State-of-the-Art

Computational Tasks in Robotics and Factory Automation *

Frank P. Biemans
Philips Laboratories, Department Robotics and Flexible Automation, 345 Scarborough Road, Briarcliff Manor, NY 10510, USA

Chris A. Vissers
Department of Computer Science, Twente University of Technology, P.O. Box 217, 7500 AE Enschede, The Netherlands

The design of Manufacturing Planning and Control Systems (MPCSs) – systems that negotiate with Customers and Suppliers to exchange products in return for money in order to generate profit, is discussed.

The computational task of MPCS components are systematically specified as a starting point for the development of computational engines, as computer systems and programs, that execute the specified computation. Key issues are the overwhelming complexity and frequently changing application of MPCSs.

Keywords: Top-down design, Reference model, Observable behavior, Formal specification, Complexity, Algorithms.

F.P.M. Biemans received his B.S. and M.S. degrees in Electrical Engineering cum laude from Twente University of Technology, The Netherlands. He worked on Reference Models and architectures for Manufacturing Planning and Control Systems at the Computer Aided Manufacturing Centre of Philips, The Netherlands, and currently works at Philips Laboratories' Department 'Robotics and Flexible Automation', 345 Scarborough Road, Briarcliff Manor, NY 10510, USA.

* Presented at the 1988 IEEE Workshop on Special Computer Architectures for Robotics and Automation, April 29th., Philadelphia, PA, USA.

North-Holland
Computers in Industry 10 (1988) 95–112

1. Introduction

This paper discusses the design of a system that negotiates with Customers and Suppliers to exchange products in return for money in order to generate profit. We call this system the Manufacturing Planning and Control System, or MPCS for short.

An MPCS has to *compute* how it should realize its goal by interacting with Customers and Suppliers, and by controlling devices that act upon material. We will show that an MPCS can be represented as an organization of cooperating components. Each of these components performs a portion of the computation of the complete MPCS; collectively they realize that MPCS computation. The design of an MPCS is hence the definition of the computational tasks of its components and the development of computational engines that perform the required computations.

Two phenomena have a considerable impact on the design of an MPCS, i.e. its *complexity* and its *uncertain and changing application*.

Complexity. The computations to be performed by an MPCS are tremendously complex. Consider, for example, the computations required for such sub tasks of the MPCS as scheduling of jobs for

C.A. Vissers earned the M.S. degree in Electrical Engineering from Delft University of Technology, The Netherlands, and holds a Ph.D. degree from Twenty University of Technology, The Netherlands. He is presently a full professor in the Department of Computer Science, Twenty University of Technology, P.O. Box 217, 7500 AE Enschede, The Netherlands. His research interests are in the area of top-down design and formal specification of interface, protocol, and service systems.

machines or planning of a path for a robot arm. Due to their combinatorial nature, these tasks soon become computationally intractable if the dimension of the search space is extended, and if exhaustive evaluations of all possible solutions are required [24,46]. There is no way that realistic investments in computational power can remedy the intractability of many of these algorithms that require exhaustive search [13].

Apart from the exceedingly large numbers of potential solutions, it is often difficult to clearly quantify the value of a given solution. This makes it difficult to test solutions which have been generated. How to assess, for example, the decision of an MPCS to buy raw materials on the basis of forecasts of Customers' demand, if finally the customers do not buy the products as was expected?

In the light of the intractable nature of computations required in an MPCS, we have to focus on methods to compute a *satisfactory* way of achieving a goal with a reasonable investment in computational power and time [41,52]. The computational goal is not specified as an optimum to be found, but rather as a range of goals, which are equally desirable solutions. The fact that a satisfactory solution is accepted implies that optimal solutions are not pursued at all costs [1].

Uncertain Application. Another source of complexity is the computational uncertainty [24]. A component performs computations to act upon its environment. Plans may be computed to that effect, but it cannot be known in advance whether the environment will respond as expected. If not, plans may no longer be feasible and may have to be computed again.

Often, however, the question is not whether the environment will respond as expected; we hardly know what to expect. An MPCS typically deals with a very dynamic environment. Customers and Suppliers come and go, change their services, demands, and negotiation strategies. Other changes are caused by the introduction of new products. Or, because the capacity of an MPCS has to be expanded or cut, or equipment has to be replaced by more advanced equipment, etc.

Since the computations of MPCS components determine how they interact with their environment, changes in the environment will have consequential effects on the computations required. For practical reasons, it is imperative that such changes of MPCS components can be kept local: it is required to be able to change portions of the MPCS without redesigning the whole MPCS.

Design. We have discussed how the MPCSs inherent complexity and the uncertainty of its application will affect the way it executes its task. However, these two phenomena also affect the MPCS in another way: they affect the approach we choose to design it.

Complexity. The complexity of an MPCS forces us to consider its design at various levels of abstraction so that we can suppress details at each level that can be considered separately. In a series of steps, specifications at successively lower levels of abstraction implement those at higher levels.

The complexity of an MPCS also influences the choice of which and how many levels of abstraction should be distinguished in a particular design process because it affects the difference in levels of abstraction a designer can successfully bridge.

The complexity also affects the optimality of design. A single designer cannot make all design decisions at once, but rather has to rely on a step by step approach, where designs at one level of abstraction constrain designs at lower levels of abstraction. As a result, optimality of design cannot be guaranteed. It is difficult to accurately foresee, for example, the consequences of a high-level design decision, such as the desired functionality, for a low level design decision such as the costs of material of a product with this functionality. It is, therefore, not possible to decide beforehand whether the desired functionality is worth the costs. The design process, therefore, aims at a satisfactory, rather than optimal design [2].

Uncertain Application. If we design a system, there will always be uncertainty as to whether the system can still as usefully be applied when its design is finished as was anticipated at the beginning of

[1] May be, we should view satisfactory solutions as optimal, if we consider the costs of computation as a parameter of the function to be optimized.

[2] May be, we should view the satisfactory design process as optimal if we consider the amount of computation needed to produce a design as a parameter in the cost function.

its design. Similarly, there is uncertainty about constraints that play a role during the design process such as the costs of material that can be used, or the availability of computer hardware, for example.

The uncertainty, as the complexity, affects the levels of abstraction that we select. We want to minimize the likelihood that designs have to be changed as a result of the changes mentioned. The generality of a design, the ability to use it as a starting point for many design processes at lower levels of abstraction, is hence an important criterion for the selection of the levels of abstraction.

Example. MPCS components have to cooperate, but they can appear in different configurations and different authorities determine the physical construction of the different components. We may therefore conclude that designs that describe the cooperation of components in terms independent of their configuration or physical construction would be applicable for all implementation authorities involved [55]. Such designs are worth pursuing.

We pursue the generality of a design because it seems economical to produce designs that can be used for the development of multiple real-life MPCSs. However, we cannot maintain our desire for generality through all phases of the design: real-life MPCSs differ. The challenge, therefore, is to defer the application-specific design decisions as long as possible.

Based on these considerations we selected the following design steps:

1) We define the task of an MPCS as a black box. It is the starting point for the design process.
2) We decompose the black box MPCS into an organization of cooperating components in such a way that the relatively simple components of the total, more complex, MPCS can be considered, and we are meanwhile allowed to keep sight of the relationship between separate components. The result of the decomposition is a hierarchical organization of relatively independent components with such tasks as: negotiating with customers, determining inventory levels, scheduling, processing parts, determining trajectories of joints, controlling energy flows, servo control, and analogous sensing tasks. This organization is designed to be applicable to all MPCSs.
3) We specify the externally-observable behavior of the components so that they can execute the tasks identified in 2) and cooperatively realize the MPCS task. Large portions of the observable behavior can be generally applicable; some portions that are application-specific can be identified and isolated.
4) We develop a description of each component, which constructively prescribes how each output of a component can be produced from its inputs. These so-called 'generative' descriptions, such as algorithms, affect the efficiency of computation. They tend to be application-specific but are independent of the physical means used to do the computations.
5) We determine which components are implemented on which physical systems, and specify the observable behavior of these physical systems. We then implement these physical systems.

In the following sections, we describe these design steps and the resulting designs in more detail. Other authors have discussed general methods to design complex systems [15,16]. The contribution of this paper lies in the result of those methods: the design of an important class of systems, the MPCSs.

Note that the steps are discussed only briefly here. References are provided for some background reading.

2. Manufacturing Planning and Control System

A Manufacturing Planning and Control System (MPCS) is defined as a system that:

- Aims at earning a target amount of money or market share by buying and selling certain types of products in certain target numbers,
- Negotiates the exchange of products for money and exchanges product for money with Customers and Suppliers.

An MPCS may be able to process various kinds of products as long as those products are not drastically different from each other in terms of the technology needed for their processing, and their demand characteristics. It would not be re-

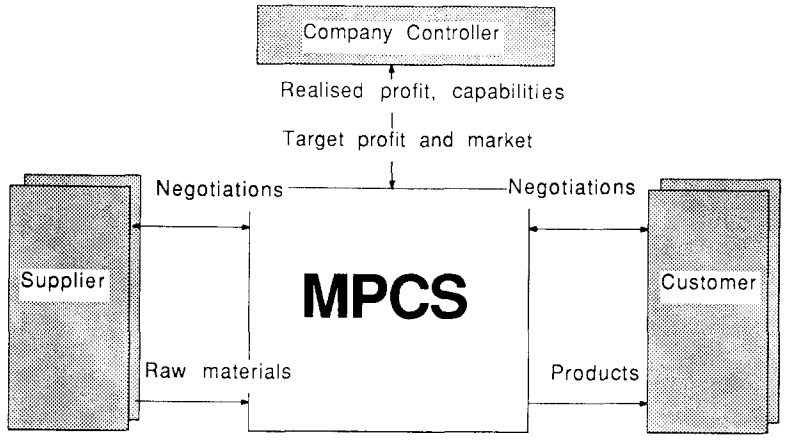

Fig. 1. A manufacturing planning and control system as a black box.

alistic to assume, for example, that an MPCS could produce television sets and petroleum.

It is assumed that an MPCS is used by a 'Company Controller' which has profit-making as one of its goals (Fig. 1). To that end, the Company Controller can command the MPCS to earn or—as part of a strategy—lose money by delivering products. The decisions about the required type and size of market share, and profit are made by the Company Controller. The MPCS can report to the Company Controller the amount and kind of profit it has generated. It can also report its capabilities in terms of the number of products it can process and the costs of processing.

A Company Controller could have several types of systems that support its profit-making, such as systems that provide service. In this paper, only the MPCS is discussed. The Company Controller could have multiple MPCSs.

'Suppliers' and 'Customers' appear to the MPCS as independent systems, although they may belong to one company. The MPCS and Customers and Suppliers negotiate the exchange of raw materials and product, taking into account aspects such as their type, number, quality, due dates, price, penalties in case of late deliveries, etc. They also exchange the actual orders for products, cancellations, and transfers of money. The exchange of raw materials and products actually means the exchange of the *authority to control* them, which does not necessarily imply their physical transfer.

3. Reference Model

The black box representation of an MPCS as presented in Section 2 does not reveal all the components of an MPCS that are required if we want multiple designers to contribute to the MPCS development. In light of the top-down design process, we need a model of an MPCS as an organiza-

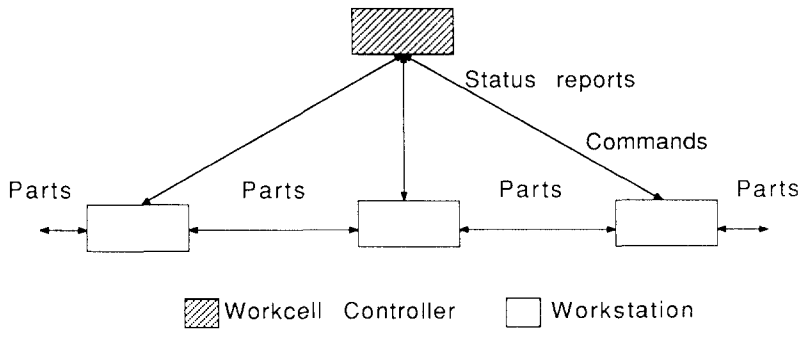

Fig. 2. Structure of workstations and workcell controller.

tion of cooperating components. The model should define components of the total more complex MPCS so that we can consider them more or less in isolation but meanwhile can keep sight of the relationship between the separate components.

Such a model is called a 'Reference Model' [3,4,8]. It describes the organization of an MPCS in terms of its structure and tasks. A *structure* defines which components may, but not necessarily have to, cooperate. A *task* defines a component's essential mission, the goal of its actions.

A Reference Model is a baseline in the top down design of an integrated MPCS. Based on the task assignments, which define only the essential, functional characteristics of a component, as well as the structure of the model, the required *behavior* of the components should be defined. The behavior defines a component's allowable inputs and outputs as well as their inter-relationship and temporal ordering.

Example. The Reference Model defines the generic task of a component 'Workstation' as accepting, processing and dispatching parts on command. A structure as illustrated in Fig. 2, determines that a Workstation may interact with a 'Workcell Controller' (to exchange commands and status reports) and with other Workstations (to exchange parts).

A specification of the required *behavior* of a Workstation should adhere to the task definition given above. It should, for example, precisely specify the commands and status reports it may exchange with the Workcell controller. Also, the messages exchanged with other Workstations to facilitate a product exchange should be specified.

The temporal ordering of the messages should be specified. It should be specified, for example, how the exchange of commands relates to the exchange of parts. A constraint could be that the Workstation should first pass a part to the next Workstation, and then report to the Workcell Controller that it has finished an operation. Meanwhile, it may accept a command for the next operation.

Finally, the protocols used to exchange messages should be specified.

Several papers discuss the need for a Reference Model for MPCSs [4,45,19,29,57]. References [25,42] describe organizations in general, but do not discuss MPCSs as such. Nevertheless, some of their general conclusions have been applied to the development of the Reference Model that is described in this paper. References [1,3,12,17,28,33, 37,39,56] describe MPCSs, but only parts of it. They focus on tasks such as production planning, or transport systems, or on robot control, or on sensing. Apart from their limited scope, references like [1,37,39] focus on control algorithms and their applications. Control algorithms and the organizational model a Reference Model is supposed to be, are different kind of models. This is not meant to be a criticism on those papers; they were written for other purposes. Nevertheless, they have been used extensively for the development of a Reference Model, as discussed in this paper, because they help to get a complete picture of the tasks that should be done by an MPCS.

References [2,4,21,32,40] describe an MPCS in a more complete sense. However, these models are insufficiently precise, unambiguous, and complete to serve as a basis for a further design of an MPCS. In fact, some of them do not define the tasks of computing agents at all. Others do, however without any justification for the definition of tasks and structure.

As we discussed before, a Reference Model is to be developed by decomposition of the black box MPCS. Decomposition has long been recognized as a piece by piece approach to analyze and design complex systems: the individual components are considered in isolation, and the behavior of the total system can be deduced from the way the components interact, whereby the components can be described with relatively few variables.

Note that there is no objective criterion or evaluation technique to assess whether a Reference Model is optimal. It is not trivial to decide how an MPCS is to be decomposed. Some decomposition techniques are discussed in [51]. However, they require algebraic relations that specify the system to be decomposed, or rely on knowledge of a physical structure inherent to the system to be decomposed. In our case of an MPCS, we have neither of them.

This strategy has led to a model that differs from all models mentioned before: it tries to cover a complete MPCS and it is justified on the basis of criteria that support the development of a consistent, conceptually uniform model, which can be easily understood. This led to quite different results.

Below, we will discuss and motivate a strategy to decompose the MPCS.

3.1. Decomposition Strategy

References [9,15] describe a few aesthetic principles for designing systems. These design principles are motivated by an overwhelming requirement that a human should be able to master and use a system, even when it has considerable, inherent complexity. The better a system meets this requirement, the better its economies in terms of cost of design, learning, and use is likely to be.

As [9] points out, the principle criterion to meet the requirement is 'consistency', the system's conceptual integrity and unity of viewpoint that allows a human to master a system as a whole. Consistency is believed to underlie all principles of quality of system design. It tells the designer not to link what is independent, not to introduce what is unneeded, and not to restrict what is essential. This leads to three maxims, i.e. 'separation of concerns', 'generality' and 'propriety', which will be briefly discussed below.

Separation of concerns. A decomposition of the MPCS should result in components that have essentially different or relatively independent tasks. This is called 'separation of concerns'. It leads to efficient cooperation of components because the more the components can execute their task independently, the less they have to interact. A component's state changes more often as a result of internal, local state transitions than as a result of interactions with its environment. Separation of concerns will, therefore, be reflected in the parsimony of interactions of components.

Separation of concerns can also minimize the effect of a modification of an individual component on other components [47]. This enhances the generality of an MPCS because changes can be kept locally. Separation of concerns exhibits the inherent parallelism or loosely coupling of the system to be decomposed.

Generality. If independent tasks have been identified, they should be defined in general terms. This leads to components that can execute their tasks for multiple purposes. It enhances the general applicability of the components and hence potentials for future use with unforeseen requirements.

Propriety. Inessential tasks should not be introduced. Such tasks would distract from the essential task and hence conflict with the requirement of consistency. They could also lead to functions that are provided in alternative ways, forcing redundant knowledge and unnecessary choice upon the user.

Analysis of interactions. The above criteria should help in achieving a consistent distribution of tasks. However, they are domain independent and do not help in task identification. Tasks should be identified on the basis of the external manifestation of the MPCS [17] and can only be identified on the basis on a thorough knowledge of the applications of an MPCS. The interactions of the MPCS and its environment, or of an MPCS component and its environment, are therefore analyzed. The basic questions are: "what are the essential temporal and semantic characteristics of the interactions?" and, "Which tasks are required to generate or process the information exchanged in the interactions?".

3.2. Application

We will now briefly review the application of the decomposition strategy and the resulting Reference Model. We will only discuss the tasks of the components and hardly consider their structure. For a more detailed discussion, refer to [8].

Analysis of interactions. If we analyze the interactions of the MPCS and its environment, as described in Section 2, it appears that:

- The negotiations of the MPCS and all its Customers and Suppliers are strongly related activities that serve to realize the commands from the Company Controller.

  The fact that all negotiations can be related is illustrated, for example, by the fact that an MPCS may try to increase its price for one Customer as a compensation for unsatisfactory deals with others. Moreover, an accepted order claims a part of the MPCS's resources and shrinks resources available for other processing other orders. To be able to conduct the negotiations, it is important to know which product transformations can be realized,when and in what time. It is not necessary to know how products are transformed.

- The acceptance of raw materials, their processing and dispatching of products are another group of related activities. It depends on a result of the negotiations—which orders for products have been accepted or are expected —which product transformations should take place. It does not depend on the way in which the negotiations are conducted.

The 'negotiating' and 'processing' tasks are fairly independent. As appears from the above, only limited information that results from executing one task is required to execute the other task. Both tasks can be executed in parallel, apart from relatively few interactions.

The negotiation and processing tasks are also essentially different. Successful negotiation requires knowledge of markets, trends, and competitors. It also requires an aptitude to deal with unexpected behavior of the environment, and abilities to hypothesize the behavior of the environment. Processing products requires knowledge how to physically manufacture these products. Dispatching them at their due dates, on the basis of actual or predicted orders also requires capabilities to plan and to assess and weigh risks of inventory, or late delivery, etc.

Decomposition. Now we have identified two distinct tasks, we can decompose the MPCS into components that execute these tasks. An MPCS is decomposed into a component which we call 'Factory Controller', concerned with the negotiation task, and a component which we call 'Shop', concerned with the processing task. Figure 3 illustrates the decomposition.

The Factory Controller accepts targets as type and number of products to be sold and profit to be realized over certain periods in time. To realize these targets, the Factory Controller negotiates with Customers the delivery of products in certain quantities, at certain due dates, against certain prices and conditions, etc. It commands the Shop to dispatch products to Customers at their due date. It may also inform the Shop about expected but not yet confirmed orders so that the Shop can already start processing them in order to reduce the procurement lead time.

A Shop needs raw material to process parts on the basis of confirmed or predicted orders. A Shop will report to the Factory Controller which raw material it needs, and when, so that the Factory Controller can negotiate their delivery with Suppliers. The Factory Controller can cancel or overrule previously-given orders to a Shop. It might do so, for example, if raw materials cannot be purchased at acceptable prices, or if orders from Customers have been cancelled. A cancellation may result into inventory of unfinished parts contained by the Shop, affecting the Shop's costs and performance.

Separation of concerns. The decomposition of the MPCS has resulted into to distinct components

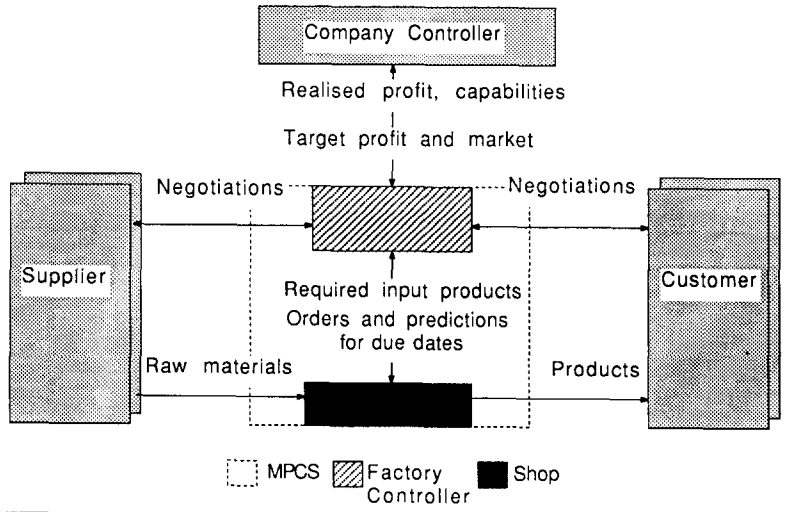

Fig. 3. The MPCS decomposed into a factory controller and shop.

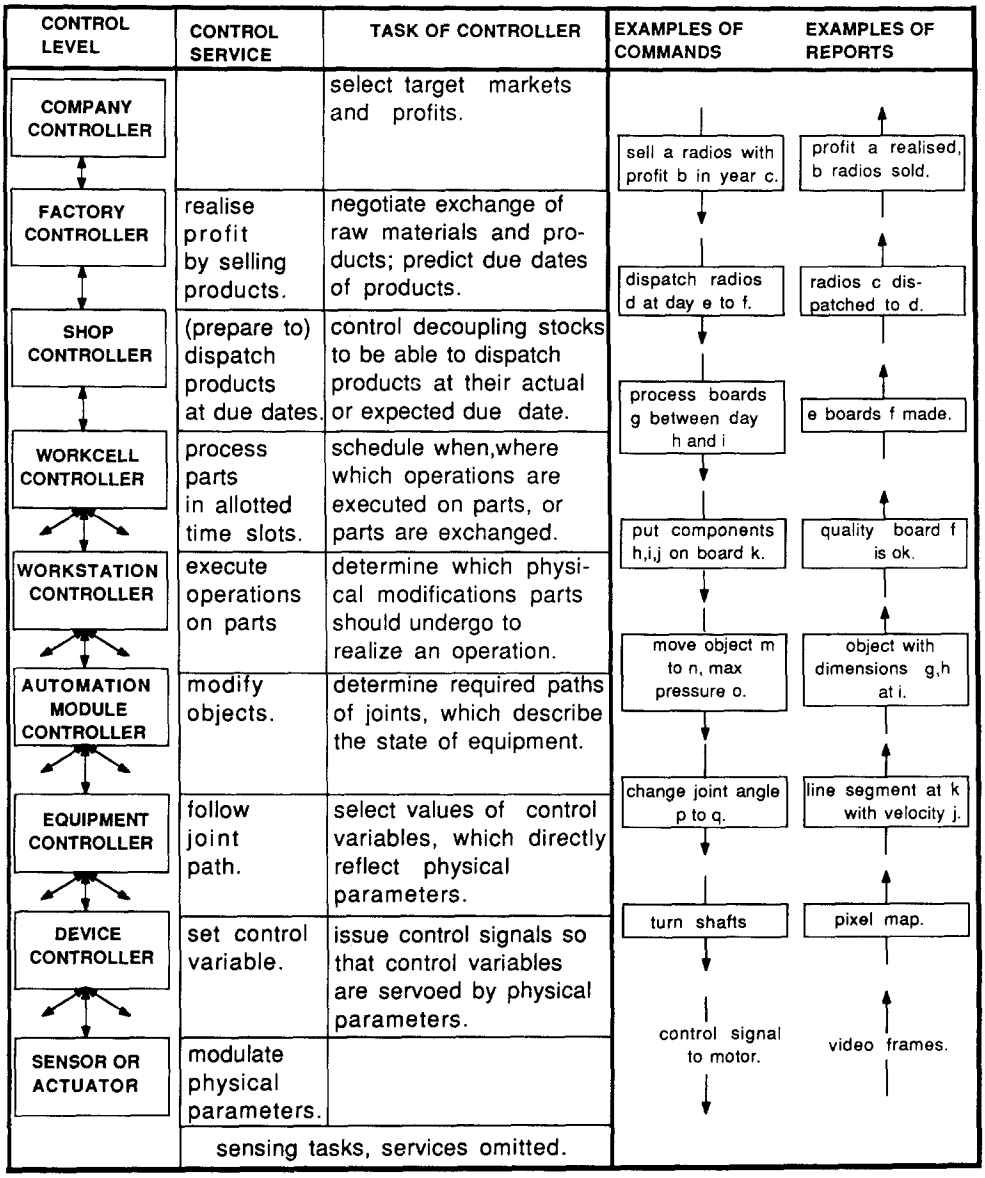

Fig. 4. Reference model for MPCSs at a glance.

that are simultaneously operational and that interact to cooperatively realize the task of the total MPCS. The environment cannot observe any difference between the MPCS as depicted in Figs. 1 and 3. Similarly, the Factory Controller deals with a Shop but does not know its internal organization. This is a result of the separation of concerns maxim, and significantly reduces the information a Factory Controller has to process compared to a situation where the Factory Controller would be concerned with the internal states of a Shop.

We call a component that does not reveal its internal organization, but actually consists of multiple components a 'service'.

Repeated decomposition. We can repeat the decomposition strategy. A service is decomposed into components, called 'controller', which we do not decompose further, and possibly into components called 'service', which can be decomposed in turn. Ultimately, we obtain an organization of controllers that do not need to be decomposed.

Our repeated decomposition of an MPCS has led to a structure of horizontally layered controllers depicted in Fig. 4. Going from the bottom to the top, each controller enhances the service provided by all its subordinate controllers. The column 'control service' characterizes the essential control task that is available at a certain level. The column 'task of controller' summarizes the basic task of a controller, that is to realize its service using the service of the aggregate of all the lower controllers.

The MPCS has a hierarchical organization in the sense that its services can be decomposed into multiple other components, i.e., controllers and services that in turn have a hierarchical organization. This is a result of the repeated decomposition and application of the separation of concerns [3]. The concept of service is also reflected in the terminology. Words as 'Shop', and 'Workcell' denote services; words as 'Shop Controller', and 'Workcell Controller' denote controllers.

We spend a few words on the *tasks* of MPCS controllers, and point to the literature that covers the kind of problems the controllers have to deal with. We do not discuss the *structure* of an MPCS, but refer to [8].

Shop Controller. A Shop Controller receives commands from a Factory Controller to dispatch products at due dates to Customers, or predictions that it may have to dispatch them. The Shop Controller may report to the Factory Controller which raw materials it needs and when. It may also report about its capability: which products can it dispatch and when, which have been dispatched, which are overdue, etc.

The time needed to dispatch a product can be shortened by having this product or parts that can be transformed into this product, in stock. Products from one stock could be transformed into products of another stock on the basis of the predictions of orders from the Factory Controller.

The Shop Controller controls the amount of inventory of some key parts to be able to dispatch products within the procurement lead time, taking into account the time needed to process the parts, but also the costs and risks of inventory. Typically, those key intermediate parts are sub-assemblies that can be used for several other sub assemblies or products so that their aggregate demands has reduced variance compared to the demands of the several products.

The stocks are called 'decoupling stocks' because they decouple goods flows, i.e:

- Parts of a decoupling stock that can be processed in time to serve Customer's demand (independent demand) or to create parts for another decoupling stock (dependent demand),
- Parts that have to be processed to create parts for another decoupling stock on the basis of predictions of the required size of that stock.

Typically, the parts have to be processed in certain *time slots*, for example, in the period that would be long enough to process a part, or between the time that an order has been received and products are due, or between the time that forecasts are available and some due dates.

The control of decoupling stocks on the basis of demand of products and predictions of demand is the essential task of a Shop Controller. Important considerations for determining whether a part should be transformed and become part of another decoupling stock are order cycles, predictions and other characteristics of demand, lead times and predictability of lead times, the inventory in 'down stream' stocks, costs of inventory and risks of non saleable inventory [27] (see also [26, Chapter 6]).

Once, the Shop Controller has decided that parts have to be transformed into other parts in a certain time slot, it commands the Workcell to do so.

Workcell Controller. The Workcell Controller receives commands from a Shop Controller to process parts in a certain time slot.

A part that is processed can be subject to a specific combination of operations. Typically, the different operations are different in nature and can best be executed separately and at different

[3] We have hence provided an example in which a strategy to design a complex system with an uncertain application results in a system with a hierarchical organization. There seems to be an interesting analogy with the discussion in [52] that natural systems with a hierarchical organization have a far better chance to survive unpredictable events as destructive impacts during evolution. Reference [20] discusses the complexity of hierarchically organized systems and discusses how the behavior of the complete, hierarchical system can be deduced if the behavior of its components is known.

locations. The Workcell Controller should decide when the operations are executed, and where (scheduling). It delegates the execution of the operations to Workstations.

The Workcell Controller has to decide *when* in the allotted time slots it actually transforms the intermediate products. The time of processing is an important factor in the costs and performance. It affects the inventory and occupied space, frequency of product change-over, etc.

Parts generally have to visit several Workstations. The Workcell Controller coordinates the exchange of parts among Workstations and tells Workstations with whom they have to exchange parts. A Transport System can best be treated as a specialized instance of a Workstation.

Note that a Workcell Controller identifies products by their name, and does not know their physical characteristics such as their geometry or colour. It commands Workstations to accept products with certain names and to dispatch products with certain names.

Workstations may report the performance of the operations, which parts they have exchanged and their availability to exchange parts at certain locations.

Workstation Controller. A Workstation Controller [7] receives commands to execute operations on parts, which are identified by their name. Those operations might imply modifications of physical characteristics of parts such as geometry, temperature, and location. A Workstation Controller determines which physical modifications a part should undergo so that the operation on the part is realized.

Example. To realize a soldering operation, different sub assemblies have to be moved in a certain order. Then, they have to be heated. Finally, a sub assembly has to be moved again.

A Workstation Controller commands Automation Modules to realize the physical modifications.

A Workstation Controller needs knowledge of parts because it should be able to determine which physical modifications a particular part should undergo and in which order. A Workstation Controller does not need knowledge of the resources with which the modification are performed other than their capability. It does not know, for example, the degree of freedom that a robot might exploit for displacing a part. An Automation Module, on the other hand, does not need to know which particular part it is processing. It only has to know the aspects relevant for the modification it has to realize. It knows 'object', collections of physical characteristics that are related. To displace an object, such aspects as its geometry and weight are to be known, but not whether the object is, for example, actually a blue or red pencil.

Automation Module Controller. An Automation Module Controller receives commands to realize physical modifications of objects.

A variable through which an Automation Module Controller can control its effects on its environment is called 'joint', a generalization of a robot's joints. A joint may be any kind of variable, such as position, temperature, or pressure, depending on the type of effects being pursued. Joints are used to efficiently describe the state of equipment that can be used to change the physical environment. To realize a required modification of an object, an Automation Module Controller needs a view of this object, the relation of this object with other objects as obstacles, and it needs to know how it can influence the objects via its joints [22].

An Automation Module Controller determines the paths of joints so that objects are modified as required. It leaves the realization of the paths to Equipment components.

Equipment may provide sensory data that describes such features as line segments or positions or temperatures. The Automation Module Controller collects those data to recognize objects, and to maintain an internal representation of objects.

Equipment Controller. An Equipment Controller receives commands to realize paths of joints as prescribed by the Automation Module Controller in the physical world and to describe this world in terms of features.

Joints describe the state of equipment, but not how this state can be changed. A robot joint, for example, can be changed by changing the angle of a shaft that is connected to the physical joint via a belt. We call such a variable, that directly reflects a parameter of the physical world that can be controlled, a 'control variable'.

An Equipment Controller realizes the required values of the joints by determining the required value of control variables and by commanding Devices (Actuators) to change physical parameters so that these servo the control variables. The Equipment Controller determines, for example, the forces (control variables) to be applied to realize a certain joint path as quickly as possible [44].

Devices (Sensors) provide sensory information as variables from which the Equipment Controller extracts features.

Example. A pixel map is a collection of variables that represent visual inspection in terms of colour or intensity as function of a location. If the feature to be extracted is a line, line-like regions of possibly noisy pixels with an intensity above a certain threshold, are to be identified. Pixels considered not part of the line, can be neglected.

Device Controller. A physical parameter has to be servoed by a control variable. A Device Controller therefore determines which control signals are to be issued to an Actuator. Actuators maintain a relation between the modulating control signal and some physical parameters.

Reversely, a Device Controller may have to provide sensory information. Sensors maintain a relation between some physical parameters and sensor signals. A Device Controller accepts sensor signals and codes the information.

Management aspects. We reviewed the tasks of MPCS controllers as defined by the Reference Model discussed in [8]. The descriptions of these tasks are assumed to be a starting point for the definition of the input and output messages of the controllers. A Reference Model would be particularly useful if it were so complete that it identified all tasks that lead to message exchanges. To achieve such a complete model, we should not only consider the primary tasks of an MPCS, such as processing materials or negotiating with customers, which directly support the MPCS goal as discussed above. We should also consider secondary tasks that are needed to monitor and change the way the primary tasks are executed. These tasks manage the MPCS and we call them 'management tasks'. Think of such management tasks as [54]:

- Fault management: tasks to detect, diagnose and correct faults in the operation of components of the MPCS.
- Accounting management: tasks to observe, calculate, and determine the costs of the operation of components of the MPCS.
- Configuration an name management: tasks to define, observe, and control the configuration of components of the MPCS and their names.

The concrete requirements for the management tasks vary in different MPCSs. Different MPCSs, for example, may utilize different methods to calculate their costs. However, we might be able to identify the management tasks already in the Reference Model, at least in a qualitative sense. We can and should leave the definition of management messages and the development of real management systems to the later phases of the MPCS design. By including management in the Reference Model, we increase the probability that it can be used as a complete basis for message definitions. The Reference Model we reviewed here does not include management tasks [4].

Conclusion. We have reviewed a Reference Model. It was developed on the basis of criteria that should allow us to understand an MPCS despite its complexity.

A Reference Model has far-reaching consequences as an early specification in a top-down design process of an MPCS, because of its impact on the subsequent design phases. Suppose, for example, that the Reference Model had defined tasks of two components that are fairly dependent. The components have to interact frequently because the execution of their tasks depends on the execution of the task by the other. If the components interact by exchanging messages, and are implemented on different physical systems connected by a data communication system, the frequency of their interactions will affect the requirements with respect to throughput of the data communication system. It will also affect the extent to which modifications of one component can be isolated from the other.

[4] The ISO-OSI Reference Model [30] does not consider management as an integral part of the model. The message definitions of the service and protocols based on this model do not support management. Instead, separate management tasks and protocols are defined. The use of service and protocols definitions as standards is limited because they do not prescribe how OSI system components cooperate on management aspects.

The following sections discuss the next design steps to be taken on the basis of the Reference Model.

4. Observable Behavior

Once we know the tasks and structure of the MPCS controllers, can define how they realize their tasks through affecting each other's behavior. We therefore define the behavior of a controller as it is observable for the other controllers. Wo do not yet define the internal mechanism of a controller that computes how to generate this behavior. The externally-observable behavior constrains the results of its computations without constraining how they are executed.

We should define a component's input and output *messages*, and their relation with respect to contents and temporal ordering. We do not yet define the physical representation of the messages since no decision has been made yet on the distribution of components over physical systems.

Design principles similar to those used for the development of the Reference Model apply to the design of the required observable behavior. Again, strict application of these principles can greatly enhance the ease of understanding of the specifications.

Example. Typically, a tool is used for multiple operations, as some sub assemblies that are broken into parts and used in multiple operations. Also, data as information where a defect can be found on a sub assembly, adds value to the product, as a sub assembly does.

The generality maxim suggests to exploit the similarity of sub assemblies, tools, and some data about sub assemblies. They can be treated similarly in many situations.

The commands that tell a Workstation to execute an operation should treat them similarly. This reduces the space required for coding the messages and the number of messages needed. It also reduces the complexity of the Workcell Controller which has to schedule and coordinate the processing and exchange of sub assemblies, tools, and data by Workstations, but can treat each of these in similar ways.

We may define the temporal ordering of messages non-deterministically; that is in terms of what can happen, rather than what will happen during an actual application. We do not wish, for example, to put improper constraints on that portion of the temporal ordering that will depend on the application, which we do not yet fully know. Think of those messages exchanged by a Workstation Controller and Automation Modules depending on the product being processed. Nor do we wish to specify temporal ordering in full detail if the actual temporal ordering is not relevant. We might, for example, specify that a Workstation responds to all status requests it gets from the Workcell Controller, without requiring that all responses are issued in the order they were requested.

4.1. Integrated behavior.

It is essential that the definitions of the behaviors of all MPCS components allow them to cooperate meaningfully, i.e. realizing the behavior of the MPCS as a whole [3]. We will now discuss some techniques that can support the development of such meaningful specifications of the observable behavior.

One is the technique of step-wise refinement. First, we draft the specification of a service (page 103) like the MPCS. Next, we draft the specification of the controllers and service that resulted from a decomposition of the service. This should occur in such a way that the composition of the newly-specified components and the service that was decomposed should render equivalent [5] behavior as observable by their common environment. (The cooperating Factory Controller and Shop should not be distinguishable from the MPCS).

Another technique to support the development of meaningful specifications is the use of formal specification languages, methods, and tools. A formal specification language allows us to make *precise* specifications of MPCS components. The precision helps us to prevent or detect errors in the specification. Equally important, precision reduces the risk that ambiguous interpretations of the specification arise during their implementation.

Some formal specification techniques could also help to proof such properties of specifications as

[5] Reference [43] provides in-depth discussions on different types of equivalence relations.

the absence of deadlocks or races, or some form of equivalence of specifications which differ in form. The latter could be used in conjunction with the step-wise refinement discussed above. Equivalence of a service and its decomposed specification is shown. If the complexity does not allow a mathematical proof of these properties, methods as simulation might still be helpful. Reference [6] discusses requirements for formal specifications of MPCS components. It discusses trial specifications of a Workcell, Workcell Controller, Workstation and Transport System written in the language LOTOS [14,31], developed by the International Standardization Organization. It also gives an equivalence proof that shows that the specified Workstations, Transport System and Workcell Controller cooperate meaningfully, and cannot be distinguished by a Shop Controller from a particular Workcell. Discussions on formal specifications of MPCSs can also be found in [48,23,53]. References [5,10,11] describe non-trivial applications.

Ideally, we could formally specify the absolute time frames between the exchange of messages. This would define the externally-observable performance of a controller. However, we are not aware of formal methods to define performance requirements of externally-observable behavior of controllers and also produce proofs about their integrated behavior.

4.2. Application

A major difficulty in the development of the specifications is to answer the question of usefulness. How useful would a specific behavior of a component be when it is applied in a real-life MPCS? Because the design of specifications is ahead of the actual use of the components, the usefulness is always a vision, not a measurement [9]. To not limit the potential use of a design, it should not be optimized towards a specific, present-day application, but should be kept general.

Each time two components exchange messages with unused parameters, expensive code space is wasted. This calls for more messages with fewer parameters. However, if components have to exchange multiple messages to convey information which could as well be conveyed by a single message, such code space as the code for addresses is used which could have been saved. This calls for fewer messages with more parameters. To decide upon an optimal set of messages, we should be able to estimate the relative frequency with which they will be exchanged.

There will always be application-dependent portions of the specifications. A specification for a Workcell Controller, for example, will describe how many Workstations it may command in a particular application. To increase the applicability of specifications, the application-dependent portions should be clearly distinguished from the generally-applicable portions and represented as parameters that can get values depending on the application, as illustrated by Fig. 5.

The application-dependent parameters can be simple data elements like addresses of the controllers. Examples of more complex parameters for the Workstation Controller described in [7] are an initial world model and recipes. The initial

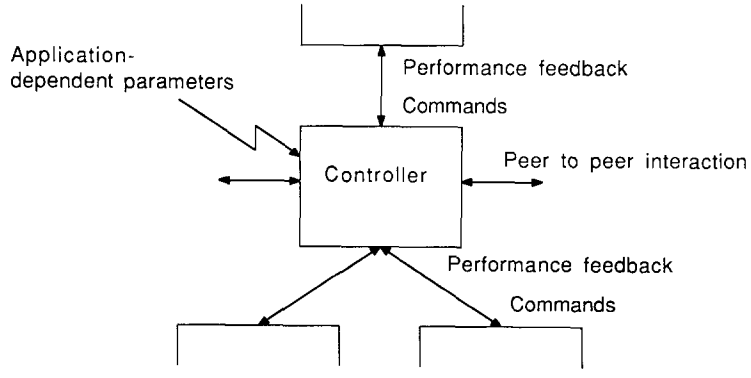

Fig. 5. Application-dependent parameters that constrain the externally observable behavior.

world model describes the state of the Automation Modules at start-up of the Workstation Controller. The recipes describe how the Workstation Controller should execute an operation on a product by commanding Automation Modules, for example, to modify characteristics of the material of the product. The recipes are a kind of programs. Such programs are sometimes referred to a process plans [18].

The parameterized, generally-applicable specification of the observable behavior of a controller serves as a starting-point for its further design. However, we should still determine the values for the application-dependent parameters and feed them into the controllers in order to achieve an operational MPCS.

Very little has been published yet about the definition of application-dependent parameters as input for controllers. Exceptions are [18,36,35,38]. They mainly focus on the process planning portions of the application-dependent data.

Much research needs to be done in this area. We discuss a few thoughts, without pretending to be complete.

It seems imperative that the application-dependent parameters can be defined in a language that allows us to efficiently express the domain dependent characteristics. The semantics of such languages need to be based on characteristics of data and control structures in the application-dependent parameters of controllers. It should also be possible to define the application-dependent parameters independent of the physical implementation of a controller.

Computational aspects of controllers need to be considered. Reference [24] argues that a particular schedule—which can be considered as an application-dependent parameter for a Workcell Controller—should be complete, compact, and allow time-efficient computation.

The application-dependent parameters for the different layers in a hierarchy as depicted in Fig. 4 can be related. Suppose, for example, that each controller requires an initial world model as part of its application-dependent parameters. Each world model contains a description of the capabilities of the lower level service. These capabilities may depend on other application-dependent parameters for lower level controllers.

The above thoughts suggest that an MPCS controller should not be considered as a computer to be programmed in a general purpose programming language. Rather, an MPCS controller is a computational engine that interacts with other controllers in an MPCS to execute and delegate tasks, and that interprets parameters that define application-dependent constraints on its behavior.

There is a strong relation between the capabilities of a controller and the kind of application-dependent data it can interpret. Compare, for example, two types of application-dependent parameters for the control system of a Transport system. One type defines how at each node of the Transport System a direction should be chosen for a product with a certain destination. The other type only defines the layout of the Transport System, assuming that the control system is capable of finding a path for a product ones it knows its layout. The first type defines constructively how a task it to be executed; the second describes constraints on the execution of the task, and assumes that the controllers can construct a plan to reach its goal given these constraints.

5. Generative Description of Controllers

A generative description of a controller describes constructively how its externally observable behavior can be realized. We refer to abstract descriptions that are independent of the technology of implementation. The less abstract descriptions, which are technology-dependent, are typically represented in computer programming languages.

A generative description of the behavior of a controller can be developed independent of the other controllers if it is developed on the basis of a specification of its observable behavior. The only constraint in this respect is that its observable behavior is realized.

From now on, our design process is less dominated by the attention to integration, since generative descriptions can be developed independently of each other. However, a different aspect of integration arises, i.e., whether it would be possible to develop generative descriptions of different type of controllers that have much in common.

Such a common applicability is more or less claimed by the 'G-H-M' organization described in [2,33] and many other papers. This organization

consists of three components with different tasks. These tasks can roughly be described as the acceptance of commands and determining the commands to be issued to subordinate controllers, maintaining a representation of the state of these controllers, and accepting their performance reports to update the representation and to issue aggregate status to a superior controller.

The generative description of a controller can be simple if the application-dependent parameters are constructive. Consider, for example, the generative description of a Workstation Controller in [50], which describes how the application-dependent parameters, state tables which step by step prescribe how an operation is to be executed, are interpreted.

The more constraint-oriented the application-dependent parameters are, and the higher their level of abstraction, the more the generative description of a controller will resemble what is commonly referred to as planning and control algorithms.

5.1. Planning and Control Algorithms

Planning and control algorithms for MPCSs give a detailed prescription of the computations to be made, and hence have a strong impact on the efficiency of the computation. The efficiency determines the perfection of the computed result given the limited computational power and limited time as constrained by the required observable behavior.

We are able to allocate algorithms to the controllers in the Reference Model depending on the type of problems they solve. For example, a typical MRP system [56] could be allocated to the Shop Controller, the scheduling algorithm in [34] to the Workcell Controller, the path planning algorithm in [22] to the Automation Module controller, the time optimal control in [44] to the Equipment Controller.

The algorithms have mostly been developed as a solution to mathematical problems without considering what an algorithm would look like if it is used by the type of controllers discussed in this paper.

Most published algorithms that are used by a controller are trivial and are not capable of complex computations. The MRP systems [56], for example, are fundamentally large-scale data bases with simple computation to translate product requirements into time phased component part requirements, taking into account existing inventories and scheduled receipts. They are not capable of exploiting forecasts of demand, taking into account the actual and limited processing capabilities, and aggregating demands for individual products to reduce variance in forecasts of demand.

It is yet difficult to see what the impact could be of the desire to have more advanced algorithms that take into account the control aspects as well. They would also have to specify how the controller should react on commands or performance feedback 'on the fly', how controllers are to be started, how they should react on errors, etc.

An important and often ignored aspect in designing control algorithms for controllers is the performance of a controller in the context of the total MPCS. The effects of further improving the performance of a particular controller are limited if other controllers in the MPCS cannot benefit from this improvement.

One of the major problems that faces the developer of algorithms for an MPCS controller is the large number of parameters that could possibly be taken into account and the resulting computational intractability [6].

Many algorithms tend to be suitable for only a few applications. A few methods to develop algorithms have been published that might have broader applications or might even be applicable across the levels of the hierarchy. Among them are:

Opportunistic Scheduling

Rather than selecting one possible way to achieve a goal from the beginning, several ways are kept in mind [24]. This gives the flexibility to switch to another way if the way currently used is no longer feasible, for example, because of unforeseen events. Complete rescheduling may not be necessary. It hence addresses the issue of uncertainty. It exploits the complexity in the sense that many instead of one way to a goal are considered to be acceptable.

Hierarchical Production Planning Systems

The word hierarchical is used here to denote a sequential decision making using aggregation

[6] Simplifications of the 'MPCS's hardware' as its transport system layout also serve to reduce the complexity.

methods. The idea is to first aggregate individual products and develop long term schedules based on the aggregate data. When short term decisions are to be made, the aggregate schedules are disaggregated to deduce schedules for individual products. Due to the aggregation the amount of data to be taken into account is reduced and the aggregate forecast of demand might have reduced variance [26].

Classification of Heuristics

Heuristics are often used if solutions with proven characteristics cannot be found. Some papers describe heuristics that seem to be applicable in a class of situations. Reference [34] for example, claims a scheduling algorithm that seems to be suitable for the common situation in which Workstations are organized in lines.

Genetic Algorithms

Reference [49] reports about an initial application of genetic algorithms to industrial optimization problems. The genetic algorithm generates trial solutions, evaluates them, perform a survival-of-the-fittest selection, followed by stochastic operations which mimic genetic recombination. It is a search mechanism that does not require, nor exploit, any domain-dependent knowledge of the parameter space.

6. Physical Implementation

In order to achieve specifications that can be given to independent implementation authorities that build physical systems, we should transform the abstract specification of the observable behavior of controllers into specifications of the observable behavior of physical systems. This involves deciding which abstract components are implemented on a single physical system, and designing the data communication of the physical systems. Reference [55] discusses a structured way to design the data communication system.

It also involves of course, the implementation of abstract specifications into, for example, computer programs. We do not discuss this part of the design process. It is the responsibility of the designer of the abstract specification, however, to proof that there is at least one possible implementation so that the implementor is not confronted with a specification that cannot be implemented. An important aspect of this proof is that the behavior of a controller can be achieved with realistic investments in computational power.

7. Conclusion

We have reviewed a systematic approach to define the computational tasks to be performed by MPCS controllers. Successful application of this approach requires a strict use of general systems engineering techniques and a thorough knowledge of the application domain. Complexity and uncertain application of MPCSs are key problems.

The Reference Model is designed to be generally-applicable. It seems that generally-applicable specifications of the observable behavior of controllers could be achieved as well. However, these will contain application-dependent parameters that get specific values for each application of the controller. It is important that the values of application-dependent parameters can be determined quickly. Systems that support the selection of such parameters are still to come.

Planning and control algorithms have been published extensively, but hardly ever as generative descriptions of *controllers*. The algorithms tend to be application-specific, and it seems challenging to reduce their dependence on the application.

A generative descriptions of a controller can only be designed after its observable behavior has been specified. It is also important to note that application of the separation of concerns (see Section 3.1) maxim in designing the Reference Model and specification of observable behavior can greatly aid the development of generative descriptions. The relative independence of controllers, which is a result of the separation of concerns, puts proper limits on the complexity with which an individual generative description has to deal. Moreover, local modifications of the algorithms could be made that do not affect other controllers if the application of an MPCS changes.

Acknowledgements

The support, constructive criticism, and advice of Sjoerd Sjoerdsma (Philips' CAM Centre, The Netherlands) and Pieter Blonk (Twente University

of Technology, The Netherlands) is greatly acknowledged. Damian Lyons (Philips Laboratories Briarcliff) is acknowledged for his review of the paper. David Marimont (Philips Laboratories Briarcliff) is acknowledged for his review of the paper and discussions on the impact of complexity and uncertainty on the design process.

References

[1] R. Akella, Y. Choong, and S. Gershwin, Performance of hierarchical production scheduling policy. *IEEE Trans. on Components, Hybrids, and Manufacturing Technology*, Vol. CHMT-7(3), 1984.

[2] J. Albus, A. Barbara, and R. Nagel, Theory and practice of hierarchical control, in: *Proc. 23rd IEEE Comput. Soc. Internat. Conf.*, 1981.

[3] F. Biemans, The design of distributed transport systems as a major standard interface in computer integrated manufacturing, *Computers in Industry*, Vol. 7(4), 1986, pp. 319–333.

[4] F. Biemans, A reference model of production control systems, in: *Proc. IECON86, IEEE Industrial Electronics Society*, 1986.

[5] F. Biemans, A trial architecture of a Workstation formally specified, Technical Report, Philips CFT-Report 02/86, 1986.

[6] F. Biemans and P. Blonk, On the formal specification and verification of computer integrated manufacturing architectures using lotos, *Computers in Industry*, Vol. 7(6) 1986, pp. 491–504.

[7] F. Biemans and S. Sjoerdsma, Description and motivation of a Workstation Controller architecture, towards integration and standardisation, Technical Report, Philips CFT-Rep 55/86, 1986.

[8] F. Biemans and C. Vissers, A reference model for manufacturing planning and control systems, *J. Manufacturing Systems*, 1988, submitted.

[9] G. Blaauw and F. Brooks, Computer architecture, 1982: lecture notes, Twenty University of Technology, The Netherlands, and University of North Carolina at Chapel Hill.

[10] P. Blonk and F. Biemans, An architecture of internal transport control systems – towards standardisation of the functional behaviour of internal transport systems, Technical Report, Philips CFT-Rep. 09/87, 1986.

[11] P. Blonk and F. Biemans, An architecture of internal transport control systems – towards standardisation of the functional behaviour of internal transport systems, Appendices, Technical Report, Philips CFT-Rep. 10/87, 1986.

[12] L. Boza, C. Chavous, J. Hsu, R. Lake, and M. Pratt, Cima-an integrated architecture for product realization, *ATT Technical Journal*, Jul–Aug 1986, pp. 17–25.

[13] H. Bremermann, Complexity and transcomputability. in: R. Duncan, editor, *The Encyclopedia of Ignorance*, Pergamon, Oxford, 1978.

[14] H. Brinksma, A tuturial on lotos, in: M. Diaz, editor, *Proc. of the IFIP WG 6.1 5th. Int. Workshop on Protocol Specification, Testing and Verification*, North-Holland, Amsterdam, 1985, also published as: Provisional LOTOS tutorial ISO/TC 97/SC 21 N.

[15] F. Brooks, *The Mythical Man-Month*, Addison-Wesley, Reading, MA, 1975.

[16] F. Brooks, No silver bullet: essence and accidents of software engineering, *Computer*, Vol. 20(4), 1987, pp. 10–20.

[17] R. Brooks, A robust layered control system for a mobile robot, *IEEE J. Robotics and Automation*, Vol. RA-2(1), 1986, pp. 14–23.

[18] P. Brown and C. McLean, Interactive process planning in the ARMF, in: *Symposium on Knowledge-Based Expert Systems for Manufacturing at ASME Winter Annual Meeting*, 1986, pp. 245–262.

[19] Computer Integrated Manufacture-Open System Architecture/AMICE Consortium, Cim-osa: a primer on key concepts and purpose, 1985, 489 Avenue Louise, B 14-B-1050, Brussels, Belgium.

[20] P. Courtois, On time and space decomposition of complex structures, *CACM*, Vol. 28(6), 1985, pp. 590–603.

[21] G. Domeignt, How to decentralize decisions through grai model in production management, in: *I.F.I.P. W.G. 5.7 International Working Conf.*, 1985.

[22] L. Dorst and P. Verbeek, The constrained distance transformation: a pseudo-euclidean, recursive implementation of the lee-algorithm, in: *Signal Processing* III: *Theory and Applications*, Elsevier Science Publishers (North-Holland), Amsterdam, 1986.

[23] J. Favrel and K. Lee, Modelling, analyzing, scheduling and control of flexible manufacturing systems by petri nets, in: *Proc. of the IFIP WC 5.7 Working Conf. Modelling Production Management Systems*, 1984.

[24] B. Fox and K. Kempf, Complexity, uncertainty, and opportunistic scheduling, in: *Proc. IEEE Conf. AI. Appl.*, 1985, pp. 487–492.

[25] J. Galbraith, *Designing Complex Organizations*, Addison-Wesley, Reading, MA, 1973.

[26] A. Hax and D. Candea, *Production and Inventory Management*, Prentice-Hall, Englewood Cliffs, 1984.

[27] S. Hoekstra and J. Romme, *To Integral Logistic Structuring*, Kluwer, Dordrecht, 1985 (in Dutch).

[28] T. Hopp and K. Lau, A hierarchical model-based control system for inspection, in: Gardener, editor, *Automated Manufacturing, ASTM STP* 862 L.B., 1985, pp. 169–187.

[29] International Standards Organisation, 1986. The ottawa report on reference models, version 0.1, ISO TC 184/SC5/WG1 N51, 1986.

[30] ISO/TC97/SC16, Information processing systems, open systems interconnection, basic reference model, international standard ISO/IS 7498, Technical Report, ISO, 1983

[31] ISO/TC97/SC21, A formal description technique based on the temporal ordering of observational behaviour, draft international standard ISO/DIS8807, Technical Report, ISO, Information Processing Systems, Open Systems Interconnection, 1985.

[32] A. Jones, A cell control system for the ARMF, 1985, Center for Manufacturing Engineering, National Bureau of Standards, Washington, DC 20234.

[33] E. Kent and J. Albus, Servoed world models as interfaces between robot control systems and sensory data, *Robotica*, Vol. 2, 1984, pp. 17–25.

[34] S. Kochbar and R. Morris, Heuristic methods for flexible flow line scheduling, *J. Manufacturing Systems*, Vol. 6(4), 1988, pp. 299–314.

[35] B. Krogh and G. Ekberg, Automatic programming of controllers for discrete manufacturing processes. in: *Proc. IFAC 10th. World Congress*, 1987.

[36] B. Krogh and A. Sanderson, Modeling and control of assembly tasks and systems, Technical Report, Carnegie-Mellon University Rep. CMU-RI-TR-86-1, 1986.

[37] C. Lin and C. Moodie, An energy-saving production scheduling strategy for hierarchical control of steel manufacture, Technical Report, Purdue Laboratory for Applied Industrial Control, report nr. 147, 1985.

[38] D. Lyons, A novel approach to high-level robot programming, in: *Proc. IEEE Workshop on Languages for Automation*, 1987.

[39] O. Maimon, Real-time operational control of flexible manufacturing systems, *J. Manufacturing Systems*, Vol. 6(2), 1987, pp. 125–136.

[40] C. McLean, H. Bloom, and T. Hopp, The virtual manufacturing cell, National Bureau of Standards Washington DC 20234.

[41] M. Mesarovic, R. Erlandson, D. Macko, and D. Fleming, Satisfaction principle in modeling biological functions, *Kybernetes*, Vol. 2, 1973, pp. 67–75.

[42] M. Mesarovic, D. Macko, and Y. Takahara, *Theory of Hierarchical Multilevel Systems*, Academic Press, New York, 1970.

[43] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, Springer, Berlin, 1980.

[44] W. Newman and N. Hogan, High speed robot control and obstacle avoidance using dynamic potential functions, *Proc. 1987 IEEE Internat. Conf. Robotics and Flexible Automation*, 1987, pp. 14–24.

[45] Danish Standards Organization, The framework for development of a generic reference model for cim systems, 1986, Contribution of the Danish member Body to ISO/TC184/SC5, Document S142/CIM(DK).

[46] R. Parker and R. Rardin, Complexity theory: Concepts, results and implications for discrete optimization, Technical Report, Columbia University, Industrial and Systems Engineering, Rep. J-81-9, 1981.

[47] D. Parnas, On the criteria to be used in decomposing systems into modules, *CACM*, Vol. 15(12), 1972.

[48] P. Scharbach, Formal methods and the specification and design of computer integrated manufacturing systems, in: *Proc. Internat. Conf. on The Development of Flexible Automation Systems*, The Institution of Electrical Engineers, 1984. Conference Publication Number 327.

[49] R. Schroeder, Gerard Verkoeijen, and J. Schaffer, Production line design optimization using simulation and automatic search procedures, Technical Report, Philips laboratories, Briarcliff, TR-87-144, 1987.

[50] H. Scott and K. Strouse, Workstation control in a computer aided manufacturing system, 1984, national Bureau of Standards, Washington, DC 20234.

[51] D. Siljak, *Large-Scale Dynamic Systems*, North-Holland, Amsterdam, 1978.

[52] H. Simon, *The Sciences of the Artificial*, MIT Press, Cambridge, MA, 1981.

[53] J. Thistle and W. Wonham, Control Problems in a temporal logic framework, *Int. J. Control*, Vol. 44(4), 1986, pp. 943–976.

[54] W. Verdonck, Network management and osi: the management information services, in: *Proc. Telematica-3*, SIC-Nederlands Genootschap Informatica, 1986, in Dutch.

[55] C. Vissers, Basic (architecture) concepts of the osi reference model, in: *Proc. Telematica-3*, SIC-Nederlands Genootschap Informatica, 1986, in Dutch.

[56] T. Vollman, W. Berry and D. Whybark, *Manufacturing Planning and Control Systems*, Richard D. Irwin, Homewood, IL, 1984.

[57] R. Yeomans, A. Choudry and P. ten Hagen, *Design Rules for a CIM System*, North-Holland, Amsterdam, 1985.