both hierarchical and heterarchical strategies; they have to play different roles and support different kinds of relations. Then, the architecture—which constitutes of those building blocks—is adjustable and the control system might decide on its own optimal global behaviour, which allows it to adapt to special situations such as machine failures.

References