A hierarchical control architecture for job-shop manufacturing systems

A Hierarchical Control Architecture for Job-Shop Manufacturing Systems

proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof. dr. J.H. van Lint
voor een commissie aangewezen door het College
van Dekanen in het openbaar te verdedigen op
dinsdag 10 maart 1992 om 16.00 uur

door

Gerrit Hendrik Smit

geboren te Laren (Gld.)

Dit proefschrift is goedgekeurd door de promotoren

prof. dr. ir. J.E. Rooda

en

prof. dr. M. Rem

copromotor

dr. ir. J.H.A. Arentsen

Acknowledgement

This research project was partly sponsored by ASM Europe b.v., Bilthoven, The Netherlands.

A Hierarchical Control Architecture

for

Job-Shop Manufacturing Systems

Print: Sansevieria, Eindhoven

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Smit, Gerrit Hendrik

A hierarchical control architecture for job-shop manufacturing systems / Gerrit Hendrik Smit. - [Eindhoven : Technische Universiteit Eindhoven]. - Ill.
Proefschrift Eindhoven. - Met lit. opg. - Met samenvatting in het Nederlands.
ISBN 90-386-0012-7
Trefw.: produktieproces.

voor mijn ouders

Summary

This thesis describes the development of a hierarchical control architecture which allows control systems to be built for manufacturing systems having a job shop character.

The control architecture is specified with the use of modelling and the Process-Interaction approach. The architecture is then developed and simulated in ProcessTalk with the aid of the ProcessTool. The total approach described permits a smooth transition to be made from modelling the system to its simulation and, finally, to the implementation of the controller.

A manufacturing system transforms raw material into finished products. The control architecture takes the specification of the physical manufacturing system as its starting point. The specification is written using the description of the operations, the material, the machines (=resources), and the manufacturing process (= recipe).

The job shop manufacturing system class is a complex system, it is characterized by universal resources and a transport system having a high degree of route flexibility. The route of the material through the job shop is not constrained. The control architecture can also be applied to manufacturing system classes in which the route of the material through the system is less free, such as the flow shop, the parallel shop and the single shop.

The job shop manufacturing system consists of a controller, one or more stores, transporters and processing resources, where the actual manufacturing takes place. Repeating the control structure of a manufacturing system inside a processing resource allows the creation of a hierarchical control structure. Such a structure has at its top a factory controller, which communicates with the outside world (consumers and suppliers) and, as the bottom layer, one finds controllers for the machines that execute the manufacturing process.

The performance of a manufacturing system has to be measured, and performance graphs are introduced for this purpose. Plotting such graphs allows an impression to be gained of the behaviour of the manufacturing system and the quality of the control system. The graphs show the lead time and the throughput as a function of the inventory level within the manufacturing system. The graphs also assist the

designer in selecting a work point (= inventory level) at which the manufacturing system should be operated.

The control functions of planning, scheduling and monitoring are discussed. The decisions a controller has to take concern material exchange, transport and the processing of material. These decisions have to be taken at a certain point in time and they have to be communicated to the controlled resources. The consequences attached to the time at which a decision is taken, as well as the possible communication protocols between controller and resource, are discussed.

The architecture is developed in the form of a general control model, the data structure and process description of which are described. The model uses requests from controlled resources in order to signal to the controller that manufacturing capacity is free. A command that work is to be done is sent to the resources in the form of a job. The resources report the results of the jobs executed back to the controller (with reports). The class of the manufacturing system is discussed in relation to the strategies which may be adopted for the generation of requests.

Finally, the control architecture is used to model an Integrated Circuit manufacturing system where wafers are fabricated by the diffusion process. Simulation studies, performed with the model, are used to demonstrate the capabilities of the architecture.

The control architecture makes the design of manufacturing systems and manufacturing controllers a more structured process, allowing the construction of hierarchical control systems. In combination with the performance graphs, the selection of a work point and the use of requests for regulating the release of jobs, the architecture also clearly shows how even complex job shops can be controlled.

Samenvatting

Dit proefschrift beschrijft een hiërarchische besturings-architectuur, die geschikt is voor het bouwen van besturingen voor "job shop" produktiesystemen.

Bij het specificeren van de besturings-architectuur is de Proces Interactie Benadering gebruikt. De architectuur is geïmplementeerd in "ProcessTalk" en gesimuleerd met behulp van de "ProcessTool". Deze werkwijze maakt de overgang van modelleren naar simuleren en implementeren eenduidig.

De specificatie van het fysieke produktiesysteem is het uitgangspunt voor de besturings-architectuur. Een produktiesysteem transformeert grondstoffen in produkten. De specificatie is beschreven in de vorm van bewerkingen, materiaal, machines (= produktiemiddelen) en het produktieproces (= recepten voor produkten).

De "job shop" produktiesystemen behoren tot een klasse van zeer complexe produktiesystemen. Ze worden gekenmerkt door het feit dat ze universele machines bevatten en dat het transportsysteem het materiaal van een willekeurige machine naar een willekeurige andere machine kan vervoeren. Elke denkbare route voor het materiaal door het produktiesysteem is toegestaan. De besturings-architectuur kan ook worden toegepast op meer eenvoudige klassen van produktiesystemen, zoals de "flow shop", de "parallel shop" en de "single shop".

Het model van een "job shop" produktiesysteem bestaat uit een bestuurder, een magazijn, een transporteur en één of meer produktiemiddelen waar de feitelijke produktie plaats vindt. Een produktiemiddel kan weer zijn opgebouwd uit de hiervoorgenoemde elementen. De besturings structuur wordt dan binnen een produktiemiddel herhaald. Op deze manier is het mogelijk een hiërarchische besturing te maken. De bovenste laag van deze besturing bestaat uit een fabrieksbestuurder. De fabrieksbestuurder communiceert met de buitenwereld (consumenten en leveranciers). De onderste laag van de besturing bestaat uit bestuurders van produktiemachines.

Voor het beoordelen van het gedrag van het produktiesysteem en de kwaliteit van de besturing is het noodzakelijk om de prestatie van een produktiesysteem te meten. Voor dit doel zijn prestatie-karakteristieken ingevoerd. Deze karakteristieken geven in grafische vorm de doorlooptijd en de doorzet als functie van de hoeveelheid onderhanden werk. De

grafieken helpen de ontwerper ook bij het selecteren van een geschikt werkpunt (gewenste hoeveelheid onderhanden werk) voor het produktiesysteem.

De besturingsfuncties "planning", "scheduling" en "monitoring" worden behandeld. De beslissingen die een bestuurder moet nemen, de consequenties verbonden met het tijdstip waarop een beslissing wordt genomen en de mogelijke communicatieprotocollen worden nader bekeken. De beslissingen hebben betrekking op het uitwisselen, transport en bewerken van materiaal.

De architectuur is beschreven in de vorm van een algemeen besturingsmodel. Hiervan zijn de datastructuur en de procesbeschrijvingen gegeven. Het besturingsmodel gebruikt aanvragen van produktiemiddelen om aan te geven dat er produktiecapaciteit beschikbaar is en dat er dus een opdracht kan worden vrijgegeven. De produktiemiddelen sturen na het uitvoeren van een opdracht een rapport waarin de resultaten vermeld staan. De strategie waarmee aanvragen worden gegenereerd hangt samen met de klasse van het produktiesysteem. Dit verband wordt nader uitgelegd.

Tot slot wordt de architectuur gebruikt bij het modelleren van een fabriek waarin het diffussieproces voor het vervaardigen van geïntegreerde schakelingen plaatsvindt. Simulatiestudies die met dit model zijn uitgevoerd demonstreren de mogelijkheden van de architectuur.

De besturings-architectuur maakt het op een gestructureerde manier ontwerpen van produktiesystemen met hun besturingen mogelijk. De besturing kan zijn opgebouwd uit een hiërarchie van bestuurders. In combinatie met de prestatie-karakteristieken, de keuze van het werkpunt en het gebruik van aanvragen voor het vrijgeven van opdrachten, laat de architectuur zien hoe zelfs complexe "job shop" produktiesystemen kunnen worden bestuurd.

Table of contents

Chapter 1
Introduction

1.1 Historical background

From the time of his appearance on earth, man has made artifacts to support his existence. In the beginning with his bare hands, and later with the aid of tools that themselves developed from simple hand tools into complex machinery. This notion of manufacturing - the making of goods by hand or with the use of machinery - is therefore as old as mankind itself. Historically, the trend has always been that work done by human muscles becomes replaced, wholly or partially, by work done by machines. This easing of the human workload by the use of machines is the process of mechanization. This process brought with it further changes: labour became divided into specific tasks and forms of organization changed. Formerly the artisan did everything himself. He ordered or collected raw, materials, made his goods, brought them to market and sold them. The central issue at this time was material.

This changed with the coming of factories during the industrial revolution. Here, work was organized in stages, and the workers brought the material to a machine, the machine performed an operation, and the processed material was then taken to another machine or to a store. Labour had become mechanized and the central issues were materials and energy. It was at this time, too, that labour became divided into specialized tasks. The worker operated only one type of machine or performed only one type of action, such as the transport of material.

The introduction of the factory concept also saw a separation between the owner-manager and the workers. The owner, who was usually also the manager, decided how the factory should look, what products were manufactured, what machines were bought, how many people were to be employed, an so on. The manager also took care of the purchase of raw material and the sale of products. He decided how much the factory produced during a certain period. The workers had to carry out the commands of the manager and had to tend the machines.

With the introduction of the assembly line in the early 1900s, a new phase in mechanization was introduced. Now not only the manufacturing of goods that was done by machines; the transport of the goods was also mechanized. Control of the manufacturing actions, however, was

still in the hands of the workers, but the speed with which these actions had to be executed was partly fixed by the speed with which the products were transported along the assembly line.

After the introduction of the assembly line the central issues remained material and energy and it was with the introduction of the computer that a new issue, information, became important. From this point (about 1950), it became possible to automate the control of machines. At the moment the first robots appeared it became easier to integrate the material handling with the processing of material on machines. The introduction of computers in factories also saw the start of the automation of the tasks of the manager. Software packages became available for such tasks as accountancy, material planning and production planning. There was a further increase of specialization in the organizational structure. Also, ownership and management was split between different persons or groups of people, leading to a reduction of management tasks and restrictions on managerial responsibility.

1.2 Automation, the present situation

The purpose of automation is to enable, lighten or replace human labour by computer controlled machinery in order to increase the quality of life of human beings and/or to increase the productivity of a company. There are economic, social and technical reasons for a company to automate.

The economic reasons have to do with increasing competition, which demands an increase of productivity. The market that used to be a sellers' market has changed into a buyers' market. The costs of human labour increase steadily, which often makes it unattractive to hire a lot of workers, so instead machines are used to manufacture products. There are social reasons to automate when the manufacturing process is unhealthy, dangerous or boring. As to technical reasons, automation will allow an increased and more constant quality of the products to be achieved.

What the manufacturer tries to achieve with automation is to manufacture at low costs and to deliver products of high quality. The change into a buyers' market, too, has resulted in a short product life cycle, which demands high manufacturing flexibility and small production runs. In order to keep costs low the productivity has to be adjusted to the consumer demand, the lead times have to be short and the inventories small. Automation is seen to provide an interesting opportunity to reduce costs and to increase the quality of products. From the viewpoint of flexibility, however, automation may also be a retrograde step, because there is no machine that is as flexible as a human being.

The automation of factories has concentrated on two aspects: the real time automation of machinery and the automation of the administrative functions [Arentsen 1989].

Automation of machinery

The control of machinery is characterized by its short cycle times (in the range of milliseconds to minutes), parallel algorithms, simple and short messages, small amounts of data, and real time execution.

A manufacturing process is split up into a sequence of operations. This differentiation of the process into steps is often taken as a basis for automation. A first attempt is made to automate machinery that performs a single step, with material handling and transport following at a later stage. But the control of the whole system is considered only at the end of the process. This approach neglects the fact that the different stages of production are not only related by the material stream, but also by the technical and organizational aspects of the process. The result of this approach is stand-alone automation of separate production units with human beings taking care of the interfacing between the automated elements, thus taking care of any inflexibilities, failings, shortcomings and imperfections in the process.

Another problem in the control of machines or aggregates of machines is the fact that control systems are based on sequentially run programs. But a manufacturing system consists of man and machines which operate in a parallel way, and this requires a control system or algorithm which is also based on parallelism [Rooda 1987].

Automation of administration

Administrative functions are characterized by a long cycle time (in the range of days to years), large data bases and batch execution.

Accountancy tasks, financial management and materials management are administrative functions. The automation of these functions often concentrates on an efficient implementation of the function, rather than considering the total system effectiveness. The result is a poor connection between the different software packages.

System approach

Both cases above indicate that automation of elements of the whole system results in a collection of subsystems that are difficult to couple

together. This automation of elements is called island automation. The differences in the approaches to automation of machinery and administration have resulted in the automation gap: it is difficult to couple automated machinery with automated administration.

When a system is going to be automated one has to consider all the elements that make up the system and the relations between these elements. When looking at the current state of automation, however, one can see that it is characterized by island automation and an automation gap. We may conclude that the control of the system as a whole is important and that this calls for an integral concept that ensures that the various parts interface with each other. Furthermore, the control of machines, and aggregates of machines and people, has to take account of the fact that many actions take place simultaneously. A concept that is appropriate for this purpose is the Process-Interaction Approach [Rooda 1987, Arentsen 1989, Rooda1991a, 1991b, 1991c, Rooda Arentsen 1991, Rooda Arentsen Smit 1992], which will be used in this thesis and which is described further in Chapter 2.

Automation and factory layout

When looking at the layout of a factory, there are two important alternatives: a very common layout is the process layout; another, more complex, layout is the functional layout.

The process layout corresponds with the flow shop. Machines are ordered in the sequence of the operations that have to be performed on the material. The route of the material through the factory is fixed and dedicated machines are used in the flow shop.

The functional layout corresponds with the job shop. Machines are ordered in groups having the same functionality. The job shop is characterized by a great route flexibility. In a job shop the machines are of a much more universal type, and they are capable of executing many different operations.

Arentsen's [1989] thesis, "Factory control architecture", describes a control architecture for flow shop factories using the Process-Interaction Approach. It was shown that island automation of factories can be avoided, and that the automation gap can be bridged. Modern factories, however, often have a job shop character. They contain many expensive and complex machines. This has resulted in complex manufacturing systems that are capable of executing many manufacturing processes. They are difficult to control, often giving rise to very long lead times. No control architecture for job shop factories has yet been described.

1.3 Objectives of the study

We have seen that there are pressing reasons for industry to automate. Modern factories are complex, they often have a job shop character, together with a hierarchical layout and control structure. But presently there is no structured method by which the control systems for such factories can be built and so there is a need for a control architecture. This architecture has to allow the building of control systems and the automation of factories without creating island automation and an automation gap. The control architecture presented in this thesis is unique in fulfilling such a need.

This study only considers control technology. The systems considered have a discrete character, in which the controllers control discrete manufacturing processes. As stated, the subject of this thesis is a control architecture for factories, an architecture being defined as a framework for the logical and functional implementation of a system [Flatau 1988]. A control architecture is a structure of algorithms and controllers that drive the machines in a factory, together with the relationships and interfaces between these algorithms, controllers and machines. The architecture comprises the specification of a general control model. We are here concerned with the programming of the algorithms; the building of control hardware is not dealt with, neither is the building of the physical manufacturing system.

We use an integral approach to the building of controllers for manufacturing systems having a great routing flexibility between universal machines. Arentsen [1989] presented an architecture for flow shop factories. Here we develop a comparable architecture for job shop factories. However, since the job shop is the most complicated, the architecture presented can also be used for other factory layouts.

The functions of the factory controller considered are capacity planning, marketing, purchasing and manufacturing control. The emphasis lies on the control of the manufacturing function; control of the other factory functions is not considered so intensively, since these are implemented in much the same way as in Arentsen's work.

In order to evaluate the quality of a controller we need measures by which we can assess the performance of the system. The method we present for the assessment of the behaviour of a manufacturing system and its controller uses the concepts of mean lead time, mean throughput, and mean inventory level and the relationship between them. One of the major problems in the control of a job shop is the limitation of the lead times in the shop. By using the developed architecture it is shown that

one good way of keeping the lead times down is to exert control on the inventory level within the shop.

The implemented control architecture is intended for a manufacturing system that has a configuration which remains unchanged: the products are designed in advance, the manufacturing processes and the machines by which the products are manufactured remain constant throughout the manufacturing cycle.

Our approach to the specification of the architecture uses modelling together with the Process-Interaction Approach [Rooda 1987, Overwater 1987]. This approach provides a language (ProcessTalk) and a tool (ProcessTool) [Wortmann, Rooda 1990, Wortmann 1991] which are used for the development and testing of the architecture. The use of ProcessTool allows the models built with its aid to be validated by simulation. The whole approach allows a smooth transition from modelling the system to its simulation and, finally, to the implementation of the controller [Overwater 1987].

The builder of a control system uses information from design engineers and production engineers, who specify the products, the manufacturing processes, and the physical manufacturing system. These specifications thus place constraints on the control architecture. Simplified general models of the physical machines are used. These machines are commanded to execute operations on material and to report when the operations are finished. It is no part of our present task to consider how machines actually execute their operations. The control system developed here is based on the idea that a hierarchical structure is useful, the factory controller being at the top, machines at the bottom, and a variable number of control layers in between. This hierarchy is constrained by the specification of the given layout of the physical manufacturing system.

Chapter 2 presents the terminology, concepts of modelling, hierarchies, the specification of physical manufacturing systems and a manufacturing system classification. Chapter 3 considers aspects of the control of a manufacturing system. Chapter 4 presents the new concepts and the resulting model of the general control architecture. The control architecture is illustrated in Chapter 5 by showing how it can be applied to a factory for the manufacture of integrated circuits on wafers (IC wafers). The uniqueness and a review of the newly developed architecture for job shop systems is presented in Chapter 6. Finally conclusions are discussed, together with suggestions as to the ways in which this present work may be continued in the future.

Chapter 2
Manufacturing systems

The present chapter deals with some general considerations relevant to all kinds of manufacturing systems, before going on to present the terminology that will be used subsequently. Afterwards the Process-Interaction approach, systems and hierarchies are discussed. Finally a description of the basic components of physical manufacturing systems and the classification of manufacturing systems is given.

2.1 The life phases of a factory

A factory passes through five life phases: orientation, specification, realization, utilization and elimination [Rooda 1987, 1990]. In reality a factory develops in a continuous way, the different phases are gone by in an iterative way and different parts of the factory may be in different phases. But by looking at the factory from one point of view, these phases can always be distinguished. This process is governed by a management system which sets goals for the factory. The main goal of a factory usually is to insure its future existence by making a reasonable profit. The factory will try to make a profit by selling the goods produced to people or to other companies. To sell goods the factory can either try to create a need for its products (technology push) and/or it can anticipate the existing needs of society (market pull).

In the life phases of a factory decisions about the total control and design of a factory have to be taken. It is the task of management to take these decisions and to check whether the decisions are executed in the way planned. The different aspects are mentioned below.

The orientation phase is used to study whether it is interesting to develop a factory. The consumers' needs and suppliers' offers are analysed. Alternatives for products, technology and raw materials are considered. The analysis of the economic expectations means that a decision has to be taken on whether the factory should or should not be built. This decision is usually supported by market research.

The problem definition subphase (which is the first component of the specification phase referred to above) includes decisions as to what kind of products are to be manufactured and what the capacity of the factory

will be (number of workers, machines, stores). In most cases these decisions are taken by the owner or the investor.

Because nature behaves in a causal way, it is possible for humans to look for causes that result in a desired effect. In the specification process this reverse causality is used to find a way to produce the new product [Hubka and Eder 1988]. The main choices, in relation to the manufacturing process, are the raw material from which the product is made, the transformation process that is used to transform the material, and the type of machines that execute the manufacturing process. The transformation process and type of the machines are based on the technology that is considered to be most suitable to transform the material.

The decision about which products a factory is to produce is called 'product planning' and is a task for the management. The specification of a factory and the manufacturing process is an engineering design task [Hubka and Eder 1988]. The design engineer decides what the product looks like. How the product is produced is decided by the production engineer [Kempf 1989].

Once the technology used in the factory has been chosen, the layout, the material transport and storage still have to be specified, as does the control system. These choices cannot be made independently of the technology chosen. When specifying and realizing a control system the specification of the physical system are considered as constraints on the problem of specifying the control system. But, in order to obtain a "good" control system when one is specifying the physical system, the related control problem has to be considered and the specification of the physical system should be adapted in such a way that the complexity of the control system is reduced as much as possible.

The way the factory is to be controlled is also incorporated in the specification of the factory. Is production to order or production for inventory used? What performance criteria are important? What control strategy is used, a fixed or changing product mix, with the possibility of producing new products, etc.

The profitability of a factory also depends on the control strategy. So the market situation, the supplier relationship and the consumer relationship are all likely to influence the control strategy and, through this, the specification of the physical system.

The realization phase is considered to be the task of an external system, the factory builder and the equipment supplier, called the realization system. Here the management instructs the realization system about how the factory is to be realized. The management checks the results and compares them with the specifications.

The system starts to operate in the utilization phase. This is controlled directly by the control system of the factory. This control system is also implemented during the realization phase. The management checks the performance of the realized system, controlled by its local control system. During the utilization phase the control system has to take care of the creation of capacity plans, the marketing of products, the purchasing of raw materials and the manufacturing of products. The factory control system has the responsibility for translating the demand and/or the predicted demand into commands for the machines. Here, too, the consumer has to be negotiated with about the terms under which a product will be delivered: think, for instance, about the due date.

During the elimination phase the management instructs the destruction system about the way the factory has to be disposed of.

2.2 Terminology

We will introduce in this section the terms and notions that are fundamental to our subject.

The transformation of natural means (such as raw material) into desired means (products) is accomplished by a **transformation system** [Hubka and Eder 1988]. One class of transformation systems is formed by industrial systems. In this thesis three forms of **industrial systems** are distinguished: the factory, the manufacturing system and the machine.

To **manufacture** means "1) the making of goods, 2) the process of making wares by hand or machinery" [Burbidge 1987]. When looking at a manufacturing system one can draw a distinction between hardware and software: the manufacturing process is considered as the software and man or machine the hardware.

A **process** is "a set of consecutive operations which complete a significant stage in the manufacture of a component". An **operation** is "the smallest unit of work taken into account for a particular planning or control purpose". And "that which is necessary for the execution of an operation" is called a **resource**. There are two types of resources: expanded resources and leaf resources. An **expanded resource** consists of an aggregate of machines, a **leaf resource** is a machine. **Material** is the operand that undergoes the process. "The materials used as input to a manufacturing system" are called **raw material**, "an end item or output from a manufacturing system" is the **product**. [BSI, 1975].

A factory is part of a bigger whole: the economy. It operates within a market and it has to do with suppliers, consumers and competitors. A **factory** is a system that sells products to **consumers** and that buys raw

materials from **suppliers**. The products are manufactured from the raw materials in one or more resources of the factory.

A **manufacturing system** is a transformation system in which the actual manufacturing of products takes place. Usually this is an aggregate of machines. A manufacturing system is a part of a factory. A manufacturing system consists of one or more resources and a control system, which controls the resources. A manufacturing system transforms one or more types of input material (raw material) into one or more types of output material (finished products). This transformation is called the **manufacturing process**.

A resource behaves in a causal way. Causes have three important components: there are conditions which have to be fulfilled in order to let something happen, there is an internal chain of actions through which something happens, and there is a trigger which starts things happening [Hubka and Eder 1988]. In manufacturing two areas of expertise may be distinguished: the processing of materials (processing technology) and the processing of information (control technology) [Rooda 1987]. The control system triggers the machines and is itself triggered by human beings. It is supposed that a manufacturing system is split into machines that transform material, which is called the physical manufacturing system or the physical system, and a control system that triggers the physical system, which is called the manufacturing control system or just the control system.

The **control system** directs and regulates the physical system in such a way that a predefined goal is achieved as closely as possible. A distinction is drawn between factory control and machine control. **Factory control** is on a more abstract level, it controls aggregates of machines and people. It considers machines as equipment that is started on a command and that gives a signal if the action is finished. **Machine control**, on the other hand, regulates the internal operation of the machine. The biggest difference between the two is the form of parallelism. In machine control the parallelism most often has a fine-grained structure with a very close synchronization between the events. The parallelism in factory control has a more coarsely grained structure with many more or less independent events.

Manufacturing control is comparable to factory control, except that it contains fewer functions. Factory control comprises capacity planning, marketing, purchasing and manufacturing. Manufacturing control comprises manufacturing only. This difference is also found in the commands that both types of controller receive. An order is a command for a factory, which specifies an amount of a certain product type. A job is a command for a manufacturing system, which specifies the material and the manufacturing process.

The **lead time** is the time it takes to execute the manufacturing of a product. In our case the manufacturing process starts with the arrival of an order or a job and finishes when the finished products are available for the requester. The **process time** is the minimum time necessary to execute the manufacturing process. This is the sum of all times the machines of the manufacturing system need to execute the operations of the manufacturing process. The **delivery time** is the difference between the date the ordered goods are delivered and the date the order arrived. The **due date** is the date at which the sender of the order wants its goods delivered. The **throughput** is the amount of products that are manufactured per unit of time. The **input** (rate) is the amount of products of which the manufacturing process is started per unit time. The **inventory level** is the number of products of which the manufacturing process is started and not yet finished.

As we have seen above, five phases characterize the **life cycle** of an industrial system: the orientation phase, the specification phase, the realization phase, the utilization phase and the elimination phase [Rooda 1987, 1990]. In the **orientation phase** the abstract objective of the system is formulated. The **specification phase** is divided into three subphases: (1) the **problem definition subphase**, where the function of the system is set down in quantitative terms; (2) the **operating method determining subphase**, where structures are sought that are able to fulfil the requirements; and (3) the **design subphase**, in which elements of the system are chosen on the basis of the structure found in the former subphase. In the **realization phase** the system is actually constructed. During the **utilization phase** the system functions, if possible in the way specified, and in the **elimination phase** the system is liquidated.

Just like the manufacturing system, the life cycle of a product is also split into five phases. The phases of the product and the phases of the manufacturing system are related to each other. The utilization phase of the manufacturing system coincides with the realization phase of the product. How the specification phases of the two are related to each other depends on which of the two was specified first. With an existing manufacturing system the specification phase of a product is related to the specification of the manufacturing system. With an existing product specification the manufacturing system specification depends on the product specification.

2.3 The Process-Interaction Approach

The Process-Interaction approach is a method for the specification of industrial systems and for the specification and realization of the associated control system [Rooda 1987, Overwater 1987, Wortmann et

al. 1989]. The related language ProcessTalk can be used to make formal and functional specifications of industrial systems [Wortmann 1991]. The ProcessTool supports the specification and realization phase of industrial systems according to the Process-Interaction approach [Wortmann 1991]. It is an interactive graphical environment for the modelling and simulation of industrial systems.

In the Process-Interaction approach an industrial system is considered to consist of a set of parallel processors connected to each other by interaction paths. The representation of an industrial system as a collection of processors, with interactions between them and with a specification of the passive elements, is called a model of an industrial system. The modelling is always done within the framework of a certain problem definition. One of the main criteria by which a model may be judged is the degree to which it represents the relevant aspects of the industrial system. Simulation is used to evaluate the model. One of the strong points of the approach is that it is possible to use the model of a control system as the actual control system.

Top-down design is supported by the process interaction approach. The design of a model usually starts, at a given level of abstraction, with the definition of the processors and the interaction paths. The processor executes certain functions, the functionality of a processor is determined by the interaction ports. These ports connect the processor to the environment.

Processors

There are two types of processors: expanded processors and leaf processors. When the model of the processor is described it may be found that there is still some parallelism inside the processor. This means that the model of the (expanded) processor consists of parallel subprocessors with interaction paths between the internal processors and interaction paths to the rest of the model (the environment), through the interaction ports of the processor. The process of expansion may be repeated, which results in a tree of processors. At the end of such a tree an unexpanded processor is found; this type of processor is called a leaf processor. A subprocessor is also called a child processor and the expanded processor is then its parent processor.

The structure language is the part of ProcessTalk that describes expanded processors in a graphical form. A circle represents a processor, while an arrow represents an interaction path. The name of a processor is printed in the circle, while the name of an interaction port may be attached to the corresponding end of an arrow.

Leaf processors are modelled by a process description. This description lists the actions a processor has to execute and the conditions under which these actions have to be executed.

The task language is the part of ProcessTalk that is used to describe the leaf processors. It is based on the object oriented programming language Smalltalk-80, which is further described in Appendix A. The task language is able, among other things, to express: send actions, receive actions and actions that take time but which are not detailed any further. The task language is described in Appendix B.

A passive element is always inside a processor. A passive element may have a value; if it does not, it is the presence of the element that counts. It is transferred from one processor to another processor through an interaction path. The state of the model depends on the passive elements present in the model. A processor changes the state of the model by the execution of actions. These actions consist of creating a passive element, receiving a passive element, changing the value of a passive element, sending a passive element, or deleting a passive element.

Interactions

The transfer of a passive element from one processor to another processor is called an interaction. The purpose of an interaction is communication and/or synchronization. There are two types of interaction: a send action and a receive action. A send action makes a passive element available for interaction, a receive action takes a passive element that is available for interaction.

The interaction (the transfer of the passive element) takes place through an interaction path. An interaction path is connected to two different processors. The connection to a processor is called a port. There are two types of ports: a send port and a receive port. In order to be able to distinguish between ports, every port has a name. The interaction path is directed, it starts at a send port and it ends in a receive port. The transfer of the passive element occurs in zero time.

The send and the receive action specify which port is involved in the interaction. More than one interaction path may be connected to a send or a receive port. If a send or receive action involves more than one interaction path, then the interaction takes place along that path where the other side was first in executing its receive or send action.

For further detail on the Process-Interaction representation of a model see [Overwater 1987], and for further information on ProcessTalk (the

structure and task language) and ProcessTool (the modelling and simulation environment), see [Wortmann 1991].

2.4 Systems

There are three types of systems: real systems, conceptual systems and formal systems. Real systems actually exist in the real world. Conceptual systems are abstractions or constructions created by the human mind. Conceptual systems exist only on paper and point towards reality. Formal systems are systems that are used to create conceptual systems [Nauta 1974]. The terms for these three types of system that will be used in this thesis are: 'system' for real system, 'model' for conceptual system and 'language' for formal system [Rooda 1990].

A model is a representation of a system which contains the essential properties of the system. The essential properties depend on the objective for which the model is intended. The model can be used to examine and to predict the behaviour of the system. The accuracy of a model depends on the effort that is put into the construction of the model and on the accuracy of the available data. These factors are influenced by the accuracy that is required.

Modelling is a means to study and design real systems. Modelling is considered to be an art, which means there are no specific guidelines which lead to a good model; in fact it is a rather intuitive process.

The use of a model helps in structuring one's thoughts about the system, in understanding the system's behaviour and in differentiating relevant data. With a model it is possible to conduct experiments to test the system's sensitivity to certain factors and to test the effect of changing the system. It is often difficult or impossible to conduct these experiments with the real system.

There are two kinds of models: iconic models, which are visual representations of a system; and symbolic models, which represent the properties of the system with help of mathematical symbols and relations [Smedinga 1988].

Symbolic models are used to perform calculations. There are three kinds of symbolic models, which are differentiated in terms of the way a solution is constructed: analytical, numerical and simulation models [Smedinga 1988]. All models represent only part of reality and are used to investigate some aspects of this reality. A disadvantage of simulation models over analytical and numerical models is that it is not possible to prove the mathematical correctness of simulation results. For complex models it is often impossible to find an analytical or numerical solution.

In that case simulation is the only possible way to solving the problem. Before simulation is started, however, a model first has to be constructed; this model is then transformed into an algorithm which is run on a computer.

Real systems are divided into two types: continuous time and discrete time systems. In discrete time systems variables change discretely with time while in continuous time systems variables change continuously with time. Whether a system is discrete or continuous depends on the view of the observer. Most systems are partly discrete and partly continuous. In a manufacturing system, for instance, the changes material undergoes inside a machine are of a continuous nature. On the level of abstraction where the inside of the machine is not considered, however, these changes can be viewed as having a discrete character.

In this thesis the systems are factories and manufacturing systems. The language used is ProcessTalk, which is supported by a tool: ProcessTool. Only the discrete nature of systems is considered and the model presented in this thesis concentrates on the control context.

2.5 Hierarchies

Different kinds of hierarchies are found in the control context. Some of these hierarchies coincide with each other, but most of them are not completely coupled. This means that the levels of the different hierarchies are usually intermixed. Much of the vagueness found in the literature about the control of manufacturing systems is caused by the fact that the different hierarchies are not distinguished.

Three forms of hierarchy may be distinguished.

- The first hierarchical form is the system hierarchy. A system hierarchy is a control hierarchy. The system controls subsystems, but the subsystems are not a part of the system. An example is found in the army, where a lieutenant commands sergeants, and a sergeant commands soldiers.

- The second form is the model hierarchy, which is a form of aggregation and decomposition. The system is split into smaller subsystems, and all subsystems together form the original system. The subsystem is part of the system. An example is time. A week is split into seven days, a day into 24 hours, etc.

- The third form of hierarchy is the inheritance hierarchy. This hierarchy classifies systems. An example is the taxonomy of biological species.

All three forms of hierarchy are found in manufacturing control. The most striking hierarchy is the system hierarchy. The manufacturing system consists of resources that are controlled by one (hierarchical) controller. The manufacturing system executes one or more manufacturing processes: a manufacturing process consists of operations that have to be performed to transform the input material into the output material. The resources of a manufacturing system are able to perform these operations. The resource may consist of a central controller and subresources. The invocation of a manufacturing process is also an operation. As a consequence the manufacturing system itself is also a resource.

Many decisions have to be taken by a controller. The controller often takes these decisions in a step by step manner. The control problem is solved with the use of an algorithm having a hierarchical nature. For instance, a piece of material can be transported to several resources and there are more than a single transporter available. In this case the controller might first allocate the material to one resource and afterwards allocate the transport job to one of the transporters.

A manufacturing system is controlled by a hierarchical control system. The manufacturing system also has a model hierarchy, which coincides with the control hierarchy. The model hierarchy is related to the configuration of the manufacturing system. The hierarchical levels that are often distinguished are: factory, facility, shop, cell, station and machine [Beukeboom et al. 1989]. The related system hierarchy contains the levels factory controller, facility controller, shop controller, cell controller and station controller.

Other model hierarchies are concerned with aggregation: aggregated information is used, especially during planning. The demand information is aggregated and a coarse plan is calculated on the basis of this information. In different phases, the information is decomposed in order finally to generate detailed manufacturing commands for the different machines. Examples of aggregation are: consumers taken together in consumer groups, products taken together in product families, machines taken together in machine groups, time intervals taken together to form longer time periods. These are all examples of model hierarchies.

The product structure also has a hierarchical nature: this is true for both the material structure and the operation structure. For the material structure three forms are distinguished: (1) the assembly structure; a product consists of one or more parts; (2) the arborescence structure; different products all are made of the same part; and (3) the general product structure, which is a combination of the former two [Joensson 1983].

The operation structure of a product has a hierarchical nature, too. The hierarchy of the manufacturing system (facility, shop, cell, etc.) is also found in the operation structure. The operation of a facility is discharged by the execution of operations for shops. The operation of a shop consists of operations for cells, and so on. Both the product structure and the operation structure form a model hierarchy. These hierarchies are related but do not necessarily correspond to each other on a one to one basis.

Smalltalk-80, which forms the basis for the task language, uses the inheritance mechanism and contains a class hierarchy. Every object in Smalltalk-80 belongs to a certain class; as a consequence all objects in the model belong to a class. These classes are part of the Smalltalk-80 inheritance hierarchy.

As has been illustrated above some hierarchies coincide with each other even if they are of a different form. Other hierarchies do not coincide although they are of the same form. In the rest of the thesis it has to born in mind that two hierarchies do not necessarily coincide.

2.6 Basic components of physical manufacturing systems

The physical manufacturing system is defined by the operations, the material, the resources and the recipes. An operation is performed on material by a resource. The recipe describes the manufacturing process as a sequence of operations. These concepts are all discussed below.

Operations

An operation is a unit of work that is executed by one resource. There are operations that are specific to the manufacturing process (which is determined by the processing technology and the manufacturing machines) and there are operations that also depend on the status of the manufacturing system. The first are called the processing or value adding operations, while the second are supporting operations. An inheritance hierarchy for operations is given in Figure 2.1.

The execution of an operation changes material; input material is changed into output material. The change is related to the material properties in the case of manufacturing operations. A verifying operation changes the data about the material. A transport operation change the spatial location of the material. The store operation is related to a

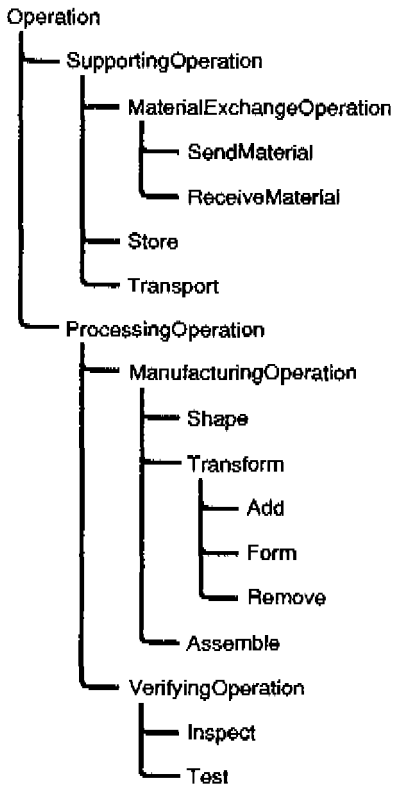


Figure 2.1. *The operation inheritance hierarchy.*

change of the time coordinate. The material exchange operations are related to the directing of material interactions between processors.

The processing operations are manufacturing or verifying operations. Transport, store and material exchange operations are supporting operations. The two material exchange operations, receive material and send material, are considered to be inherent actions which are performed before and after the execution of another operation.

The processing operations only change if the manufacturing technology changes. Processing operations are static and determined in advance. The execution of supporting operations depends on the status of the manufacturing system; for this reason supporting operations are not a part of the manufacturing process description. Verification operations are considered as processing operations because these operations usually form an integral part of the manufacturing process.

Several classifications of manufacturing operations have been published [Spur and Stoeferle 1981, Ehrlenspiel 1985, Buma 1987]. Here

the following types are distinguished: shape, add, remove, form and assemble. These operations transform material. For clarity two forms of material are distinguished: discrete products and bulk goods. Bulk goods are also considered to include small parts, fluids, gases and so on. The control system is considered to direct only the manufacturing of discrete products.

The shape operation creates a discrete product from bulk goods. An add operation specifies the addition of some material to a discrete product. This means that a discrete product and some bulk good is transformed into another discrete product. Remove means that some material is removed from a discrete product: this results in waste (a bulk good) and a discrete product. Form means the transformation of a discrete product into another discrete product. The form operation is supposed to change the geometrical properties or the internal material properties. The assemble operation means the putting together of two or more discrete products to form a new discrete product. Because bulk goods are neglected, the operations add, remove and form are abstracted into the general term transform.

The verifying operations are necessary to achieve a specified quality. The verification is split into two categories: test and inspect. The test operation checks the functional performance of a piece of material, while an inspect operation checks a piece of material in a visual manner (surface and geometric properties) [Herroelen 1985]. A verifying operation makes specific information about the state of material available. The route of material through the manufacturing system may be influenced as a consequence of a verifying operation; for example the material may be reworked, repaired or rejected. Rework means that the material is processed once again on some resources it has already passed; repair means some additional operations are exerted on the material; reject means the material is removed from the manufacturing system and not processed any further. Other uses of the verification information are to modify the performance of an upstream resource (feedback) or to modify the way the material will be processed in a downstream resource (feedforward).

The transport operation is needed to move material from one resource to another. The material route through the (physical) manufacturing system is not part of the manufacturing process. The destination of the material is determined by the controller. This way it is possible to let the controller decide which resource executes an operation and to realize dynamic material routes.

The store operation is in fact not a real operation: it is the passage of time between the receive operation and the send operation. So every resource

can store material. During a store operation (in general) the material does not change.

The manufacturing operations assemble, transform and shape completely specify the product. The technology chosen, and the resource types inside the manufacturing system, dictate which operations are used to manufacture a product. These operations add value to the product. The verifying operations are necessary to achieve a certain quality standard. The verifying operations and the manufacturing operations together form the processing operations. These operations form the recipe and describe the manufacturing process. Material transport is needed in order to execute a manufacturing process. Store operations are used to allow the manufacturing process to proceed in a smoother or optimum manner. Material exchange operations command the exchange of material between processors. The transport, the store and the material exchange operations are not part of the manufacturing process description (the recipe), they depend on the status of the manufacturing system.

Material

A manufacturing system transforms raw material into finished products. A resource performs a part of the total transformation. Both raw material and finished products are called material. The material in the manufacturing system has a discrete nature. Materials having a non-discrete nature, such as small parts, bulk goods, fluids, gases and waste, are not modelled; it is supposed that they are taken from an unlimited supply, or placed in an unlimited store.

Every resource has the same material interface. It is supposed there is a unit load that is the same in the whole manufacturing system. All resource batch sizes and capacities are expressed in terms of this unit; all material fits into this unit load.

For a piece of material, a material unit, there is a route through the resource, which is expressed in terms of operations. The operation that a material unit has to undergo indicates which type of resource the material unit needs for the execution of the next operation. It forms the basis on which the destination of the material unit is determined. The status of the material depends on the progress of the manufacturing process; the operations that have been exerted on the material, and on the quality of the material; the way the operations have been exerted on the material.

Resources

A resource is a system that executes operations. In a manufacturing system the operation is performed on a load of material. When the resource is to execute an operation, the material on which this operation has to be performed must also be specified. A resource performs three basic actions: receive material, process material and send material. The process material action is the actual execution of a store, transport or processing operation. Receive and send material actions are the execution of a material exchange operation.

Resources are divided into expanded and leaf resources, leaf resources being machines. Another classification of resources is into processing resources and supporting resources. Stores and transporters are supporting resources; all other resources (shaper, transformer, verifier, assembler and expanded resources) are processing resources. The resource inheritance hierarchy is described in Figure 2.2.

The machine is characterized by the fact that it consists of a physical system (the device) and a control system (the machine controller) [Rooda 1990]. The machine controller is not considered: only the interface of the machine with the control system is of interest. The capabilities of a machine are the operations a machine can execute. The status of a machine is either idle, receiving, processing, sending or

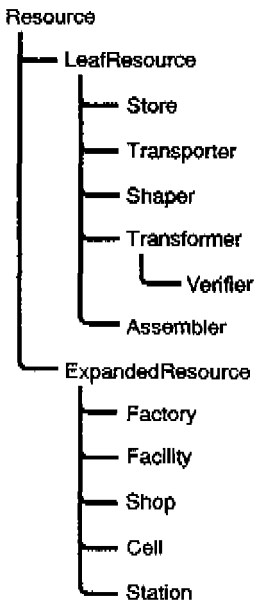
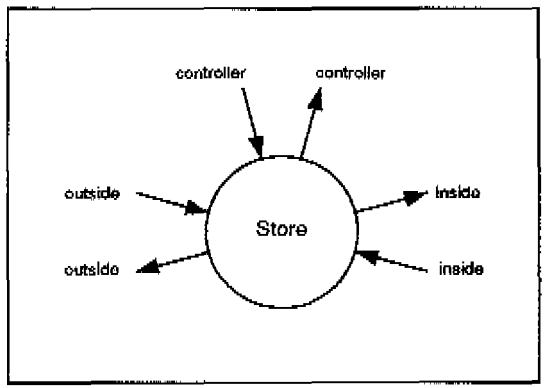


Figure 2.2. *The resource inheritance hierarchy.*

down. The amount of material in a machine is also a determinant of its status. All machine types are characterized by a maximum amount of unit loads they can contain. Another characteristic of the processing machines is their batch size. Processing machines are able to process loads of material and this load is characterized by a minimum and maximum amount. The maximum amount of material a machine can contain is equal to its (maximum) batch size. The capacity of a machine is the amount of material a machine can process per unit of time.

The function of a store can be manifold. A store is used to store goods before and after processing in order to compensate fluctuations in the arrival and departure of goods, caused by down times, stochastic variations in process times and so on. A store is used to prevent deadlock (see Section 3.5). It is used as an interface between transport systems and it is used to collect material in order to manufacture a load of material as a batch. The model of a store is described in Figure 2.3. A resource of type store only performs the actions receive material and send material. A send material action is invoked by a send material operation (material request), the receive material is not invoked by an operation. The store is connected to an internal transport system (with the ports inside) and to an external transport system (with the ports outside)




```
Store > body
    | material |
        self receiveFromOneOf: #('outside' 'inside' 'controller' ) do:
            [:portName :item |
            (item isKindOf: MaterialRequest)
                ifTrue:
                    [material := self removeFromBuffer: item.
                    self send: material to: item destination].
            (item isKindOf: Material)
                ifTrue:
                    [self addToBuffer: item]]
```

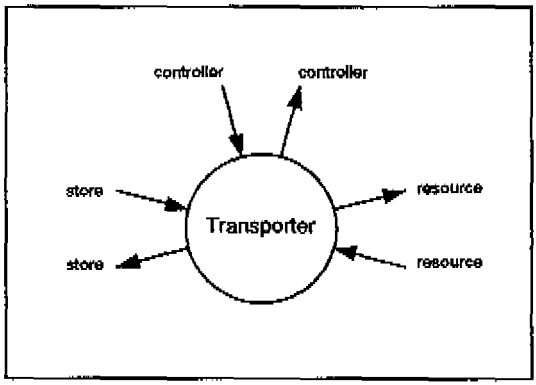
Figure 2.3. The model of a store.

A machine of the transporter type moves material from one place to another without changing the properties of the material. The model of a transporter is described in Figure 2.4.

A machine of the transformer type changes the material it is processing and delivers it in a new state. The model of a transformer is shown in Figure 2.5. A machine of type verifier does not transform material, it collects information about the state of the material it is verifying. Its model, however, is the same as the model of a transformer.

A machine of the shaper type creates new discrete pieces of material. Its model is similar to that of the transformer, with the exception that it does not receive material from the transporter before the execution of a job.

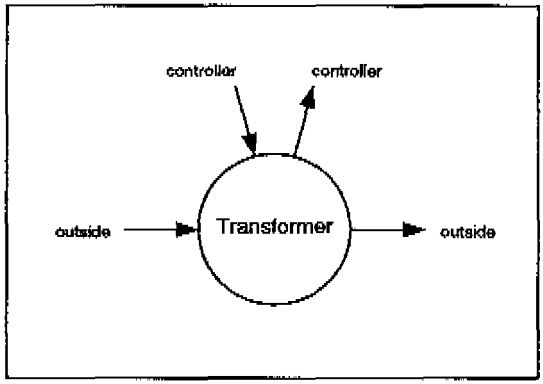
A machine of the assembler type combines two or more pieces of material and delivers it as one piece. Its model is also similar to the transformer, an assembler only receives more pieces of raw material.



Transporter > body

```
| request transportJob origination destination materialData material report |
request := self formulateRequest.
self send: request to: 'controller'.
transportJob := self receiveFrom: 'controller'.
origination := transportJob origination.
destination := transportJob destination.
materialData := transportJob material.
self moveTo: origination.
material := self pick: materialData from: origination.
self moveTo: destination.
self place: material at: destination.
report := self formulateReportFrom: transportJob and: material.
self send: report to: 'controller'
```

Figure 2.4. *The model of a transporter.*




```
Transformer > body
    | request rawMaterial job operation finishedMaterial report |
    request := self formulateRequest.
    self send: request to: 'controller'.
    rawMaterial := self receiveMaterialFrom: 'outside'.
    job := self receiveFrom: 'controller'.
    operation := job operation.
    finishedMaterial := self execute: operation on: rawMaterial.
    report := self formulateReportFrom: job and: finishedMaterial.
    self send: report to: 'controller'.
    self sendMaterial: finishedMaterial to: 'outside'
```

Figure 2.5. *The model of a transformer.*

Recipes

The product structure describes a product, either in the form of an operation structure or a material structure. The operation structure, or the 'recipe', is a list of operations that have to be executed in order to make the product. The material structure is a list of parts that make up the product. In this study only the recipe is used to specify products.

The operation structure describes the manufacturing process. The description varies from very simple to very complex and is called a recipe. A list of sequential operations is a relatively simple form, the assembling of subassemblies followed by a series of operations on the assembly is already more difficult. In the following a number of recipe forms are discussed: sequence, concurrency, alternative and block. The inheritance hierarchy of recipes is printed in Figure 2.6. These recipe forms may be nested to realize complex operation structures. A recipe consists of a collection of recipe steps; a recipe step is either an operation or a recipe. A recipe indicates which operations have to be performed,

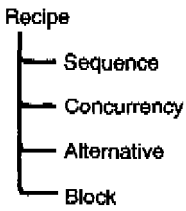


Figure 2.6. *The inheritance hierarchy of recipes.*

what type of material is involved and in which sequence these operations have to be executed.

The two basic recipe structures are the sequence and the concurrency. A sequence is a list of recipe steps that have to be executed one after the other (sequentially). A sequence usually refers to one type of material. An example is

Sequence
(transform (1) material of type A,
transform (2) material of type A,
transform (3) material of type A).

A concurrency means that all the recipe steps of the recipe may be executed at the same moment, but also at different times, for instance:

Concurrency
(transform (1) material of type A,
transform (2) material of type B,
transform (3) material of type C).

A concurrency, most of the time, refers to different types of material. If a concurrency refers to one type of material, the recipe steps of the concurrency are executed one after the other in a random sequence. These two recipe structures allow the description of the assembling of a product, for example:

Sequence
(Concurrency
(Sequence
(transform (1) material of type X,
transform (2) material of type X,
transform (3) material of type X),
Sequence
(transform (4) material of type Y,
transform (5) material of type Y)),
assemble (6) material of type X and Y into Z,
transform (7) material of type Z,
transform (8) material of type Z)

Other recipe structures permit more sophisticated operation sequences to be implemented. The alternative indicates that only one of the recipe steps in the list has to be executed. If, for instance, a product can be manufactured in two ways (way I and way II) then the recipe is as follows:

Alternative
(recipe specifying way I,
recipe specifying way II)

The block is meant to indicate that more operations have to be executed simultaneously. All the recipe steps of a block have to be executed at the same moment. The recipe steps of a block have to be operations, unlike in other recipes the steps may not form a recipe themselves. This kind of structure is necessary in some control situations in order to prevent deadlock (see Section 3.5) of resources; all the resources involved have to be ready before the execution of a block may start.

A second situation in which the block is useful is when the second operation has to start directly after the finishing of the first operation. A block mostly refers to one type of material, which means that the operations of the block are executed one after the other. If the recipe steps of the block refer to different types of material the recipe steps are executed simultaneously. An example:

Block
(transform (1) material of type A,
transform (2) material of type A)

The basic components presented in this section -operations, material, resources and recipes- are used to specify the physical manufacturing system and manufacturing process. The leaf resources, the material, the operations of the leaf resources, and the recipe with these operations form the parameters for the control architecture.

2.7 Classification of manufacturing systems

The control architecture is intended for the control of manufacturing systems. The following classification of manufacturing systems will be used to discover demands which the architecture has to fulfil. A rough distinction between manufacturing systems is made by considering the universality of the resources and the route flexibility inside the system; this results in the classes flow shop and job shop. A flow shop is characterized by dedicated resources and a fixed route, while in a job shop there are universal resources which can be used for many different operations and many possible routes. For a more complete classification

of manufacturing systems there are more factors to be taken into account.

Several classifications of manufacturing systems have been presented in the literature [Kittel 1982, Van Rijn 1986, Van Rijn 1988]. The following parameters are distinguished: the material in a manufacturing system, the recipes of the manufacturing system, the resources of the manufacturing system (with which the universality of resources and flexibility of material routes are meant) and the control situation of the manufacturing system. These parameters are explained below.

The material in a manufacturing system is split into the categories raw material and finished products. For classification purposes the commonality of raw material, the assortment of finished products, the standardization of finished products and the material structure of the finished products are distinguished. On one hand there are manufacturing systems which use only a few types of raw material for the manufacturing of their products (large commonality) while, on the other hand, there are manufacturing systems that use many different types of raw material in their products (small commonality). The assortment gives an indication of whether the manufacturing system makes many or few types of products. The standardization of products plays a role in an assortment, too: do the products differ only in colour or are they manufactured in completely different ways? The products of a manufacturing system also vary in the complexity of the material structure. There are manufacturing systems that assemble complex products and there are manufacturing systems that simply transform the raw material.

The control architecture has to be capable of handling the different categories of material flow in the manufacturing system. Only the effort required to implement the architecture will differ: a small commonality and big assortment result in the need for a lot of data, which makes it a big effort to implement the architecture.

The recipes classify manufacturing systems in two ways: (1) the number of recipes the manufacturing system is capable of executing; and (2) complexity of the recipes. There are manufacturing systems having a few recipes and those with many recipes. This classification parameter is related to the product assortment. The complexity of a recipe has to do with the nesting structure of the recipes and with the number of operations that have to be executed to make the product. This parameter may be related to the complexity of the material structure, but there are also products that contain no assemblies but that still have a lot of operations involved in their manufacturing. Further in this section a distinction is made between recipes with a single operation and recipes with multiple operations.

The control architecture has to be capable of handling all kinds of recipe structures: the number of operations in a recipe is limited only by the computer power. The number of recipes in a manufacturing system is also only limited by the capacity of the control computer. The architecture, however, starts from the static situation, where the recipes are known in advance. Even for situations where many different product types are manufactured in small series, the architecture will be applicable, if these recipes are available in advance.

The classification of manufacturing systems with the help of the resources allows us to distinguish two related topics: the universality of resources and the flexibility of the transport system. On the one hand there are dedicated resources that are capable of only one operation and on the other hand there are resources that can execute a lot of different operations. The flexibility of the transport system influences the route flexibility in the manufacturing system. If the transport system is only capable of transporting material from one resource to the next, the route is fixed. If the transport system is capable of transporting material from any resource to any resource, then one has maximum route flexibility. The route flexibility also depends on the decoupling of resources, by which is meant the possibility of processing material in a different sequences on different resources.

The universality of machines and the route flexibility are usually related. Dedicated machines mostly form part of manufacturing systems with a small route flexibility, for instance the flow shop where the route of the material through the manufacturing system is fixed. Universal machines often form part of manufacturing systems with a large route flexibility, which then results in a job shop.

The control architecture is intended for the job shop situation, but the flow shop and other manufacturing configurations having a lower degree of route flexibility or which contain less universal machines are in fact simplifications of the job shop and also will be handled with this control architecture.

The control situation delivers classifying parameters, such as the uncertainty in the market situation, manufacturing job sequencing strategy, the production rate and series sizes. The uncertainty in the market situation is composed of the uncertainty in demand and the uncertainty in delivery. The uncertainty in the market situation is an important factor for determining the production strategy, which can be based on consumer orders, on a fixed production program or on production for inventory. Production rate and series sizes are usually related. Production rate is the number of products manufactured in unit time; series size is the number of products of the same type, manufactured consecutively.

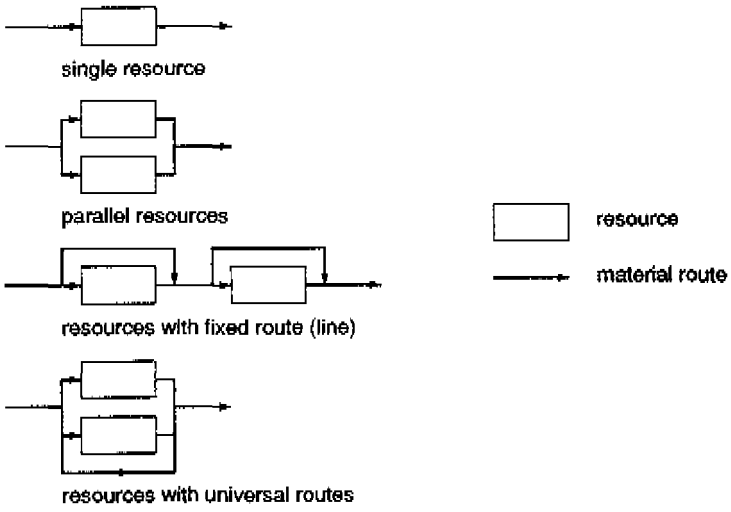


Figure 2.7. *The basic resource layouts of a manufacturing system.*

The control architecture focuses on manufacturing control and pays little attention to the interactions with consumers and suppliers. These interactions are part of the factory controller; all control situations have to be handled by adapting the control algorithms of the factory controller, without the need to change the rest of the control architecture.

Up to now we have seen four parameters that classify manufacturing systems. In the next part only the recipe and the resources of the system are used to distinguish four classes of manufacturing systems, which are of interest for the control situation. For the recipe two categories are distinguished: recipes consisting of a single operation and recipes containing more than one operation. The resources in the manufacturing system are characterized by the resource layout. The resource layout is a function of the number and types of processing resources and the material routes that can be realized by the transport system.

There are four basic layouts; single resource, parallel resources, resources with a fixed route and resources with universal routes. The layouts are represented in Figure 2.7. The resources with a fixed material route form a line. It is supposed that in a line material is allowed to pass a resource on its route without being processed on the specific resource (skipping). Many layouts forms exist between resources with a fixed route and resources with universal routes. These forms are all considered to belong to the universal route case.

The recipe is based on the manufacturing process, the recipe has to correspond with the resource layout. The recipes with a single operation

Table 2.1. *The relation between resource layout, recipe and manufacturing system classes.*

resource layout	recipe	
	single operation	multiple operations
single resource	single shop	-
parallel resources	parallel shop	-
resources with fixed route (line)	parallel shop	flow shop
resources with universal routes	parallel shop	job shop

can be executed by all four resource layouts, but the resources with a fixed route and the resources with universal routes behave as parallel resources to the recipe with a single operation. This results in two different manufacturing classes: the single shop and the parallel shop. The recipes with multiple operations can only be executed by the resources with a fixed route and by resources with universal routes. This results in the flow shop and the job shop, respectively.

Four basic manufacturing system classes are considered in the rest of this thesis. These are the single shop, the parallel shop, the flow shop and the job shop. The classes in relation to resource layout and recipe are represented in Table 2.1. The parallel shop can be equipped with identical or different resources, the flow shop may be a pure flow shop (no skipping) or a flow shop with skipping. A manufacturing system which does not belong to the classes single shop, parallel shop and flow shop is always regarded as a job shop. In the job shop the same resource may be visited more than once by a piece of material, which means cyclic material routes are allowed. This view of the manufacturing configuration has not considered the stores. The stores are supposed to be placed at the interface of the manufacturing system with the outside world, and inside the resources if these are expanded.

Chapter 3
Control of manufacturing systems

3.1 Control concepts

A distinction is drawn between the control system and the controlled system. The control system regulates the actions of the controlled system. To do this the control system has to have some conception of the goal that has to be reached. In order to control, there have to be interactions between the control system and the controlled system. The control system influences the controlled system with stimuli and it needs to know the status of the controlled system and the response of the controlled system to a stimulus. By comparing the response and the status of the controlled system with the goal, the control system determines the stimuli for the controlled system. These stimuli are also influenced by the stimuli the control system receives from the outside world, called the environment.

The activities in a discrete system have to be triggered and coordinated. These activities are themselves triggered by signals from the environment. The triggering of the internal actions as a response to stimuli from the environment is the task of the control system. In some cases the control system sends stimuli to the environment in order to influence constraints imposed by the environment and to influence the stimuli the environment sends to the control system.

The goal of a control system is often defined as a value or criterion that has to be optimized. The control system uses a strategy, which describes the way the goal should be achieved. The way the control system and the controlled system react to the stimuli from the environment is called the behaviour of the system. The behaviour of the system is determined by the goal and the strategy of the control system, by the capability and the capacity of the controlled system and by the constraints and stimuli from the environment.

In manufacturing the control system processes information and the controlled system processes material. The control system directs the material flow through the controlled system. In the leaf resources, or machines, the information and material meet each other. Before the working of the manufacturing controller is elaborated further, we discuss the information in a manufacturing system.

The types of information in a manufacturing system are configuration information, the system status, the system history and objects that are used for communication. The control system has knowledge about the configuration: the resources that it controls, the physical layout of the resources and the manufacturing process that is performed by the resources (the recipes). These data about the configuration are considered to be unchanged during the control of the manufacturing system.

In order to be able to take its control decisions, the control system needs to keep a record of the status of the physical manufacturing system. This record includes information about the material in the manufacturing system, which operations have been performed on the material, which operations are being performed on the material by which resources, and which operations still have to be performed on the material. To be able to evaluate the performance of the manufacturing system, the controller has to keep track of the history of the manufacturing system. Things like the up, down and idle times of resources, the process time and the wait time of material are represented in the form of statistical excerpts. Records of the events that happen in the manufacturing system are kept in logs to be able to replay events in order to trace causes of errors. The statistical excerpts and the logs change over time.

The control system of the manufacturing system regulates the manufacturing of products: this is done by commanding the resources in the physical manufacturing system and collecting data from these resources and sending data to the environment. With the help of the resources the material is directed through the manufacturing system. The control task is divided into planning, scheduling and monitoring. These subtasks result in actions taken by the control system. The control system starts the manufacturing process, distributes the work to the resources, takes care of the progress of the manufacturing process and signals the finishing of the manufacturing process. The controller of a manufacturing system commands the resources of the manufacturing system and in this way it directs the material through the system.

The monitoring of the manufacturing system is split into three functions. The first function is the momentary recording of the progress of the manufacturing process: the recording of the status of the material that is being processed. The second function is the recording of the activities of the resources. Related to these functions is the signalling to the environment about the status of the manufacturing system and the process. In the third place, the control has to record and check the overall performance of the manufacturing system over a longer time interval.

Factories and manufacturing systems have a certain capability and capacity. They are able to manufacture products, they are designed for a certain throughput, the manufacture of a product has a more or less

defined lead time. These things are fixed in the configuration of the systems (resources, layout and recipes). The control of the systems has to fulfil certain criteria concerning such matters as costs and quality. And the systems have to react to stimuli. The control system uses the system, with its fixed capability and capacity, by reacting to the stimuli while optimizing the criteria.

The stimuli for factories are orders for amounts of products; the stimuli for manufacturing systems are jobs describing loads of work to be executed. The goals of factories and manufacturing systems differ. A factory tries to secure its future existence by fulfilling orders as completely and as well as possible, while a manufacturing system has to execute jobs as rapidly and as well as possible. These goals are translated into criteria like lead time, throughput and due date reliability. The factory is an independently operating system; a manufacturing system is a kind of slave system which is part of the factory. This makes the factory controller more complex than the manufacturing controller.

A factory receives stimuli: the orders from consumers and material from suppliers. The factory sends stimuli to its environment: orders for suppliers and, perhaps, advertisements to consumers. The factory control system has to adjust the performance of the factory to the market demand. The capacity of the factory is fixed at the moment the factory is built, the capacity is based on market forecasts. If the demand deviates from the forecast it may be impossible for the control system to fulfil the demand. If demand is higher than supposed, the factory is overloaded with orders or orders have to be refused. If demand is lower than predicted the factory is underutilized and this leads to a lower return on investment. Another difficulty arises from the purchasing of material from suppliers. If suppliers are unable to deliver or deliver too late, it may also be impossible for the factory to satisfy the demand.

The manufacturing system receives stimuli from its master. This is a control system, which sends jobs to the manufacturing system and takes care of the supply of material necessary for the execution of the jobs. The manufacturing system sends stimuli to this master in the form of signals that it wants work. The capacity of a manufacturing system is fixed, which means that the manufacturing control system has to ensure that the system is neither overloaded nor underutilized. But, to a large degree, this is the responsibility of the master. The manufacturing controller has to signal to its master that it needs more or less work. It is the responsibility of the master to use the manufacturing system in the way it was designed to be used. The manufacturing controller has to enable the master to do this by realizing a behaviour that is easy to understand and to predict, so that the master can observe the consequences of proper or improper usage of the manufacturing system.

Measures of performance

In order to use resources efficiently it is necessary to attune the work load to the capacity. An ideal factory and an ideal manufacturing system are perfectly balanced (see below), which means that the resource capacity is adjusted to a certain throughput level. At this throughput level it is possible to match the work load perfectly to the resource capacity. The inventory level is decisive for the work load of the system: the ideal work point of an ideal manufacturing system is an inventory level which is equal to the sum of the batch sizes of all machines. In the ideal manufacturing system, operating at its ideal work point, resources are never idle and material never has to wait to be processed. In the ideal case the work load is equal to the capacity all the time.

The work load, however, is not exactly known in advance: neither the work load nor the capacity are constant in time. A factory is confronted with fluctuations in demand, uncertainties in purchasing raw material and variations during manufacturing. The demanded products, amounts, order times and due dates required by the consumer vary in time. The delivery dates of material from suppliers usually are difficult to predict in advance. The manufacturing process itself is also confronted with disturbances; capacity fluctuations caused by machine failures, maintenance and repair, and yield fluctuations which result in rework, repair and rejection of material. These disturbances cause the manufacturing times to vary. Other factors that contribute to variations are differences in process time for operations, differences in recipes for products, differences in batch size of machines, differences in batch quantities for jobs, variations in the product mix that a factory or a manufacturing system manufactures, and variations in the amount of work a factory or a manufacturing system is processing.

The performance of a factory is related to the amount of products that the factory manufactures, the costs that are incurred in the manufacturing of those products, and the prices received for the manufactured products. This performance has to result in (positive) profits, because otherwise the existence of the factory is jeopardized. The performance of a manufacturing system, on the other hand, has to do with the relation between the specified behaviour and the actual behaviour. A manufacturing system is designed to be able to manufacture a certain amount of product per hour in a certain process time, under defined constraints (throughput, lead time, inventory level). If the manufacturing system is operated in the way specified, the performance has to be close to the specification.

There are three measures that are frequently used to assess the performance of a factory: due date reliability, mean lead times of products and utilization degree of machines.

Due date reliability is a difficult measurement, because due dates are usually created in a negotiation process between the factory and its consumer. Whether due dates are realizable depends on the agreed due date (whether it is a realistic due date), on the work available in the factory, and the due dates of all of the jobs in the factory.

The lead time of products as a performance measure is not trivial. Different products usually have different process times. If, within a period of time, many products with short process times are produced, the mean lead time is smaller than in a period in which most of the manufactured products have long process times. Another important influence on lead time is the inventory level within the manufacturing system. The bigger the inventory level of a manufacturing system, the longer the mean lead time.

Utilization is a difficult performance measure because it concentrates on the performance of single machines. The utilization is a function of the available capacity and load (the amount of orders) assigned to the factory. If the available capacity is fixed and the load is adapted to the capacity, the scheduling algorithm has to ensure that the resources are utilized as efficiently as possible: this means that idle times of resources have to be minimized, just like set up times.

In order to measure the performance of a manufacturing system, the intention of the manufacturer has to be born in mind. The maximum throughput of a manufacturing system is fixed, a manufacturer wants his system to manufacture close to the maximum throughput, with short lead times, small inventories and reliable due dates. In the ideal system, as we have seen, the wait times of products are zero, as also are the idle times of resources. The inventory level is equal to the amount of material all resources in the system are able to process simultaneously. The performance of the manufacturing system is found by studying the relationships between lead time, throughput and inventory level.

The lead time as a function of the inventory level is shown in Figure 3.1. The throughput as function of the inventory level is shown in Figure 3.2. These relations are found if the manufacturing system is in a steady state. The steady state is reached if the input rate and the output rate of the manufacturing system, measured over a period of time, are equal. This means that during the measurement the inventory level is more or less constant. There is a relation between throughput, inventory level and lead time [Little 1961, Wiendahl 1987]:

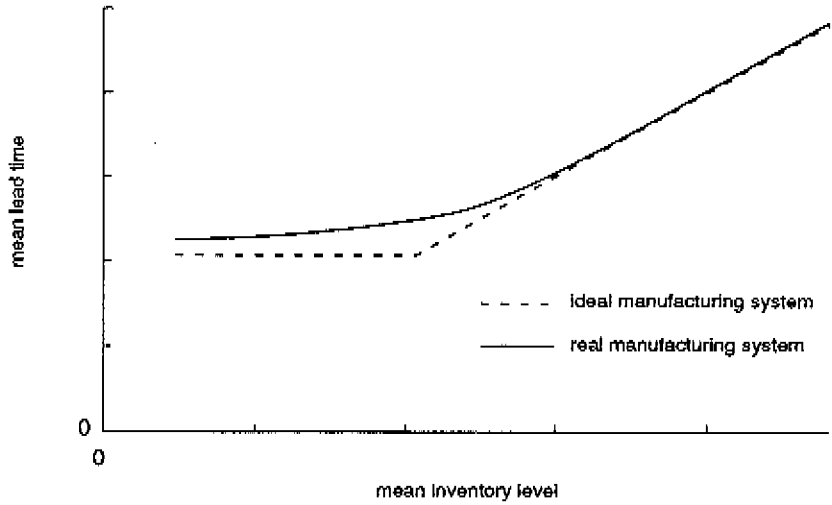


Figure 3.1. Mean lead time as a function of the mean inventory level.

$$\text{mean throughput} = \frac{\text{mean inventory level}}{\text{mean lead time}}$$

In an ideal manufacturing system the throughput and the lead time as function of the inventory level are like the dotted lines in Figures 3.1 and 3.2. The lead time is equal to the nominal process time until the system is saturated. This happens when the inventory level is equal to the sum of the batch sizes of the machines. From here on the lead time increases linearly with the inventory level. The throughput, on the other hand, increases linearly with the inventory level until the system is saturated; from the saturation point on the throughput remains constant and is equal to the maximum throughput.

The behaviour of less ideal systems is shown as solid line in Figure 3.1 and 3.2. In such systems the lead time always increases with the increase of the inventory level but, from a certain point of inventory level, this increase is linear. This linear increase is equal to the inverse of the maximum throughput per piece of inventory. The throughput of the system does not increase linearly but more smoothly, and from a certain point it remains constant. This saturation point coincides with the point from where the lead time increases linearly.

The work point of a manufacturing system is determined by the choice of the mean inventory level. The control system has to try to minimize the difference between the ideal performance and the real performance. But the lead time and throughput performance measures depend for the biggest part on the chosen work point. So, for instance, it makes little

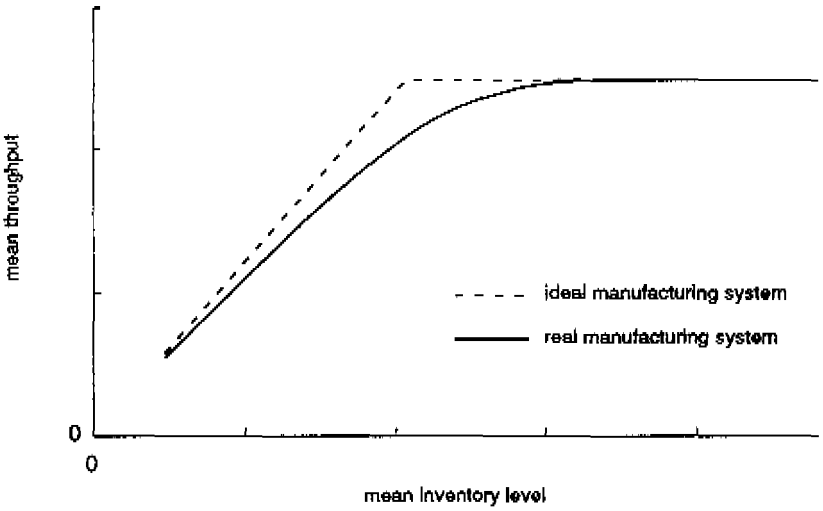


***Figure 3.2.** Mean throughput as a function of the mean inventory level.*

sense to increase the inventory level any further if the system has reached its maximum throughput. The point where the ideal factory reaches its maximum throughput at minimum lead time is called the ideal work point. The representations of the lead time and the throughput against the inventory level are called the performance graphs. It depends on the market situation whether the work point for a manufacturing system is chosen on the "left side" or on the "right side" of the performance graph. On the left side the lead time is relatively short at the cost of idle machines. On the right side the machines are intensively utilized at the cost of long lead times. The control system tries to keep the inventory level close to a work point. The quality of the controller is determined by the difference between the ideal lead time at the work point and the measured lead time at that work point. The best controller minimizes the difference between these values.

The control system of a manufacturing system tries to operate at the work point it was designed for. If, however, the master has too few jobs, this results in an inventory level that lies below the work point, which means that the mean lead time is also lower than that at the work point. If there are more jobs than are allowed by the work point, these jobs will have to wait at a higher level (in the master), thus introducing wait queues at that higher level. The lead time of jobs in the system remains equal to the lead time given in the performance graph. If the manufacturing system allows hot jobs (jobs having a higher priority), this results in a transfer of the mean lead time. The mean lead time at the work point will become longer. Thus, the throughput will become lower. The hot

jobs disturb the manufacture of the other jobs so much that the mean lead time increases [Conway et al. 1967].

Balancing

The process of determining the capacity of a factory or a manufacturing system in relation to an expected work load is called balancing. The balancing of a factory states goals for throughput levels and determines the necessary resources in a factory. The recipes and the throughput levels are used to calculate the static work load of the resources. These static loads are used to calculate the number of resources needed to achieve the throughput level. These calculations allow the ideal work point to be defined. By constructing the ideal performance graph, an impression of the real behaviour of the manufacturing system at the different inventory levels can be obtained.

3.2 Control functions

The control system is often divided into subsystems that are related to the functions that have to be performed. Some functions exist as separate subsystems while others are incorporated in more than one subsystem. The factory controller incorporates capacity planning, marketing, purchasing and manufacturing control. Design, accounting and quality control are not incorporated in this study. These functions increase the complexity of the model, without adding to our insight. If need be, these functions can be added in the future.

To coordinate the activity of the resources in the manufacturing system, the controller has to take decisions on what actions have to be performed by what resources. The tasks of the controller are split into the subtasks of planning, scheduling and monitoring.

The difficulties associated with planning and scheduling are the combinatorial nature of the problems, their size and complexity and the uncertainty in the data, all of which cause deviations between the generated plan or schedule and reality. Because of the size of the planning problem it is usual to aggregate information, in order not to have to consider every detail of the manufacturing process. The possible aggregations are many. Aggregation of time, aggregation of products, aggregation of resources, aggregation of operations, aggregation of consumers, aggregation of suppliers. Aggregation helps in certain ways to overcome uncertainties; the demand of a group of consumers behaves less unpredictably than the demand of one consumer. But aggregation

also introduces new uncertainties: the decomposition process cannot always be executed in a way that is consistent with the aggregated data.

Aggregation is closely related to the notion of hierarchy. In hierarchical planning more than one aggregation is applied. A popular hierarchy in planning is the division into strategic, tactical and operational levels. These levels have different time horizons, but often also consider different details for material or products, operations and resources [Hax and Candea 1984, Joensson 1983]. The hierarchical planning will not coincide with the control hierarchy presented here. The control architecture does not contain a hierarchical planning algorithm, if it is to be implemented in the future it will have to be implemented in one controller, probably the factory controller.

Planning

The term planning, as it is used in the literature, has a lot of different meanings. Here we use the definition given in Hax and Candea [1984]: the manufacturing process and the capacity of the resources are supposed to be fixed and planning is related to the optimal utilization of the resources under the constraints of fluctuating demand requirements. The term capacity planning will also be used. In Burbidge [1987] planning is defined as the function that provides the control system with the information to be used to manufacture products, it is related to the design of the manufacturing process and the provision and arrangement of production resources (layout). This is considered to be part of the specification phase of the system. In the controller a process planning function is implemented that generates process plans (which are called tasks) on the basis of the command the system has received and the capabilities (recipes) of the system. Kempf [1989] relates planning to the decisions concerning the use of capabilities of resources in order to manufacture products which are described by design engineers. Kempf's [1989] definition is used here for the term balancing.

In this thesis planning is divided into subfunctions: capacity planning, purchasing, process planning and process interpreting. Capacity planning ensures that the resources are utilized on a more or less constant level in time. This function is only found in the factory controller, it decides whether a new order is accepted or refused and it generates, if necessary, manufacturing jobs for which no orders have yet been received. The purchasing function takes care of the ordering of raw material from the suppliers. The purchasing function is also only found in the factory controller. The process planning function, with help of the recipes, defines the way material is manufactured. The process interpreting function is needed to determine the next operation a piece of

material has to undergo. This new operation is determined from the process plan and the information about the status of the material.

In this thesis the planning of the recipes is not considered. It is related to the product and the technology that is used to manufacture the product and lies outside the scope of the present work. The balancing problem is concerned with the attuning of the production capacity of a factory to the expected market demand. The balancing of a factory states goals for production levels and considers the necessary capacity of the factory. Because the configuration of the factory is supposed to be static, this function is performed before the control system starts to operate and it is not considered as a part of factory control.

The term capacity planning is used for the function of the factory controller that states norms for the manufacturing system, which are based on expected and real demand. Capacity planning tries to satisfy demand as well and as far as possible and to utilize the system as well as possible. Problems in satisfying consumer demand have two aspects. Demand is characterized by its random character, with peaks that are often bigger than the maximum production rate of the manufacturing system. On the other hand, the consumer demands delivery times that are often not realizable with production to order. The means by which the capacity planning can absorb demand fluctuation are the variation of production rate by adding production capacity or introducing idle times, or the spreading of production over time and introducing inventories. If these do not work, management has to negotiate with the consumer to introduce delays and backlogs, or has to refuse orders. The means by which delivery times are reduced are based on the same principles.

Scheduling

Scheduling is the assigning of a start time and a completion time to an operation, together with the specification of the material and the resource involved. This assignment depends on the operations the material has to undergo, on the operations the resource is able to execute, and on the capacity available for operations in the manufacturing system.

The scheduling problem is classified in the literature [Graves 1981] as being either dynamic or static and either deterministic or stochastic. In practice a manufacturing system is dynamic and stochastic. 'Dynamic' means that jobs arrive during the time considered, 'stochastic' means that some events in the system have a random nature. These events are the demand of jobs, the process time of a job, the breakdown of

machines, and errors during the execution of an operation which lead to rework, repair or rejection of material.

In a manufacturing system material is directed across the resources as efficiently as possible. The efficiency is usually realized by the optimization of a measure of performance (the optimization criterion) that is related to the resources or to the material in the manufacturing system. Scheduling analyses the future in order to take decisions that are to be executed in the future. The optimization attempts to maximize the performance of a manufacturing system. Unlike planning, scheduling is not concerned with negotiations with consumers, but rather with the work that has to be executed by the resources. Scheduling receives its jobs from planning or from a master controller.

Scheduling is about taking decisions concerning the sequence in which operations are executed when, and on what resources. The subfunctions of scheduling are releasing, allocating, sequencing and dispatching. Releasing has to do with the moment of release of new material in a manufacturing system. In order to do this the controller has to take decisions and to communicate. The allocating function decides on which resource material is processed. The sequencing function decides on the sequence in which material is processed on a resource. Both allocating and sequencing are related to taking decisions. The dispatching function takes care of the sending of a command to a resource. Dispatching has to do with the communication, no decisions are involved.

The releasing of a command or material in the manufacturing system is very important. The material that is released has to be processed on the necessary resources. If too much material is released into the manufacturing system, the system becomes overloaded. An overload results in extremely long lead times of the material, and in high inventory levels.

In the literature the sequencing of commands to resources has been more extensively researched than the releasing of jobs to a system. But recent findings show that a releasing strategy is as important as, if not more important than the sequencing strategy [Lou and Kager 1989, Lozinski and Glassey 1988, Glassey and Resende 1988, Wein 1988, Wiendahl 1987].

When looking at scheduling (especially the subfunctions releasing, allocating and sequencing) one finds that there are two extreme approaches: predictive scheduling and reactive scheduling [Kempf 1989]. But both methods have their disadvantages: predictive - the future is never the way you planned it; and reactive - if you had known things in advance you always could have done better. A purely predictive schedule

is difficult to calculate and has to be recalculated every time something goes wrong. With big, complex systems in particular, this results in a lot of useless calculations, which is why a reactive approach is preferred. This means that a decision is taken at the moment the actual choice occurs. If system performance can be increased by including prediction this may be done by signalling future choices. But this signalling, or claiming, has to be kept to a minimum.

Predictive scheduling is characterized by the fact that the scheduler calculates a schedule in advance. This schedule states the time at which a resource has to perform operations on material. The calculation of an optimal schedule is in practice not possible, it takes too much time. Even the calculation of semi-optimal schedules, with reasonable scheduling problems, requires long computation times. Large problems often remain impossible to solve. Once a schedule has been calculated it is used to start all action on time. But small disruptions make the rest of the schedule invalid, precautions have to be taken so that the deviation between the reality and the schedule is kept to a minimum, or a new schedule is calculated with the disruption included. A schedule is, for instance, only valid for the static case if no new jobs arrive during the processing of the jobs in this schedule.

A different approach to calculating a schedule in advance is to take decisions when a choice problem appears: reactive scheduling. This approach is used in this study. When considering a reactive algorithm two kinds of decisions have to be taken. Material becomes available and has to be allocated to a resource and there is more than one resource idle and capable of processing the material: the allocating problem. Or a resource comes available and there is more than one piece of material waiting to be processed. Some material is chosen: the sequencing problem. In this case sequencing rules are used to differentiate between the different possibilities. The use of sequencing rules results in non-delay schedules. A lot of sequencing rules have been described in the literature [Panwalker and Iskander 1977, Montazeri 1987] but no simple rule has been found that functions well in all situations. Every case has to be considered separately to find good sequencing rules. The effectiveness of a sequencing rule depends, among other things, on the optimization criterion that is used, on the resource configuration and on the manufacturing process.

The dispatching function is concerned with communicating the decisions of the controller to the resources. Because of this, dispatching strategy is closely related to the allocating and sequencing strategy of the controller. The dispatching algorithm of a controller has to be consistent with the releasing algorithm of the resources in the manu-

facturing system. The releasing algorithm of a controller has to be consistent with the dispatching algorithm of the master controller.

In order to solve the scheduling problem the controller may want to know what the performance of the resources is. Estimations of process times, down times and yield are needed. These data are recorded during the functioning of the manufacturing system. If the recorded data are used for scheduling purposes, great care has to be exercised because this recording results in a feedback loop which may lead to instabilities.

Release strategies

A common release strategy is the use of uniform starts, where new work into the manufacturing system is released at a constant rate equal to the desired throughput. The uniform starts rule is an open loop strategy, the release rate is independent of the status of the manufacturing system (e.g. the inventory level). Another release strategy is the fixed-work-in-process rule (Fixed-WIP), new work is released at the moment the manufacturing of a product is finished. The work load regulating input policy [Wein 1988] releases new work in the manufacturing system if the total amount of remaining work for the bottleneck resource falls below a prescribed level. The release rate is derived from the throughput of the bottleneck. The starvation avoidance (SA) rule [Glassey and Resende 1988, Lozinski and Glassey 1988] is similar to the work load regulating input policy, but it uses a virtual work load which is the work content expected to arrive at the bottleneck within a given time. New work is released if the virtual work load falls below a given level. Work load oriented job release [Wiendahl 1987] uses a fixed plan period for which planned values of throughput, lead time and inventory level are determined. Work is released in such a way that the start inventory plus the released work (both expressed in hours) is equal to the sum of planned mean inventory and the planned finished work. Per resource an estimate of the work load is made and if this work load exceeds a predetermined limit, the release of work in the planning period is stopped.

Priority rules

Some priority rules and their classification are mentioned below. All these rules are intended for the sequencing problem: the allocating problem is usually solved on a first-come-first-served basis or randomly. Of course it is possible to use other priority rules to solve the allocating problem.

Priority rules related to processing time use the process time of a command or the process time of the supercommand (the command from the master) or the order to which the command belongs, to discriminate between commands. The SPT rule (shortest-process-time) gives the highest priority to the command with the shortest process time. The SRPT rule (shortest-remaining-process-time) gives the highest priority to the command that belongs to the supercommand that needs the least process time to finish.

Priority rules related to due dates base their choice on the due date of the command or of the supercommand or the order to which the command belongs. In the first case a due date for every command is derived from the due date of the supercommand or order. Examples of priority rules are the EDD rule (earliest-due-date) which gives the highest priority to the command which belongs to the supercommand or order with the earliest due date. The OPNDD rule (earliest-operational-due-date) uses the due date of the command.

Priority rules related to arrival times and random rules use the sequence of arrival of commands to discriminate between them. A very well known rule is the FIFO rule (first-in-first-out) or FCFS rule (first-come-first-served). The command that arrives first has the highest priority. LIFO (last-in-first-out) or LCFS (last-come-first-served) gives the highest priority to the command that arrived last. Instead of the arrival time of the command the arrival time of the order or supercommand can also be used. The RANDOM rule chooses a command from the queue at random.

Other priority rules are related to the number of operations or are related to costs, for instance by using the economic value of commands to discriminate between them. Priority rules related to slack use the difference between the time needed to execute a command and the time available to execute the command to assign a priority to a command. Priority rules related to resources use the work in the queue of the resource which is to be visited after this resource by the command, in order to discriminate between commands.

Simulation

Besides priority rules, simulation may also be used to decide which material has to be processed first. In this case the different possible choices are simulated in a model. The simulation results in a performance report for a number of possible choices. The choice that leads to the best performance report is chosen. For the simulation an exact status of the system has to be fed to the simulator in order to obtain a reliable result [Doulgeri 1987, Doulgeri et al. 1987, Steyns 1991]. Simulation is used

to determine differences between the transforming of one piece of material and the transforming of other pieces of material. The simulation uses priority rules and the material used in the simulation with the best result is also used for the real command.

Monitoring

Monitoring is concerned with the processing of information from resources. When we speak about the job progress recording subfunction we mean the recording of the progress of a job that informs the process interpreting function about the new status of the material and that signals the finishing of a job. The resource activity recording function keeps track of the momentary status of resources in order to be able to allocate new jobs to resources. The performance measuring function collects statistical excerpts of the resources to be able to keep track of the performance of the system.

The events in the manufacturing system are recorded. To be able to direct the material through the manufacturing system information on the material is kept (job progress recording). This information contains the operations that are performed (past), the operation that is being performed, together with the resource involved (present), and the operations that have to be performed (future). Further information needed for scheduling is the due date and the arrival date of the material and information related to processing time: the process time (left to be executed), the wait time and the lead time of the material.

To be able to command the resources information is kept about the material processed by the resources (past), the material in progress in the resource (present) and the material waiting for the resource (future) (resource activity recording). Because the material processed by a resource increases in time, it is possible to keep only statistical excerpts about the past or to record only the material that was processed the last 24 hours. It also has to be known of a resource whether there is still free capacity and, perhaps, when this capacity became available. To evaluate the performance of a manufacturing system information per resource is collected and information about the way jobs from the master are executed is collected.

When considering the manufacturing system performance (performance measuring) the mean lead time, the mean input, the mean throughput, the mean inventory level, the due date reliability and the yield are most interesting. Because not all jobs have the same work contents, the parameters lead time, input, throughput and inventory level are also kept in a weighted form, with their process time (= work contents) as their weight. The due date reliability plays an important role at the factory

level; the other measures are important at all control levels. Every resource has to record these values. For leaf resources the idle time, the busy time and down time are also of interest. Looking at the jobs from the master, the process time, the wait time and the lead time per type of job are of interest. The type of a job is determined by the operation.

Part of the performance measuring is a logging function for material and for resources. These lists are used to replay the events in the manufacturing system. This information is especially interesting for an analysis of the behaviour of the manufacturing system in relation to errors that occurred.

3.3 Control configuration

In this section we discuss the factors that influence the control configuration, dealing in turn with the controller hierarchy, the possible classification of controllers by distinguishing the stimulator (consumer or supercontroller) and the controlled resources (expanded or leaf resources). This results in four controller categories, which are applied one or more times to obtain a hierarchical controller. The distribution of the control tasks in relation to the availability of information is discussed, just like the kind of decisions that have to be taken and when these decisions have to be taken.

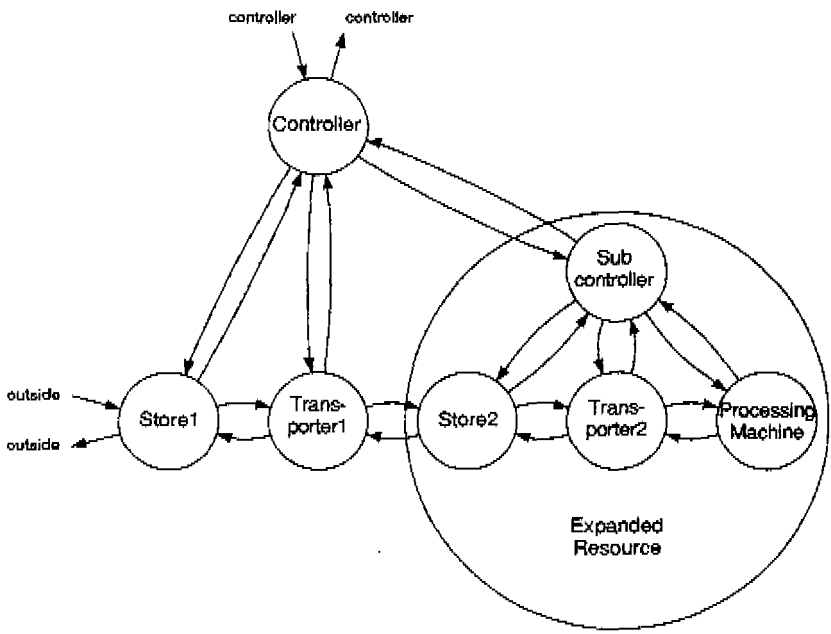


Figure 3.3. *The system and model hierarchy of a manufacturing system with one expanded resource.*

The relation between recipe structure, system hierarchy and the influence of the controller is discussed with reference to an example, followed by a discussion of the controller decisions. These decisions are related to material exchange, transport and the processing of material. The generation of these operations and the time at which the commands are sent will be described.

Here, a manufacturing system is considered to be a system with one central controller, one transporter, one store and one or more processing or expanded resources (Figure 3.3). A manufacturing system itself forms an expanded resource. The control system of a manufacturing system has a tree structure; the nodes are formed by the controllers, the branches by the connection between the controllers, the leaves are formed by the leaf resources (= machines). Two controllers which are connected by a branch have a super/sub relation. The highest of the two is the supercontroller (master), the other the subcontroller. The topmost node is the central controller (the factory controller), this controller has no supercontroller. Controllers that are equally distant from the central controller (which means the number of branches between the central controller and the controllers is equal) form a so-called control layer. The vertical direction is related to the super/sub connection between controllers. The horizontal direction has to do with controllers of the same layer.

The processors of a manufacturing system exchange two kind of objects: information and material. In the machines the material and the information meet. Controllers process only information. The material is transported by transport systems, the stores of resources constitute interfaces between the different transport systems. Material is exchanged between resources and there are always machines involved.

A disadvantage of an extra control layer is the introduction of extra transport systems and stores. This complicates the material route and it increases the material handling time and the material waiting time. It is possible to share transporters and stores between expanded resources; this, however, complicates the control problem substantially, and from that point of view it has to be avoided as far as possible. In this study a controller will always be associated with its own private transporter and store.

The character of the control system is determined by two factors. First, the controller is either independent and receives its stimuli from the consumer, or else the controller is a slave and receives its stimuli from a supercontroller. Second, the controller controls expanded resources which are decoupled by their internal stores, or the controller controls leaf (processing) resources which are coupled. Leaf (processing) resources or machines are coupled because these do not contain internal

Table 3.1. Controller categories.

resources	stimulator	
	consumer	supercontroller
decoupled processing resources	factory controller controlling expanded resources	manufacturing controller controlling expanded resources
coupled processing resources	factory controller controlling leaf resources	manufacturing controller controlling leaf resources

stores (see Section 2.6), which means that a leaf resource cannot start receiving and processing new material until the processed material has been removed. This leads to four controller categories, as shown in Table 3.1.

The (independent) factory controller is concerned with the interaction of the factory with consumers and suppliers. The interest is directed at the allocation of work to capacities; here the goal is to meet consumer demands as well as possible. Factors like lead time, throughput and due date play a role. The factory controller accepts or rejects orders, with the help of the configuration information, the status information of the resources and the already accepted orders. The capacity planning plays a key role in the acceptance of orders. The (slave) manufacturing controller has to execute the work it is offered; the goal is to minimize the lead time at a given throughput. The controller has no possibility to refuse the execution of work, it has to allocate material to resources and to sequence material on resources. The process planning specifies with help of the recipes how the material is to be manufactured, and is found in both types of controllers, just like the other functions, scheduling and monitoring.

The controller of leaf processing resources has to avoid a system deadlock. A machine has no internal store, it is necessary to remove the material at the moment a machine has finished processing and the destination machine has room for the material and has not started processing. The problem of deadlock is not considered for the controller of expanded resources: it is supposed that the stores are big enough to store the material that is waiting to be processed. The actions that have to be directed have a much more parallel character, many actions are independent of each other, in some circumstances the actions disturb each other.

In the hierarchical control system, the responsibility for material is transferred from one controller to another controller or to machines. The control strategy specifies when the responsibility is delegated. By using more control layers it is possible to distribute the control of a manufacturing system. The control task of the highest controller is more general. A controller in a high layer processes aggregated data. The decisions that have to be taken are spread more widely in time and concern less details. In the sublayer only a subset of the control problems is concerned, but in greater detail and also in smaller time ranges. As a consequence there are more but simpler controllers and the control tasks are distributed over several controllers. A disadvantage of more hierarchical layers is that a controller not only has to take more general decisions, these decisions are also based on more general information. This can only be avoided with a heavy increase of the communication volume between controllers.

Sometimes detailed data that allow the best decision to be taken are not available to the responsible controller. This information is either present inside one of the resources, or else outside the system. In the first case extra communication is necessary to make the information available to this controller. In the second case two situations are distinguished. First, the information is available at a higher level, which means that the decision has to be taken at that higher level. Second the information is available in the same layer or in a lower layer, in which case the information has to be transferred to a higher common controller. This controller also has to take the decision.

Information is exchanged in the vertical direction, status information from bottom to top and commands from top to bottom. Decisions that involve global information have to be taken at the top. If information from inside the resource only is needed, the decision can be taken in the resource controller. A controller may know what happens in the resources or in the expansion of resources. Information from resources that are not downward connected is not present in a controller.

Resources do not necessarily process material from their store in a first-come-first-served order. This is one of the reasons why a resource has stores. The sequence in which material is processed is a responsibility of the resource controller. This has the disadvantage that the sequence depends only on the status inside the resource. If status information from outside the resource is necessary to determine the sequence then this decision has to be taken at the level where this status information is available. On the other hand the suboptimization by a controller may disturb the optimization that the supercontroller intended to reach.

The optimization of the performance of a manufacturing system depends to a degree on the scheduling. In order to realize a global

optimization, scheduling decisions have to be taken at the top level, taking into account all detailed status information at the lower levels, or else the resources have to execute operations in a determined time. The first situation leads to controllers between the top and the machines that merely pass on information. In the second case the resources have to be inflexible with no scheduling possibilities for the controllers below the top controller. In the ideal case (to reach global optimization) scheduling problems are solved as far as possible in the top layer of the controller structure, the number of control layers is kept as small as possible, and the lower layers are increasingly inflexible with decreasing uncertainty in their behaviour.

The hierarchical structure found in the control system is also found in the description of the manufacturing process: the recipes. A recipe is started or invoked by an operation and the manufacturing system is the resource that executes this operation. Usually the recipe contains a number of sub (processing) operations that have to be executed on the

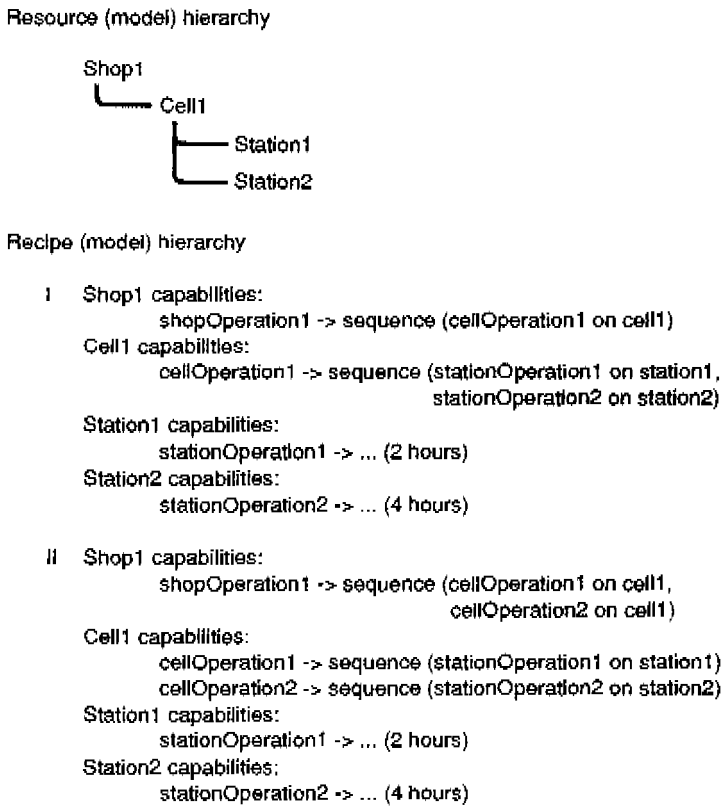


Figure 3.4. Example of relation between system configuration hierarchy and recipe hierarchy.

resources in the manufacturing system. The recipe hierarchy and the control system hierarchy, however, do not necessarily correspond in a one-to-one relation. There may be fewer resources than operations or fewer operations than resources.

The recipe configuration influences the scheduling possibilities of a controller. The next example illustrates this problem. A shop contains one cell and the cell contains two stations. The factory control level is omitted in this case because it does not add any value to this control aspect. The manufacturing process consists of an operation on station1 followed by an operation on station2 (Figure 3.4).

In situation I the shop controller starts the manufacturing process with the sending of operation cellOperaton1 to the cell. The cell controller sends first stationOperation1 to station1 and afterwards it sends stationOperation2 to station2. In situation II the shop controller first sends cellOperation1 to the cell, the cell controller sends in reaction stationOperation1 to station1. Afterwards the shop controller sends cellOperation2 to the cell and the cell controller then sends stationOperation2 to station2.

It is seen that the influence of the shop controller on the progress of the manufacturing process differs for both recipe configurations. In situation I the cell controller decides the start of operations on the stations. The shop controller is able to influence the start of stationOperation1, by delaying the dispatching of cellOperation1. If, for the execution of shopOperation1, it is important that the shop controller is able to influence the start of operation stationOperation2, the recipe configuration of situation II has to be used. The shop controller needs the status of the two stations and the commands from the shop have to be sent straight away to the stations. Both pass via the cell controller. This situation II degrades the cell controller to an information passer, the transport between the two station operations now becomes a responsibility of both the shop and the cell controller. The shop controller has two consecutive operations without transport in between. This is an unusual situation, most of the times between two operations the material has to be transported from one resource to another. Material flows from cell store to station1, to the cell store, to station2 and back to the cell store.

Types of operations

To control the manufacturing process three kinds of operations are used: processing operations, transport operations and material exchange operations (see Section 2.6). These operations are defined before the start of the manufacturing process or on line during the execution of the manufacturing process. All operations the material has to undergo may

be fixed beforehand. A second possibility is to fix only the processing and the transport operations and generate the material exchange operations on line. The third possibility is to generate both the transport and the material exchange operations on line. This means that only the processing operations are fixed. The most complex situation arises if the processing operations are also variable and depend on the way the manufacturing process is executed and the manufacturing system status. Verifying introduces the possibility of changing the course of the manufacturing process.

The processing operations are deduced from the manufacturing technology that is chosen, and from the resources that are applied in the manufacturing system. The freedom of choice for these operations is restricted. The support operations (transport and material exchange), too, depend on the operational status of the manufacturing system. Because of this it is preferred to leave the support operations out of the manufacturing process description (the recipe) and let the controller generate the appropriate support operations on line. It also means that the recipes are usable for all manufacturing systems that are based on the same manufacturing technology.

When commands are issued

The controller directs the actions in the manufacturing system; there are several strategies possible for the performance of this task. In this section we discuss the moment at which a command is issued. The commands are divided into the categories: material exchange, transport and processing.

Material exchange commands

The material exchange is considered first. With material exchange a transporter is always involved. The moment to issue a material exchange command is evident. This has to happen when both sender and receiver of material are physically connected and ready to pass material. The send or receive material command is given at the moment the transporter has arrived. An exchange command may only be given to a resource if the material is present in the resource. If material exchange commands are given in an earlier stage resources become blocked, which disables further material exchange with the resources involved.

The send and receive material commands may be send by the controller to the transporter and another resource. The other resource is either a store, an expanded resource or a processing machine. If the resource is

expanded the subcontroller has to pass the exchange command to the store involved.

Material exchange commands are needed because material does not necessarily leave stores and processing machines in the sequence it entered. The simplest way to control material exchange is by letting the transporter send information about the material it wants to receive from a resource (send material command). The receive material command is superfluous, it is incorporated in the sending of material, under the condition that the material receiver (a store or a processing machine) is ready and able to receive and identify the material. This solution makes storage independent of the manufacturing controller. The manufacturing controller only invokes the transport command and, after the execution of the command, it receives a transport report, which confirms that the material is stored in the destination resource. As a consequence the manufacturing controller does not know what material has arrived in and has left its store. In our model the transporter will send "send material" commands (or material requests) to stores and processing machines and "receive material" commands will not be issued.

Transport commands

A transport command is issued after the material has become available for transport. There are two extreme moments in time to send these transport commands. Start transportation at the earliest possible moment, when a processing resource has completed its operation or when material arrives at a manufacturing system (early transport). Or start transportation at the latest possible moment when the destination processing resource is ready to start a new operation (late transport). The transport command is sent by the controller to the transporter.

With early transport the controller has to know what the destination of the material is at the moment a processing operation is finished. In case of multiple destinations, a choice between destinations may have to be made before the resource is available. This disadvantage can be overcome by avoiding the possibility of multiple destinations; by configuring resources in such a way that resources which perform the same operation have a common input store, for instance. An advantage of early transport is the good overview that results. All material is waiting in the store of the resource that is going to process the material.

With late transport material always arrives too late, the transport starts when the processing resource becomes idle. The resource remains idle during the transport of the material. The overview of the manufacturing system decreases. The material that has to be processed on a resource is waiting in expanded or processing resources that just have processed the

Table 3.2. *The activities as a function of material position and responsible control level for early transport.*

position of material	responsible control level	
	manufacturing controller	resource
store	schedule transport capacity	
transporter	transport material	
resource1	schedule resource1 capacity	
"		process material
"	schedule transport capacity	
transporter	transport material	
resource2	schedule resource2 capacity	
"		process material
	etc.	

material or it is in the store of the manufacturing system. Multiple destinations, however, do not cause any problems.

Tables 3.2 and 3.3 show at what material position control decisions are taken. With early transport the material arrives in the store, here the controller schedules the material to the transport capacity. The material is transported to the resource. After the material has arrived in the resource, the controller schedules the material to the resource capacity. The process command is sent to the resource and the material is processed in the resource. After the material has been processed the material is scheduled to the transporter again and so on. From Table 3.2 it is clear that it is impossible to use early transport for the control of (coupled) leaf resources. It is impossible to put material into a busy leaf resource and schedule it to the leaf resource afterwards. The capacity of a leaf resource has to be available before the material is transported to it. With late transport the material is scheduled to resource capacity and afterwards the material is scheduled to transport capacity. Then the material is transported, the process command is sent and the material is processed. This cycle repeats until the manufacturing process is finished. Only after the last processing operation the material is scheduled right away for transport to the store. Other strategies for sending transport commands are possible: use late transport for the operations

Table 3.3. *The activities as a function of material position and responsible control level for late transport.*

position of material	responsible control level	
	manufacturing controller	resource
store	schedule resource1 capacity	
"	schedule transport capacity	
transporter	transport material	
resource1		process material
"	schedule resource2 capcity	
"	schedule transport capacity	
transporter	transport material	
resource2		process material
	etc.	

that can take place on multiple resources and early transport for operations that take place on unique resources, or start transportation a short time before the destination resource becomes idle. In our model (factory and manufacturing) controllers of leaf resources apply a late transport strategy. For controllers of expanded resources either an early or a late transport strategy may be chosen. The late transport strategy is implemented in such a way that it is possible to start transportation before capacity is available in the destination resource.

Processing commands

A resource cannot process material for which it did not receive a command or execute a command for which the material has not arrived. The issuing of processing commands leads to two extreme strategies: material driven and command driven manufacturing. The earliest moment to issue processing commands is when the manufacturing process is started. This way of issuing commands leads to a material driven manufacturing system. The other possibility is to issue processing commands in order to start the processing of the material. This is a command driven manufacturing system.

In a material driven manufacturing system the route of the material is fixed at the start of the manufacturing process. The fixing of the material route has a big disadvantage, it reduces the on line route flexibility. In case of machine failure, for instance, it is impossible to change the route to a resource that has not broken down. The arrival of material at a resource triggers the execution of the manufacturing process. The bottom controller (of the leaf resources) has to collect all processing operations and is able to schedule these, thus realizing local optimization. The scheduling of transport operations is the only way to enable some global optimization. One way to implement a material driven strategy is by connecting the recipe to the material, and to use this for the control of the resources.

A command driven strategy sends a command at the moment that an operation has to be started on a resource. This means that the controller decides at the latest possible moment which resource is to be used to process what material. Global optimization is possible and material routes are determined during the manufacturing process. The controller needs the status of the resources to start operations, so the resources have to send their status information to the controller. The command driven manufacturing system has to take care of transport before the actual process command is issued. Our model is based on a command driven control strategy.

The configuration of the controller is influenced by the way in which it is stimulated and by the type of the resources controlled. Two categories of control configuration may be distinguished on the basis of the stimulator: the factory controller and the manufacturing controller. The factory controller implements the following functions: capacity planning, marketing, purchasing, process planning, process interpretation, allocating, sequencing, dispatching order progress registration, resource activity registration and performance measurement. The manufacturing controller implements the functions of process planning, process interpretation, releasing, allocating, sequencing, dispatching, job progress registration, resource activity registration and performance measurement. Other factors influencing the controller configuration are the strategies chosen for commanding the material exchange, the transport and the processing. The total control system configuration is determined by manufacturing process constraints, resource constraints and control constraints. Here factors such as the number of control layers required, the need for information, the availability of information, the need for global optimization and the way the recipes are built up play a role. In the next section the communication between controllers is considered.

3.4 Communication protocol

The controller of a manufacturing system communicates with its master (supercontroller) and its resources. The purpose of communication is the exchange of information and the synchronization of actions. The communication provides information from and to the supercontroller to and from the controller, to enable the controller to receive commands for executing a manufacturing process (releasing), and it provides information to resources, to start the execution of operations in the resources (dispatching). The communication takes place with the exchange of objects, which are either commands from the controller or statuses and results from the resources. The commands for resources start the processing of material or request information. The results are responses to commands, while statuses are not necessarily related to a command. The status or result contains information about the status of material and/or information about the status of resources: information about the available resource capacity is of particular interest to the controller. The resource status information is used to decide what new work has to be done by a resource (allocating and sequencing). Material status information is used to discover what new operation has to be performed on material (process interpretation). Status information is also used for the monitoring function.

The way the communication takes place is defined by a communication protocol. The communication protocol defines the contents of the messages and at what point in time the messages are sent. The communication protocol has to be able to control all kinds of manufacturing classes (job shop, flow shop, parallel shop and single shop). The contents of the messages have to be limited and the number of messages sent has to be limited.

Usually information is exchanged between a controller and its supercontroller and between a controller and its resources. The resource is either a machine or an expanded resource; in the latter case the information is sent to a subcontroller. The exchange of information in this fashion is in the vertical direction. Sometimes the volume of communication can be restricted by letting resources communicate with each other, rather than using a common controller. Such interactions carry with them the risk that interaction paths between resources have to be introduced which depend on the specific manufacturing process. For this reason resources are not connected to each other. The only exception is the transporter. The transporter is physically connected to other resources in order to exchange material; to control the material exchange an information exchange from the transporter to the involved processing machine or store is also allowed.

A controller of a manufacturing system, which knows the exact status of all the resources, is able to control the whole manufacturing system in the best way. But it has to deal with every signal and control all activity, which means a lot of communication and calculation. The resources send excerpts of their status to the controller, in order to reduce the communication volume. The controller is able to use these excerpts to help allocate and sequence material. The status of a resource is represented at two places, the status representation in the manufacturing controller has to be correct and consistent with the status of the resource. This places certain demands on the communication protocol. The sophistication of the communication protocol has to be weighed against the communication volume. Complex communication protocols allow the controller to acquire more information and to control on a more detailed level. Simpler communication protocols go hand in hand with smaller communication volumes and the delegation of detailed control decisions.

Four communication protocols are considered for the execution of work on a resource. These protocols use three different sets of messages:

1 command, result (protocol 1)
2 status, command, result (protocol 2)
3 statusRequest, status, command, result (protocol 3 and 4)

The messages are sent in the sequence that they are mentioned. The manufacturing controller uses the protocol for the release of work from the supercontroller. The supercontroller also has to use the same protocol. StatusRequests and commands are sent to the controller; status and result are sent from the controller to the supercontroller. The manufacturing controller also uses the protocol to dispatch work to resources; the controllers of the resources use a matching protocol.

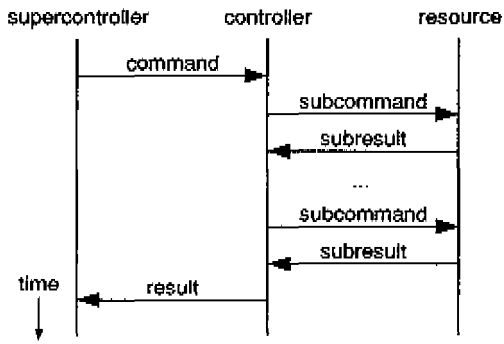


Figure 3.5. *Messages between control levels as function of time for protocol 1.*

Subcommands and substatusRequests are sent to the resources; substatuses and subresults are sent from the resources to the controller.

In the first protocol (see Figure 3.5), the manufacturing process is invoked by the command. The subcommand dispatches work to the resources. The subresults are used to inform the controller about the capacity of the resource and the progress of the manufacturing process. The finishing of the manufacturing process is confirmed by a result. The result also signals the capacity of the manufacturing system. A command results in a subcommand. The subresult is used to generate a new subcommand or a result.

In protocol two (Figure 3.6) the status is used to signal the capacity of the manufacturing system to the supercontroller. The status or the capacity available is calculated with help of the substatuses and/or the subresults of the resources, which give indications about the capacity of the resources. The work for the manufacturing system is sent with the command. The dispatching of work to resources is done with help of the substatuses. The work is sent with the subcommands. The subresults indicate the progress of the manufacturing process. The completion of the manufacturing process is announced with a result. In protocol two substatuses are used for two purposes, to generate a status and to generate a subcommand.

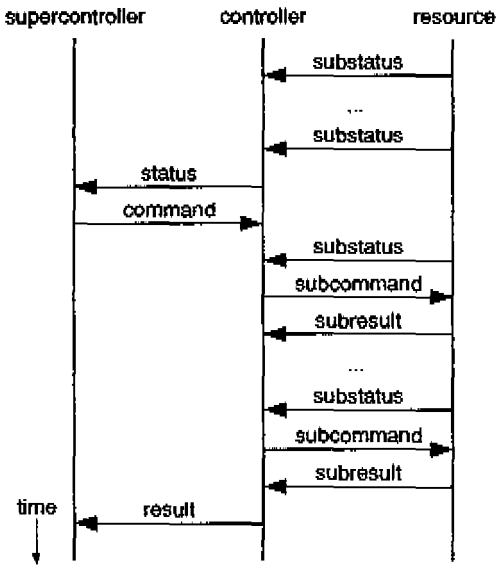


Figure 3.6. *Messages between control levels as function of time for protocol 2.*

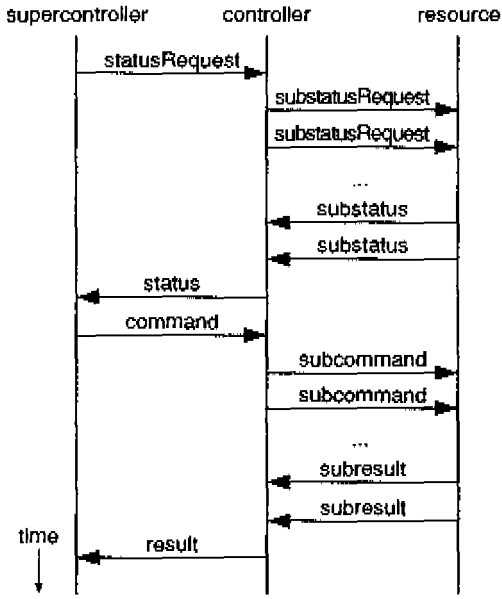


Figure 3.7. Messages between control levels as function of time for protocol 3.

The set of messages statusRequest, status, command and result leads to two different protocols. The first (protocol 3 is printed in Figure 3.7) uses the messages statusRequest and status for the determination of the capacity planning function of the factory controller. The second (protocol 4 is printed in Figure 3.8) uses the messages statusRequest and status for the on-line allocation and sequencing of work on resources.

The statusRequest is used to test whether it is possible to execute work. The statusRequest generates substatusRequests; the substatuses answer these subrequests, they give information about the capacity of the resources and are used for the generation of the status which declares the capacity available in the manufacturing system. The work the manufacturing system receives depends on the status sent and comes in the form of a command. The work is distributed in the form of subcommands. This has to be done upon the arrival of the command to inform the resources about the work they have to execute. The resources need to know their work load in order to be able to answer future substatusRequests concerning their capacity. The progress of the manufacturing process is regulated by the material transport and recorded by the subresults. The completion of the manufacturing process is reported with a result. This manner of control leads to a material driven manufacturing system, where the load of the manufacturing system is

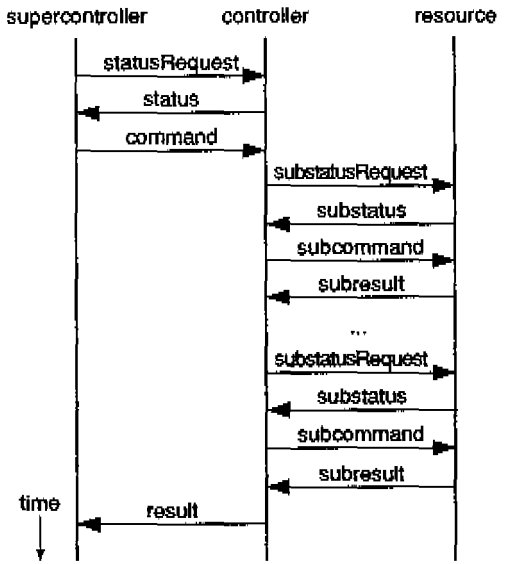


Figure 3.8. *Messages between control levels as function of time for protocol 4.*

controlled with help of a question/answer protocol preceding the start of the manufacturing process.

In protocol 4 the statusRequest and status are used for the on-line releasing of work in the manufacturing system. The statusRequest is answered by the controller with help of the information available, without consulting the resources. The status indicates the capacity of the manufacturing system and influences the work released in the manufacturing system. The manufacturing process is started with a command. To allocate and sequence the work the controller consults the resources with help of substatusRequests. The answers of the resources, the substatuses, are used to allocate and sequence the work and to dispatch it to the resources in the form of subcommands. The progress of the manufacturing process is recorded by the subresults and the completion of the manufacturing process is signalled with a result.

The choice of a protocol

Protocol 1 allows no separate communication about material status and resource capacity. In expanded resources this is a particularly severe drawback, which is why protocol 1 has been rejected. Protocol 3 leads to a material driven manufacturing system and the decisions about resource capacity are always based on the information available in the resource, which leads to local optimization. Unforeseen problems may

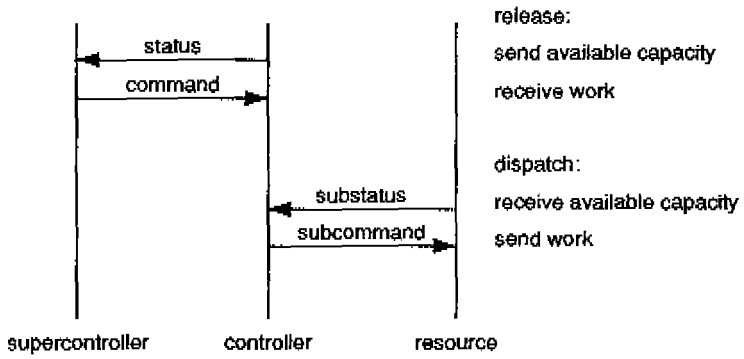


Figure 3.9. The releasing and the dispatching of work in protocol 2.

arise during the execution of the manufacturing process. On the basis of these disadvantages protocol 3 has to be rejected.

Protocols 2 and 4 are more closely comparable. A possible implementation of protocol 4 sends all available work as potential work to the resource (= a statusRequest). The resource filters out the infeasible work and accepts a part of the potential work (= a status). The controller uses the accepted work to decide what work the resource has to execute (= a command). In protocol 2 the resource requests for work by sending all possible work it can execute to the controller (= a status), the controller matches this with the available work and decides what work the resource has to do (a command).

For both protocols both the releasing and the dispatching of work is schematically reproduced in Figure 3.9 and Figure 3.10.

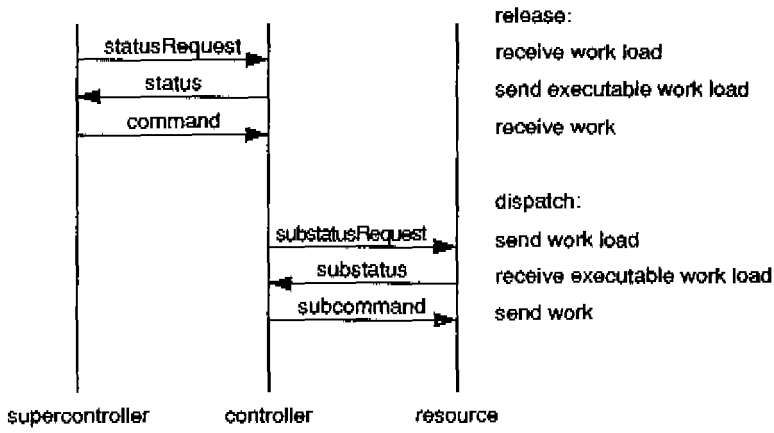


Figure 3.10. The releasing and the dispatching of work in protocol 4.

Table 3.4. *Examples of expression medium for protocol 4.*

expression medium	jobs
statusRequest	10 potential jobs
status	five accepted jobs
command	one job

expression medium	operation
statusRequest	a job with any operations
status	a job with a set of possible operations
command	a job with one operation

expression medium	amount of material
statusRequest	a job with any amount of material
status	a job with a minimum of one and a maximum of 10 pieces
command	a job with four pieces

expression medium	due date
statusRequest	a job to be executed as fast as possible
status	a job which can be finished within one or two months
command	a job that has to be finished within six weeks

expression medium	resource
statusRequest	a job to every resource
status	jobs from the resources that can execute the job
command	a job for one resource

In protocol 4 potential work is sent to the resource. Here the work is filtered and accepted work is sent back to the controller. This work is filtered again and real work is sent to the resource. This work can be expressed in many forms. Table 3.4 shows different means of expression. A job contains an operation, material, a due date and is intended for a resource. With protocol 2 it is possible to use the same manner of expression without the use of a statusRequest, under the condition that the freedom with which the status can be expressed can be restricted in meaningful way (as is the case for at least operation and resource as expression medium).

Protocol 4 has the disadvantage that all available work has to be communicated to the resources, which leads to a lot of communication. In protocol 2 the resources have to express all possible work or available capacity to the controller; with flexible and universal resources this may be a difficult task. Protocol 2, however, confers a double function on the status messages: they can be used both for the signalling of capacity to a higher controller and for the distribution of work to the resources. This means that the use of protocol 2 supports releasing of work with help of information send by the resources. A relatively uncomplicated version of protocol 4 will support releasing only on the basis of information available information in the controller. Because of this, protocol 2 will serve as the basis for communication between controllers and between controller and resource in the control architecture.

3.5 Problems related to parallelism

The physical manufacturing system consists of parallel machines. The control system consists of parallel controllers: there is a single controller only if the control system consists of one layer. The parallel controllers are implemented on parallel computers and/or as parallel processes on one computer. The controllers and machines are both called processors. Problems related to interacting parallel processors discussed are: deadlock, starvation, combining messages, data consistency and the modelling of the controller.

Problems of deadlock or starvation may arise between synchronized communicating processors. In case of deadlock or starvation one or more processors become blocked indefinitely. These problems have been addressed in the literature on operating systems [Maekawa et al. 1987, Peterson and Silberschatz 1986]. A deadlock situation occurs when two or more processors are waiting for an event that can only be caused by one of the waiting processors. The starvation problem occurs if a processor is indefinitely waiting for an event to happen. In contrast to deadlock, this event happens regularly, but it is always allocated to

another processor. There are two levels of interaction between the processors: the information level and the material level. The deadlock and starvation problem occur on both levels. The cause of starvation has to do with the scheduling policy. Here no further attention is paid to this aspect.

Three policies are used to handle deadlock: prevention, avoidance and detection [Meakawa et al. 1987]. With prevention the system design excludes deadlock. With avoidance the behaviour of the processors is restricted in such a way that deadlock does not occur; to accomplish this one needs to have a knowledge of the future processor behaviour. In the third case a detection algorithm is used to identify a group of deadlocked processors. The system is recovered by breaking the deadlock.

Deadlock on the information level is prevented by blocking the controllers only in a receive action; any processor can send an object to a controller, which the controller is always able to handle. The controller is willing to receive any object. The handling of the received object is never blocked. This is done by using the asynchronous send and by disallowing receive actions in the handling of received objects.

The manufacturing system becomes deadlocked on the material level when two or more resources want to exchange material and none of the resources involved can store new material before the old material is removed. On the material level deadlock cannot be prevented, since this would constrain the recipes of a manufacturing system. So deadlock on the material level has to be avoided, or detected and recovered from. Deadlock is avoided by only releasing new jobs that will not cause any circular wait. The use of an avoidance policy may lead to a less efficient usage of resources. If one ensures that the stores in the manufacturing system are large enough, then the occurrence of deadlock is less probable and recovery from deadlock is always possible. With sufficient storage space the transporter is always able to empty itself and/or to remove the material from on of the blocked resources.

The problem of combining messages that arrive at a processor is discussed next. In parallel processor systems communication may take place simultaneously, but a processor handles a single message at a time. In some cases a choice has to be made between handling messages separately or together even if they arrive after each other. As stated above the controller handles an object without blocking, i.e. without waiting for the next object. If objects have to be combined, they have to be stored until the combination is complete. If an object may be handled either alone or in combination with an object that is still to arrive, the disadvantages of waiting for the next object or handling the object at once, have to be weighed against each other.

In parallel processor systems information is sometimes stored in more than one processor. A problem is to keep the data consistent. A controller, for instance, records the status of its resources. The controller can change the status of a resource by sending it a command, so that the controller may update the status record. The status kept in the controller is also changed by the status message from the resource. This means that the status is kept in two processors and is changed by two processors. Attention has to be paid to the way the status record is changed, in order that it stays consistent with the status of the resource.

The controller may be modelled as an expanded processor or as a leaf processor. A rule of thumb is that processors that contain parallelism should be expanded. In our case, however, we have chosen not to expand the controller even though there is still some form of parallelism. A controller, as defined above, receives objects and handles these objects. These objects stem from different ports and can in many cases be handled in parallel. In order to handle these objects, however, common data are used. So, if the objects are handled in parallel, there has to be a central data base processor. This data base processor will in some sense have the same structure as the controller with parallelism. This will mean a shift of the parallelism from the controller to the data base. This is why we have chosen not to expand the controller in parallel processors. Another form of parallelism in the control algorithm has to do with the sending of objects after a delay. The task language of the ProcessTool has a messages for this kind of interaction. Because of this we have chosen not to use a separate processor that delays objects.

3.6 Summary

Chapter 2 has revealed the way in which the physical manufacturing system is specified. This is done using the operations, the material, the resources and the recipes. The various manufacturing classes, too, have been introduced: single shop, parallel shop, flow shop, and job shop. A manufacturing system consists of a controller, one store, one transporter, and a number of processing resources. The resource layout together with the recipe corresponds with one of the classes mentioned above. Because resources may be expanded, it is possible to create control layers. There are four controller categories. These are based on whether the controller communicates with consumers and suppliers (= factory controller), or if it is commanded by a supercontroller (= manufacturing controller) and on whether the controller controls expanded resources or machines (leaf resources).

Performance graphs have been introduced to measure the performance of a manufacturing system. These represent the lead time and through-

put as a function of the inventory level. The use of these graphs allows the ideal performance and the ideal work point of a manufacturing system to be determined. The control functions planning, scheduling and monitoring have been discussed. It has been seen that the releasing strategy is the most important aspect of scheduling.

As described in Section 3.3 the configuring of the control systems and the distribution of control decisions depends on the availability of information, the decisions that a controller has to take, control transport and processing. The material exchange is commanded by the transport system. The control configuration is influenced by the moment at which operations are transferred. Two strategies have been mentioned for the sending of transport commands: early and late transport. Likewise, command driven and material driven manufacturing are distinguished for the sending of the processing commands.

Section 3.4 discusses the different communication protocols more closely. Finally, the problems related to parallelism are discussed. Here factors such as deadlock, starvation, combining messages, data consistency and the modelling of the controller in ProcessTalk play a role.

In the next chapter a model of a general control architecture will be presented. This is intended for a job shop manufacturing system, but it is also suitable for the other classes that have been discussed. Both communication with consumers and suppliers, as well as with a supercontroller are considered. Processing operations are generated at the start of the manufacturing process, and the controller uses a command driven manufacturing strategy. Transport operations are generated on line by the controller. The controller of leaf resources applies a late transport strategy, while controllers of expanded resources may use either late or early transport. Material exchange operations are generated on line by the transporter.

The communication protocol between the controllers of different layers use status, command and result messages. A request is used as status, it expresses in terms of operations the momentary capacity of a resource. A job is used as command and a report is used as result. The release strategy of the manufacturing controller is implemented with the sending of requests to the supercontroller. Release may be delayed by the supercontroller by not answering a request immediately. The controller uses the subrequests from the resources to take allocation and sequencing decisions and dispatches the work in the form of a job. The allocation and sequencing is done on the basis of reactive scheduling and simple priority rules are used.

Chapter 4
The control architecture for manufacturing systems

In this chapter the ideas developed in the previous chapters are applied in the control architecture. It presents the data structure which is used to represent the objects in the model, together with the model itself, defined according to the Process-Interaction Approach. The model gives an insight into the workings of the manufacturing controller. The final sections discuss the application of the model to four different manufacturing system classes: single shop, parallel shop flow shop and job shop. Here we describe how they are to be controlled and how requests have to be generated. The last section goes further into the matter of configuring a controller hierarchy.

The total model can be used in the design of manufacturing systems, to test their control systems and to simulate their behaviour. If the simulation shows that the performance of the manufacturing system is satisfactory, then the model can be used to control the actual system; either an existing one, or one which still has to be constructed. The most important parameters that can be changed in this model are the operations, the material, the resources and the recipes which specify the physical manufacturing system. Other parameters influence the control configuration: the control layers with the different types of controllers, the recipes and the manufacturing system class of every expanded resource, and the control strategy (particularly the releasing and the sequencing strategy).

4.1 The data structure

In order to implement the model of a manufacturing system, we need data objects. These, together with the data structure, are described in what follows. The model of the factory is constructed using the ProcessTool [Wortmann 1991]. The process descriptions of the leaf processors are written in ProcessTalk, a Smalltalk-80 based language used by the modelling tool. As a consequence, the data structure is also written in Smalltalk-80. In Smalltalk-80, class names start with a capital letter; by convention, instances of a class receive the same name as their class or superclass, but they start with a small letter.

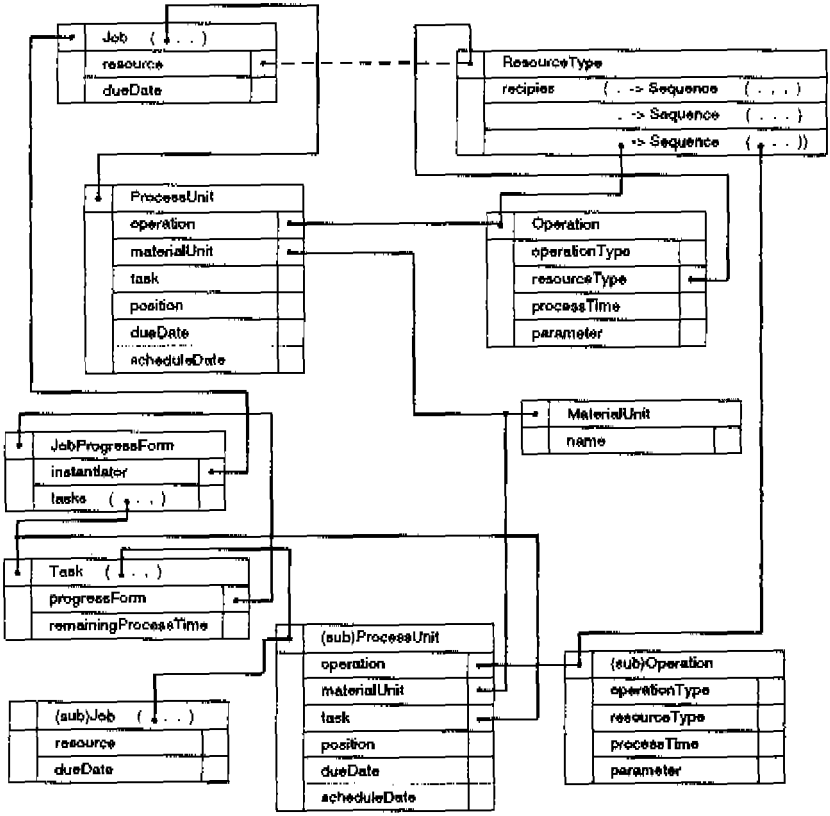


Figure 4.1. *Data structure: job - subjob relation. The blocks are objects with their instance variables. A solid line ending in the left corner of an object reflects the relation is contained by. A dotted line ending in the left corner reflects the relation is of type.*

Several types of objects are distinguished in the model. A short overview of the objects in the data structure is given below. Subsequently these objects are treated more elaborately. The processors of the model are instances of the classes Resource, ManufacturingController, Consumer and Supplier or subclasses of these classes. The processors are part of the model and are not treated extensively in this section.

The processors communicate with the help of objects. The objects that are transferred between Factory, Consumer and Supplier are instances of PotentialOrder, Quotation, RealOrder and Invoice.

Between controllers and between controller and processing resources instances of Request, Job and Report are transferred. These will also be called ProcessRequest, ProcessJob and ProcessReport.

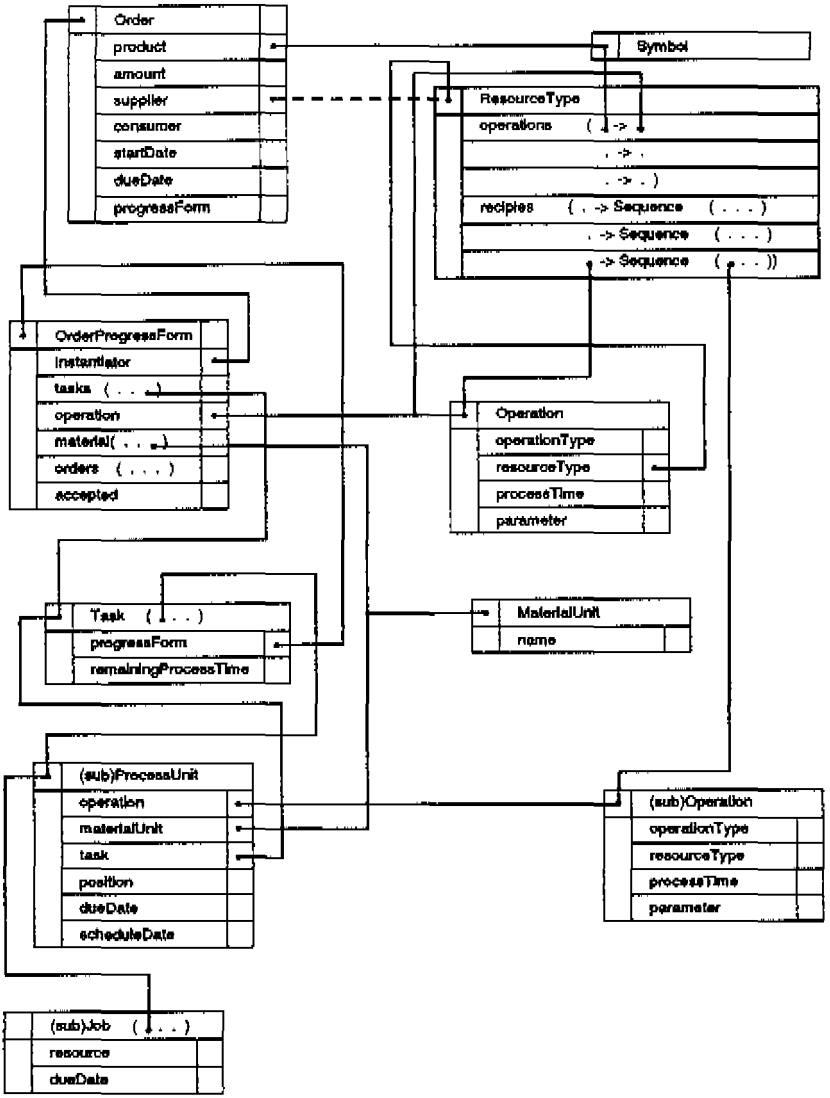


***Figure 4.2** Data structure: order - subjob relation.The blocks are objects with their instance variables. A solid line ending in the left corner of an object reflects the relation is contained by. A dotted line ending in the left corner reflects the relation is of type.*

Between controllers and transporters instances of TransportRequest, TransportJob and TransportReport are communicated.

Transporters exchange objects of the class Material and MaterialRequest with stores and leafResources.

The objects of type ProcessJob and TransportJob are collections. A processJob contains instances of ProcessUnit, a transportJob instances of TransportUnit. Both ProcessUnit and TransportUnit have a common superclass: WorkUnit.

A workUnit contains an instance of MaterialUnit and an instance of Operation. A processUnit contains a processOperation and a materialUnit, a transportUnit contains a transportOperation and a materialUnit.

Also the classes Material and MaterialRequest are collections, they contain instances of the class MaterialUnit.

Other categories of objects are used in the controller to record the progress of orders and jobs. The progress of orders is recorded in instances of OrderProgressForm, the progress of jobs is done with JobProgressForm. OrderProgressForm and JobProgressForm are subclasses of ProgressForm.

The manufacturing process is specified with objects of class Sequence which is a subclass of Recipe. The model only contains sequential recipes. The sequence is a collection of instances of ProcessOperation. The ProcessOperation is a subclass of Operation.

The actual process plan for the manufacturing of a product is kept in the controller. The process plan is an instance of Task. Because the recipes are always sequential, a task is also sequential. It contains a collection of processUnits.

The controller contains objects which do the calculating and deciding. These objects implement the functions of planning, scheduling and monitoring. In the manufacturing controller an instance of the class ProcessPlanner makes the process plans (tasks with processUnits) and monitors the manufacturing process. In the factory controller this is done by an instance of FactoryPlanner. The factoryPlanner does the same as the processPlanner and it incorporates the capacity planning function. The sequencing and allocating functions for resources of the same type are implemented by instances of the class JobScheduler. The total sequencing and allocating in a controller, and the generation of transportUnits, is done by instances of LateScheduler and EarlyScheduler. Where the first one uses a late transport strategy and the second an early transport strategy. Both LateScheduler and EarlyScheduler use instances of the class JobScheduler to take sequencing and allocating decisions.

The communication between controllers and between controller and resource, is based on the communication protocol that uses status, command and result (protocol 2 described in Section 3.5). These are implemented with the objects request, job and report, respectively. The

communication protocol between the factoryController and the environment (Consumer and Supplier) is implemented with the help of potentialOrders (consumer enquiries), quotations (responses to a consumer enquiry), realOrders and invoices. This way it is possible to achieve an interface between these processors, where the refusal of orders is possible. An order represents a command for the factoryController. The manufacturingController receives jobs as commands.

Both the order and the job invoke the manufacturing process. Below we first describe the way the different objects are related for the job (Figure 4.1) and the order (Figure 4.2). After that a more detailed description of the different classes is given.

The job invokes a manufacturing process. The job is a collection of processUnits. A processUnit is an operation associated with a materialUnit. The operation of the processUnit points to a recipe. The recipe is a collection of operations and a specification of the materialType. The materialType of the recipe has to correspond with the class of the materialUnit. This recipe is used, together with the materialUnit, for the creation of the task. The task is the process plan, it administrates the progress of the manufacturing process. The task is kept in a controller and it is not sent to other controllers. The task consists of a collection of processUnits. The processUnits of the task are used for the creation of subjobs. The task is linked to the job with help of a jobProgressForm (see Figure 4.1).

The order also invokes a manufacturing process. The order contains, among other things, a product name and an amount. The orderProgressForm contains all orders that have to be delivered in order to purchase the raw material needed to manufacture the product. The product name is used to find an operation that points to a recipe for the manufacturing process. The operation and the material are kept in the orderProgressForm. These two are used to create the task for the manufacturing of the products. The task is linked to the order with help of an orderProgressForm (see Figure 4.2).

Now follows a detailed description of the objects in the manufacturing model. All processors of the model are instances of subclasses of the class ProcessorObject. The objects transferred between processorObjects are either instances of a subclass of InteractionObject or of InteractionCollection. An interactionCollection is a collection of objects. These objects need not be instances of a subclass of InteractionObject. An interactionObject and an interactionCollection contain the address (the processorObject for which it is intended), and the arrival time (the time it arrives at the intended processorObject). A processorObject only receives an interactionObject or an

interactionCollection if it is addressed to itself. When sending an interactionObject or an interactionCollection, the processorObject addresses the object for the appropriate processorObject. A processorObject records the arrival time of an interactionObject or an interactionCollection; as a consequence these objects know the time during which they stay in a processorObject.

In a manufacturing system there is a distinction drawn between material and information. All pieces of material in the model are instances of a subclass of the class MaterialUnit. These instances refer to a piece of physical material. This is used to model the physical material flow and to model the reference to a piece of material in the information flow. This means that a processor can identify a piece of physical material. Every piece of material has a unique name. To transfer material between processorObjects, material is sent as a collection of materialUnits. This collection contains one or more instances of MaterialUnit. In this way it is easy to send different batch sizes of material to a resource. A material collection only exists for a short while and has no special name. The collection is an instance of the class Material, which is a subclass of InteractionCollection.

The class Material is not intended for the creation of hierarchical material structures. In the model it is supposed that all materialUnits are of the same type. If a hierarchical material structure is needed, this has to be created by adding instance variables to subclasses of the class materialUnit.

Example of a hierarchical material structure. A cassette with 0 to 25 wafers is built with the classes Wafer and Cassette.

class name: Wafer
superclass: MaterialUnit
class name: Cassette
superclass: MaterialUnit
instance variable names: wafers

The instance variable wafers is an instance of the class Array (a Smalltalk-80 class) of size 25 and contains 0 to 25 instances of Wafer.

In order to be able to draw material from a store, the store has to know which material is to be withdrawn, and the processor to which the material has to be sent. An instance of MaterialRequest commands a Store to send the specified material to the destination. The class MaterialRequest is a subclass of InteractionCollection. A materialRequest contains materialUnits, and has an instance variable that specifies the processorObject to which the material has to be sent (the destination).

The instances of the class Operation are used to tell a resource which recipe has to be executed. There are two kinds of operations: the TransportOperation, which specifies a transport movement for a transporter; and the ProcessOperation, which specifies a recipe of a processing resource. The processOperation is of a certain type and has a specific parameter. Two processOperations of the same type with different parameters refer to the same recipe, but with different adjustments of the resource. The processOperation is executed on a resource of a certain type. This resource type is specified in the processOperation. The processOperation knows the time it takes to execute the related manufacturing process. This time, the process time, refers to the nominal time required to execute the processing suboperations. The actual time needed to execute the processOperation is increased by waiting times and by the transport. The instances of the class TransportOperation specify the origin and destination of material.

The work a resource has to perform is specified by an operation and a piece of material on which the operation has to be performed. This combination is part of a Task, which specifies the complete manufacturing process plan on the materialUnit in the manufacturing system. The combination of operation and materialUnit is called a workUnit. The workUnit contains a reference to the task to which it belongs. For the purposes of administration, the workUnit has an instance variable arrivalTime. Like operations, there are also two types of workUnit: TransportUnit and ProcessUnit. For scheduling purposes it is possible to assign a startDate, a dueDate and a priority to a workUnit. In order to keep a record of the position of the material, the workUnit keeps the position in the instance variable position.

The class TransportUnit has two methods by which it can access the origin and the destination of the materialUnit. Although the transportUnit belongs to a task, it is not placed in the task when the task is created; rather, it is created at the moment the materialUnit of the task has to be transported. In this way it is possible to choose the route of the materialUnit at the latest possible moment.

The manufacturing process that the material has to undergo is specified by a task, which consists of a collection of processUnits. A task belongs to a progressForm. The task is related to material, which is specified in the processUnits. The task is derived from a recipe. The task contains processUnits, the recipe contains operations. The materialUnits, together with the operations from the recipe, form these processUnits. The materialUnits are specified in the processUnit(s) of the job that invoked the task. Tasks are not transferred between processors, they stay inside the controller. From a task subjobs are derived, which are transferred to resources. The task structures are equivalent to the recipe structures.

Only sequential tasks are implemented in the model. In a sequential task all processUnits have to be executed one after the other (in sequence), and all processUnits are related to the same piece of material. The material and position are represented in the processUnit because, in other types of task, more than one piece of material can form part of the task.

A finished processUnit is removed from the task. In the case where the last processUnit of a task has been executed, the resulting materialUnit has to be transported to a store. For scheduling purposes a record of the remaining process time of a task is kept. A task contains only processUnits. The transportUnits are dynamically created by the controller during the execution of the task. The processUnits are created upon the arrival of the job that invokes the task.

A progressForm is used to administrate the execution of an order or a job. The reception of an order or a job is a signal for a controller to start certain actions. A record of the progress of these actions is kept in the instances of OrderProgressForm and JobProgressForm, respectively. Both are subclasses of ProgressForm. The class ProgressForm implements the common messages of OrderProgressForm and JobProgressForm.

The class ProgressForm has an instance variable for the instantiator of the progressForm. This is either an order or a job. To execute the order or the job the resources have to execute one or more tasks. These tasks are kept in an instance variable. To be able to administrate the performance of the manufacturing system, the time when the execution of the tasks started and finished is kept in instance variables.

The material content of a progressForm is equal to the number of materialUnits that are manufactured in the tasks. The work content of a progressForm is thus the material content multiplied by the manufacturing time of one materialUnit.

The instances of class JobProgressForm register the progress of the execution of a job. At the start the jobProgressForm is created with the help of a job. After the finishing of the tasks the jobProgressForm delivers a report on request.

The factory has no supercontroller that dispatches jobs associated with operations; rather, it has consumers that dispatch orders associated with products. This means the progress of orders in the factory has to be recorded differently from the progress of jobs in the manufacturing system (see Figure 4.1 and 4.2). The instances of OrderProgressForm are used to record the progress of the execution of an order. The progress of the purchasing and of the manufacturing process are kept in the

orderProgressForm, in the instance variables orders and tasks, respectively. Because the relation between a task and an order differs from the relation between a task and a job, an orderProgressForm contains some extra instance variables in comparison with the jobProgressFrom. A job contains the material that has to be processed and the operation that points to the recipe that has to be used to create the task. An order contains a product name and the amount that is wanted. Raw material has to be ordered in order to manufacture the products. The orders are kept in the instance variable orders. The material that has to be transformed is kept in the instance variable material. The operation that represents the interface between the product name and recipe is kept in the instance variable operation. This operation is introduced in order to be able to represent the recipes in the factory in the same) as in a manufacturing system.

For the ordering of products a protocol with potentialOrders, quotations, realOrders and invoices is used. The consumer enquires whether it is possible to deliver products; the quotation states whether the potentialOrder is feasible or not. A quotation that accepts a potentialOrder may be answered with a realOrder by the consumer. The realOrder is answered with an invoice (after delivery of the products). The orderProgressForm is used for potentialOrders as well as for realOrders. For a potential order, the orders sent to the suppliers are also potential. With the help of the quotations from the suppliers, the controller decides whether the potential order of the consumer is acceptable. A potential order is only acceptable if all potential orders are accepted by the suppliers.

The instance variable orders keeps track of all orders that have not yet been responded to. The instance variable accepted keeps track of whether the received quotations have accepted or rejected the potentialOrders that have been sent. If the collection orders is empty, all outstanding orders have been answered and the instance variable accepted contains whether the potential consumer order is acceptable or not.

In the case where the orderProgressForm belongs to a realOrder, the collection orders is empty if all the raw material that has been ordered has actually been delivered. In this situation the manufacturing of the products starts.

Instances of the class Order specify the product and the amount of products that a consumer wants from a supplier. An order contains its sender in the instance variable consumer and its receiver in the instance variable supplier. The order contains a dueDate, which is the date before which the products have to be delivered. For administrative reasons the order also contains a reference to the progressForm to which it belongs.

The class Order has two subclasses: PotentialOrder and RealOrder, to be able to distinguish between both types of orders.

The response to a potentialOrder is an instance of the class Quotation. The quotation contains the order to which it is related. A quotation also has an instance variable that indicates whether the order is accepted or rejected.

After the manufacturing and the distribution of the products for an order, an instance of Invoice is sent. An invoice contains a reference to the order that invoked the manufacturing of the products and a reference to the material that has been delivered to the consumer.

A job is a collection of workUnits that have to be executed (simultaneously) by a resource. There are two kinds of jobs: the processJob and the transportJob. The class TransportJob is a subclass of (Process)Job. All the workUnits of a job have to comprise the same operation. This means that the operation type and operation parameter of the processOperations have to be the same, or in case of a transportJob the origin and the destination of the transportOperations have to be the same. A job is destined for a certain resource, so the job has a reference to the resource for which it is intended. The job specifies an amount of material and an operation that has to be performed on the material. Although the processUnits of a processJob have each their own dueDate, these dates do not necessarily have to be the same. For this reason a processJob has its own dueDate. The class TransportJob has two methods by which it can access the transport origin and destination.

An instance of Request specifies the conditions which a new job for the resource has to fulfil. A requests gives information about the operationTypes a resource is willing to execute. The minimum and maximum batch size of the job is specified and a request contains information about the resource from which the requests stems. A request has to be answered by exactly one job. This may occur immediately, or after a time interval.

After a job has been executed, an instance of Report is sent to the sender of the job. A report contains the job that has been executed. There are two types of Report: the (Process)Report and the TransportReport. A transportReport belongs to a transportJob. Because a task always finishes with a transportJob, the transportReport has a method to investigate whether it belongs to a finished task.

The class Resource is used to specify which behaviour and properties a Resource has. A resource is an instance of a subclass of the class Resource. The class Resource is a subclass of ProcessorObject. The behaviour specified by the instance protocol of Resource are methods

every resource knows. All resources have different names. An expanded resource consists of subresources. The resource knows its subresources and it knows the type of the subresources.

A resource has properties that are the same for all resources of the same type. These properties are specified in the class protocol of Resource. Every resource is of a certain type: the resourceType. The resourceType corresponds with the class name of the resource. A resource type has a minBatchSize and a maxBatchSize which specifies the number of materialUnits the resource expects in one job. A resource has a limit to the amount of material and amount of work it can process simultaneously. In the case where the resource is of type machine the maxInventoryLevel is the same as the maxBatchSize. A resourceType has a dictionary of recipes, which lists, for every operation, the suboperations that have to be executed by the subresources. The recipe for a certain operation is used by the resource controller to construct the task that specifies the manufacturing process the resource has to perform in order to execute a job. For the resourceType there is also a set of operationTypes that specifies which operationTypes the resource is capable of executing. An operationType can only be executed on one type of resource.

The controller uses objects to perform calculations and decisions. The planner is one of these objects. There exist two types of planners. A planner for the manufacturing controller and a planner for the factory controller. The first is of the class ProcessPlanner, the second is of the class FactoryPlanner, which is a subclass of ProcessPlanner.

A processPlanner makes process plans for the jobs the controller receives. These process plans are represented in tasks, and are coupled to jobs via progressForms. When a task is finished the processPlanner receives a transportReport which is used to see whether a report for a job has to be generated. These reports may be requested from the processPlanner. The processPlanner also keeps track of the performance of the manufacturing system. It monitors the input, the throughput, the inventoryLevel and the leadTime.

The factoryPlanner plans the available capacity in the controller, it records the progress of the purchasing of material for orders and it makes process plans for the received orders. The factoryPlanner may be asked whether there is still capacity available to manufacture an order. The purchasing is done both for potentialOrders and for realOrders. From a potentialOrder the factoryPlanner creates potentialOrders for the supplier. From a realOrder it creates realOrders for the supplier. From quotations it formulates quotations for the consumer. From invoices from the supplier it formulates tasks for the manufacturing of products. From a transportReport, which signals the completion of a task, an

invoice is formulated for the consumer. The factoryPlanner also keeps track of the capacity available in the factory. Just like the processPlanner, the factory planner monitors the input, the throughput, the inventoryLevel and the leadTime.

Three classes are responsible for the allocating and sequencing functions: there is a class that allocates and sequences the work for one type of resource, called the JobScheduler; and there are two classes that schedule all work for the processing resources, called the LateScheduler and the EarlyScheduler. The last two classes differ only in the way the transport takes place: late transport or early transport.

The jobScheduler schedules requests (allocating) and it schedules workUnits (sequencing). The jobScheduler uses these objects to formulate new jobs for the resources of one type. A request is related to a resource, a workUnit is related to material. The resource and the workUnit are combined with help of the operationTypes mentioned in the request and the operation mentioned in the workUnit. The jobScheduler is used for both processingResources and for transporters. The jobScheduler implements sequencing and allocating. For these it uses simple sequencing rules.

The class EarlyScheduler is a subclass of LateScheduler. Both have the same message interface but differ in their implementations and are used in different ways. Here only the message interface is discussed and only the lateScheduler is mentioned further. The lateScheduler takes care of the scheduling of work on the processing resources, which is done by formulating new transportJobs and processJobs. The lateScheduler uses one jobScheduler for every resourceType in the manufacturing system. The lateScheduler schedules requests and it schedules tasks; as a response it formulates new transportUnits. The controller lets the transportScheduler (which is an instance of JobScheduler) schedule these. The lateScheduler treats processReports, which also result in new transportUnits. The lateScheduler formulates new processJobs for the resources from transportReports.

4.2 The control model

The environment in which a factory operates is called the market. The market is characterized by a sequence of consumers and suppliers. The market model contains a consumer, a factory and a supplier (Figure 4.3). The factory behaves as a consumer to the supplier and as a supplier to the consumer. The factory receives orders for products from the consumer, and it orders raw material from the supplier. The supplier delivers raw material to the factory. The factory manufactures products from the raw material and delivers the products to the consumer.

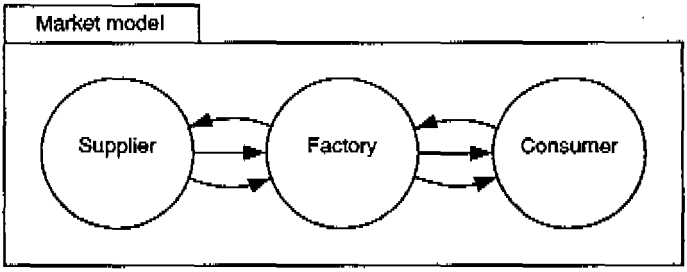


Figure 4.3. *Market model.*

In the model the protocol of ordering and delivering between a consumer, factory and supplier is similar to the protocol 4 mentioned in section 3.4. The supercontroller corresponds to the consumer, the controller corresponds to the factory controller and the subcontroller to the supplier. The messages statusRequest, status, command and result are replaced by respectively potentialOrder, quotation, realOrder and invoice.

The factory model consists of a factory controller and factory resources (Figure 4.4 a, b and c). The factory controller commands the factory resources and it handles the administrative interactions with the supplier and the consumer. The factory controller's task consists of capacity planning, marketing which is about handling consumer orders, purchasing of raw materials, manufacturing control and distributing control. In Figure 4.4 d the process description of a factory controller is given, it

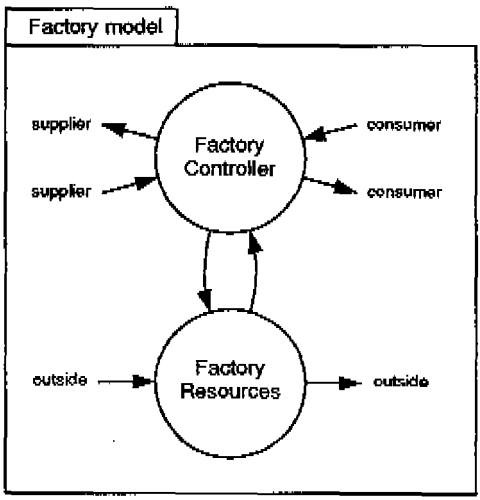


Figure 4.4. *Model of the factory.* ***a)*** *Factory model.*

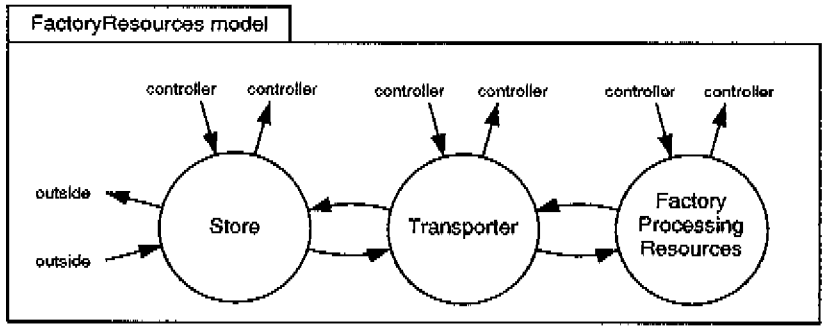


Figure 4.4. *Model of the factory.* ***b)*** *FactoryResources model.*

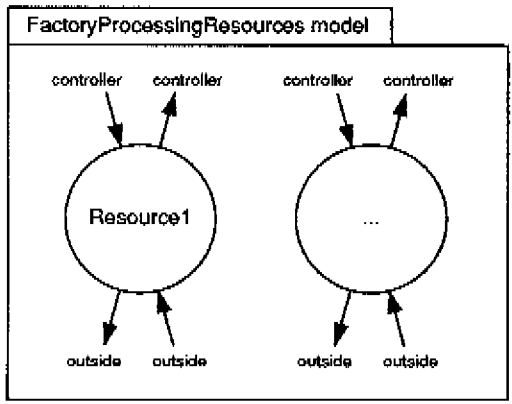


Figure 4.4. *Model of the factory.* ***c)*** *FactoryProcessingResources model.*

handles orders in a sequential fashion, one at a time. The actual process description of the factory controller is shown in Appendix C.

If the factory controller receives a potentialOrder, it checks whether there is capacity available to manufacture the products. If there is capacity free to manufacture the products before the due date demanded, the factory controller formulates a potentialOrder for the supplier. The potentialOrder is sent to the supplier to check whether the supplier can deliver the needed raw material in time. Only if the supplier accepts the potentialOrder of the factory, does the factory accept the potentialOrder of the consumer. In all other cases the potentialOrder is rejected by the factory controller. When the consumer receives a quotation that accepts the potentialOrder, it decides whether it wants to place a real order.

The consumer sends the realOrder to the factory controller. The factory controller formulates a realOrder for raw material and sends it to the

```
FactoryController » body
    | order |
    order := self receiveOrder.
    order isPotential
        ifTrue:
            [(planner hasCapacityFor: order)
                ifTrue:
                    [self handlePotentialOrder: order.
                    self sendPotentialOrder.
                    self handleQuotation: self receiveQuotation].
            self sendQuotation]
        ifFalse:
            [self handleRealOrder: order.
            self sendRealOrder.
            self handleInvoice: self receiveInvoice.
            self handleSubrequest: self receiveSubrequest.
            self handleTransportRequest: self receiveTransportRequest.
            self sendTransportJob.
            self handleTransportReport: self receiveTransportReport.
            self sendSubjob.
            self handleSubreport: self receiveSubreport.
            self handleTransportRequest: self receiveTransportRequest.
            self sendTransportJob.
            self handleLastTransportReport: self receiveTransportReport.
            self sendInvoice]
```

Figure 4.4. *Model of the factory.* ***d)*** *Process description of a simplified FactoryController.*

supplier. The factory controller also reserves capacity for the manufacturing of the products. After the receipt of the raw material and the invoice from the supplier, the controller commands the factory resources to manufacture the products. The description of the control of the manufacturing process by the factory controller is analogous to the control in a manufacturing system controller and is described below. After the manufacturing process is completed, the factory controller sends a distribution command to the factory store and an invoice to the consumer. The factory store sends the products to the consumer.

This model does not contain a transport system between factories. The exchange of material between the supplier and factory and between factory and consumer is modelled as an interaction path between processors.

The factory model is based on the factory control architecture described by Arentsen [1989]. It is possible to connect more than one supplier or more than one consumer to the factory, and it is possible to use a forecast controller to realize different ordering strategies. In Arentsen's model the processor FactoryResources is modelled as a single processing machine; in the present model it is modelled as a factory store, a transporter and processing resources (Figure 4.4b).

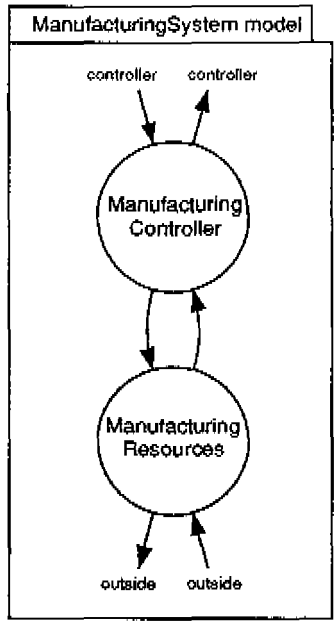


Figure 4.5. *The model of a manufacturing system.*
a) *ManufacturingSystem model.*

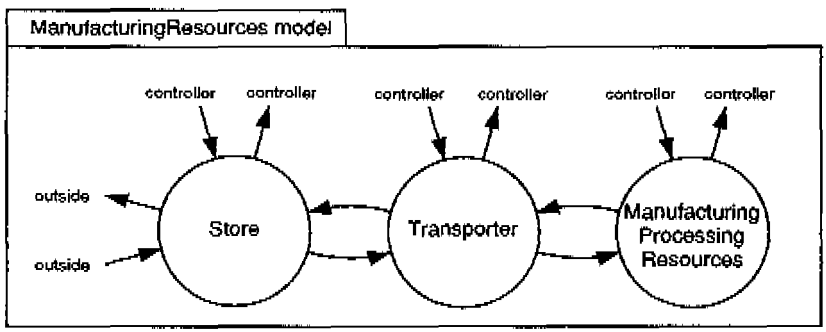


Figure 4.5. *The model of a manufacturing system.*
b) *ManufacturingResources model.*

The processing resources in a factory are either leaf resources or expanded resources (Figure 4.4c). A leaf resource (a machine) performs one operation on all the material in the machine and it has no capability to store other material. If the processing resource in a factory is an expanded resource, it consist of a group of processing machines. An expanded resource is also called a manufacturing system; it does have the capability to store material that is not being processed.

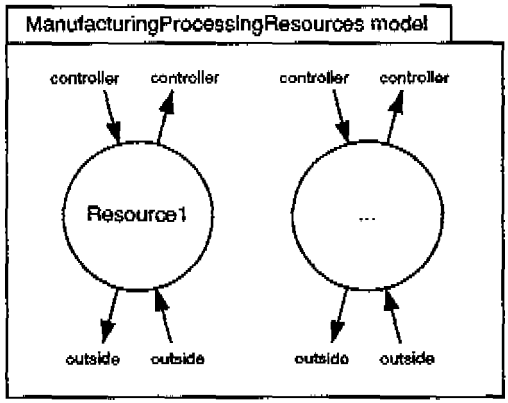


Figure 4.5. *The model of a manufacturing system.*
c) ProcessingResources model.

A manufacturing system (Figure 4.5a) and a processing machine have the same interactions with their environment. A manufacturing system contains at least one store and one transporter (Figure 4.5b). The processing resources of a manufacturing system are leaf resources or expanded resources (Figure 4.5c). It is possible to repeat the control structure of the manufacturing system recursively in the expanded processing resources. The recursion starts with a factory controller at the top and ends with a processing machine at the bottom. The number of control layers between the factory controller and the processing machine is a design parameter.

In the next part a distinction is made between job, subjob, request, subrequest etc. The meaning of these terms are given in Figure 4.6.

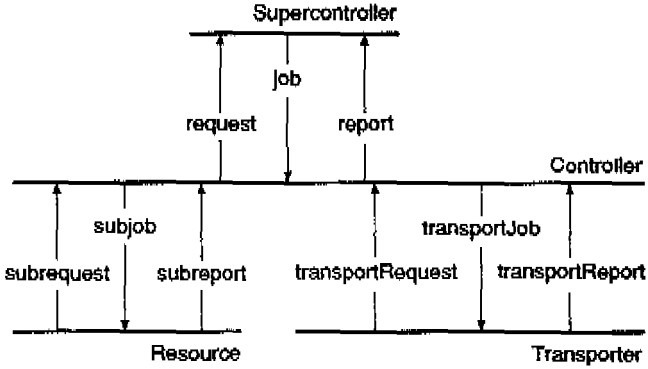


Figure 4.6. *The names of objects sent between the different processors.*

```
ManufacturingController > body
    self handleSubrequest: self receiveSubrequest.
    self sendRequest.
    self handleJob: self receiveJob.
    self handleTransportRequest: self receiveTransportRequest.
    self sendTransportJob.
    self handleTransportReport: self receiveTransportReport.
    self sendSubjob.
    self handleSubreport: self receiveSubreport.
    self handleTransportRequest: self receiveTransportRequest.
    self sendTransportJob.
    self handleLastTransportReport: self receiveTransportReport.
    self sendReport
```

Figure 4.7. *Process description of a simplified (sequential) manufacturing controller.*

We now go on to explain the description of the execution of a job in a simple manufacturing system, which executes jobs sequentially one after the other. The related process plan (a task) consists of a single processUnit, which means a job leads to one subjob. This manufacturing system has one processing resource. The controller handles only one job at a time. The process description of this controller is printed in Figure 4.7.

The manufacturing controller starts with the receipt of the subrequest of the processing resource. The subrequest is given to the process scheduler. The subrequest is also used to generate a request for a job from the supercontroller. The request is sent to the supercontroller. The supercontroller takes care of transport of material to the store. The transportation of material to the system store is either a reaction to the request from the system, or else the material is already available in the store. The processJob for the manufacturing controller is a response to the request. This means that the material always arrives before the processJob and the processJob always arrives after a request has been sent.

The controller lets the process planner create the description of the manufacturing process (the process plan) that has to be executed in order to execute the processJob. The process plan or the task is here a collection of one processUnits. The process planner records what jobs are in progress. The process planner administrates the new processJob and generates a progressForm that contains the task that has to be executed.

The processUnit of the task is scheduled. For this purpose the process scheduler uses the subrequest from the processing resource. The process scheduler monitors the status of the processing resources (with help of subrequests) and generates subjobs for processing resources with the

process plan (from the process planner). The process scheduler uses the process plan to create transportUnits and to dispatch the processUnits to the processing resources. The scheduler in this controller uses a late transport strategy. This means that the transportUnits are generated after the sequencing of the processUnits.

The processUnit and the subrequest taken together result in a subprocessJob for the processing resource. Before the subprocessJob is sent to the processing resource, the material has to be transported to the processing resource. The process scheduler generates from the subprocessJob the transportUnit that specifies the transport of material to the processing resource. The transport scheduler uses this transportUnit and waits for a transportRequest from the transporter to generate a transportJob. This transportJob is sent to the transporter and the transporter moves the material to the processing resource. When the transporter has finished its transportJob it sends a transportReport to the controller. The transportReport is a sign to the process scheduler that the subprocessJob may be sent to the processing resource. The controller dispatches the subprocessJob to the processing resource.

When the processing resource has finished the processing of the material, it sends a subprocessReport to the controller. The process scheduler uses the subprocessReport to generate a new transportUnit. This unit specifies the material that has to be moved from the processing resource to the store. The transport scheduler waits for a transportRequest, then generates a new transportJob from the transportUnit and the request. This transportJob is sent to the transporter. The transporter picks the material from the processing resource and places it in the store. After the completion of the transport, the transporter sends a transportReport to the controller. The transportReport belongs to a finished job. For this reason the transportReport is given to the process planner that administrates the finished processJob and delivers a processReport. This processReport is sent to the supercontroller. The supercontroller has to take care of the removal of the material from the store of the manufacturing system.

After the processJob has been finished the controller waits for a new subrequest and sends a request for a new processJob to the supercontroller. In the case where the manufacturing process is controlled by a factory controller, the factory controller has to take care of the material flow itself and no requests are sent to get jobs: the jobs stem from the supplier and the capacity planner.

The model of a sequential manufacturing controller has been described above. However, the manufacturing controller has to handle some complications. First, it has to control more than one processing resource and to execute more than one processJob at a time. This is handled by

```
ManufacturingController > body
    self
        receiveFromOneOf: #('controller' 'resource' )
        before: self requestSendTime
        do:
            [ :portName :item |
            portName = 'controller'
                ifTrue:
                    [self handleJob: item.
                    self sendAvailableTransportJobs].
            portName = 'resource'
                ifTrue:
                    [item isRequest
                        ifTrue:
                            [self handleSubrequest: item.
                            self sendAvailableTransportJobs].
                    item isReport
                        ifTrue:
                            [self handleSubreport: item.
                            self sendAvailableTransportJobs].
                    item isTransportRequest
                        ifTrue:
                            [self handleTransportRequest: item.
                            self sendAvailableTransportJobs].
                    item isTransportReport
                        ifTrue:
                            [item belongsToFinishedTask
                                ifTrue:
                                    [self handleLastTransportReport: item.
                                    self sendAvailableReports]
                                ifFalse:
                                    [self handleTransportReport: item.
                                    self sendAvailableSubjobs]]]]
        ifTimedOut: [self sendRequest]
```

Figure 4.8. *Process description of the manufacturing controller.*

using a parallel algorithm where the execution of all tasks is progressed by the events that happen. Second, a task can consist of more than one processUnit. This means that the control algorithm has to repeat the execution of processUnits until the task is finished. Third, the manufacturing controller has to handle batch size differences between processing resources. In order to do this material has to be split and/or to be combined. The splitting of material is made possible by allowing more tasks for one processJob. Thus the material can be processed in smaller quantities. To process material of different tasks on one processing resource, processUnits have to be combined. This is possible because a subprocessJob may consist of more than one processUnit. The fourth complication is the generating and the sending of the request, which will be handled in the next sections.

The control algorithm of the manufacturing system has been described in a purely sequential fashion. By rewriting the algorithm in a parallel version, in our case an event driven controller it possible to control all

kinds of manufacturing systems. The controller is event driven: the reception of an object (an event) precedes a part of the manufacturing process. The parallelism of the control algorithm results in a controller that is continuously waiting to receive objects. As a response to these objects it sends, if possible, other objects. The objects a manufacturing controller can receive are: a processJob, a subrequest, a subprocessReport, a transportRequest or a transportReport (Figure 4.6). The objects a controller sends are: a request, a processReport, a subprocessJob and a transportJob. The sending of the requests is coupled to a timer in order to be able to send requests after some delay. The process description of the manufacturing controller is given in Figure 4.8. Hereafter the parallel control algorithm is discussed, i.e. the different actions to be undertaken after the receiving of an object.

If the potential actions cannot be executed because other objects are missing, the received object or derived objects are stored by one of the schedulers. The scheduler stores processUnits and requests, the transport scheduler stores transportUnits and transportRequests.

processJob

The process planner generates a progressForm for the processJob and specifies the task that has to be executed in order to execute the processJob. The task is scheduled by the process scheduler. The process scheduler uses subrequests from the processing resources to find which resource is able to execute the processUnit from the task. The process scheduler generates the transportUnits. Because material may be split, the arrival of a processJob may lead to more than one transportUnit. It has been chosen to let the process scheduler plan the route of the material, because this way fixed routes and flexible routes are handled in the same way. This is not possible if the process planner generates the routes.

The transportUnits from the scheduler are scheduled by the transport scheduler. If the transport scheduler has a transportRequest which can execute the transportUnit, this results in a transportJob, which is sent to the transporter.

subrequest

If the controller receives a subrequest, this may enable the process scheduler to sequence a processUnit on a processing resource. As a consequence the process scheduler generates a transportUnit. Together with a transportRequest this results in a transportJob for the transporter.

subprocessReport

The controller receives a subprocessReport if a processing resource has processed the material. As a consequence the material from the resource has to be transported to the next processing resource. To do so the process scheduler has find out the next operation to be executed on the material (process interpreting) and to allocate material to a processing resource. If the task (process plan) is finished, there are no more processUnits to be executed by any processing resource and the material has to be transported to the store. The process scheduler that handles the subprocessReport tries to generate the transportUnits. This is only possible if the task is finished or if there is a subrequest from the resource to which the next processUnit can be allocated. A consequence of combining material is the fact that a subprocessReport may lead to more than one transportUnit. If the transport scheduler has transportRequests, the transport scheduler generates a transportJob.

transportRequest

The transportRequest is used by the transport scheduler to generate transportJobs. If there are transportUnits the receiving of a transportRequest results in the sending of a transportJob. Because of combining of material a transportJob may transport more materialUnits at one time.

transportReport

With the arrival of a transportReport at the controller two cases have to be distinguished. The material has been transported to a processing resource where a processUnit has to be executed, or the manufacturing process on the material is finished and the material has been transported to the store. If the material has been transported to a processing resource, then the process scheduler allocates the subprocessJob for that resource and it is sent to the resource. Because of the combining of material a transportReport does not automatically lead to the allocating of a subprocessJob. The subprocessJob has to be sent if all material of the subprocessJob has been transported to the resource. Transport of the material to the store means the manufacturing process of the material is finished. A report has to be generated only after the arrival of the last piece of material of a processJob, this is a consequence of the splitting of material. The planner generates a processReport for the processJob and the processReport is sent to the supercontroller.

In many cases the process scheduler knows the material route beforehand. This enables the early generation of transportUnits. With early transport, the process scheduler generates the transportUnit on arrival of the processUnit. The transportReport releases processUnits and the process scheduler schedules these with subrequests from the processing resources. The transport is executed before the scheduling. A manufacturing controller with such a process scheduler behaves differently from the controller with late transport, and these differences are discussed below.

The differences are found in the process scheduler and in the actions taken after the reception of a processJob, a subrequest and of a transportReport. After the reception of the processJob the process scheduler always generates a transportUnit and, if there is a transportRequest available, the transport scheduler generates a transportJob. After a transportReport material has to be allocated to the resource, it depends on the received subrequests whether a subjob has to be sent or not. After the reception of a subrequest no transportUnit is generated but material has to be sequenced on the resource, if material present in the resource, a subjob is created and this is sent to the resource. A transportReport of a finished task is handled in the same way as by the late scheduler.

The request send strategy implements the release strategy. The actual release of jobs is equal to the sending of a request or, if no jobs are available, it is worse than the request send strategy. The sending of requests depends on the status of the manufacturing system. The sending of a request may be strongly coupled to the sending of subrequests, or not coupled by using a fixed time interval for sending a request (open-loop). A simple request send strategy is to send a request for every subrequest. This only works if every processJob results in one subprocessJob. Then there is the possibility to use the subrequest of one specific processing resource of the manufacturing system to send a new request, or a combination of subrequests of different processing resources. The progress of processJobs can be used for the generation of subrequests. For instance: send a request if a job is finished. This seeks to achieve a fixed number of jobs in progress (Fixed-WIP). Another way is to send a request after the start of a specific subprocessJob, it is also possible to delay such a request, which results in a so-called request generation with delay. To send a request some time interval after the last request is a so-called uniform starts strategy. This strategy uses no status information of jobs and resources. It sends a new request after a time interval. The fixed time interval has to be adapted in advance to the capacity of the manufacturing system. A possibility to use dynamic time intervals is presented by Mommers [1990]. He calculates the time interval using

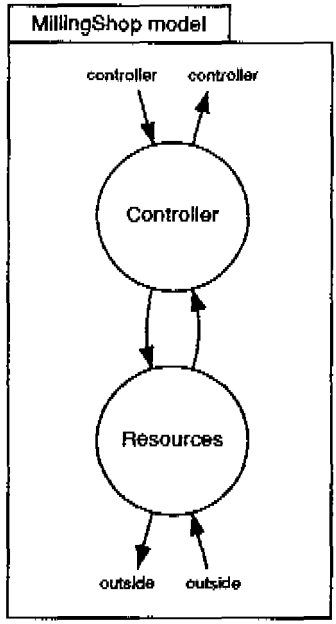


Figure 4.9. *Model of the single milling shop.*
a) *The MillingShop model.*

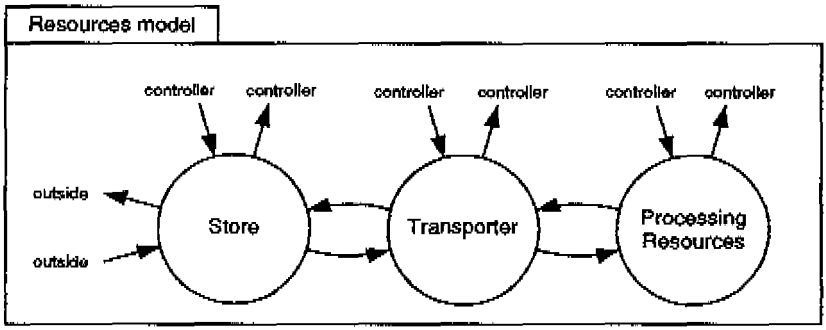


Figure 4.9. *Model of the single milling shop.*
b) *The Resources model.*

performance indicators such as lead time and inventory level, thus establishing a closed-loop strategy.

A manufacturing system with processing machines has to avoid blockage of resources (deadlock). To avoid deadlock complex simulators may be used which only generate a request if the execution of the job will never block the system.

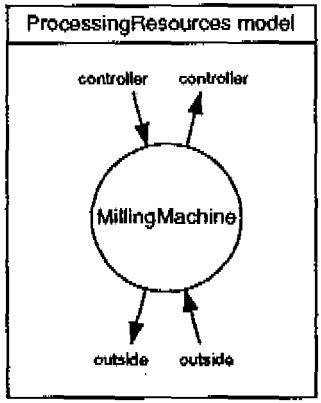


***Figure 4.9.** Model of the single milling shop.*
c) The ProcessingResources model.

The next sections discuss the relation between request generation and the class of the manufacturing system. In the last section the relation between the request generation and multiple control layers is discussed.

4.3 Single shop

In this section the model of a single shop is presented, followed by its recipe table, which consists of operations the shop is able to execute, coupled to recipes. After that the request generation strategy is discussed and the performance graphs of a single shop are studied. The example starts from an ideal single shop in which all recipes take the same time. Finally, the consequences of deviations in process times are discussed.

In this example a single milling shop contains one milling machine. Its model is illustrated in Figure 4.9. The shop consists of a controller and resources. The resources consists of a store, a transporter and processing resources. The processing resources consist of a single milling machine. The milling machine processes one piece of material at a time. The manufacturing jobs for the shop have to contain only one piece of material. The manufacturing processes in the shop consist of one operation on the milling machine. The recipe table is given below.

recipes:
millProductA1 -> Sequence (mill1)
millProductB1 -> Sequence (mill2)

The shop is capable of executing the operations millProductA1 and millProductB1. The milling machine executes the operations mill1 and mill2. The total manufacturing process in the shop consists of transport-

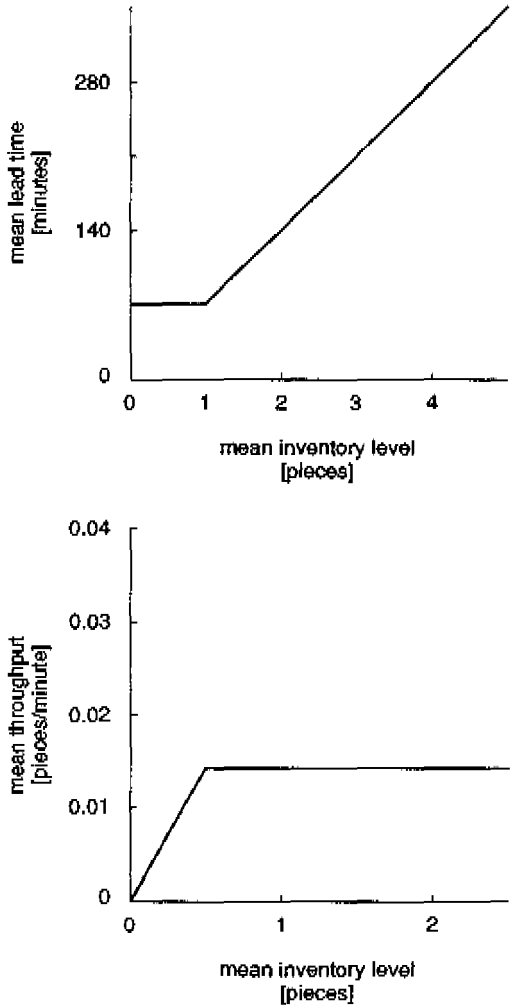


Figure 4.10. *The performance graph of the milling single shop. The mean lead time as a function of mean inventory level and the mean throughput as a function of mean inventory level.*

ing material from store to the milling machine, milling the piece of material and transporting the material from milling machine to store. It is supposed that the transport takes 5 minutes per movement and the milling takes 60 minutes. This means the total manufacturing process takes 70 minutes. The shop controls a machine, so a late transport strategy has to be used (see Section 3.3).

Several possibilities exist for the generation of requests. The first is to use the subrequest from the milling machine for the generation of a new request for the supercontroller. This works well if the supercontroller is

immediately capable of sending a new job for the shop (if the supercontroller uses the early transport strategy) else the shop has to wait a while before a new job is available: during this period of time the milling machine remains idle. Let us suppose it takes 15 minutes before the supercontroller is able to send a new job in response to a request. In this case the request strategy should try to make sure the new job arrives at the shop at the moment the manufacturing process is finished. The manufacturing process in the shop is ready 65 minutes after the milling machine starts processing. So a new request should be send 65 - 15 = 50 minutes after the sending of the subjob to the milling machine. Thus the controller sends the subjob and uses this event to start a timer which signals the moment to send a request to the supercontroller. This manner of requesting is called request generation with delay. A simpler way to request would be to request a new job 70 - 15 = 55 minutes after the arrival of a job. However, this is an open-loop policy: the arrival of a job at the milling shop is no guarantee that the current job for the milling shop is finished within a certain time. The request of a single shop contains all operation types for which the single shop has recipes. In the example the request contains the operation types millProductA and millProductB.

The milling machine is idle during the transport of material (loading and unloading). This is due to technological constraints. In order to reduce this idle time the transport time has to be reduced, or else another way of transporting has to be implemented. No attention has been paid to this problem.

The performance graphs of the milling single shop are printed in Figure 4.10. The lead time of a job is always 70 minutes or more. If the shop always contains exactly one job the maximum throughput of one piece of material per 70 minutes is reached. If the mean inventory level becomes larger the lead time increases by 70 minutes per piece of material. It is clear that an inventory level of exactly one is the best work point for the controller. If the time between the sending of a request and the receipt of a job varies, a smaller delay may be chosen to be sure the job has arrived before the machine runs idle. This smaller delay results in an inventory level that becomes larger than one and a lead time that becomes larger than 70 minutes. If the delay is too big the lead time remains 70 minutes but the throughput of 1 piece of material per 70 minutes is not reached.

The consequences of variable process times are restricted to the performance graph and the length of the delay. The delay is equal to the process time minus a safety margin. If the process time is known, the delay can be calculated, and material is requested in time. If the process time is not exactly known an estimation has to be made, and the

performance will depend on the estimation. The controller behaviour, in principle, remains the same and the lead time does not necessarily have to be larger than the (deterministic) process time of the jobs together with the transport time.

Two ways to generate requests have been discussed above. One was based on the use of subrequests, the other on the use of a delay, which started at the sending of a subjob to a resource. In both request mechanisms the resource influences the controller in requesting new jobs. The lead time of a job for a single shop is known exactly if the behaviour of the resource in the shop is known. If process times are random, then lead time has to be weighed against throughput. A high inventory level reduces the chance that the resource becomes idle at the cost of an increase in lead time. The controller of the single shop tries to keep the inventory level at exactly the batch size of the resource, because this minimizes the lead time in the single shop.

4.4 Parallel shop

The parallel shop is in many ways similar to the single shop. The parallel shop presented here is a parallel milling shop which consists of a controller and resources (Figure 4.11). The resources consist of a store, a transporter and processing resources. This time, the processing resources are two equivalent milling machines. These milling machines have the same characteristics as in the single shop.

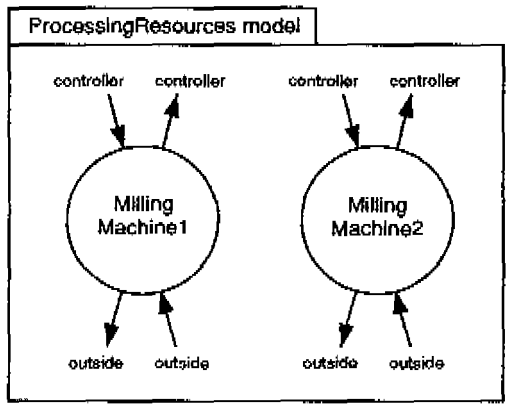


Figure 4.11. *Model of the processing resources of the parallel milling shop. The controller, store and transporter are modelled according to Figure 4.9 a and b.*

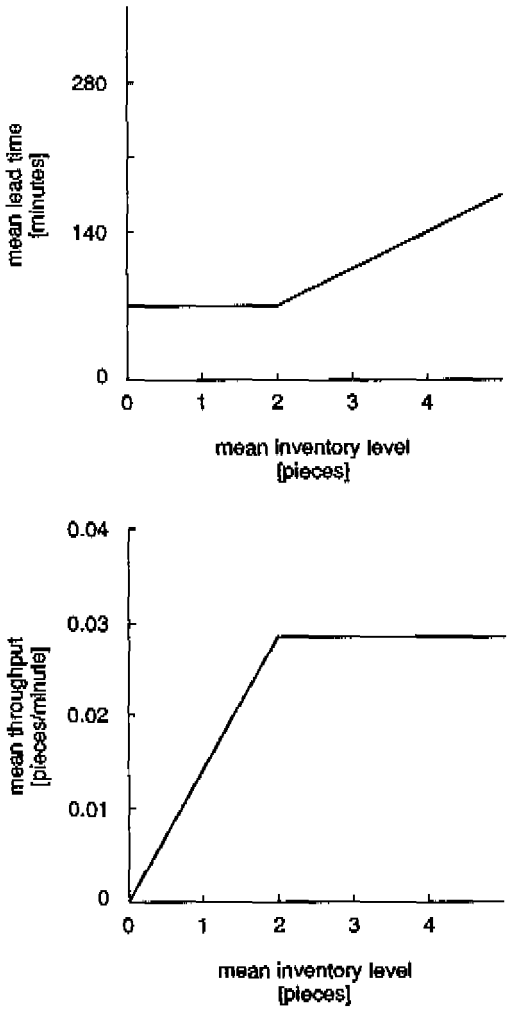


Figure 4.12. *Performance graphs of the parallel milling shop.*

The most important characteristic of the parallel shop is the fact that all manufacturing processes in the shop consist of one processing operation, performed on one processing resource. The manufacturing process consists in this case of transport from store to one of the milling machines, processing of the material on the milling machine, and transport from the milling machine to the store. The recipe table of the shop is equivalent to that of the single milling shop. The recipe table of the shop is printed below.

```
recipes:
    millProductA1 -> Sequence (mill1)
    millProductB1 -> Sequence (mill2)
```

The controller of the shop is able to process two pieces of material in two different resources; these resources are in fact independent capacities. So if one resource sends a subrequest, the controller sends a request that states the capabilities and capacities of that resource. If, in our case for example, one milling machine can only execute the mill1 operation and the other only the mill2 operation, a subrequest of the first machine results in a request from the controller for a job with a millProductA1 operation. The shop controller sends as many requests as it has capacity in its processing resources. This capacity of the processing resources is stated by the subrequests from these resources, so one subrequest is equal to one request. Just like the single shop, the parallel shop may also use a delay before sending a request in order to reduce the idle times of the machines. The sending of a subjob is a reasonable reference point for the start of the delay.

The performance graphs of the parallel milling shop are printed in Figure 4.12. The mean lead time remains 70 minutes, as long as the inventory level remains below two pieces of material. With a mean material level above the two pieces the mean lead time increases by 35 minutes per piece of material. The mean throughput of the shop is at a maximum if the inventory level is two pieces or more. The maximum throughput is equal to one piece of material every 35 minutes.

The parallel milling shop has the transporter as a common resource. If two machines receive a job at the same moment one job has to wait for the transport of the material of the other job. To prevent this it seems logical not to request two jobs at the same time, but rather to force a delay between two requests equal to the time it takes to load the material of a job.

The request generation in a parallel shop is analogous to the generation of requests in a single shop. A possible refinement to the request generation mechanism is the introduction of a minimum time interval between the sending of two requests. This time interval is equal to the load time of material and prevents wait times due to transport in the shop.

4.5 Flow shop

Two working methods can be followed to model a flow shop. The flow line can be built by coupling a number of factories with a single processing resource behind one another. This is in fact the factory control architecture proposed by Arentsen [Arentsen, 1989]. Because Arentsen's factory control architecture is compatible with the architecture presented in this thesis, it follows that Arentsen's theory is also valid for this architecture. The disadvantages of the use of the architecture in

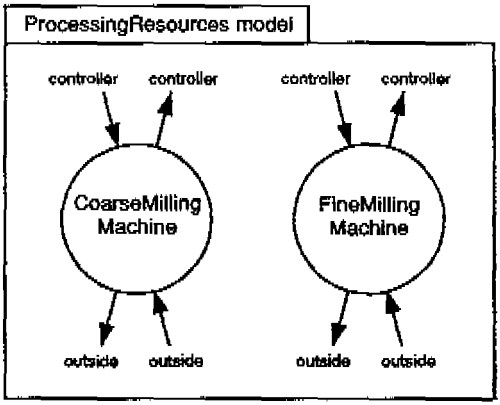


Figure 4.13. *Model of the processing resources of the milling flow shop. The controller, store and transporter are modelled according to Figure 4.9 a and b.*

this way is that, with a change of the manufacturing process, the control structure of the manufacturing system has to be changed and that every resource is involved in the capacity planning.

The other way to model the flow shop is with the help of multiple processing resources. It is modelled as a manufacturing system with a central controller; all the recipes consist of a number of operations, which are executed in sequence on the resources. This method is illustrated below.

The milling flow shop has two milling machines: a coarse milling machine and a fine milling machine. The model of the shop is found in Figure 4.13. The milling parallel shop has similar structure as the milling single shop and the milling parallel shop. The resources of the shop are a store, a transporter and processing resources. The processing resources are formed by a coarse milling machine and a fine milling machine.

The recipes of the flow shop differ from the single and parallel shop. The manufacturing of a product is now done by executing two processing operations. The recipe is shown in the recipe table below.

```
recipes:
    millProductA1 -> Sequence (coarseMill1 fineMill1)
    millProductB1 -> Sequence (coarseMill2 fineMill2)
```

The manufacturing process consists of transporting material from store to the coarse milling machine, the coarse milling of material, transport from the coarse milling machine to the fine milling machine, fine

milling of material, and transport from the fine milling machine to the store. It is supposed that both the coarse and the fine milling take 30 minutes, the transport of a piece of material takes 5 minutes. The transporter is modelled as one common resource which executes all transport in the station. This is not a necessity: other solutions, e.g. a distributed transport system, are also possible.

The request generation of a flow shop differs from that of the single and parallel shop. Not every subrequest leads to a request for the supercontroller. This is not possible because one job of the supercontroller is executed on two resources. There are two subrequests needed for the execution of one job. So only half as many requests have to be generated as there are subrequests received. One way of generating requests is to use subrequests of the first resource in the manufacturing process, in this case the coarse milling machine. The flow shop will not request more than one job at a time because the first milling machine can only process one piece of material at a time. Just as in the single shop a delay may be used to request a new job in order to be sure the material arrives in time at the flow shop. For the start time of the delay the sending of a job to the first resource is a good reference.

If the first machine is not a bottleneck, this manner of request generation does not work: it leads to an everlasting increase of inventory in the flow shop, because the input rate of jobs becomes bigger than the throughput of the flow shop. The start of a subjob at the bottleneck station is a reference point, which circumvents this problem. Now the length of the delay has to be adapted in such a way that jobs arrive in time at the bottleneck resource. This time depends on the jobs that are in process on the resources in front of the bottleneck and on the process time of these jobs. The maximum time interval between two requests is equal to the inverse of the maximum throughput (in this example 40 minutes, see below). This is a maximum length, which has to be used if the inventory level of the flow shop is equal to the desired work point. In some cases (e.g. with the start up of the shop) it makes sense to increase the inventory level of the flow shop. In order to do this the length of the time interval has to be smaller than the inverse of the maximum throughput. To make the time interval smaller than the process time of the subjob on the first resource has little use, because it would introduce a wait queue in front of the first resource.

By using the start time of a subjob on the bottleneck resource for generating a new request, establishes a link between the behaviour of the resources and the generation of requests (closed loop). If, for instance, a resource in front of the bottleneck goes down, the bottleneck resource does not receive a new subjob. If subjobs arrive too early at the bottleneck, they have to wait until the bottleneck is idle. In both cases

the bottleneck will not generate new subrequests and, because of this, the controller will not send any new requests.

Another way of generating requests is to start by generating so many requests that the flow shop becomes filled to the work point. From then on the finishing of a job can be used to generate a new request. This leads to the Fixed-WIP release strategy (see Section 3.2). In the milling flow shop this means two requests are sent at the beginning and afterwards, after the finishing of a job, a new request is sent. Another possible request generating mechanism is the use of a constant time interval. The interval should be equal to the inverse of the desired throughput. This leads to the so-called uniform-starts release strategy, where every so many minutes a new job is started in a manufacturing system. In our example a request has to be generated every 40 minutes. This mechanism is an open-loop policy and bears the disadvantage that no coupling exists between the execution of subjobs on the resources and the sending of requests.

The request of a flow shop normally contains all operation types that the flow shop is capable of executing. A fixed product mix may be realized by alternatingly sending requests with different capabilities.

To calculate the performance graph of the milling flow shop a possible optimal behaviour of the shop is given. This is the behaviour of a filled shop and repeats every 40 minutes, although the operations on both the coarse and the fine milling machine take only 30 minutes. At time 0 the fine milling machine has finished a product and the transporter transports the finished piece of material from the fine milling machine to the store. At time 5 the coarse milling machine finishes its operation and the transporter transports the piece of material from the coarse milling machine to the fine milling machine. At time 10 the material arrives at the fine milling machine and the fine milling is started. The fine milling finishes at time 40, the start of the new cycle. At time 10 the transporter also transports a new piece of material from the store to the coarse milling machine. At time 15 the material arrives at the coarse milling machine and coarse milling is started. The coarse milling will be finished at time 45, just in time for the material to be transported to the fine milling machine.

The capacity of the machines is one product every 30 minutes, the extra time being due to the transport of products. Once again, this can be reduced with another implementation of the transport system.

The minimum lead time of a job is equal to 75 minutes, and the maximum throughput is equal to one piece of material every 40 minutes. The flow shop inventory level, however, is not two pieces of material all the time. During a period of five minutes, when the material from the

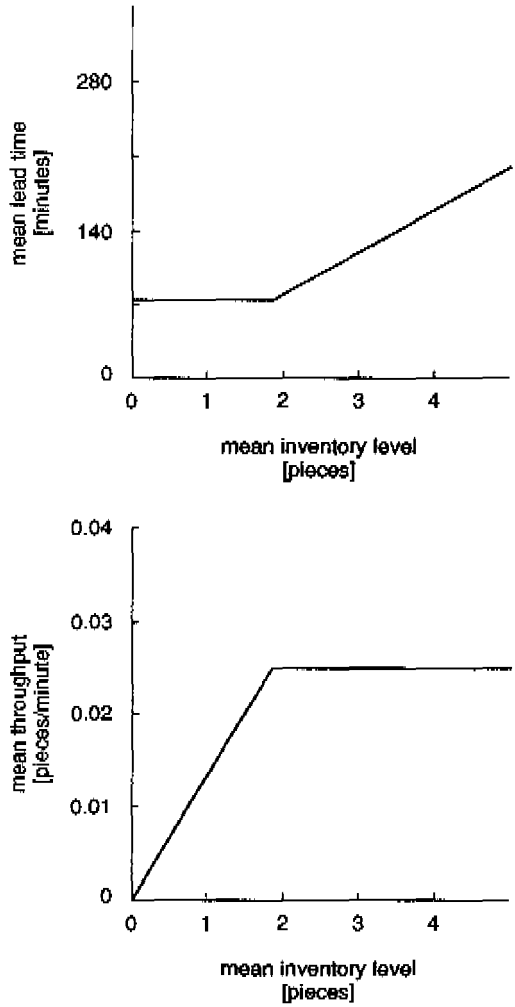


Figure 4.14. *Performance graphs of the milling flow shop.*

coarse milling machine is transported to the fine milling machine, there is only one piece of material in the shop. The mean material contents at the ideal work point is therefore 1.875. The performance graph of the flow shop is given in Figure 4.14. An inventory level above the ideal work point causes the lead time to increase by 40 minutes per piece of material.

The example above mentions new mechanisms for the generation of request. The subrequest from the bottleneck resource may be used or the sending of a subjob to the bottleneck may be a reference for a delayed request. Then there is the possibility to use the finishing of a job to send

a request (Fixed-WIP). The last mechanism mentioned uses a fixed time interval between the sending of requests (uniform starts).

4.6 Job shop

The most complex manufacturing system is a job shop. The job shop is characterized by great route flexibility and universal resources. In a job shop there are no constraints to the route of the material through the shop and there is no limit to the number of times a resource is visited, thus cyclic routes are allowed.

The propagation of requests from resources through the manufacturing controller to the supercontroller in a job shop is difficult. Because the recipes consist of more than one operation, it is not possible to send a request for every subrequest received from the resources. There is also not necessarily one bottleneck that is always visited and that can be used as a trigger for the requesting of new jobs from the supercontroller. A way to request new jobs in a job shop is the use of the inventory level in the manufacturing system. This leads to the generation of requests when a job finishes, and a filling mechanism for loading the shop (Fixed-WIP, as discussed in Section 4.5). The bigger the job shop the smaller the influence of one job on the behaviour of the manufacturing system, and the better general observers, such as inventory level, can be used as a request generating mechanism. The work point is related to the sum of the batch sizes of the machines in the manufacturing system. This mechanism only gives satisfactory results if the execution of the jobs is distributed smoothly over all resources in the manufacturing system.

The mechanism mentioned above does not function well if the work contents of the different jobs show large variations. In this case the use of work content may work better. The work content is the sum of the process times of all the jobs in the system. Manufacturing control with the use of work content is discussed in [Wiendahl 1987]. A disadvantage of the use of work content is the difficulty of finding a relation between the batch size of the machines, which is expressed in material units, and the optimal work content of the manufacturing system, which has to be expressed in hours. Here also the jobs have to be distributed evenly over the total manufacturing system.

An even more loosely coupled request mechanism is the use of a fixed time interval between the sending of two requests (uniform starts). If, for instance, the manufacturing system has been designed to manufacture a product every two hours, then the controller may send a request every two hours. A disadvantage of this open-loop mechanism is the lack of feedback between the actual capacity of the manufacturing system and the loading request of the controller.

The time interval between two requests also has to be related to the input rate. It is no use to request two jobs at the same time or shortly after each other, if there is no capacity available to process these jobs at that moment. The example of a job shop is presented in the next chapter, so no further example is given here.

The single shop and the parallel shop are able to guarantee a certain lead time, when they request a job. The flow shop also, under certain conditions, guarantees a deterministic lead time. For the job shop, however, the lead time tends to vary. By operating the job shop with a controlled inventory level (Fixed-WIP) at a work point, this variation in lead time is often kept within limits.

The resources of a job shop should not be leaf resources in order to prevent deadlock and long idle times. On the other hand resources with a job shop structure in a job shop seems also unwise. This increases uncertainty and lead times, with bad throughput figures. It is best if all job shop decisions are taken at one level, as high in the hierarchy as possible, so that one controller has a view of the total scheduling problem and is able to keep the total system at a work point which is determined in advance.

4.7 Configuring a hierarchical control system

The integration of the concepts treated above results in the new hierarchical control architecture. This architecture allows a structured approach to the design of manufacturing control systems, especially for manufacturing systems having a job shop character.

Four manufacturing system classes have been presented in Chapter Two: a single shop, a parallel shop, a flow shop, and a job shop. In Chapter Three four controller categories have been described: a factory controller controlling expanded resources, one controlling leaf resources, a manufacturing controller controlling expanded resources, and one controlling leaf resources.

This chapter has discussed the data structure, the general control model and the strategies for the generation of requests. The following six request generation strategies have been described: a subrequest causes a new request, a subrequest of the bottleneck causes a new request, start of the first subjob of a task causes a new request after a time interval or delay, start of a subjob on a bottleneck causes a new request after a time interval or delay, the completion of a job causes a new request (Fixed-WIP), or a new request is sent at fixed time interval (uniform-starts). In the last four sections the relation between request generation and the manufacturing system class has been discussed.

The controller of the described architecture controls processing resources, a transporter and a store. It has been shown that the processing resource may be expanded, itself containing a controller, processing resources, a transporter and a store. In this way a hierarchical control system is built up.

A controller belongs to a certain category, and a manufacturing system class has to be determined. A request generation strategy and a transport strategy have to be chosen for every controller. These aspects are related. The number of control layers is adjustable. The top of the hierarchy is formed by a factory controller. The bottom is formed by (leaf) processing resources controlled by a manufacturing controller controlling leaf resources. If there is only one layer, then the controller is a factory controller controlling leaf resources.

A controller that controls leaf resources has to use late transport, because a leaf resource has no way to store material that is not being processed. In order to avoid deadlock and to reduce idle times the single shop, the parallel shop and, perhaps, a well balanced flow shop are the most obvious classes to use for a manufacturing system with leaf resources. These classes are associated with few sequencing problems: a scheduler using the FIFO sequencing rule is normally sufficient. In case of a job shop extra attention will have to be paid to deadlock avoidance.

The factory controller has to take care of capacity planning. To do this it needs a clear view of the behaviour of the resources it controls. In order to achieve such insight, one should use a relatively simple class of manufacturing system. The most obvious classes here are also the single shop, the parallel shop, or perhaps a flow shop, if it is well balanced. A job shop manufacturing system class on this level would introduce a lot of complications. On this level, one tries to keep the scheduling problems to a minimum: calculations are concentrated on the capacity planning problem. A necessary condition is that the behaviour of the resources in the factory behave reasonably predictably. A simple scheduler using a FIFO sequencing rule or a rule for due date control (e.g. EDD) is used on the factory level.

In controllers of expanded resources, early transport reduces the idle times of resources and increases the clarity of the manufacturing system. But it reduces the allocating possibilities at an early stage because material is allocated to a machine before the material is transported. Requests sent after a time interval may be used to request material before a job is finished, to prevent the idleness due to late transport. The job is requested too early, where the interval between request and actual finishing of the job is a safety margin. This margin is, if possible, only used by the controller that controls leaf resources. The margins used by this controller may settle transport times in more than one supercontrol

layer. The transport times of more layers can only be incorporated in one margin if the supercontrollers generate requests on the basis of subrequests. If in one of the supercontrollers requests are not generated with help of subrequests (e.g. uniform starts or Fixed-WIP), only the transport delays up to this superlayer are accounted for in the margin. Here delays in layers above this superlayer can not be accounted for. This problem may arise in controllers of a flow shop or a job shop class.

If a controller controls expanded resources and there are no duplicate resources of one type, then early transport is an interesting strategy to use. A controller of a parallel shop class may be inserted in order to prevent duplicate resources . A problem that has to be solved with any transport strategy is the level at which material in the factory is stored. In the manufacturing controller the material is considered to form the inventory of the controller only if a job for the material has been received. Normally the material is stored in the resources of the controller where the sequencing and allocating decisions have to be taken. This sequencing and allocating has to be done in a layer where a global survey of the necessary system information is available. This is usually one of the highest layers. The lower layers use late transport and are of a simple manufacturing system class in order to keep the internal inventory small and to achieve deterministic lead times, which enables sequencing and allocating decisions to be made on the basis of relatively certain data. This layer, where sequencing and allocating problems are solved, may be of a job shop or an (unbalanced) flow shop class.

The control architecture which has been presented in this chapter takes as its starting point the specification of the machines and the recipes. Together with the specification, one may also give a rough sketch of the controller configuration and the number of the hierarchical layers. The control architecture applied to this information results in the controller configuration. The control architecture assumes that the transport capacity of the transport system is over-dimensioned. The control configuration, and especially the transport systems and the stores, have to be adapted to each other. The designer of the physical manufacturing system has to determine the transport systems and the stores in concert with the designer of the control system. The control configuration is created on the basis of the insights presented above. This leads to a specific model. The use of simulation allows one to check the behaviour of the control system. The simulation also generates performance graphs, which may be used to determine a suitable work point for the factory.

This approach is used in Chapter Five. With help of the control architecture a hierarchical control system is developed for a complex IC factory with a job shop character. The unique properties of the hierarchical control architecture are discussed in Chapter Six.

Chapter 5
A case: an IC manufacturing system

5.1 Introduction

The architecture presented in the previous chapters will now be applied to an example, in order to illustrate the architecture in practice. In this chapter an Integrated Circuit (IC) manufacturing system is modelled and the performance of the system is studied with help of simulation experiments. The diffusion process is used in the factory to produce wafers containing ICs. The factory applies the CMOS (Complementary Metal Oxide Semiconductor) technology [Sze 1983].

The manufacturing of ICs is difficult for many reasons. The process consists of hundreds of steps (manufacturing operations). Many process steps have to be performed on the same machine. The durations of the process steps vary from less than one hour to almost one day. The factory has a job shop character. The machines have a poor reliability and the process is subject to random yield crashes. Operator availability and unpredictable repair times further complicate the manufacturing process [Lozinsky, Glassey, 1988]. The nominal time required to manufacture an IC can be up to several weeks and, in practice, average lead times of more than six times the nominal process time are no exception [Miller, 1990].

IC manufacturers have put a great deal of effort into the development of new technologies and the improvement of the manufacturing process, in order to increase the scale of integration of ICs. Nowadays, they have started to realize that, in order to reduce the cost, attention has also to be paid to the control of the manufacturing system.

In the future factories will contain cell orientated manufacturing systems and automated material handling systems, in order to reduce the number of processing steps and to exclude contact between operator and processing material. Computers will control the material flow and the manufacturing process to reduce lead times [Warnecke, 1990].

5.2 The IC manufacturing system

The IC manufacturing process to be dealt with is described briefly below. More extensive descriptions can be found in the literature [Sze 1983, Burman et al. 1986, Kessler 1988].

The manufacturing of an IC is split into four parts: the manufacturing of the raw wafers; the wafer fabrication; a probe and dice operation; and packaging and testing. The first two parts are also called the front end, the last two the back end. To produce raw wafers, molten silicon is transformed into crystalline ingots which are sawn into wafers. The wafer fabrication is the part with which this chapter deals, and is explained in brief below. The wafer fabrication is done in an IC manufacturing system, often called a wafer fab. After the wafer fabrication the ICs on the wafer are tested, and sawn into individual chips. The approved chips are encapsulated into packages and tested again before shipping.

The basic building block of an CMOS IC is the MOS transistor. An IC contains up to millions of these transistors. The way to make an IC is by depositing or growing layers of material on the wafer. With help of a photolithographic process a pattern is applied on this layer and with help of an etch process the layer is partially removed, the photoresist being removed afterwards. By repeating these steps several layers are grown on the wafer. Another process used is ion implantation, where ions are shot into the wafer and introduce so-called dopants into the silicon. These dopants diffuse through the silicon by heating the wafer. In order to manufacture an IC hundreds of process steps have to be executed. Many of these steps are executed on the same resources, which makes the process cyclic, i.e. it is a job shop.

The machines of a wafer fab are usually divided into five categories: lithographic equipment, diffusion-CVD (Chemical Vapour Deposition) equipment, etch equipment, implantation equipment and metalization equipment. A step in the manufacturing process is usually preceded by a clean step and followed by an inspection step. Etch steps are not preceded by a clean step, and implantation steps are neither preceded by a clean step nor followed by an inspection step.

The lithographic machines put patterns on the wafer. The wafer is first coated with a light-sensitive photoresist. The stepper exposes the wafers, which are then developed. The coater, stepper and developer are usually integrated into one machine.

The diffusion-CVD machines grow or deposit layers of material on the wafer. This happens by putting the wafers in a furnace, heating it and

leading gases through the furnace. Because of the long setup times furnaces are used for only a limited number of process steps.

The etchers remove material from the wafer. The developed photoresist protects a part of the wafer and the uncovered areas are removed. This is usually done with a dry etch process. An etcher is also used to strip the photoresist from the wafer.

In the implanter charged ions of the right dopant are fired at the wafer. This is a low temperature process. After the implantation the surface of the wafer may be damaged. This damage is healed by heating the wafer for a short while, so-called thermal annealing. To diffuse the implanted ions into a larger doped region the wafers are heated in a furnace (one of the diffusion-CVD machines).

The metalization machines deposit metal on the wafer. The process metalization or sputtering is done to connect the components of the IC with each other and to provide bonding pads, were the IC is connected to the outside world (pins of the encapsulation).

Besides to the process machines (steppers, furnaces, etchers, implanters and sputterers), the chip fab also contains cleaners, inspectors, stores and transporters. The material in the factory consists of cassettes with wafers. These cassettes usually contain 25 wafers. The machines in the IC factory have different batch sizes, some process single wafers such as the steppers. Furnaces process up to two, three or four cassettes at one time.

5.3 Control of IC manufacturing systems

The control of IC manufacturing systems is the subject of much research. Several control, releasing and sequencing strategies have been described and tested with help of simulation. Work load regulation [Wein 1988, Lawton et al. 1990] uses the work load in front of the bottleneck to decide when to introduce a new job in the manufacturing system. Starvation avoidance [Glassey and Resende 1988, Lozinski and Glassey 1988] tries to start new jobs as late as possible, in order to let material arrive at the bottleneck station just before the bottleneck runs idle and to minimize work in progress levels. An important conclusion of such studies is that scheduling influences the average lead time significantly, where larger improvements are obtained with job release than with subjob sequencing. Flow rate control [Kager and Lou 1989, Lou and Kager 1989] calculates loading rates at each job step in the wafer manufacturing system by comparing inventory levels and surplus levels with predetermined values. The shifting bottleneck approach [Uzsoy et al. 1989] approximates the general job-shop problem by

Table 5.1. *Manufacturing process of CMOS (process time 207 hours).*

number	operation	parameter	machine type	process time [hours]
1	IntlDiffusion	CMOS1	DfIntl	6
2	Mask	CMOS2	Stepper	1
3	CF4Etch	CMOS3	EtchCF4	1
4	HCurrImplant	CMOS4	HCurr	1
5	O2Etch	CMOS5	EtchO2	1
6	DrivDiffusion	CMOS6	DfDriv	15
7	CF4Etch	CMOS7	EtchCF4	1
8	Mask	CMOS8	Stepper	1
9	MCurrImplant	CMOS9	MCurr	1
10	O2Etch	CMOS10	EtchO2	1
11	DryDiffusion	CMOS11	DfDry	4
12	NtrdDeposition	CMOS12	LPNtrd	2
13	Mask	CMOS13	Stepper	1
14	CF4Etch	CMOS14	EtchCF4	1
15	O2Etch	CMOS15	EtchO2	1
16	MCurrImplant	CMOS16	MCurr	1
17	WetDiffusion	CMOS17	DfWet	12
18	CF4Etch	CMOS18	EtchCF4	1
19	GateDiffusion	CMOS19	DfGate	4
20	DPolDeposition	CMOS20	LPDPol	3
21	Mask	CMOS21	Stepper	1
22	Cl2Etch	CMOS22	EtchCl2	1
23	O2Etch	CMOS23	EtchO2	1
24	Mask	CMOS24	Stepper	1
25	MCurrImplant	CMOS25	MCurr	1
26	O2Etch	CMOS26	EtchO2	1
27	TEOSDeposition	CMOS27	LPTEOS	2
28	CF4Etch	CMOS28	EtchCF4	1
29	TEOSDeposition	CMOS29	LPTEOS	2
30	Mask	CMOS30	Stepper	1
31	MCurrImplant	CMOS31	MCurr	1
32	O2Etch	CMOS32	EtchO2	1
33	Anneal	CMOS33	DfAnnl	4
34	Mask	CMOS34	Stepper	1
35	MCurrImplant	CMOS35	MCurr	1
36	O2Etch	CMOS36	EtchO2	1
37	Anneal	CMOS37	DfAnnl	3
38	LTODeposition	CMOS38	LPLTO	3
39	Mask	CMOS39	Stepper	1
40	CF4Etch	CMOS40	EtchCF4	1
41	O2Etch	CMOS41	EtchO2	1
42	AlDeposition	CMOS42	LPAl	2
43	Mask	CMOS43	Stepper	1
44	BCl3Etch	CMOS44	EtchBCl3	1
45	O2Etch	CMOS45	EtchO2	1
46	OxIdDeposition	CMOS46	PEOxid	3
47	ResistDeposition	CMOS47	Coater	1
48	CF4O2Etch	CMOS48	EtchCF4	1
49	OxidDeposition	CMOS49	PEOxid	2
50	Mask	CMOS50	Stepper	1
51	CF4Etch	CMOS51	EtchCF4	1
52	O2Etch	CMOS52	EtchO2	1
53	AlDeposition	CMOS53	LPAl	2
54	Mask	CMOS54	Stepper	1
55	BCl3Etch	CMOS55	EtchBCl3	1
56	O2Etch	CMOS56	EtchO2	1
57	OxidDeposition	CMOS57	PEOxid	3
58	Mask	CMOS58	Stepper	1
59	CF4Etch	CMOS59	EtchCF4	1
60	O2Etch	CMOS60	EtchO2	1

Table 5.2. *Manufacturing process of SRAM (process time 197 hours, this process time is inclusive cleaning, inspecting and transporting).*

number	operation	parameter	machine type	process time [hours]
1	IntlDiffusion	CMOS1	DfIntl	6
2	Mask	SRAM2	Stepper	1
3	CF4Etch	CMOS3	EtchCF4	1
4	HCurrImplant	CMOS4	HCurr	1
5	O2Etch	CMOS5	EtchO2	1
6	DrivDiffusion	CMOS6	DfDriv	15
7	CF4Etch	CMOS7	EtchCF4	1
8	Mask	SRAM8	Stepper	1
9	MCurrImplant	CMOS9	MCurr	1
10	O2Etch	CMOS10	EtchO2	1
11	DryDiffusion	CMOS11	DfDry	4
12	NtrdDeposition	CMOS12	LPNtrd	2
13	Mask	SRAM13	Stepper	1
14	CF4Etch	CMOS14	EtchCF4	1
15	O2Etch	CMOS15	EtchO2	1
16	MCurrImplant	CMOS16	MCurr	1
17	WetDiffusion	CMOS17	DfWet	12
18	CF4Etch	CMOS18	EtchCF4	1
19	GateDiffusion	CMOS19	DfGate	4
20	MCurrImplant	SRAM20	MCurr	1
21	DPolDeposition	SRAM21	LPDPol	3
22	TaSi2Deposition	SRAM22	SpTaSi2	1
23	Mask	SRAM23	Stepper	1
24	Cl2Etch	SRAM24	EtchCl2	1
25	O2Etch	SRAM25	EtchO2	1
26	TEOSDeposition	SRAM26	LPTEOS	2
27	CF4Etch	SRAM27	EtchCF4	1
28	Mask	SRAM28	Stepper	1
29	MCurrImplant	SRAM29	MCurr	1
30	O2Etch	SRAM30	EtchO2	1
31	Mask	SRAM31	Stepper	1
32	MCurrImplant	SRAM32	MCurr	1
33	O2Etch	SRAM33	EtchO2	1
34	TEOSDeposition	SRAM34	LPTEOS	2
35	Mask	SRAM35	Stepper	1
36	CF4Etch	SRAM36	EtchCF4	1
37	O2Etch	SRAM37	EtchO2	1
38	PolyDeposition	SRAM38	LPPoly	2
39	Mask	SRAM39	Stepper	1
40	Cl2Etch	SRAM40	EtchCl2	1
41	O2Etch	SRAM41	EtchO2	1
42	NtrdDeposition	SRAM42	LPNtrd	2
43	Mask	SRAM43	Stepper	1
44	CF4Etch	SRAM44	EtchCF4	1
45	O2Etch	SRAM45	EtchO2	1
46	HCurrImplant	SRAM46	HCurr	1
47	PSGDeposition	SRAM47	LPPSG	3
48	FlowDiffusion	SRAM48	DfFlow	3
49	Mask	SRAM49	Stepper	1
50	CF4Etch	SRAM50	EtchCF4	1
51	O2Etch	SRAM51	EtchO2	1
52	AlDeposition	SRAM52	LPAl	2
53	Mask	SRAM53	Stepper	1
54	BCl3Etch	SRAM54	EtchBCl3	1
55	O2Etch	SRAM55	EtchO2	1
56	OxidDeposition	SRAM56	PEOxid	3
57	Mask	SRAM57	Stepper	1
58	CF4Etch	SRAM58	EtchCF4	1
59	O2Etch	SRAM59	EtchO2	1

solving a sequence of single machine problems and using a disjunctive graph representation to capture interactions between machines. Elleby et al. [1989] introduce a constraint-based framework with an adaptive mechanism to allow the operator to express interactively its criteria and with a least commitment approach to prevent superfluous scheduling activities. An application of the JIT manufacturing management philosophy in an etch shop is presented by Martin-Vega et al. [1988].

The simulation of IC manufacturing systems has received an increasing amount of attention in the literature [Miller 1990, Tullis et al. 1990, Denekamp et al. 1990, Matuyama and Atherton 1990, Burman et al. 1986, Atherton et al. 1989, Atherton 1988, Atherton 1987, Pollak 1989]. Simulation has been used to gain an insight into the behaviour of the manufacturing system, both performance and dynamic capacity having been analysed. Simulation results have also been used to take control decisions and to design future factories. Simulation programs are mostly used in combination with a material tracking system, which delivers input information for the simulation program. To validate the model, simulation output is compared with information from the tracking system. Attempts are being made to integrate output results with the tracking or control system in order to use simulation as a tool for scheduling and control [Tullis et al. 1990].

In all these studies, the most important performance measure of an IC manufacturing system is the lead time of jobs. This is the time between the start of a job in the wafer manufacturing system and the finishing of the job. The lead time influences the yield, inventory costs and time to market. Long lead times influence the yield negatively [Miller 1990]. Yield is the percentage of products manufactured, that fulfil the requirements. The lead time influences the time during which foreign particles have a chance to contaminate wafers. Variations in lead times and in times between processing steps cause process variability. Lead times determine learning time, and thus the time required to solve manufacturing problems. Long lead times go with high work in progress levels. With high inventories, the capital invested in the partially processed wafers is large, there is much space needed for storage and extra resources for product tracking. Long lead times result in long times before a new product comes on the market, which influences the competitive capability of the company.

On the other hand IC manufacturing systems are expensive and, to keep costs per wafer low, high throughputs are required. The relation between lead time, throughput and inventory is found in the performance chart, mentioned in Chapter three. The model of the IC factory is shown in the next section. The capability of the model is shown in the last section of this chapter, together with the calculation of some performance charts

for different configurations and different control, releasing and sequencing strategies.

5.4 The control model of an IC manufacturing system

The IC manufacturing system we model here is based on data taken from the literature, compiled by Denekamp [Denekamp 1989]. It is based on parts of an existing factory, so that the process has all the properties of a real IC manufacturing process and can probably be used for a real IC manufacturing system without much alteration.

The facility manufactures two product types, both being manufactured with CMOS technology. One is called CMOS, the other SRAM. Both processes are described in Tables 5.1 and 5.2. These tables only contain the processing steps, the cleaning steps and the inspection steps having been omitted.

The figures accompanying this chapter represent schematically the model of the important parts of the chip factory. The complete model contains 792 processors, 582 leaf processors and 210 expanded processors. The model will be presented in a bottom up manner. The

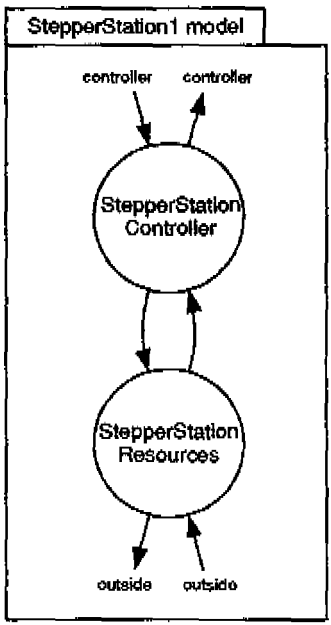


Figure 5.1. *A station model. a) StepperStation1 model.*

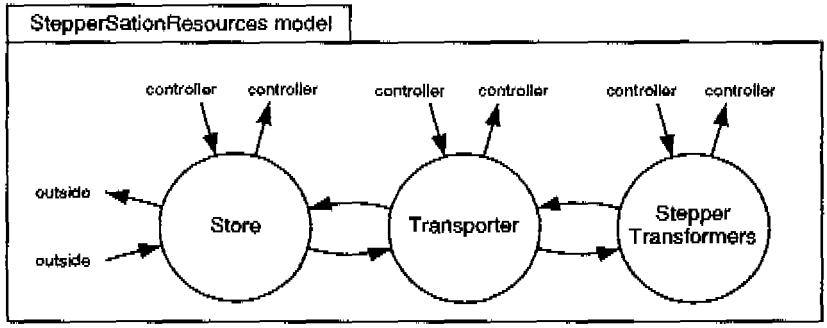


***Figure 5.1.** A station model. **b)** StepperStationResources model.*

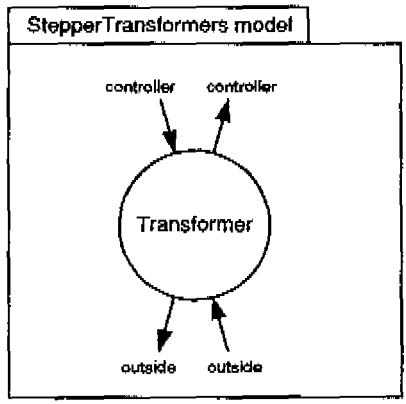


***Figure 5.1.** A station model. **c)** StepperTransformers model.*

hierarchical control layout of the factory has five control levels: station, cell, shop, facility and factory.

Station

The lowest level controls machines. These are the cleaners, the processing machines and the inspectors. A machine, together with a store and a transporter, forms a station. An example of a station (the StepperStation) is shown in Figure 5.1. A station is of a single shop class.

Cell

The stations are put together in a flow shop manufacturing system class, the cell. A cell is a line with a cleaning station at its head, then the processing station and finally the inspection station (Figure 5.2). The cells in the etch shop do not contain cleaning stations, and the cells in the implant shop contain no cleaning and no inspection station. A cell also contains a store and a transporter.

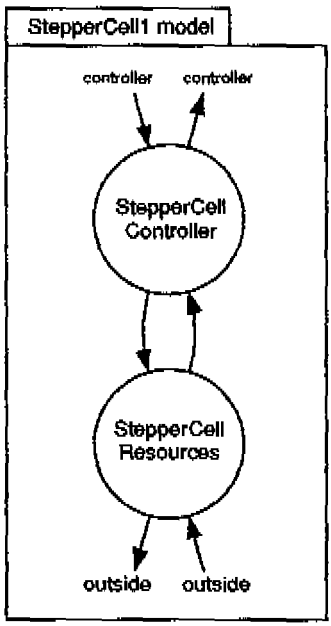


Figure 5.2. *A cell model.* ***a)*** *StepperCell1 model.*

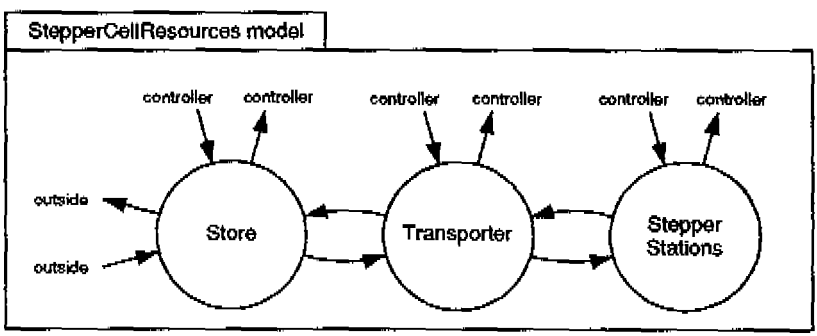


Figure 5.2. *A cell model.* ***b)*** *StepperCellResources model.*

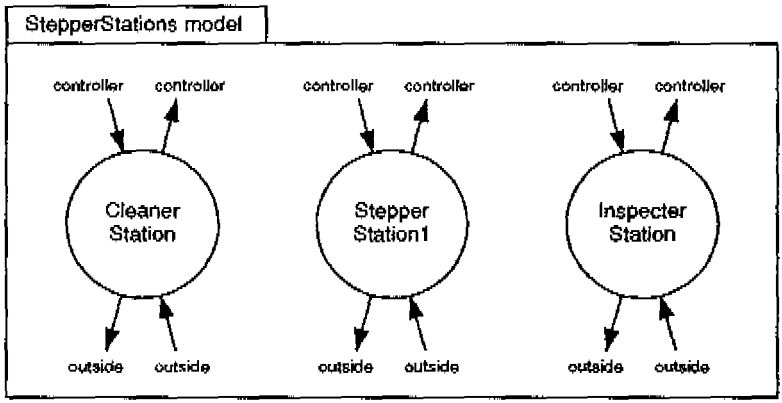


Figure 5.2. *A cell model.* ***c)*** *StepperStations model.*

Shop

The cells are grouped in a parallel shop manufacturing system class: shops (Figure 5.3). This means that, in the shop, material visits one cell and leaves the shop afterwards. Besides the cells, a shop contains, of course, one store and one transporter.

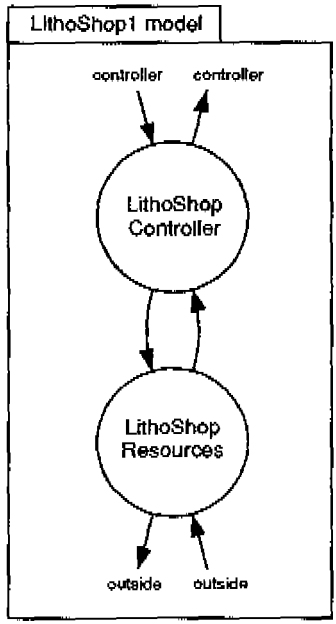


Figure 5.3. *A shop model.* ***a)*** *LithoShop1 model.*

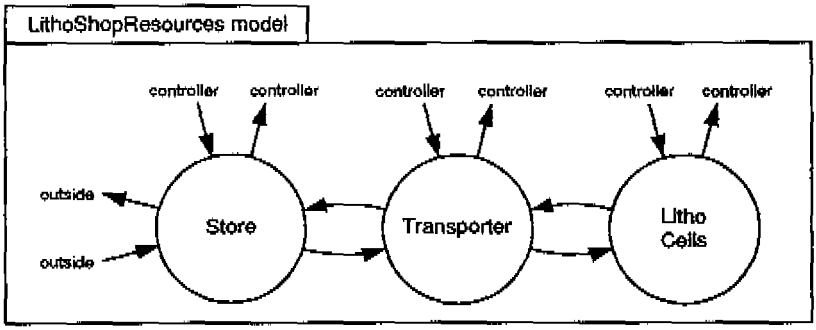


***Figure 5.3.** A shop model. **b)** LithoShopResources model.*

Facility

The facility is of a job shop class. The cells that use the same technology and the same chemicals are grouped together. This leads to five shops in the facility: litho-shop, diff-CVD-shop, etch-shop, implant-shop and metal-shop. These shops, together with a transporter and a store, form the entire facility (Figure 5.4).

Factory

The IC factory contains one facility, a store and a transporter (Figures 5.5). The factory has a single shop configuration.

From the control point of view we can say that the factory is a single shop, the facility is a job shop, the shop is a parallel shop, the cell is a flow shop, and the station is a single shop.

The material in the IC factory consists of cassettes of wafers. The material unit is one cassette containing 25 wafers. All machines in the factory have batch sizes of one or more cassettes. The maximum batch size is four cassettes. The process times of the material on the machines is supposed to be known and to be deterministic. The process times mentioned in Tables 5.1 and 5.2 refer to one batch size. The machines do not fail, the facility operates 24 hours a day and no (scheduled) maintenance takes place. Yield losses during manufacturing are in the first instance not considered. This means that, during the processing of a cassette, no wafers are reworked, no wafers are damaged and, after the execution of an order, the specified number of products are manufactured. Operator availability is also not included in the model: every operation on a machine starts at the moment the material arrives at the idle machine. Tools are also not considered and setup times are part of

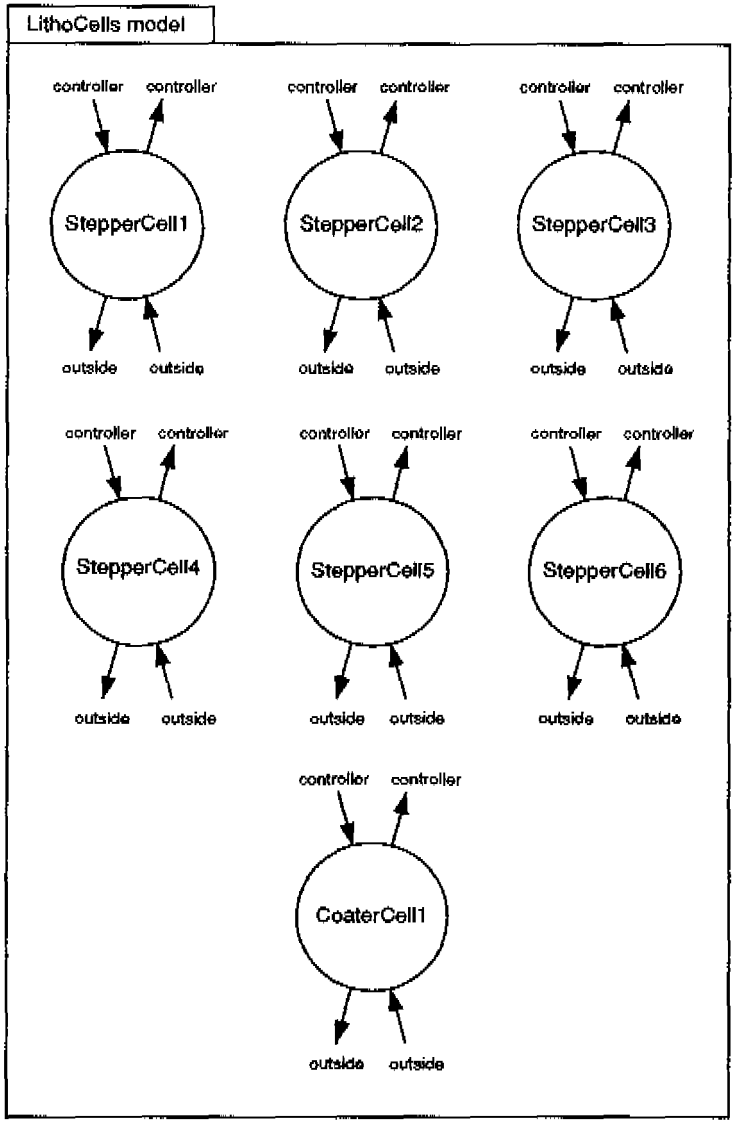


Figure 5.3. *A shop model. c) LithoCells model.*

the process time. Machine failure, yield loss and operator availability are left out of account, because the intention of this study is to gain an impression of how to configure the control of a complex factory, and how useful the described control architecture is. At this stage these factors only serve to confuse the issue. Afterwards, when the control theory of an ideal factory has been established, these factors will, of course, have to be taken into account.

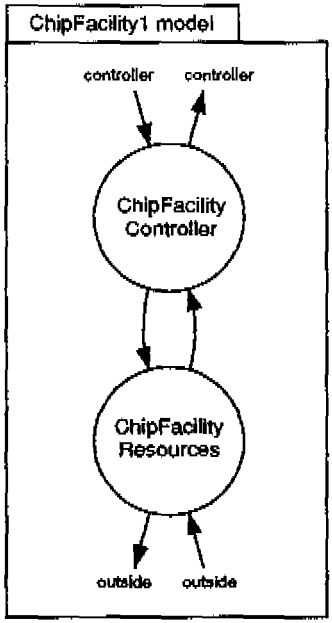


Figure 5.4. *A model of the facility.* ***a)*** *ChipFacility1 model.*

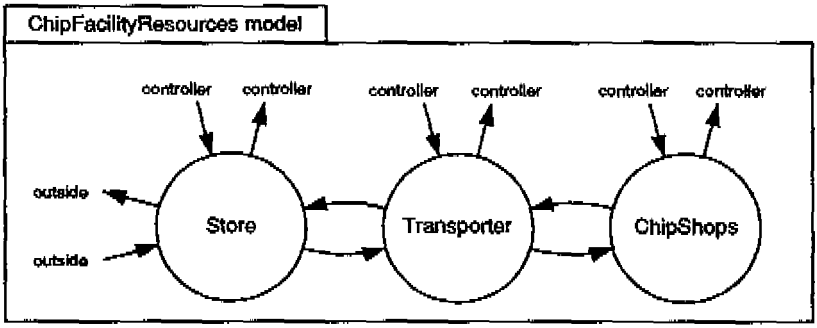


Figure 5.4. *A model of the facility.* ***b)*** *ChipFacilityResources model.*

The factory is to produce 0.5 cassettes per hour (= throughput). The load of the different processing machines can be calculated from Tables 5.1 and 5.2, as is explained below. The target throughput is used to calculate the necessary number of processing machines. With the number of machines and the load for the machine for both product types, the utilization of the machines is calculated for three product mixes. The first mix produces 0.5 cassette CMOS per hour and no SRAM (CMOS:SRAM = 1:0). The second mix produces 0.25 cassette CMOS and 0.25 cassette SRAM per hour (CMOS:SRAM = 1:1). In the third

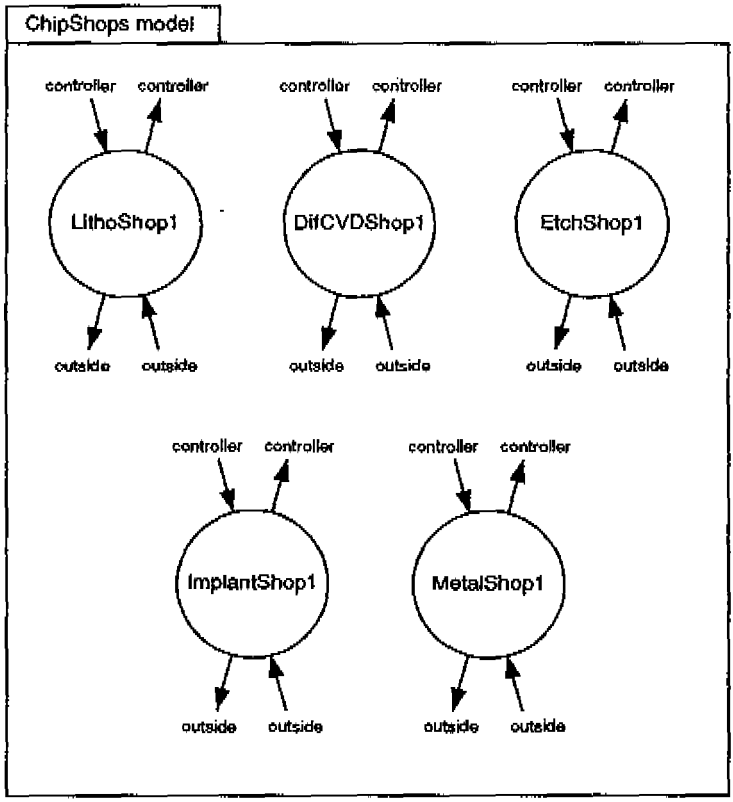


Figure 5.4. *A model of the facility. c) ChipShops model.*

situation 0.5 cassette SRAM and no CMOS is manufactured per hour (CMOS:SRAM = 0:1).

The loads for the different processing machines are represented in Table 5.3, the cleaners and inspectors of each cell being omitted. A cleaning step and an inspection step are supposed to last one hour. All cleaners and inspectors of a cell have a batch size that is equal to the processing machine in the cell, so the capacity of the cleaner and the inspector is the same as the capacity of the processing machine. Because the load is equal to or less then the load of the processing machine, the utilization of the cleaner and the inspector is also equal to or less than the utilization of the processing machine.

To calculate the values of Table 5.3 the following formulas are used:

$$l_{xy} = \sum_{z=1}^{O_{xy}^{n}} p_{xyz}$$

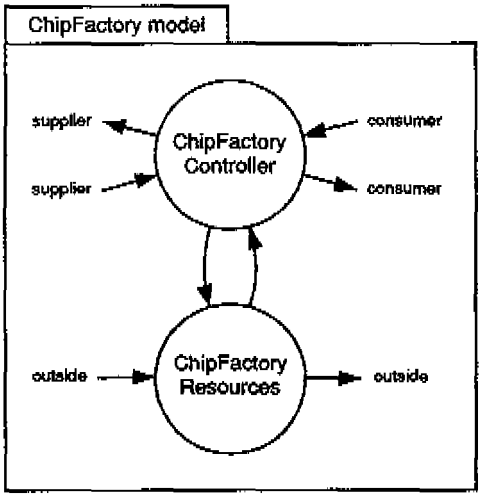


***Figure 5.5.** A model of the factory. **a)** ChipFactory model.*

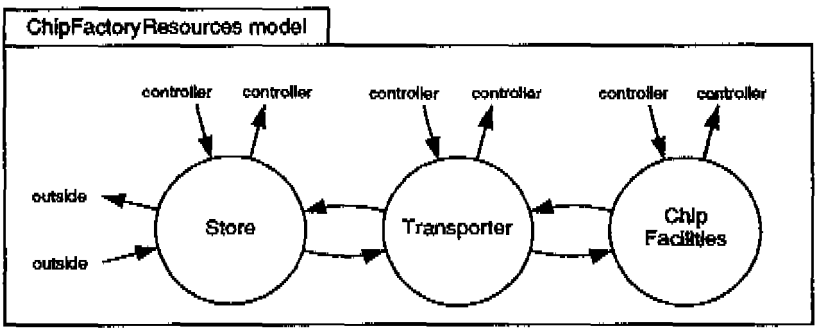


***Figure 5.5.** A model of the factory. **b)** ChipFactoryResources model.*

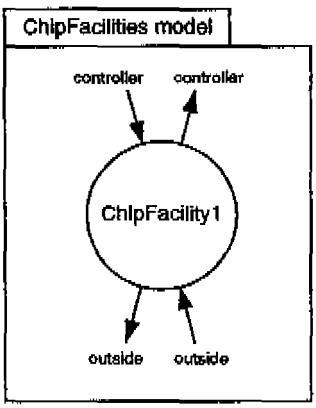


***Figure 5.5.** A model of the factory. **c)** ChipFacilities model.*

***Table 5.3.** Load and utilization of the different machine types for different product mixes (CMOS:SRAM) 1:0, 1:1 and 0:1.*

shop	machine type	number of machines	batch size	capacity	load		utilization		
					CMOS	SRAM	1:0	1:1	0:1
LithoShop	Stepper	6	1	6	12	12	1	1	1
"	Coater	1	1	1	1	0	0.5	0.25	0
DiffCVDShop	DfAnnl	1	4	4	7	0	0.88	0.44	0
"	DfDrive	2	4	8	15	15	0.94	0.94	0.94
"	DfDry	1	4	4	4	4	0.5	0.5	0.5
"	DfFlow	1	4	4	0	3	0	0.19	0.38
"	DfGate	1	4	4	4	4	0.5	0.5	0.5
"	DfIntl	1	4	4	6	6	0.75	0.75	0.75
"	DfWet	2	4	8	12	12	0.75	0.75	0.75
"	LPDPol	1	2	2	3	3	0.75	0.75	0.75
"	LPLTO	1	2	2	3	0	0.75	0.38	0
"	LPNtrd	1	2	2	2	4	0.5	0.75	1
"	LPPoly	1	2	2	0	2	0	0.25	0.5
"	LPPSG	1	2	2	0	3	0	0.38	0.75
"	LPTEOS	1	2	2	4	4	1	1	1
"	PEOxid	2	3	6	8	3	0.67	0.46	0.25
EtchShop	EtchBCl3	1	1	1	2	1	1	0.75	0.5
"	EtchCF4	5	1	5	9	9	0.9	0.9	0.9
"	EtchCl2	1	1	1	1	2	0.5	0.75	1
"	EtchO2	6	1	6	12	12	1	1	1
ImplantShop	HCurr	1	2	2	1	2	0.25	0.38	0.5
"	MCurr	2	2	4	5	5	0.63	0.63	0.63
MetalShop	LPAl	2	1	2	4	2	1	0.75	0.5
"	SpTaSi2	1	1	1	0	1	0	0.25	0.5

$$M_x^n = \text{first natural number bigger than } \frac{\sum_{y=1}^{p^n} l_{xy} \cdot t_y}{b_x}$$

$$c_x = M_x^n \cdot b_x \qquad \text{and} \qquad u_x = \frac{\sum_{y=1}^{p^n} l_{xy} \cdot t_y}{c_x}$$

with: x = type of machine, y = product and z = operation,

O^n = number of operations, P^n = number of products,

M^n = number of machines of a type,

p = process time of operation of a product on a type of machine,

l = load of a type of machine due to a product,

t = throughput of a product,

b = batch size and c = capacity of a type of machine,

The controllers of the hierarchical layers generate requests according to different strategies. The station controller sends a request shortly before the machine has finished processing. A cell controller sends a request shortly before the bottleneck station has finished processing. The bottleneck station is most of the time the station where the actual processing takes place. Only if the processing step lasts 1 hour are the stations in the cell perfectly balanced, and the first station is used as bottleneck. A shop controller sends a request when it receives a cell request. The shop requests only work that can be executed by the cell which sent the cell request. The facility controller sends a request when its inventory level is below a certain level and there is no request outstanding (Fixed-WIP). The facility request is either sent when the facility receives a job or when the facility has finished a job. The cells and stations request a material amount that is equal to their (maximum) batch size. The shop requests the same amount as was requested by the cell. The job for the facility always contains four cassettes. This is done because there are many furnaces that process batches of four cassettes or common dividers of four. Only the PEOxidCell (a furnace in the diff-CVD shop) has a batch size of three. By letting this cell request 1 to 3 cassettes, instead of its maximum batch size, a batch of four cassettes can be processed by all stations without the station having to wait for cassettes from another job.

5.5 Simulation experiments and results

To demonstrate some possibilities of the control architecture, the model of the IC manufacturing system is used for different experiments. The experiments consider the following aspects: the product mix, the transport time, the batch size, the request generating strategy and the sequencing strategy. The performance of the facilities is emphasized during these experiments. The influence of the market is not considered. To decouple the behaviour of the consumer and the supplier from the performance of the facility, the factory controller ensures that an excess amount of blank wafers and product orders are available. The consumer demand is bigger than the maximum throughput of the factory and the supplier delivery time is very short. Because of these conditions the factory controller is able to answer all requests from the facility with a job.

To judge the quality of the model, it has to be verified and validated first. The verification is used to ascertain whether the model functions as intended. The validation is then used to investigate the correctness of the model in relation to the modelled system. The verification is done with the help of modular programming and testing of the different modules (in this case the processors of the model). The model is executed in steps

in order to check its behaviour. With the complete model some extreme experiments are performed. These experiments do not need to be realistic, but they do serve to reveal the robustness of the model. If the model seems to be reliable, some simple experiments are executed. The results of these experiments may be deduced analytically. The results of the experiments are compared with the analytical solution. The last check is to collect redundant data during the simulation experiments and to test whether the collected data are consistent. The validation of the model is in this case not possible because the modelled system does not exist in reality. The only possible validation is by comparison of the results with results reported in the literature for comparable experiments.

Simulation runs have been done with the complete model and once it was found to be correct, it was simplified in order to save execution time. The simplification was possible because the cells of the model behaved in a predictable way. The cells were flow shops containing a minimum of one and a maximum of three stations, all having the same batch size. The control of the cell and the stations is adjusted in such a way that a request is generated when the cassette can be processed directly after each other on all stations. The lead time of the cassette is easy to predict, the only insecurity being related to the transport time. The compressed model contains 80 processors: 66 leaf processors and 14 expanded processors. The model runs more than six times faster. The compressed model is validated with help of the complete model.

The performance of the facility has been studied with the compressed model. The results of the experiments are recorded in the form of graphs. The lead time and the throughput are set out against the inventory level. These are called performance graphs (see Section 3.1). The performance graphs contain redundant information. In most cases both graphs are reproduced because they give insights from different viewpoints. The performance graphs only have validity if the material input rate is equal to the throughput. To get truthful results this condition has to be fulfilled. A performance graph is made for every experiment. To construct performance graphs an experiment consists of runs with different inventory levels. A measurement at a certain inventory level gives only one point on the performance graph. For the complete graph usually 10 measurements are performed.

The simulation model is deterministic, but nonetheless the results of the simulation behave in some sense stochastically. The experiments involve non-terminating experiments, where the lead time and the throughput of the facility is determined at a more or less constant value of the inventory level. Before the measuring of these values is started, the facility has to be in a steady state situation. This state is achieved only after the elapse

of a certain time, called the start-up interval: the model is then in a transient state. To determine the length of the transient state the change in the distribution of the observations is studied by plotting the observations against time, and by comparing histograms of different sets of observations. When the distribution of the observations remains constant, the steady state has been reached.

The observations generated by a deterministic model show a regularity: the measurement of one cycle is enough to obtain a reliable result. Sometimes the cycle is difficult to distinguish, and then the model seems to behave stochastically. In these cases the run is split into subruns. The measurements of the different subruns is determined by looking at the sample autocorrelation function; the size of the confidence interval depends on the number of subruns.

The purpose of the experiments is to reveal the possibilities of the control architecture. Because an extended statistical analysis of the simulation output calls for many simulation runs and because the time to perform these runs is limited, for most runs only one subrun is done and, in case the observations showed large variances, three subruns are performed. The size of the 90% confidence interval is always less then 10% of the measured value. A more extensive analysis of the results is to be found in De Jonge [De Jonge 1991].

An experiment is done to determine a performance graph. The experiment starts with an empty facility, whereupon the facility is filled with material up to a certain level. The loading of material is distributed over a time interval, the length of the interval is equal to the average lead time at that moment. After the loading one waits until the steady state is

Table 5.4. *The default setting of the facility.*

product mix (CMOS:SRAM)	1:0
transport time	1 minute
batch sizes	1, 2, 3 and 4
minimum amount requested	equal to maximum batch size
minimum amount PEOxid	equal to 1
shop requesting strategy	shop request derived from cell request
facility requesting strategy	Fixed-WIP
sequencing rule	FIFO

reached. During the waiting at least all material in the facility has to be processed. This lasts at least as long as the average lead time. After the waiting the measuring starts.

Five aspects are investigated: the product mix, the transport times, the batch sizes of the machines, the request generating strategy and the sequencing strategy. A default experiment is performed to compare the different experiments. The default factory manufactures only CMOS at a throughput of 0.5 cassettes per hour. All transporter movements in the factory take one minute. The machines have batch sizes ranging from 1 to 4 cassettes. The number of cassettes requested by a machine is equal to its maximum batch size, except for the PEOxid furnace, which requests 1 to 3 cassettes. The shop controller sends the requests from the cells immediately to the facility controller. The facility controller uses a Fixed-WIP strategy to generate requests. The facility controller (and all other controllers) use a FIFO sequencing rule. The default settings of the facility are shown in Table 5.4. These settings have been varied during the different experiments. With product mix and transport time the settings of product mix and transport time are changed. With batch size the number and batch size of machines and the minimum batch size of a request is varied. With the request generating strategy, three ways of generating requests by the shop controller are studied. With the sequencing strategy different sequencing rules are used by the facility controller.

Product mix

The first experiment concerns the product mix. The facility is capable of manufacturing CMOS and SRAM. The capacity of the facility is independent of the product mix: 0.5 cassettes per hour. The two products have different process times: CMOS 199 (inclusive transport 207) hours and SRAM 189 (inclusive transport 197) hours. The lead time of a product consists of its process time and a wait time:

$$l_{CMOS} = p_{CMOS} + w_{CMOS}$$

$$l_{SRAM} = p_{SRAM} + w_{SRAM}$$

$$p_{CMOS} = 207 \text{ [hours]}$$

$$p_{SRAM} = 197 \text{ [hours]}$$

with: l the lead time, p the proces time and w the wait time.

From the formula: $l = \frac{i}{t}$ (i = inventory level, t = throughput)

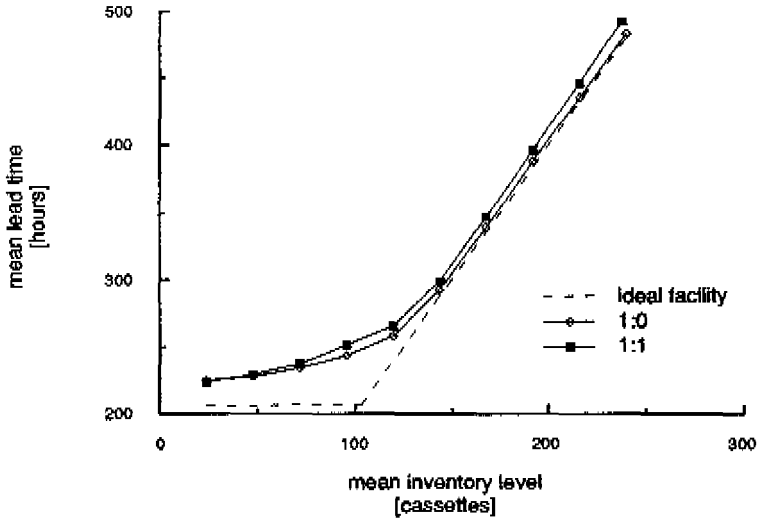


Figure 5.6. *Performance graphs of the facility for different product mixes (CMOS:SRAM = 1:0, 1:1 and 0:1).* ***a)*** *Mean lead time of CMOS cassettes versus the mean inventory level.*

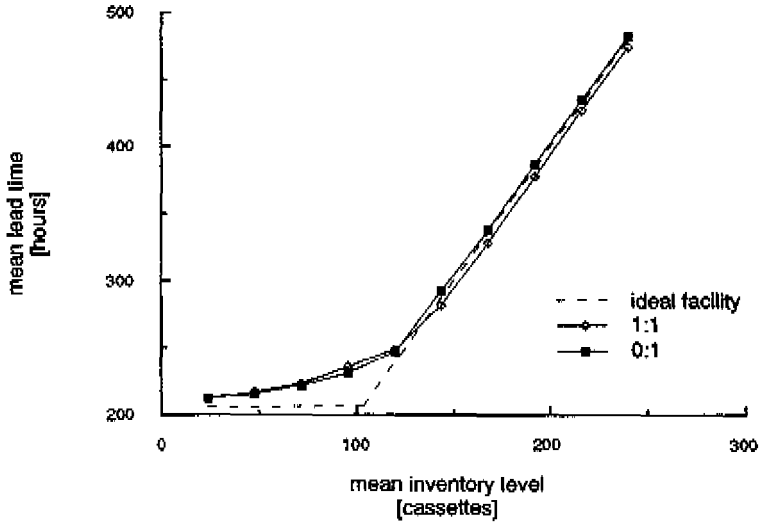


Figure 5.6. *Performance graphs of the facility for different product mixes (CMOS:SRAM = 1:0, 1:1 and 0:1).* ***b)*** *Mean lead time of SRAM cassettes versus the mean inventory level.*

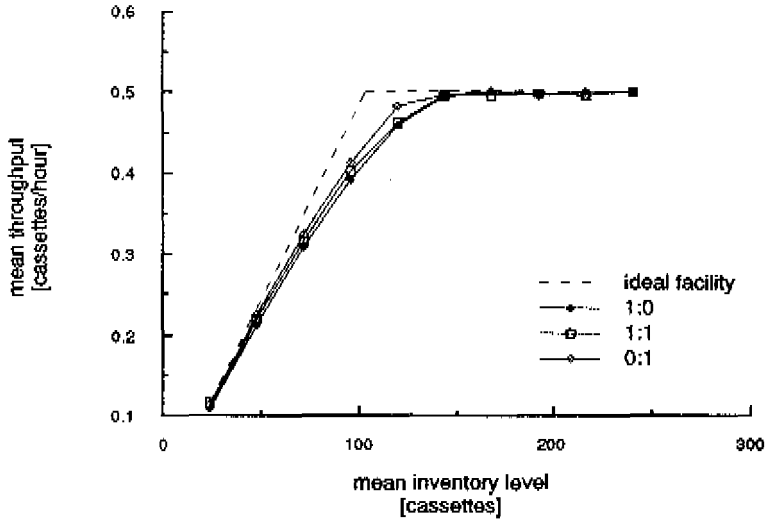


Figure 5.6. *Performance graphs of the facility for different product mixes (CMOS:SRAM = 1:0, 1:1 and 0:1). c) Mean throughput versus the mean inventory level.*

one can derive the average lead time independently of the product mix. Another expression for this lead time is:

$$l = a \cdot l_{CMOS} + (1 - a) \cdot l_{SRAM}$$

with a being the proportion of CMOS manufactured in the facility. It is supposed that the wait time is equally distributed over the two products, so:

$$w_{CMOS} = w_{SRAM} = w$$

This leads to the following expression for the wait time:

$$w = (\frac{i}{t} - p_{SRAM}) - a \cdot (p_{CMOS} - p_{SRAM})$$

The value of the wait time depends on the product mix. With ProcessTimeCMOS being larger than ProcessTimeSRAM it follows that with an increase of the CMOS ratio (a) the wait time becomes smaller, so with an increase of the CMOS ratio the lead times of the two product types also become smaller.

To see the influence of a change in the product mix on the throughput and the lead time, runs with CMOS : SRAM equal to 1:0, 1:1 and 0:1 are compared. The results are shown in Figure 5.6.

The CMOS lead time graph (Figure 5.6a) shows that at the higher inventory levels the mean lead time of CMOS cassettes decreases if the percentage CMOS produced increases. The SRAM lead time graph give a similar result. This confirms our expectations. The throughput graph show only small differences.

Transport time

All moves of the transporters in the (default) model take one minute. A transport job takes either one or two minutes, depending on the start position of the transporter. If the transporter is at the place were the material has to be removed, it takes one minute. In all other cases it takes two minutes. A transporter is supposed to be able to carry up to eight cassettes. The utilization (u) of a transporter (= percentage of the time the transporter is busy) is calculated from the load for the manufacturing of one cassette expressed in hours (= move time (m) times the number of moves (n)) divided by the batch size (b) (= number of cassettes the transporter carries per transport movement), times the throughput (t) (= number of cassettes manufactured per hour). This leads to the following expression:

$$u = \frac{n \cdot m}{b} \cdot t$$

The maximum utilization is calculated with a move time of two minutes and an inventory level of one cassette. The desired throughput is equal to 0.5 cassette per 60 minutes. Both the worst case and the observed (after simulation) utilization of the transporter are represented in Table 5.5.

From this table it appears that in worst case the facility has a heavily loaded transport system, but because the transporter can transport

Table 5.5. *Utilization of the transporters for CMOS manufacturing.*

resource	number of transport moves	worst case			observed		
		time [minute]	batch size [cassette]	utilisation	time [minute]	batch size [cassette]	utilisation
ChipFacility	61	2	1	1.02	1.73	1.27	0.69
LithoShop	26	2	1	0.43	1.59	1.00	0.34
DifCVDShop	30	2	1	0.50	1.62	2.60	0.16
EtchShop	48	2	1	0.80	1.65	1.00	0.66
ImplantShop	12	2	1	0.20	1.52	2.00	0.08
MetalShop	4	2	1	0.07	1.00	1.00	0.03

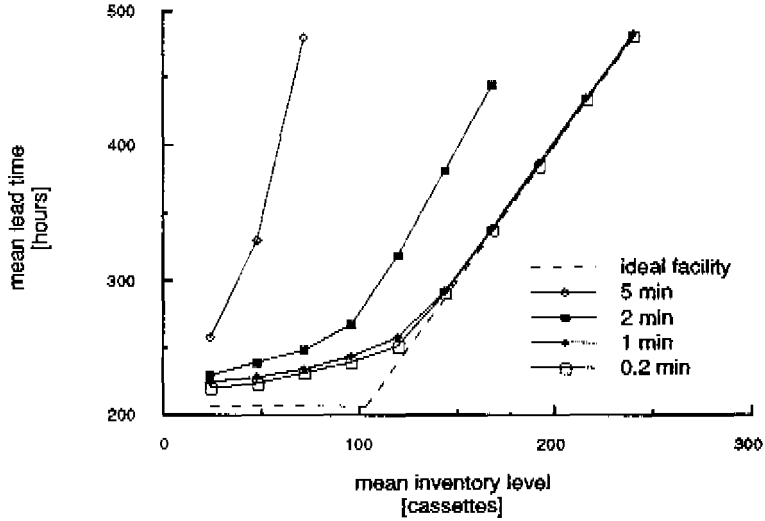


Figure 5.7. *Performance graphs of the facility for different transport times.* ***a)*** *Mean lead time versus the mean inventory level.*

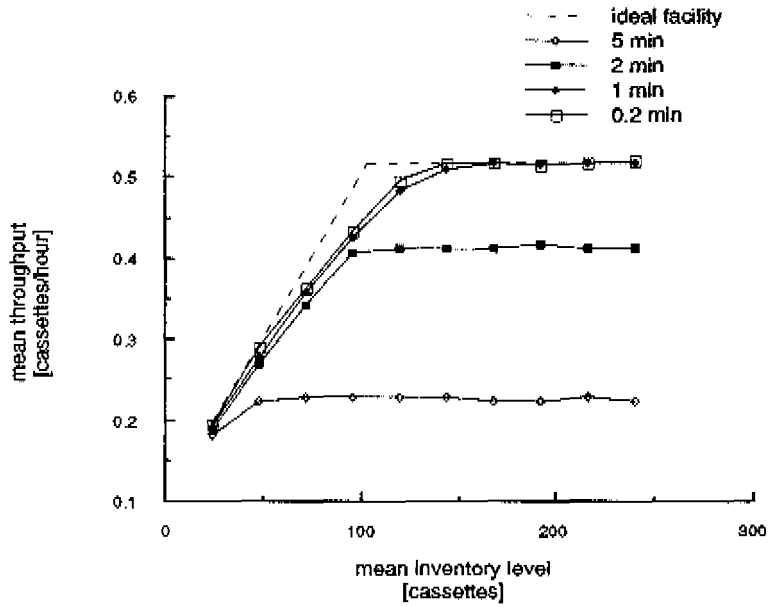


Figure 5.7. *Performance graphs of the facility for different transport times.* ***b)*** *Mean throughput versus the mean inventory level.*

batches of more than one cassette in less than two minutes, the actual capacity of the facility is bigger than the worst case capacity. The transporter in the etch shop, however, has no possibility to move more than one cassette at a time: all etch cells have a batch size of one. The only way to improve the capacity of the etch transporter would be the use of a scheduling strategy that increases the chance that the transporter waits in the right start position. With the increase of the transport time the transporter in the etch cell is expected to become a bottleneck.

Experiments have been performed with four different transport times: 0.2, 1.0, 2.0 and 5.0 minutes. The expected utilization of the transporter of the etch shop for the four transport times is calculated. The average transport time is calculated by multiplying the transport time by 1.65 (the mean time a transport movement in the etcher last according to simulation results). If the utilization of the transporter comes out to be greater than 1, it is the bottleneck and the throughput is limited by the transporter. Then the throughput wanted has to be divided by the utilization of the transporter to find the expected throughput. These values are given in Table 5.6.

The differences in transport time influence the lead time graph in two ways. An increase of the transport time increases the process time and, if the transporter is the bottleneck, the increasing sloping line becomes steeper. The change in throughput is deducible from the change in lead time.

The results of the change in transport time are reproduced in the Figure 5.7. The lead time graph shows that, if the transporter is not a bottleneck, an increase in transport time influences the lead time only for small inventory levels. If the transporter is a bottleneck the increase in lead time is dramatic. The throughput levels observed in the throughput graph at high inventory levels correspond with the expected values in Table 5.6.

Table 5.6. *Expected utilization of the transporter of the etch shop and expected throughput of the facility.*

transport time	utilization	throughput
0.2	0.132	0.5
1.0	0.66	0.5
2.0	1.32	0.38
5.0	3.3	0.15

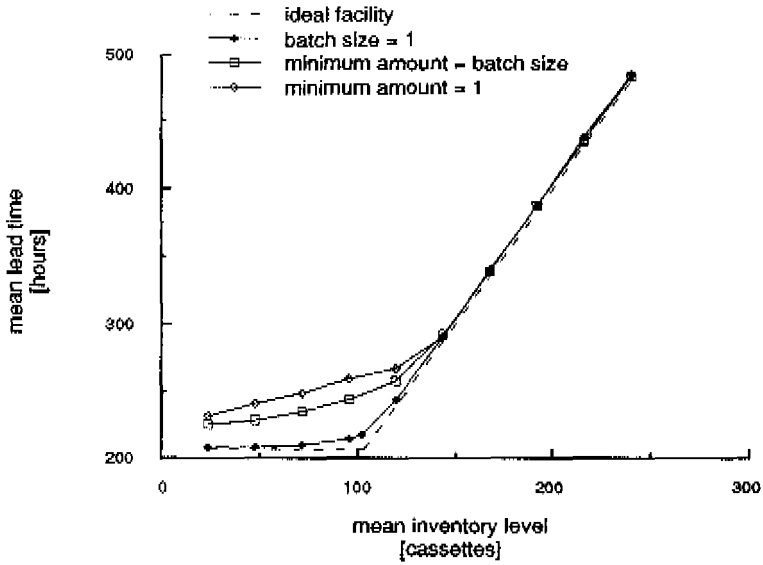


Figure 5.8. *Performance graphs of facility with machines with batch sizes of one cassette and for different values of the minimum bach size of a request.* ***a)*** *Mean lead time versus the mean inventory level.*

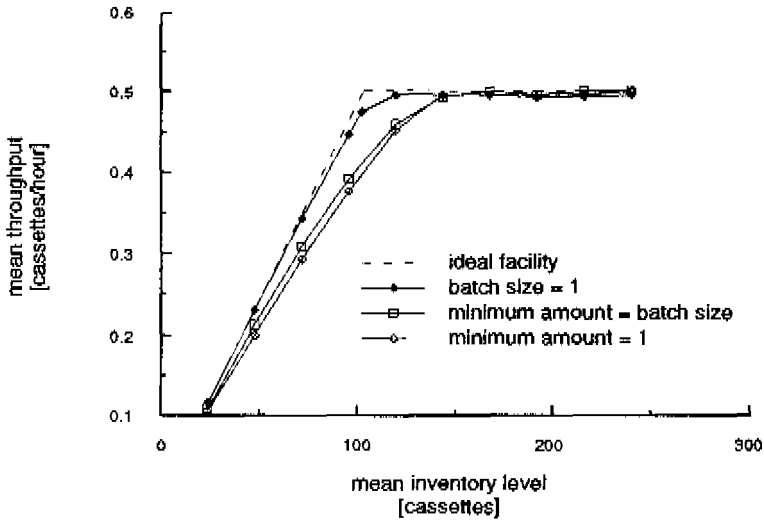


Figure 5.8. *Performance graphs of facility with machines with batch sizes of one cassette and for different values of the minimum bach size of a request.* ***b)*** *Mean throughput versus the mean inventory level.*

Batch size

The facility contains machines with batch sizes that vary from one to four cassettes. The machines with the biggest batch size also have, in general, the longest process time. Despite this, many machines with a big batch size are underutilized. Two effects have been studied in relation to the batch size: what happens if all machines have a batch size of one? And what is the influence of the minimum amount of material that is requested by a machine?

In the first experiment the capacity of all cell types is kept the same, by replacing a cell with the number of cells equal to its batch size. Thus, the cell consists of a flow shop with stations, a stations is of class single shop and contains a machine with a batch size of one cassette. The existing overcapacity is kept the same. In the second experiment the batch sizes are put back to the default values and the minimum amount of cassettes requested by a cell is changed. The new minimum amount is one cassette instead of the usual maximum batch size. The minimum amount of PEOxid is always kept at one cassette.

The results of the experiments are plotted in Figure 5.8. It appears that the facility where all transformers have a batch size of one, comes close to the ideal facility. Differences are now mainly caused by process time differences. The consequence of requesting a minimum of one cassette causes a worse performance in the low inventory level area. This situation often leads to the processing of one cassette while, shortly afterwards more cassettes arrive, which have to wait for the finishing of the first cassette. This wait time appears to be longer than the wait time that arises if machines have to wait until a batch is complete. This also has to do with the fact that a facility job contains four cassettes.

Request send strategy

It has been shown above that it is possible, using the control architecture, to manufacture different product types, to vary transport times, to change the configuration by adding machines, and to change the control parameters of resources (minimum batch size requested). In the next example the behaviour of the shop controllers is changed. Up to now a shop request has been sent after the reception of a cell request (the cell request strategy). This situation is compared to generating shop requests depending on the inventory level in the shop (a Fixed-WIP request strategy). The control strategy variants used in these cases refer to the request send strategy of the five shops. As stated in Section 4.4 for a manufacturing system of class parallel shop, which the shop in the IC factory is, the request generation is best based on the requests of the

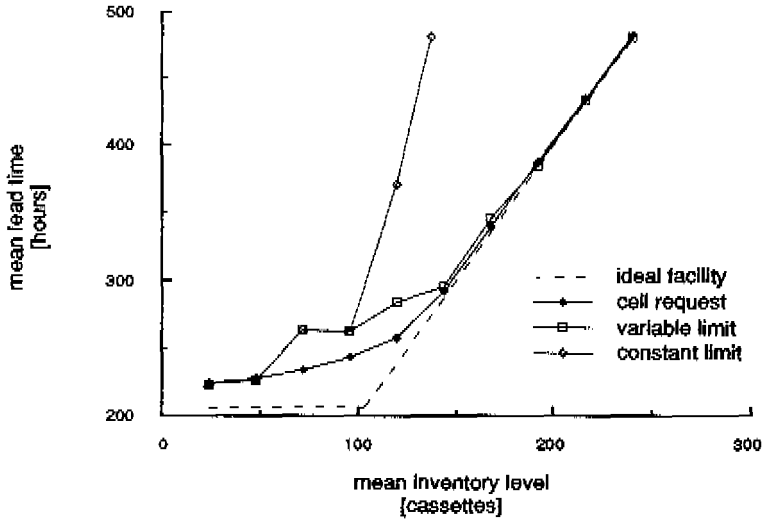


Figure 5.9. *Performance graphs of the facility for different shop request strategies. The constant and the variable material limit strategy are compared with the cell request strategy.* ***a)*** *Mean lead time versus the mean inventory level.*

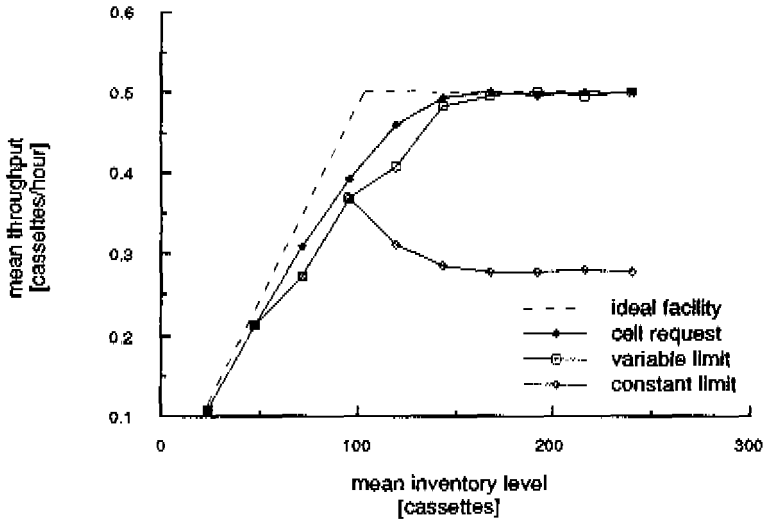


Figure 5.9. *Performance graphs of the facility for different shop request strategies. The constant and the variable material limit strategy are compared with the cell request strategy.* ***b)*** *Mean throughput versus the mean inventory level.*

resources (the cells). So these experiments may show if this assumption is correct.

The Fixed-WIP strategy uses two kinds of limits: a limit that remains the same during the experiment (constant limit) and a limit that depends on the inventory level during the measurement (variable limit). In both cases the Fixed-WIP level is constant during the simulation run. The constant limit is calculated using the work load of the shop. The division of the material depends on this work load. This is based on the assumption that the facility is perfectly balanced and every shop needs its processing time share of the material. The share is increased by one cassette, to also allow a load for the transporter of the shop. The load per shop is calculated with the formula:

load = total process time in shop . maximum throughput

Thus the sum of the process times of all operations that have to be performed in a shop times the wanted throughput is the amount of the inventory that the shop should contain. Table 5.7 shows the inventory limits that result for the maximum wanted throughput in the constant limit case. In the variable limit case the material limit in the shop during a measurement is found by multiplying the maximum inventory level of the facility and the ratio given in Table 5.7. The constant limits are equal to the variable limits at the ideal work point. The ideal work point is equal to the nominal lead time times the maximum throughput: this is 207 x 0.5 = 103.5 cassettes.

The sending of requests according to a Fixed-WIP release strategy, uses less information than the sending of request based on cell requests. A shop request based on a cell request asks only for jobs the cell is able to execute. A request based on the inventory level (Fixed-WIP) asks for jobs that the shop is able to execute. As a consequence the facility controller might send a job to the shop for a cell that is not idle. This job takes the place of a job for the idle cell. So material in the shop blocks material waiting on the facility level. With the variable limit, all material

Table 5.7. *Work loads of the different shops for one CMOS cassette.*

shop	process time [hour]	load [cassette]	limit [cassette]	ratio
litho	39	20	21	0.196
difCVD	98	49	50	0.492
etch	48	24	25	0.241
implant	6	3	4	0.030
metal	8	4	5	0.040

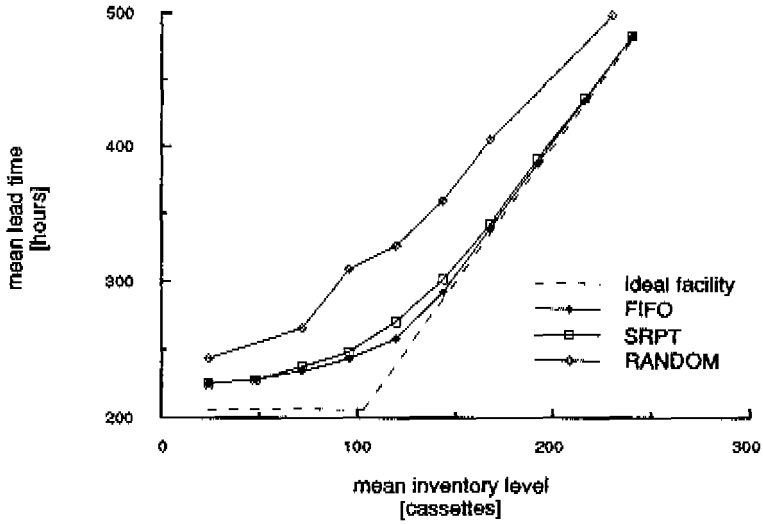


Figure 5.10. *The performance graphs of the facility for different sequencing rules.* ***a)*** *Mean lead time versus the mean inventory level.*

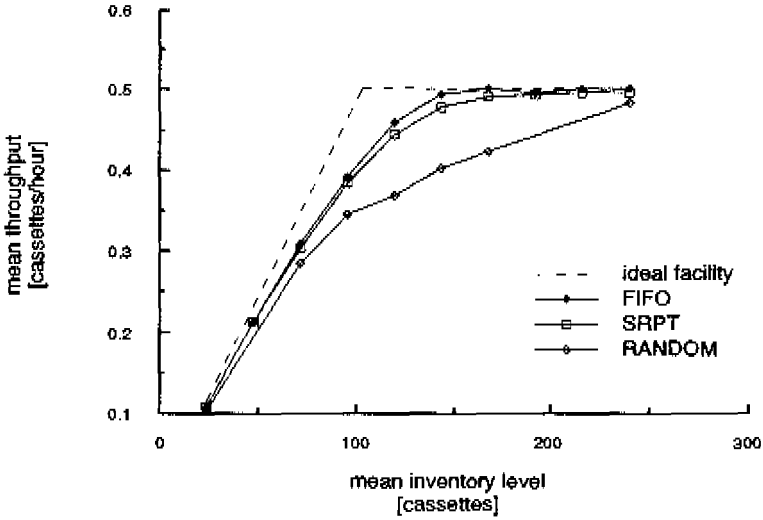


Figure 5.10. *The performance graphs of the facility for different sequencing rules.* ***b)*** *Mean throughput versus the mean inventory level.*

is divided over the shops in a proportional way. If the facility is reasonably balanced all material will be waiting in the shops. With the constant limit the shop only contains the amount of material it is supposed to be able to process. Below the ideal work point all material is probably waiting in the shops; above this work point part of the material is waiting at the shop level and another part at facility level. With the cell request strategy the shop only contains material that is being processed in the resources of the shop; the rest of the material is waiting on facility level. The last control strategy ensures that the right jobs are sent to the shop and there is a view of the material that is waiting to be processed. The results show that the performance of this strategy is better than the use of a material limit. Besides this, the strategy also offers the biggest opportunity to schedule material in the facility.

The results in Figure 5.9 show that the cell request strategy is best. Below the ideal work point the Fixed-WIP request strategy using constant limits behaves the same as the one using variable limits. Above the ideal work point the experiments with variable limit approach the cell request strategy while the experiments with a constant limit clearly show a diminution in throughput. This shows that a parallel shop can use a request generation strategy, where the requests are generated at the moment when they are received from the resource. However, this is only true if delays due to transport of material have been accounted for in the requests of these resources.

Sequencing strategy

The last example shows that it is possible to apply different sequencing rules in the control architecture. The sequencing of jobs in the facility is examined. Three sequencing rules are applied. FIFO (first-in-first-out), SRPT (shortest-remaining-process-time) and RANDOM. A rule like SPT (shortest-process-time) has no use in the facility because there are only a few operations which have to be executed on different resources and which also have different process times. And if one is interested in other sequencing rules, they can easily be implemented and tested with help of simulation.

Because some shops request jobs with more than one cassette, the sequencing rule is combined with an algorithm to join cassettes. A job always contains the cassette with the highest priority and if necessary it contains more than one cassette. So sometimes the situation arises that the cassette with the highest priority cannot form a complete batch, while other waiting cassettes do form a batch. The cassette with the highest priority now blocks these other cassettes and no job is sent. In case of FIFO, the cassette with the highest priority (the first cassette) is

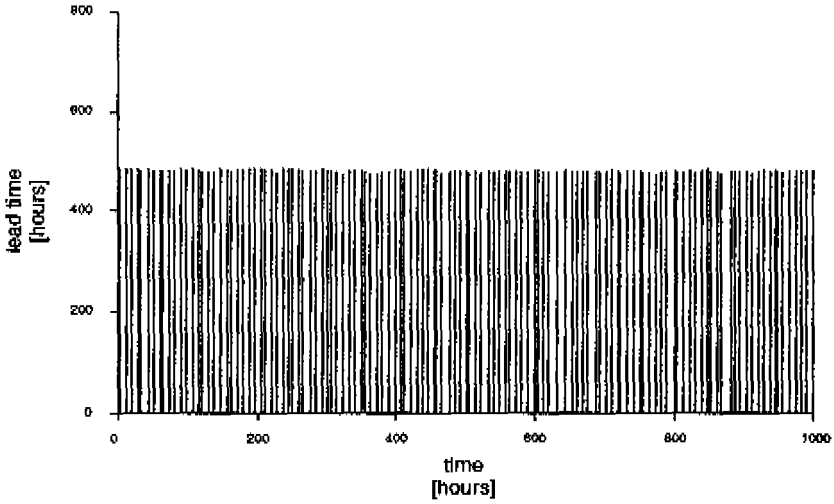


Figure 5.11. *Observations of the lead time of cassettes in the facility against time for different sequencing rules.* ***a)*** *Lead time against time for FIFO.*

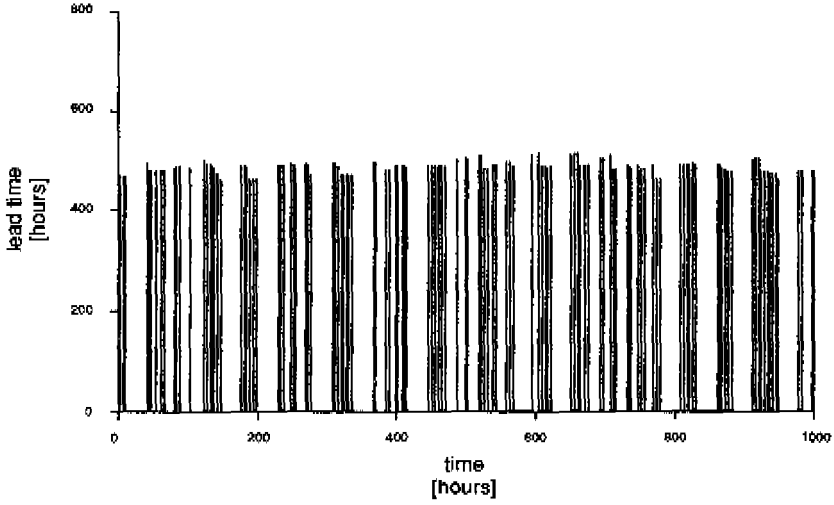


Figure 5.11. *Observations of the lead time of cassettes in the facility against time for different sequencing rules.* ***b)*** *Lead time against time for SRPT.*

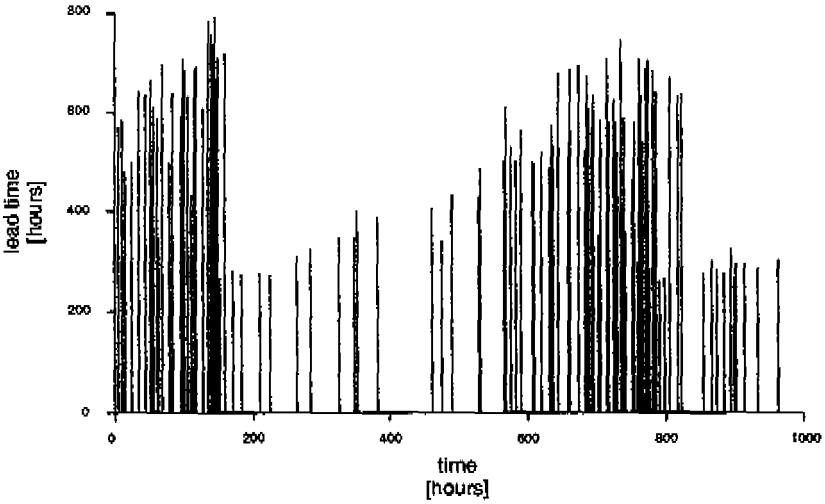


Figure 5.11. *Observations of the lead time of cassettes in the facility against time for different sequencing rules. c) Lead time against time for RANDOM.*

probably also the first batch. With SRPT, and especially with RANDOM, it is less probable that the cassette with the highest priority will form the first batch. The results in Figure 5.10 show that FIFO performs best, even a little better than SRPT and much better than RANDOM.

The Figures 5.11 a, b, and c show the lead time of the jobs and the time at which the specific job was finished. The figures show the lead times for an inventory level of 240, but the behaviour found is also valid for other inventory levels. These figures show that SRPT and RANDOM cause a bigger variation in the lead time than FIFO. Not only the lead time varies more, the time between the finishing of two jobs also shows a bigger variation. It appears that especially RANDOM sequencing causes the facility to oscillate. This oscillation is also related to the Fixed-WIP release strategy, where a new job is started when a job is finished. So if many jobs finish shortly after each other, many jobs are also started during a short interval.

This chapter has illustrated how a hierarchical control system can be built using the control architecture. The constructed factory contains five control layers, where the second layer has a job shop character. The factory contains 1 facility, 5 shops, 43 cells, 110 stations, 110 transformers, 160 stores, 160 transporters and 160 controllers. With the help of simulation experiments the use of performance graphs and a number of possibilities of the control model have been demonstrated.

Chapter 6
Conclusions

6.1 Review of the study

The study presents a new hierarchical control architecture and a structured method by which it is possible to specify, simulate and implement controllers for actual or planned manufacturing systems having a job shop character. A job shop is characterized by universal machines having great flexibility in terms of material routes. A property related to job shop manufacturing systems are the serious problems in controlling throughput and lead time.

The investigation was initiated in view of the increasing need for automation, coupled with the problems associated with the automation solutions currently available. Two current automation problems have been signalled by Arentsen [1989]: the gap between the automation of machinery and the automation of administration, and the island automation which results in stand alone solutions of automated elements which are difficult to combine into an efficient system. With the advance of technology the functionality of machines has increased, which makes the complexity of manufacturing systems greater and the control of manufacturing systems more difficult. One of these control problems is the long lead times that are often encountered in manufacturing systems with a job shop character. There is a clearly perceived need for computer control of these manufacturing systems. But the results of research into manufacturing controllers for such systems, as these have been presented in the literature to date, are very vague, and the research into sequencing, which is a part of the control problem and is thought to be able to reduce the long lead times, often lacks any close relation with practical application in manufacturing systems.

Arentsen's [1989] work demonstrated the value of modelling and simulation in the building of control systems, as well as how to avoid island automation and the automation gap. His factory control architecture concentrates on the highest control level of a factory and on the interactions between factories. It consists of a chain of transformers which are controlled by their own controller. It is, however, only applicable to manufacturing systems with a flow shop character. This study presents a control architecture that is applicable to the much more

complex job shop manufacturing systems. It uses recipes to specify the manufacturing process. The control architecture is constructed in such a way that it can be used for any manufacturing system for which the manufacturing processes are expressible in (sequential) recipes. The architecture is a framework that can be used when building and/or automating manufacturing systems.

The starting point for building controllers for industrial systems is modelling. The modelling is performed according to the Process-Interaction approach [Rooda 1987, Overwater 1987]. This approach provides a language (ProcessTalk) and a tool (ProcessTool) [Wortmann, Rooda 1990, Wortmann 1991], which have been used for specifying, developing and testing the architecture. The approach and the tool together allow a smooth transition from modelling the system to simulation of the system and finally to implementation of the controller. The building blocks of the control architecture are implemented in software in the tool. In this way it is easy to simulate future manufacturing systems, and to use the (simulated) controller in the implementation of the manufacturing system.

Modelling involves the setting of boundaries for the system and concentrating on those aspects of the system that are of interest to the modeller. In this case the flow of discrete pieces of material, the machines that manipulate the material, the controllers that drive the machines and the interactions necessary for directing the material through the factory are modelled, together with the necessary data structures and the control algorithms. Energy, gases, liquids and small parts such as screws are not modelled. Operator availability, tools, the set up of machines, their maintenance and repair are not part of the model, just like machine breakdown and yield losses. The industrial system that results is an idealized and simplified system in which the main material flow is represented. The study is based on static manufacturing systems, by which is meant that the configuration of the manufacturing system and the manufacturing process remain constant in time.

In this study a classification of hierarchies is introduced (Section 2.5). The architecture is based on a hierarchical approach. The centralization of control enables a decoupling between the controller structure and the manufacturing process. This decoupling is realized with help of recipes, which can be executed by a controller. Centralized control also opens the possibility to let all resources aim at the same goal. Multiple hierarchical layers correspond with layouts of factories and allow the distribution of the control effort. The unique feature of the control architecture is the fact that the controllers of the different hierarchical levels all have an

equivalent model, which is recursively repeatable. As a result of this the number of hierarchical layers can be adjusted arbitrarily.

The building of a manufacturing system starts with the specification of the physical system configuration and the specification of the manufacturing process. These specifications are used in the control system. The architecture uses standardized models of the physical manufacturing machines, which are presented in Chapter Two. These are the basic physical building blocks. Machines are called leaf resources. There are two categories: processing leaf resources (shapers, transformers and assemblers), and supporting leaf resources (transporters and stores). A multi-layer control system comprises aggregates of machines. An aggregate of machines and a control system is called an expanded resource. An expanded resource is also a processing resource. Operations are used for the specification of the manufacturing process steps. Resources execute operations by transforming material. The list of operations that have to be executed by a group of processing resources to manufacture a product is called a recipe and is the representation of the manufacturing process. In this thesis new structures for recipes have been presented.

A manufacturing class is the structure of the top layer of a manufacturing system. The class of a manufacturing system depends on the physical layout of the resources and on the recipes that the system can execute. Four classes of manufacturing system are introduced: single shop, parallel shop, flow shop and job shop. The control architecture is especially suitable for the control of a job shop, but because the other three layouts are simpler variants of the job shop, the architecture can also be applied in the other cases. An expanded resource also belongs to a certain manufacturing class, so the total factory may contain different types of manufacturing classes.

The controller has to direct resources and material, and to monitor their status. Decisions are taken as late as possible in order to minimize the differences between reality and anticipated reality. The decisions are subdivided into categories concerning material exchange, transport and processing. The moment to take a decision depends on the decision freedom and the time at which information becomes available. In relation to this the strategies of early and late transport are distinguished, just like material driven and command driven manufacturing. The place where information is available is an indication of the level at which decisions have to be taken. To make information available, the controller communicates with the resources. In this study four communication protocols between the controller and the resource are introduced and looked at. The aspects related to control are more extensively handled in Chapter Three.

In order to be able to judge the performance of the manufacturing controller, performance graphs have been used (Section 3.1). The behaviour of a manufacturing system is recorded by simulating the model of the manufacturing system and measuring the mean lead time, mean throughput and mean inventory level. The performance graphs plot lead time against inventory level, and also throughput against inventory level. In a balanced system these three variables are related to each other. An essential addition to the performance graphs are the lines describing the ideal factory. By plotting both the actual performance and the ideal performance, a clear insight is gained into the behaviour of the manufacturing system in relation to the ideal, best attainable performance.

The divergence between reality and anticipated reality is kept to a minimum by making use of a reactive scheduling strategy. This means that a decision is taken at the moment the actual choice occurs. Scheduling is divided into subfunctions: releasing, allocating, sequencing and dispatching (Section 3.2).

Chapter Four presents the development of the hierarchical control architecture in the form of the data structure and the control model. A manufacturing system consists of processing resources, a transporter, a store and a central controller. The release strategy, which defines the way new jobs are started in a manufacturing system, is implemented with the help of requests. A request from a resource asks for work. In fact the request indicates free capacity in the resource. In the control architecture the momentary capacity is calculated from bottom to top. Resources send requests to the central controller. The requests are used to allocate and sequence manufacturing jobs and to calculate momentary capacity, and thus for the generation of requests for the control level above, which influences the release of jobs. In this way a new formalization for the releasing of jobs in a hierarchical controlled manufacturing system is given. By controlling the release of manufacturing jobs, the jobs are executed with lead times that fall within predictable limits.

The control architecture is illustrated in Chapter Five, where an IC factory is modelled. Here it is seen that the newly developed architecture and the control model present a structured method for the implementation of a hierarchical control system for a complex job shop manufacturing system. The experiments show that the performance of the manufacturing system can be controlled within acceptable limits.

6.2 The advantages of hierarchical control

The thesis presents a detailed description of the control architecture. The architecture enables manufacturing control system builders to model, simulate and implement controllers for manufacturing systems. It is suitable for any kind of manufacturing system, ranging from a single shop to a job shop. It contains building blocks for the modelling of manufacturing systems, and it makes the design of manufacturing systems and manufacturing controllers a more structured process, allowing the construction of hierarchical control systems.

The control problem of many manufacturing systems calls for solutions that are only valid for the specific situation. The architecture presented here does not offer a general solution with which every specific problem can be tackled, but it does offer a framework which can be adapted to most situations, which takes account of the whole system, and which offers a possibility to adjust the different controllers to a common goal. This architecture for control systems eases the task of the control system builder, in that it offers a structured approach and building blocks with which systems can be designed in shorter times.

The integration of new concepts has resulted in this control architecture. One of these concept is the use of the same controller model in the different hierarchical layers. The control architecture uses the concept of recipes to specify the manufacturing process. The recipes allow a decoupling between controller structure and the manufacturing process: a necessary condition to be able to control job shop manufacturing systems. Recipes are a type of software which instructs the manufacturing controller. The behaviour of the manufacturing system and the control system is measured with the help of the concept of performance graphs. The resources influence the work load, due to this concept the progress of the manufacturing of products is influenced in a bottom-up manner. In the literature manufacturing systems are usually controlled in a top-down manner. The progress of the manufacturing process is regulated with help of the request concept. This has resulted in a new formalization of the release of jobs in a hierarchically controlled manufacturing system.

The repeatability of the control model in hierarchical layers, the decoupling between controller structure and manufacturing process, the use of performance graphs and the formalization of hierarchical releasing with help of requests, are unique to this control architecture. These concepts make the control architecture fit for the control of job shop manufacturing systems.

Useful new ideas have been developed during the realization of the control architecture. The three hierarchical forms - system hierarchy, model hierarchy and inheritance hierarchy - are a new contribution to the architecture. The specification of a manufacturing process with help of the four recipe structures (sequence, concurrency, alternative and block) is a new idea. The classes single shop, parallel shop, flow shop and job shop are introduced to classify manufacturing systems.

Performance graphs have been used in more places in the literature, but the idea of introducing the line of the best attainable behaviour of a manufacturing system in the performance graph is new. This addition gives a much clearer insight into the behaviour of the actual manufacturing system. For simple manufacturing systems the graphs have a trivial character, but for large, complex manufacturing systems the performance graphs are useful for the assessment of the control system and for setting out control strategies.

In considering controller decisions and decision moments, new control strategies have been found. First the early and late transport strategies which are related to the transport decisions. Second the command driven and material driven manufacturing which are related to the way processing decisions are taken.

The study of communication between controllers and between controller and resources has resulted in four new communication protocols. The formalization of these protocols and the different possibilities and implementations of these protocols are new notions.

It is seen that long lead times are caused by high inventory levels. To reduce lead times it is not so much a new sequencing strategy that is needed as a better control of the inventory level. The work point of a job shop manufacturing system is adjusted by controlling the inventory level. With a simple releasing strategy (Fixed-WIP releasing) the inventory level can be kept constant. As regards sequencing, first-in-first-out seems to be an excellent strategy: it is fair, every piece of material flows in a natural way through the manufacturing system, and it causes small deviations in the lead time.

The control problem has to do with the divergence between reality (the status of the factory) and the anticipated reality (the production plans and schedules for the factory). This divergence makes corrective actions necessary, which makes the control problem complicated, and this divergence should be kept to an absolute minimum. The divergences between reality and anticipated reality are caused by uncertainties in consumer demands, supplier deliveries and manufacturing system behaviour.

The capacity of a manufacturing system is based on a forecast of demand. As long as the capacity of a manufacturing system is constant, the performance of the manufacturing system should not be related to its ability to fulfil the demand. Market demand has to be responded to in such a way that the manufacturing capacity is not exceeded. This is a problem that has to be solved on the factory level. It seems better to avoid the manufacturing control having to absorb demand fluctuations, since this obscures both demand uncertainty and manufacturing system performance. The manufacturing control has to minimize the uncertainty in the manufacturing system's behaviour. A manufacturing system that behaves in a predictable way enables the capacity planning function in the factory controller to see the consequences of its decisions.

The control is not based on the idea of executing detailed production plans, which are implemented from top to bottom. In the control architecture the momentary capacity, which is calculated from bottom to top, is used to progress production. Control decisions are taken on a reactive basis. This way, any divergence between reality and anticipated reality is minimal.

To control a hierarchical manufacturing system, it is required that the uncertainty in the behaviour in a layer is small, as small as possible, and that the momentary capacity of a layer is clearly expressed. For single shop and parallel shop layers, these conditions are easily fulfilled. For layers with a flow shop structure some constraints are needed in order to fulfil these conditions. In job shop structures, however, the instantaneous capacity is the result of changes and often cannot be unambiguously expressed. For this reason it seems to be advisable to avoid a job shop structure as far as possible in the control layers. The example of the IC factory showed that the architecture does work for a job shop structure. In case a job shop is implemented within a multi-layer control system, the architecture allows a job shop structure in every layer. From the control point of view and for efficiency purposes, the job shop structure should be implemented in as few layers as possible, and preferably only in one of the top layers, with little uncertainty in the layers below. This allows scheduling decisions to be centralized and the behaviour to be optimized on a global basis.

Although there is no maximum to the number of hierarchical layers, it is advisable to limit their number. Every layer includes extra transport systems, mechanical interfaces between transport systems, and extra stores. The sharing of stores and transport systems is possible in the control architecture, but because this complicates the control problem considerably, it seems to be a undesirable solution.

Decentralization of the control problem can be viewed as the moving of responsibility from top controllers to controllers in lower hierarchical

layers. A common goal and consistent data inside a factory will not allow a totally distributed control system: a central controller remains necessary, even if it might be reduced to something like a central database.

Planning and scheduling research is of little use without a uniform performance measure, a general control strategy and a control framework. The control architecture presented in this thesis offers the possibility to fit such research in with the whole. To improve performance, in most cases it is not the control strategy or algorithm which has to be changed; rather, it is the physical manufacturing system which will have to be adapted. The batch size of resources and the process times of operations, in particular, should be attuned to each other.

The strength of the control architecture has been demonstrated by the simulation of an IC manufacturing system. This is considered to be a complex system, containing five hierarchical layers and more than one hundred processing machines. No simulations found in the literature have considered the hierarchical control levels in such detail. The simulation demonstrated the capabilities of the control architecture. In the example, the product mix, the transport times, the batch sizes, the request generating strategy and the sequencing strategy have all been varied.

The performance graphs of the IC facility show that, when the inventory level is low, the choice of a sequencing strategy is not interesting; all sequencing strategies behave more or less the same, because there are only very short queues with no or few options. When the inventory level is very high the sequencing strategy is also not very interesting; the manufacturing system in this case is saturated and manufactures at its maximum level.

6.3 Recommendations for further research

The architecture is versatile, and may be adapted to a great variety of control situations. It allows the systematic design, modelling and simulation of manufacturing control systems, and can be used directly to implement control, when a factory is built. Nevertheless, a fully integrated and automated manufacturing system has not been realized. Neither has it been demonstrated that the control architecture may resolve every control situation. The architecture is, however, a firm base to which many additions are possible. Some of these additions are discussed in this section.

A lot of aspects of manufacturing control still remain to be investigated. Functions like quality control, product development, investment policy

and accountancy will have to be dealt with. Deadlock is at present circumvented by using stores with large enough space for material. If deadlock cannot be solved in this way, the controller has to incorporate calculation before requesting a new job to see whether a new job will cause deadlock or not. Capacity is only available if deadlock is excluded.

The architecture does not contain complex planners and schedulers. A plea has been entered for simple reactive schedulers. In some cases new scheduling techniques, based on artificial intelligence research or on neural networks, seem to offer opportunities in controlling manufacturing systems.

The performance graphs shown in the example refer to a few product types with small differences in work content. The significance of these performance graphs, in the case of jobs with large differences in work contents, is the subject of ongoing research which is also considering other performance criteria.

The control architecture manages the discrete material which undergoes the manufacturing process. In a later stage the architecture will have to be extended to take care of the management of bulk materials (e.g. gases and screws), scrap and energy. The simulation model did not consider tools and machine breakdown. The control of ideal manufacturing systems has to be understood before these aspects can be considered more closely. Extensions, which have to do with tools, setup of machines, maintenance and repair, will have to be integrated in the future, if they are not avoidable.

In order to get closer to a totally automated manufacturing system further development of the data structure and the control structure are necessary. The recipes used in the example did not contain assembly operations. To include these operations in the manufacturing processes, the data structure has to be extended with parallel recipes and with parallel tasks. Furthermore, the interpreting function of the manufacturing controller might also have to change. Another extension in this area is the possibility to change material routes, as a consequence of test results.

The control architecture is applicable to non-changing manufacturing systems. An interesting extension to the architecture is the possibility to handle changes in the manufacturing processes and changes in the configuration of the manufacturing system. Such extensions will increase the applicability of the control architecture considerably. The control architecture represents the manufacturing process independently of the control structure. If the manufacturing system contains a transport system that is able to realize general material routes, the control architecture is suitable for dynamic manufacturing processes: only functions like manufacturing process specification and the distribution

of the information of the manufacturing process specification (the recipes) to the different controllers have to be incorporated in the architecture.

The change of the manufacturing system configuration is probably easy to incorporate because of the request mechanism. A newly connected resource only has to send requests to state its capacity and to receive jobs. The controller initialization may have to be changed, but the control algorithm remains the same.

The results of this study have not been implemented in an existing factory: to do this it will be necessary to perform a closer study of organizational structures and management tasks. Some of these may have to be incorporated in the architecture. The implementation of a control architecture in a new or existing organization is probably worthy of a study in its own right.

During the development of the control architecture ideas for further research into manufacturing control became available. Students have explored some of these ideas, of which three are mentioned here. The communication protocols between the controller and the resources offer so many opportunities that further research on the different protocols seems justified [De Jonge 1991]. The implementation of the control architecture without the use of computers, but with cards, looks possible and interesting [Vincenten 1991]. During the control of a manufacturing system simulation can be used to take allocating and sequencing decisions. A further investigation of the possibilities of simulation as a tool for controlling and decision support seems to offer great opportunities [Steyns 1991].

The architecture developed here can handle very complex factory configurations - job shops - and thus formes an important contribution to the theory of factory and manufacturing control. The architecture has proved to be reliable and robust. It has given rise to a great number of new ideas, some of which are the subject of ongoing research, and it promises well for the future: extensions are likely to make it even more powerful.

References

Arends N. W. A., Taminiau D. A.,
Een aanzet tot het modelleren van de produktiebesturing van chipfabrieken (in Dutch),
Memorandum, Faculty of Mechanical Engineering,
Eindhoven University of Technology, Eindhoven (1989).

Arentsen J. H. A.,
Factory control architecture, A systems approach,
Dissertation,
Eindhoven University of Technology, Eindhoven (1989).

Atherton R. W.,
Factory scheduling using simulation models,
Proceedings of the Third Symposium on Automated Integrated Circuit Manufacturing,
Electrochemical Society 333-345 (1988).

Atherton R. W.,
Dynamic capacity planning using simulation models,
Proceedings of the Fourth Symposium on Automated Integrated Circuit Manufacturing,
Electrochemical Society (1989).

Atherton R. W., Dayhoff J. E.,
Signature analysis: simulation of inventory, cycle time and throughput trade-offs in wafer fabrication,
IEEE Transactions on Components, Hybrids and Manufacturing Technology, 9 (4) 498-507 (Dec 1986).

Atherton R. W., Pool M. A., Mukherjee S., Hodgeman R.,
Validated simulation models for factory control,
Int'l Semiconductor Manufacturing Science Symposium,
Proceedings, 118-122 (1989).

Baker K. R.,
Introduction to sequencing and scheduling,
John Wiley & Sons, New York (1974).

Bertrand J. W. M., Wortmann J.C.,
Production control and information systems for component-manufacturing shops,
Elsevier Scientific Publishing Company, Amsterdam (1981).

Beukeboom J. J. A. J., Biemans F. P. M., Hehl C. J. G., Sjoerdsma S., Veen H. J. van,
CAM reference model,
CFT Report 01/89,
Philips, Eindhoven (1989).

Biemans F. P. M.,
A reference model for manufacturing planning and control,
Dissertation,
University of Twente, Enschede (1989).

Bitran G. R., Tirupati D.,
Development and implementation of a scheduling system for a wafer fabrication facility,
Operations Research, 36 (3) 377-395 (1988).

BSI
Glossary of production planning and control terms,
BS 5191: February 1975,
British Standards Institution, London (1975).

Buffa E. S., Sarin R. K.,
Modern production/operations management,
John Wiley & Sons, London (1987).

Buma J. T.,
Materialen onderzoek in een stroomversnelling (in Dutch),
De ingenieur, (1), 13-21 (1987).

Burbidge J. L.,
IFIP glossary of terms used in production control,
North-Holland, Amsterdam (1987).

Burman D. Y., Gurruola-Gal F. J., Nozari A., Sathaye S., Sitarik J. P.,
Performance analysis techniques for IC manufacturing lines,
AT&T Technical Journal, 65 (4) 46-57 (Jul/Aug 1986).

Conway R. W., Maxwell W. L., Miller L. W.,
Theory of scheduling,
Addison-Wesley, Reading MA (1967).

Dal J. C. H. M. van,
'Automated Guided Vehicles' in wafer-fabrieken (in Dutch),
Master's thesis, Faculty of Mechanical Engineering,
Eindhoven University of Technology, Eindhoven (1989).

Denekamp B. R.,
ChipFab recipes,
Personal communication (1989).

Denekamp B. R., Rooda J. E., Wortmann A. M., Smit G. H.,
How to model and simulate a wafer processing facility,
Semiconductor International, 13 (10) 109-111 (1990).

Doulgerie Z.,
Production scheduling policy for flexible manufacturing systems,
Dissertation,
Imperial College, London (1987).

Doulgerie Z., Hibberd R. D., Husband T. M.,
The scheduling of flexible manufacturing systems,
Annals of the CIRP, 36 (1), 343-346 (1987).

Elleby P., Fargher H. E., Addis T. R.,
A constraint-based scheduling system for VLSI wafer fabrication,
In: Knowledge Based Production Management Systems, Brown J. (Editor),
Elsevier Science Publishers, North-Holland (1989).

Ehrlenspiel K.,
Kostengunstig Konstruieren - Kostenwissen, Kosteneinflusse, Kostensenkung (in German),
Konstruktionsbucher Band 35, Springer-Verlag, Berlin (1985).

Flatau U.,
Designing an information system for integrated manufacturing systems,
In: Design and Analysis of Integrated Manufacturing Systems, Compton W. D. (Editor),
National Academy Press, Washington DC (1988).

French S.,
Sequencing and scheduling,
John Wiley & Sons, New York (1986).

Glassey C. R., Resende M. G. C.,
Closed-loop job release control for VLSI circuit manufacturing,
IEEE Transactions on Semiconductor Manufacturing 1 (1), 36-46 (1988).

Goldberg A.,
Smalltalk-80, The interactive programming environment,
Addison-Wesley, Reading MA (1984).

Goldberg A., Robson D.,
Smalltalk-80, The language,
Addison-Wesley, Reading MA (1989).

Graves S. C.,
A review of production scheduling,
Operations Research 29 (4), 646-675 (1981).

Groover M. P.,
Automation, production systems, and computer integrated manufacturing,
Prentice-Hall International, Englewood Cliffs NJ (1987).

Hax A. C., Candea D.,
Production and Inventory Management,
Prentice-Hall, Englewood Cliffs NJ (1984).

Herroelen W.,
Computergeintegreerde produktie: mythe of werkelijkheid? (in Dutch),
Informatie 27 (1), 336-341 (1985).

Hubka V., Eder W. E.,
Theory of Technical Systems,
Springer-Verlag, Berlin (1988).

Janssen J. H. J.,
Een besturingsconcept voor een lithoshop-controller (in Dutch),
Master's thesis, Faculty of Mechanical Engineering,
Eindhoven University of Technology, Eindhoven (1989).

Joensson H.,
Simulation studies of hierarchical systems in production and inventory control,
Dissertation,
Linkoeping University, Linkoeping (1983).

Jonge M. A. C. de,
Modellen voor hierarchische fabrieksbesturingssystemen (in Dutch),
Master's thesis, Faculty of Mechanical Engineering,
Eindhoven University of Technology, Eindhoven (1991).

Kager P., Lou S. X. C.,
Wafer fabrication scheduling using flow rate control strategy,
Int'l Semiconductor Manufacturing Science Symposium, Proceedings, 21-24 (1989).

Kelton W. D.,
Statistical analysis methods enhance usefulness, reliability of simulation models,
IE (Industrial Engineering), 74-84, September (1986).

Kempf K. G.,
Manufacturing planning and scheduling: where we are and where we need to be,
IEEE-89, Artificial Intelligence Applications, Proceedings, 13-19 (1989).

Kessler A.,
Semiconductor and semiconductor equipment industries,
Paine Webber, (Nov 1988).

Kittel Th.,
Produktionsplanung und -steuerung im Klein- und Mittelbetrieb,
Chancen und Risiken des EDV-Einsatzes (in German),
Dissertation,
Expert Verlag, Grafenau (1982).

Lawton W. L., Drake A., Henderson R., Wein L. M., Whitney R., Zuanich D.,
Workload regulating wafer release in a GaAs fab facility,
Int'l Semiconductor Manufacturing Science Symposium,
Proceedings, 33-38 (1990).

Law M. D., Kelton W. D.,
Simulation modelling and analysis
McGraw-Hill Book Company, New York, (1982).

Little J. D. C.,
A proof for the queueing formula: L = lambda W,
Operations Research 19 (3), (1961).

Lou S. X. C., Kager P. W.,
A robust production control policy for VLSI wafer fabrication,
IEEE Transactions on Semiconductor Manufacturing 2 (4), 159-164 (1989).

Lozinski C, Glassey C. R.,
Bottleneck starvation indicators for shop floor control,
IEEE Transactions on Semiconductor Manufacturing 1 (4), 147-153 (1988).

Maekawa M., Oldehoeft A. E., Oldehoeft R. R.,
Operating systems, Advanced concepts,
The Benjamin/Cummings Publishing Company, Menlo Park CA (1987)

Martin-Vega L. A., Pippin M., Gerdon E., Burcham R.,
Applying just-in-time in a wafer fab: a case study,
IEEE Transactions on Semiconductor Manufacturing 2 (1), 16-22 (1988).

Matsuyama A., Atherton R. W.,
Experience in simulating wafer fabs in the USA and Japan,
Int'l Semiconductor Manufacturing Science Symposium, Proceedings, 113-118 (1990).

Miller D. J.,
Simulation of a semiconductor manufacturing line,
Communications of the ACM 33 (10), 98-108 (Oct 1990)

Mommers E. P. M.,
Een regelaar voor het besturen van discrete produktiesystemen (in Dutch),
Master's thesis, Faculty of Mechanical Engineering, Eindhoven University of Technology, Eindhoven (1990).

Montazeri M.,
A modular simulator for design, planning, and control of flexible manufacturing systems,
Dissertation,
Katholieke Universiteit Leuven, Leuven (1987).

Nauta D.,
Logica en model (in Dutch),
Wetenschappelijke Uitgeverij, Amsterdam (1974).

Overwater R.,
Processes and interactions, An approach to the modelling of industrial systems,
Dissertation,
Eindhoven University of Technology, Eindhoven (1987).

Panwalker S. S., Iskander W.,
A survey of scheduling rules,
Operations Research 25 (1) 45-61 (1977).

Penning W.,
Vergelijkend onderzoek naar de invloed van een automatisch transportsysteem op het logistiek gedrag van een ets-cel (in Dutch),
Master's thesis, Faculty of Mechanical Engineering, Eindhoven University of Technology, Eindhoven (1991).

Peterson J. L., Silberschatz A.,
Operating system concepts,
Addison-Wesley Publishing Company, Reading MA (1986).

Pollak S.,
Wafer fabrication factory simulation language,
Int'l Semiconductor Manufacturing Science Symposium, Proceedings, 114-116 (1989).

Rijn Th. M. J. van,
Produceren door informeren, Informatie-eisen voor verschillende produktie situaties (in Dutch),
Kluwer, Deventer (1986).

Rijn Th. M. J. van,
Het ontwerpen van een systeem voor produktiebeheersing; een balancerings-vraagstuk (in Dutch),
Besturingsconcepten wat zijn ze uw bedrijf waard?, Themadag Nevem, V.V.W., Proceedings, (Sep 1988).

Rodammer F. A., White K. P. sr.,
A recent survey of production scheduling,
IEEE Transactions on Systems, Man, and Cybernetics 18 (6) 841-851 (1988).

Rooda J. E.,
De kunst van het automatiseren (in Dutch),
Inaugural Address,
Eindhoven University of Technology, Eindhoven (1987).

Rooda J. E.,
The modelling of industrial systems,
Eindhoven University of Technology, Eindhoven (1990).

Rooda J. E.,
Procescalculus, Indeling van industriele systemen (in Dutch),
I2 Werktuigbouwkunde 7 (5) 13-15 (1991a).

Rooda J. E.,
Procescalculus, Systemen, modellen en geschiedenis (in Dutch),
I2 Werktuigbouwkunde 7 (8) 36-39 (1991b).

Rooda J. E.,
Procescalculus, Definities en begrippen (in Dutch),
I2 Werktuigbouwkunde 7 (10) 35-40 (1991c).

Rooda J. E., Arentsen J. H. A.,
Procescalculus bij modelleren van flow-shop fabrieken (in Dutch),
Mechanische Technologie 1 (1) 10-20 (1991).
Rooda J. E., Arentsen J. H. A., Smit G. H.,
Procescalculus bij modelleren van job-shop fabrieken (in Dutch),
Mechanische Technologie 2 (2) (1992).
Ruissen E. M.,
Het besturen van flexibele produktiesystemen (in Dutch),
Master's thesis, Faculty of Mechanical Engineering,
Eindhoven University of Technology, Eindhoven (1986).
Smedinga R.,
Simulatie en implementatie (in Dutch),
Addison-Wesley, Amsterdam (1988).
Smit G. H.,
De besturing van waferfabs (in Dutch),
Memorandum, Faculty of Mechanical Engineering,
Eindhoven University of Technology, Eindhoven (1988).
Smit G. H., Vaes H. J., Rooda J. E.,
Control of an IC production facility,
Posters for IOP-FOM days,
IC-EWT89.041 (1989).
Spur G., Stoeferle Th.,
Handbuch der Fertigungstechnik (in German),
BD. 1 - 5,
Carl Hanser Verlag, Munchen (1981).
Steijns E. M. H.,
Besturing van een hierarchische job shop met behulp van simulatiescheduling (in Dutch),
Master's thesis, Faculty of Mechanical Engineering,
Eindhoven University of Technology, Eindhoven (1991).
Stokey R. J.,
AI factory scheduling: multiple problem formulations,
Sigart Newsletter, (110) 27-30 (1989).
Sze S. M.,
VLSI technology,
McGraw-Hill Book Company, Singapore (1983).
Tullis B., Mehrotra V., Zuanich D.,
Successful modeling of a semiconductor R&D facility,
Int'l Semiconductor Manufacturing Science Symposium,
Proceedings, 26-32 (1990).

Uzsoy R., Martin-Vega L. A., Brown S. M., Leonard P. A.,
Production scheduling algorithms for a semiconductor test facility,
Int'l Semiconductor Manufacturing Science Symposium, Proceedings, 25-31 (1989).

Vaes H. J.,
Vertex en scheduling (in Dutch),
Master's thesis, Faculty of Mechanical Engineering, Eindhoven University of Technology, Eindhoven (1989).

Verhoef H. A.,
Onderzoek naar de verbetering van jobscheduling op celniveau binnen een waferfabriek (in Dutch),
Memorandum, Faculty of Mechanical Engineering, Eindhoven University of Technology, Eindhoven (1989).

Vincenten J. F. P. M.,
Besturing van een job shop fabriek met behulp van een op het Kanban-principe gebaseerd besturingssysteem (in Dutch),
Master's thesis, Faculty of Mechanical Engineering, Eindhoven University of Technology, Eindhoven (1991).

Warnecke, H. -J., Fruehauf W., Schmutz W.,
New manufacturing concepts for the production of integrated circuits,
Semicon/Europa '90 Technical Conference, Proceedings, 169-179 (Mar 1990).

Wein L. M.,
Scheduling semiconductor wafer fabrication,
IEEE Transactions on Semiconductor Manufacturing 1 (3), 115-130 (1988).

Wiendahl H. -P.,
Belastungsorientierte Fertigungssteuerung:Grundlagen, Verfahrensaufbau, Realisierung (in German),
Carl Hanser Verlag, Munchen (1987).

Wortmann A. M., Rooda J. E., Boot W. C.,
Basisbegrippen van de process-interactie benadering (in Dutch),
Memorandum, Faculty of Mechanical Engineering, Eindhoven University of Technology, Eindhoven (1989).

Wortmann A. M., Rooda J. E.,
The process-interaction environment user manual,
Memorandum, Faculty of Mechanical Engineering, Eindhoven University of Technology, Eindhoven (1990).

Wortmann A. M.,
Modelling and simulation of industrial systems,
Dissertation,
Eindhoven University of Technology, Eindhoven (1991).

Appendix A
An introduction to Smalltalk-80

The Xerox Palo Alto Research Center started in the early 1970's a project to create a powerful information system, which made it possible to use computing power effectively and easy. This research resulted in the Smalltalk-80 system which is described in Goldberg and Robson [1989] and Goldberg [1984]. This appendix is based on the first text.

The Smalltalk-80 system is more than a programming language. The researchers at Palo Alto concentrated on two areas: a programming language in which a human can describe the models he has in his mind, and that can be executed on a computer, and a user interface which enables the human to communicate with the computer in a user friendly way.

The Smalltalk-80 system is an interactive programming environment. In this environment every component can be observed and manipulated. In order to use the Smalltalk-80 system a high resolution graphical display screen and a pointing device (such as a mouse) are essential.

The Smalltalk-80 system is a large system, that includes objects that can perform functions which are usually provided by the computer operating system. The system is based on the concepts object, class, message, method and inheritance. Smalltalk consists of communicating objects. The interaction between objects is viewed the same way on every level of complexity. Because the functioning of objects does not depend on the internal details of other objects, modularity is supported. The complexity is reduced by minimizing of interdependencies between objects and by grouping similar objects together in classes. The subclassing mechanism supports inheritance which avoids repetition of the same code in different places. Classes and instances are units for organizing and sharing information. Subclassing is a means to inherit and to refine existing capabilties.

A.1 Basic Smalltalk-80 concepts

Programming in Smalltalk-80 is facilitated by the fact the user can use everything that is already in the system. In order to write a new program the programmer has to find out what existing concepts he can use and

which new situation he wants to model, then establishing the new situation means programming the difference.

Before introducing the syntax of the Smalltalk-80 language an overview of the main concepts will be presented.

Object

The Smalltalk-80 programming language is an object oriented programming language. Every component in Smalltalk-80 is an object. This may be a representation of something physical or non-physical. An object consists of some private memory and a set of operations. This set of operations represent the functions that an object can perform. The private memory represents the data structure of an object.

Class

If every single object in Smalltalk would have a description of its properties the system would not be manageable. To avoid this problem abstraction is used. Objects with equal properties get a generalised description: a class. The concept class is comparable to the type definition concept in Pascal. In a Smalltalk class the implementation of the private memory and the set of operations of objects of the same kind are described. Every object in Smalltalk belongs to a class. An object which is described by a class is called an instance of that class.

Message

Most objects have a contents, the private memory. The contents of an object is not directly available to other objects. The contents of an object can only be manipulated with the help of messages. A message tells the object what operation it should perform. The object can react in an appropriate way by executing the requested operation. A message does not tell how to perform an operation. A message can be accompanied by arguments. Messages are the only way to access the operations of an object (encapsulation) and operations are the only way to acces the private memory of an object (data hiding). Because of the message mechanism, the implementation of operations an object can perform, remain private to the object, just like the data structure of its internal memory. All messages an object understands are called its interface.

Method

An object knows for every message it can perform a method that describes the way in which the requested operation has to be executed. A method is a procedure abstraction comparable to a function in Pascal. A method may specify changes of the object's private memory or it may contain messages for other objects. The methods an object knows belong to the class, which the object is an instance of, or to a superclass of this class.

Inheritance

A class is a specialization of another class: it is a subclass. On the other hand it may be a generalization of another class: then it is a superclass. A property of the subclassing mechanism is called inheritance. Inheritance means that every object of a certain class will have all properties of that class and all the properties of all superclasses of that class. This inheritance mechanism counts both for the methods and the private memory of a superclass. An object understands the messages defined in its class and those defined in its superclasses. The correct execution of the method that belongs to a message is handled by the language mechanism.

Polymorphism and late binding

A variable may refer to an instance of any class, this is called polymorphism. The method, that is to be executed when a message is sent to an object, is selected at run-time. The concept is called late binding.

A.2 The Smalltalk-80 syntax

The Smalltalk-80 system components are represented by objects. These objects, which are instances of classes, interact with each other with the help of messages. A message causes a method to be executed. Now the syntax for describing objects and messages will be presented.

An object is described by a sequence of characters this is called an expression, the object is called the value of the expression. In the Smalltalk-80 programming language there are four types of expressions: literal expressions, variable names, message expressions and block epressions.

Literal expressions

Literals describe constant objects. There are five kinds of objects that can be referred to by a literal expression: numbers, individual characters, strings of characters, symbols and arrays of literal expressions.

Examples of numbers:

```
1 -10 3.5 7.8e3
```

individual characters:

```
$a $b $c
```

strings of characters:

```
abc' 'defghi'
```

symbols:

```
#idle #busy
```

arrays of literal expressions:

```
#($a $b $c)
#( #( #a11 #a12) #( #a21 #a22))
```

Variable names

The private memory of an object consists of variables (instance variables). Most of these variables have names. Each variable remembers a single object and the variable's name can be used as an expression referring to that object. A variable name is a simple identifier, a sequece of letters and digits beginning with a letter.

There are two kinds of variables in the system, distinguished by how widely they are accessible. Private variables are accessible only to a single object. Instance variables are private. Shared variables can be accessed by more than one object. Private variable names are required to have lowercase initial letters; shared variable names are required to have uppercase initial letters.

Examples of variable names:

```
contents, name
```

shared variable names:

```
Pi
```

A literal constant will always refer to the same object, but a variable name may refer to different objects at different times. The object referred to by a variable is changed when an assignment expression is evaluated. Any expression can become an assignment by including an assignment prefix. An assignment prefix is composed of the name of the

variable whose value will be changed followed by a colon and an equal sign.

Example

asignment expression:

```
name := 'Charles'
```

A pseudo-variable name is an identifier that refers to an object, a pseudo-variable name is different from a variable name in that its value cannot be changed with an assignment expression. Some of the pseudo-variables in the system are constants; they always refer to the same objects. Three important pseudo-variable names are nil, true and false.

Message expressions

Messages represent interactions between components of the Smalltalk-80 system. A message requests an operation on the part of the object which gets the message.

A message expression describes an object, which should perform the message, a selector and possibly some arguments. The object and arguments are described by other expressions. The selector is specified literally. A message's selector is a name for the type of interaction which is requested from the object. The selector of a message determines which operation will be invoked. The arguments are other objects that are involved in the selected operation.

Example of message expression:

```
aCollection add: anObject
```

where aCollection should perform the message, add: anObject is a message, add: is a selector and anObject is an argument

Unary messages are messages without arguments.

Example of unary message:

```
size
```

The general type of message with one or more arguments is the keyword message. The selector of a keyword message is composed of one or more keywords, one preceding each argument. A keyword is a simple identifier with a trailing colon. When the selector of a multiple keyword message is referred to indepently, the keywords are concatenated.

Example of keyword message:

```
at: index put: anObject
```

where at:put: is a selector, at: is a keyword, put: is a keyword

There is one other type of message expression that takes a single argument, the binary message. A binary message selector is composed of one or two nonalphanumeric characters.

Example of binary message:

```
< 10
```

Smalltalk-80 messages provide two-way communication. The selector and argument transmit information to the object about what type of response to make. The object transmits information back by returning an object that becomes the value of the message expression. If a message expression includes an assignment prefix, the object returned by the object that performs the message, will become the new object referred to by the variable. Even if no information needs to be communicated back, an object always returns a value for the message expression.

Parsing rules:

1. Unary expressions parse left to right.

For instance `2 sin sqrt` evaluates as: `(2 sin) sqrt`.

2. Binary expressions parse left to right.

For instance `1 + 2 * 3` evaluates as: `(1 + 2) * 3`.

3. Binary expressions take precedence over keyword expressions.

For instance `1 + 2 raisedTo: 0.5` evaluates as: `(1 + 2) raisedTo: 0.5`.

4. Unary expressions take precedence over binary expressions.

For instance `1 + 2 sqrt` evaluates as: `1 + (2 sqrt)`.

There is one special syntactic form called cascading that specifies multiple messages to the same object. A cascaded message expression consists of one description of the object which should perform the messages, followed by several messages separated by semicolons.

Example of cascading:

```
aCollection add: objectOne; add: objectTwo; add: objectThree
```

Block expressions

Blocks are objects used in many of the control structures in the Smalltalk-80 system. A block represents a deferred sequence of actions. A block expression consists of a sequence of expressions separated by

periods and delimited by square brackets. When a block expression is encountered, the statement enclosed in the brackets are not executed immediatly. The value of a block expression is an object that can execute these enclosed expressions at a later time, when requested to do so. The sequence of actions will take place when the block receives the unary message value.

Example of block:

```
[i := i + 1. sum := sum + i]
```

A control structure determines the order of some activities. The fundamental control structure in the Smalltalk-80 language provides that a sequence of expressions will be evaluated sequentially. Many non sequential control structures are invoked either by sending a message to a block or by sending a message with one or more blocks as arguments. The response to one of these control structure messages determines the order of activities with the pattern of value messages it sends to the block(s).

An example of a control structure implemented with blocks is simple repetition, represented by a message to an integer with timesRepeat: as the selector and a block as the argument. The integer will respond by sending the block as many value messages as its own value indicates.

Example of simple repetition:

```
i := 0. sum := 0.
10 timesRepeat: [i := i + 1. sum := sum + i]
```

Two common control structures implemented with blocks are conditional selection and conditional repetition. Conditional selection is similar to the if-then-else statement in Pascal and conditional repetition is similar to the while-do and repeat-until statements in this language. These conditional control structures use two Boolean objects named true and false. Booleans are returned from messages that ask simple yes-no questions (for example, the magnitude comparison messages: =, <, <=, >, >=, ~=).

The conditional selection of an activity is provided by a message to a boolean with the selector ifTrue:ifFalse: and two blocks as arguments. The only objects that understand ifTrue:ifFalse: messages are true and false. They have opposite responses: true sends value to the first argument block and ignores the second; false sends value to the second argument block and ignores the first. The value returned from ifTrue:ifFalse: is the value of the block that was executed.

Example of conditional selection:

```
x > max
    ifTrue: [y := max]
```

```
    ifFalse: [y := x]
```

Other conditional selections are: ifTrue:, ifFalse: and ifFalse:ifTrue:.

The conditional repetition of an activity is provided by a message to a block with the selector whileTrue: and another block as an argument. The receiver block sends itself the message value and if the response is true, it sends the other block value and then starts over, sending itself value again. When the receiver's response to value becomes false, it stops the repetition and returns from the whileTrue: message.

Example of conditional repetition:

```
i := 0. sum := 0.
[i < 10]
    whileTrue:
        [i := i + 1. sum := sum + i]
```

Other message for conditional repetition is: whileFalse:.

In order to make some nonsequential control structures easy to express, blocks may take one or more arguments. Block arguments are specified by including identifiers preceded by colons at the beginning of a block. The block arguments are separated from the expressions that make up the block by a vertical bar.

A common use of blocks with arguments is to implement functions to be applied to all elements of a data structure. For example many objects representing different kinds of data structures respond to the message do:, which takes a single-argument block as its argument. The object that performs a do: message evaluates the block once for each of the elements contained in the data structure. Each element is made the value of the block argument for one evaluation of the block.

Example of enumeration over an array:

```
sum := 0.
#(1 2 3 4 5 6 7 8 9 10) do: [ :i | sum := sum + i]
```

where #(1 2 3 4 5 6 7 8 9 10) is an array that can perform a do: message, [:i | sum := sum + i] is a single argument block, i is a block argument, sum := sum + i is the expression in the block.

Other enumeration messages are: collect:, select:, reject:, detect: and inject:into:.

The objects that implement these control structures supply the values of the block arguments by sending the block the message value:. A block with one block argument responds to value: by setting the block argument to the argument of value: and then executing the expressions in the block.

Appendix B
Basic task language methods

This appendix lists the most important methods of the task language. It is based on Wortmann [1991].

Bubble > task frame

initializeTasks
"This method is called before any processor executes initialActions or body. It should not contain any send or receive actions, as the processes are not running yet. It is mainly intended to initialize instance variables."

initialActions
"This method is called once before the first execution of body."

body
"This method is called repeatedly during simulation. It must be redefined by all subclasses. Calling stopProcess prevents further calls of this method. "

haltSimulation
"Stop the present simulation. The effect is analogous to pushing the stop-button on the control panel."

stopProcess
"Prevent this bubble from executing any further actions during the present simulation."

Bubble > activity

These methods are used to simulate an activity in no more detail than the fact that it takes a certain amount of time.

workDuring: timeDelay
"The process will be busy for timeDelay time units. The status associated with this activity is busy."

workDuring: timeDelay forReason: workStatus
"The process will be busy for timeDelay time units. The status associated with this activity is workStatus."

workDuring: timeDelay forReason: workStatus interruptFrom: portName
"The process will be busy for timeDelay time units, unless an item is received from portName before timeDelay is expired. The status associated with the activity is provided by workStatus. Return whether the activity terminated without interrupt."

workDuring: timeDelay forReason: workStatus interruptFrom: portName if: condition
"The process will be busy for timeDelay time units, unless an item that satisfies condition is received from portName before timeDelay is expired. condition is a block that will be evaluated with candidate items as the single argument. It should have no side-effects. The status associated with the activity is provided by workStatus. Return whether the activity terminated without interrupt."

Bubble > sending objects

send: object to: portName
"The most basic send action. Sends object synchronously to the port specified by portName. The process blocks until a matching receive is performed by another processor."

send: object immediateTo: portName
"Behaves exactly like a normal send when sending at this moment is possible. If it blocks, which is detected a little later, an error message appears in the console."

send: object immediateTo: portName then: thenBlock else: elseBlock
"Try to send object to portName at this moment. If that succeeds, evaluate the thenBlock, if it does not succeed, evaluate elseBlock. Thus this send cannot block."

send: object before: aTime to: portName then: thenBlock else: elseBlock
"Try to send object to portName before aTime. If that succeeds, evaluate the thenBlock, if it does not succeed, evaluate elseBlock."

send: object to: portName1 then: block1 orTo: portName2 then: block2
"Try to send object to portName1 or portName2. Block until sending to one of the ports succeeds. If sending to both portnames would be possible, use the one that has the longest waiting receiver. Evaluate the corresponding block when sending has succeeded."

send: obj toOneOf: portNames
"portNames is a collection of send port names. Try to send to one of these. When there are waiting receivers, the longest waiting is used, otherwise the first receiver that becomes available will be used. After a succesful send the method returns, the processor will not try to send to the other ports as well. Return the portName that was used for the send."

send: obj toOneOf: portNames then: actionBlock
"actionBlock is a one argument block, the argument specifies the portName to which the send succeeded."

send: object continuousTo: portName
"Send object to portName. It will be available for an unlimited number of receivers until it is replaced by a new call to this method. This send does never block."

send: object asynchronousTo: portName
"Send object to portName. object will be buffered until a receiver is available. So this processor will not block. The size of the buffer is unlimited."

Bubble > receiving objects

receiveFrom: portName
"Receive from the specified port. Block until some sender is available for communication. Return the item received."

receiveFrom: portName if: conditionBlock
"conditionBlock is a one-argument block. It is evaluated with the candidate item as argument. Evaluation should have no side effects and must return a Boolean. The condition must not involve values which change on their own, such as the simulation time. This message returns the item received."

receiveImmediateFrom: portName if: conditionBlock
"Behaves exactly like a normal receive when receiving at this moment is possible. If it blocks, this blocking is only detected a little later, then an error message appears in the console."

receiveImmediateFrom: portName if: conditionBlock then: thenBlock else: elseBlock
"Try to receive an object that has been sent at an earlier moment and that satisfies conditionBlock. If there is such an object, execute thenBlock with the received object as the single argument. Otherwise execute elseBlock (no arguments)."

receiveFrom: portName before: time then: thenBlock ifTimedOut: timeOutBlock
"If an item is received before time, thenBlock is evaluated with that item as the single

argument. Otherwise, timeOutBlock is evaluated (no arguments)."

receiveFrom: portName before: time if: conditionBlock then: thenBlock ifTimedOut: timeOutBlock

"If an item is received that satisfies conditionBlock before time, thenBlock is evaluated with that item as the single argument. Otherwise, timeOutBlock is evaluated."

receiveFrom: portName within: anInterval then: thenBlock ifTimedOut: timeOutBlock

"If an item is received within anInterval, thenBlock is evaluated with that item as the single argument. Otherwise, timeOutBlock is evaluated."

receiveFrom: portName within: anInterval if: conditionBlock then: thenBlock ifTimedOut: timeOutBlock

"If an item is received that satisfies conditionBlock within anInterval, thenBlock is evaluated with that item as the single argument. Otherwise, timeOutBlock is evaluated."

receiveFrom: portName1 then: block1 or: portName2 then: block2

"Receive an item from either one of two ports. Evaluate the corresponding block with the received item as the single argument."

receiveFromOneOf: portNames

"portNames is a collection of receive port names. Try to receive from one of these. When there are waiting senders, the longest waiting is used, otherwise the first sender that becomes available will be used. After the succesful receive the method returns the item received; the processor will not try to receive from the other ports as well."

receiveFromOneOf: portNames do: actionBlock

"actionBlock is a two-argument block. When an item is available, actionBlock is evaluated with the name of the port involved as the first argument and the item as the second argument."

receiveFromOneOf: portNames if: conditionBlock do: actionBlock

"When an item that satisfies conditionBlock is available, actionBlock is evaluated with the name of the port involved as the first argument and the item as the second argument."

Bubble > special communications

send: item to: sendPort then: sendBlock orReceiveFrom: receivePort if: cond then: receiveBlock

"Send the item to the sendPort, or receive an item from the receivePort that satisfies cond, depending on which communication succeeds first. The condition is either nil (no condition) or a one argument block."

send: item to: sendPort then: sendBlock orReceiveFrom: receivePort if: cond then: receiveBlock within: timeInterval ifTimedOut: timeOutBlock

"Send the item to the sendPort, or receive an item from the receivePort that satisfies cond, depending on which communication succeeds first. If communication does not succeed within timeInterval, evaluate timeOutBlock."

send: item toOneOf: sendPorts do: sendBlock orReceiveFromOneOf: receivePorts if: cond do: receiveBlock

"Send the item to one of the sendPorts, or receive an item that satisfies cond from one of the receivePorts, depending on which communication succeeds first. sendBlock is a one argument block; the argument specifies the portname at which sending succeeded. receiveBlock is a two argument block; the arguments are the portname and the received object."

send: item toOneOf: sendPorts do: sendBlock orReceiveFromOneOf: receivePorts if: cond do: receiveBlock within: timeInterval ifTimedOut: timeOutBlock

"Send the item to one of the sendPorts, or receive an item that satisfies cond from one of the receivePorts, depending on which communication succeeds first. If communication does not succeed within timeInterval, evaluate timeOutBlock."

Bubble > accessing

The following messages do not implement an action but provide control or access to the model.

children
"Return a collection with all the processors of my expansion"
name
"Return my name, a String"
newName: string
"Set the name of the receiver to the argument."
parent
"Return the parent processor of the receiver."

Bubble > testing

IsPartOfClass: class
"Answer whether the receiver is a child (or a child of a child etc.) of a processor of the class specified by the argument."
IsPartOfProcessorNamed: aName
"Answer whether the receiver is a child (or a child of a child etc.) of a processor named as the argument."

Appendix C
The hierarchical control model methods

This appendix lists the objects and the most important methods of the data structure and the control model. The aim of this appendix is to give the interested reader a more detailed description of the hierarchical control model and its data structure. It gives implementations of interesting methods. Most of the objects have been discussed in Section 4.1 and 4.2.

The class hierarchy of the objects is given below.

The items that are exchanged between processors (Section C.1).

```
Object
    InteractionObject
        Order
            RealOrder
            PotentialOrder
        Quotation
        Invoice
        Request
            TransportRequest
        Report
            TransportReport


OrderedCollection
    InteractionCollection
        Job
            TransportJob
        MaterialRequest
        Material
```

The object for the representation of material (Section C.2).

```
Object
    MaterialUnit
```

Items needed for the administration of the manufacturing process (Section C.3).

```
Object
    ProgressForm
        JobProgressForm
        OrderProgressForm
    Operation
        TransportOperation
        ProcessOperation
```

WorkUnit
ProcessUnit
TransportUnit

OrderedCollection
Task

The objects where the calculation for the different decisions is done (Section C.4).

Object
ProcessPlanner
FactoryPlanner
JobScheduler
LateScheduler
EarlyScheduler

The processors of the model (Section C.5).

Bubble
ProcessorObject
ManufacturingController
FactoryController
Resource
LeafResource
Transformer
Store
Transporter
Consumer
Supplier

EnvironmentProcess
Market

C.1 Interaction items

InteractionObject - instance protocol

Object subclass: InteractionObject
instanceVariableNames: 'address arrivalDate'

accessing methods

address
"Return the address (processor to which this object has to be sent) of the object.."
arrivalDate
"Return the time the object arrived at its address."
stayTime
"Return the time passed since the arrival of the object.
setAddress: aBubble
"Assign a value to the address of the object."
setArrivalDate: aTime
"Assign a value to the arrival date of the object."

Order - instance protocol

InteractionObject subclass: Order
    instanceVariableNames: 'product amount supplier consumer startDate dueDate progressForm'

accessing methods

```
amount
consumer
dueDate
product
progressForm
startDate
supplier
```

Order - class protocol

instance creation method

```
product: aProductName amount: aNumber supplier: aSupplierName consumer: aConsumerName startDate: startTime dueDate: dueTime progressForm: aProgressForm
    "Create a new order."
```

RealOrder - instance protocol

Order subclass: RealOrder
    instanceVariableNames: ''

testing methods

```
isPotential
    ^false
isReal
    ^true
```

RealOrder - class protocol

instance creation method

```
from: aQuotation
    "create a new order from a quotation"
```

PotentialOrder - instance protocol

Order subclass: PotentialOrder
    instanceVariableNames: ''

testing methods

```
isPotential
```

```
    ^true
isReal
    ^false
```

Quotation - instance protocol

InteractionObject subclass: Quotation
    instanceVariableNames: 'order accepted'

accessing methods

```
order
progressForm
testing methods
isAccepted
    ^accepted
isInvoice
    ^false
isQuotation
    ^true
```

Quotation - class protocol

instance creation methods

```
accept: anOrder
    "create a quotation that accepts the order"
reject: anOrder
    "create a quotation that rejects the order"
```

Invoice - instance protocol

InteractionObject subclass: Invoice
    instanceVariableNames: 'order material'

accessing methods

```
dueDate
material
order
progressForm
```

testing methods

```
isInvoice
    ^true
isQuotation
    ^false
```

Invoice - class protocol

instance creation methods

```
order: anOrder material: anOrderedCollection
```

```
"Create a new invoice. The delivered material is mentioned in anOrderedCollection,
the order for the material in anOrder."
```

Request - instance protocol

InteractionObject subclass: Request
    instanceVariableNames: 'resource operationTypes
minBatchSize maxBatchSize'

accesssing methods

```
maxBatchSize
minBatchSize
resourceName
resourceType
scheduleDate
newJob
    ^Job for: resource
```

testing methods

```
isAbleToExecute: aWorkUnit
    ^(operationTypes includes: aWorkUnit operationType)
isReport
    ^false
isRequest
    ^true
isTransportReport
    ^false
isTransportRequest
    ^false
```

Request - class protocol

instance creation method

```
resource: aResource operationTypes: operationTypeSet minBatchSize:
minInteger maxBatchSize: maxInteger
    "Create a new request."
```

TransportRequest - instance protocol

Request variableSubclass: TransportRequest
    instanceVariableNames: ''

accesssing methods

```
newJob
    ^TransportJob for: resource
```

testing methods

```
isRequest
    ^false
isTransportRequest
```

```
    ^true
```

Report - instance protocol

InteractionObject subclass: Report
    instanceVariableNames: 'job'

accessing methods

```
job
```

testing methods

```
isReport
    ^true
isRequest
    ^false
isTransportReport
    ^false
isTransportRequest
    ^false
```

Report - class protocol

instance creation method

```
job: aJob
    "Create a report belonging to aJob."
```

TransportReport - instance protocol

Report subclass: TransportReport
    instanceVariableNames: ''

accessing method

```
destination
origination
testing method
belongsToFinishedTask
    ^job destination = 'store'
isReport
    ^false
isTransportReport
    ^true
```

InteractionCollection - instance protocol

OrderedCollection variableSubclass: InteractionCollection
    instanceVariableNames: 'address arrivalDate'

accessing methods (see InteractionObject)

address
arrivalDate
setAddress: aBubble
setArrivalDate: aNumber
stayTime

Job - instance protocol

InteractionCollection variableSubclass: Job
instanceVariableNames: 'resource dueDate'

adding method

```
addWorkUnit: aWorkUnit
    (dueDate isNil or: [aWorkUnit dueDate < dueDate])
        ifTrue: [dueDate := aWorkUnit dueDate].
    self add: aWorkUnit
```

accessing methods

dueDate
material
operation
operationType
processTime
resourceName

Job - class protocol

instance creation method

```
for: aResource
    "Create a new job for aResource."
```

TransportJob - instance protocol

Job variableSubclass: TransportJob
instanceVariableNames: ''

accessing method

destination
origination

MaterialRequest - instance protocol

InteractionCollection variableSubclass: MaterialRequest
instanceVariableNames: 'destination'

accessing method

destination

```
^destination
```

MaterialRequest - class protocol

instance creation method

```
destination: aBubble
    "Create a new materialRequest, the destination of the requested material is
    aBubble."
```

Material - instance protocol

```
InteractionCollection variableSubclass: Material
    instanceVariableNames: ''
```

C.2 Material object

MaterialUnit - instance protocol

```
Object subclass: MaterialUnit
    instanceVariableNames: 'name'
```

MaterialUnit - class protocol

```
MaterialUnit class
    instanceVariableNames: 'instanceCount'
```

class initialization method

```
initialize
    instanceCount := 0
```

instance creation method

```
nameForInstance
    instanceCount := instanceCount + 1.
    ^name, instanceCount printString
new
    "Create a new instance of MaterialUnit with a unique name."
    ^super new setName: self nameForInstance
```

C.3 Administrative objects

ProgressForm - instance protocol

Object subclass: ProgressForm
instanceVariableNames: 'instantiator tasks'

"The instantiator is the object to which the progressForm belongs. The tasks contain the process plans which have to be executed in order to execute the instantiator."

tasks-accessing methods

```
addTask: aTask
    ^tasks add: aTask
isFinished
    ^tasks isEmpty
removeTask: aTask
    ^tasks remove: aTask
tasks
    ^tasks
```

administrating methods

```
materialContents
    ^instantiator materialContents
workContents
    ^self materialContents * self operation processTime
```

JobProgressForm - instance protocol

ProgressForm subclass: JobProgressForm
instanceVariableNames: ''

accessing methods

```
job
material
operation
report
    "Create a new report when the job is executed."
    ^Report job: instantiator
```

JobProgressForm - class protocol

instance creation methods

```
job: aJob
    "Create a new instance of JobProgressForm."
```

OrderProgressForm - instance protocol

ProgressForm subclass: OrderProgressForm
instanceVariableNames: 'operation material orders accepted'

"The instance variable operation contains the operation that has to be executed in order to produce the products ordered. The instance variable material contains the raw material that is ordered for the manufacturing of the product. The instance variable orders contains the orders for raw material, accepted keeps the results of the orders in case these orders are potential."

accessing methods

```
order
material
operation
invoice
    "Return a new invoice when the order is fulfilled."
    ^Invoice order: instantiator material: material
```

orders accessing methods

```
addOrder: anOrder
    orders add: anOrder
handleInvoice: anInvoice
    "Administrate the answer of the supplier to a realOrder."
handleQuotation: aQuotation
    "Administrate the answer of the supplier to a potentialOrder."
isAccepted
    ^self isDelivered and: [accepted]
isDelivered
    ^orders isEmpty
isPotential
    ^instantiator isPotential
```

OrderProgressForm - class protocol

instance creation methods

```
order: anOrder operation: anOperation
    "create a new orderProgressForm"
```

Operation - instance protocol

Object subclass: Operation
    instanceVariableNames: 'operationType resourceType processTime'

accessing methods

```
operationType
processTime
resourceType
setProcessTime: aTime
    processTime := aTime
```

TransportOperation - instance protocol

Operation subclass: TransportOperation
    instanceVariableNames: 'origination destination'

accessing methods
destination
origination

TransportOperation - class protocol

instance creation method
from: anOrigination to: aDestination in: aTime
"Create a new transportOperation."

ProcessOperation - instance protocol

Operation subclass: ProcessOperation
instanceVariableNames: 'parameter'

accessing method
parameter

ProcessOperation - class protocol

instance creation method
operationType: anOperationType parameter: aParameter resourceType: aClassName
"Create a new operation."

WorkUnit - instance protocol

Object subclass: WorkUnit
instanceVariableNames: 'operation materialUnit task position dueDate scheduleDate'
"The instance variable scheduleDate is used for the implementation of the fifo rule, it contains the time at which the workUnit is added to a jobScheduler."

accessing methods
dueDate
materialUnit
operation
operationType
position
processTime
resourceType
scheduleDate
task
setPosition: aBubbleName
position := aBubbleName
setScheduleDate: anScheduleDate
scheduleDate := anScheduleDate
setTask: aTask

```
task := aTask
```

WorkUnit - class protocol

instance creation methods

operation: anOperation materialUnit: aMaterialUnit position: aBubbleName task: aTask dueDate: aDueDate
"Create a new workUnit"

ProcessUnit - instance protocol

WorkUnit subclass: ProcessUnit
instanceVariableNames: ''

ProcessUnit - class protocol

instance creation method

operation: anOperation materialUnit: aMaterialUnit dueDate: aTime
"Create a new instance of ProcessUnit"

TransportUnit - instance protocol

WorkUnit subclass: TransportUnit
instanceVariableNames: ''

accessing method

destination
origination

Task - instance protocol

OrderedCollection variableSubclass: Task
instanceVariableNames: 'progressForm remainingProcessTime'

accessing methods

progressForm
remainingProcessTime
testing methods
isFinished
^self isEmpty

adding methods

addProcessUnit: aProcessUnit
super add: aProcessUnit.
aProcessUnit setTask: self.
remainingProcessTime := (remainingProcessTime + aProcessUnit processTime)

```
removeProcessUnit: aProcessUnit
    super remove: aProcessUnit.
    remainingProcessTime := (remainingProcessTime - aProcessUnit processTime)
```

Task - class protocol

instance creation method

progressForm: aProgressForm
"Create a new instance of task which is part of aProgressForm."

C.4 Calculators

ProcessPlanner - instance protocol

Object subclass: ProcessPlanner
instanceVariableNames: 'resource reports input weightedInput throughput weightedThroughput leadTime weightedLeadTime inventoryLevel weightedInventoryLevel'

"The processPlanner creates process plans for a particular resource. It also generates the report if the processPlans belonging to a job have finished. In the instance variables input, weightedInput, throughput, weightedThroughput, leadTime, weightedLeadTime, inventoryLevel and weightedInventoryLevel the performance of the resource is administrated."

handling methods

handleTransportReport: aTransportReport
"Handle aTransportReport, update the finished task, update the related progressForm. If the progressForm is finished, administrate the ready job and generate a report for the finished job."
makeProcessPlansFor: aJob
"Create the tasks (process plans) that have to be executed in order to execute aJob. aJob is also administrated as a new job."
reports
"Return the reports of the jobs that are ready."

administrating methods

administrateNewJobWith: aProgressForm
"Administrate the incoming new job."
administrateReadyJobWith: aProgressForm
"Administrate the leaving ready job."

ProcessPlanner - class protocol

instance creation method

for: aResource
"Create a new processPlanner."

FactoryPlanner - instance protocol

ProcessPlanner subclass: FactoryPlanner
instanceVariableNames: 'orders quotations invoices'

handling methods

handlePotentialOrder: aPotentialOrder
"Create new potential orders for raw material."
handleRealOrder: aRealOrder
"Create new real orders for raw material, and administrate the new aRealOrder."
handleQuotation: aQuotation
"Administrate aQuotation and generate a new quotation if all potential orders are answered."
handleInvoice: anInvoice
"Administrate anInvoice from the supplier. If all material is delivered, the tasks (process plans) are created."
handleTransportReport: aTransportReport
"Handle aTransportReport, update the finished task, update the related progressForm. If the progressForm is finished, administrate the ready job and generate an invoice for the finished order."

planning method

hasCapacityFor: aPotentialOrder
"Check if the resource has capacity to manufacture the amount of products requested in aPotentialOrder."

accessing methods

invoices
"Return the invoices of the orders that are ready."
orders
"Return the orders for the supplier that are created on the receipt of orders from the consumer."
quotations
"Return the quotations to the orders of the consumer. The quotations can be answered if the capacity planner has capacity and if the potential orders for raw material have been answered."

administrating methods

administrateNewOrderWith: anOrderProgressForm
"Administrate the incoming new order."
administrateReadyOrderWith: anOrderProgressForm
"Administrate the leaving ready order."

JobScheduler - instance protocol

Object subclass: JobScheduler
instanceVariableNames: 'controller resourceType requests workUnits'

initialize method

setController: aBubble setResourceType: aResourceType
```
controller := aBubble.
resourceType := aResourceType.
```

```
requests := SortedCollection sortBlock: self fifo.
workUnits := SortedCollection sortBlock: self fifo
```

priority rule methods

edd
```
^[:a :b | a dueDate <= b dueDate]
```
fifo
```
^[:a :b | a scheduleDate <= b scheduleDate]
```
srpt
```
^[:a :b | a task remainingProcessTime <= b task remainingProcessTime]
```

adding methods

scheduleRequest: aRequest
```
requests add: aRequest
```
scheduleWorkUnit: aWorkUnit
```
aWorkUnit setScheduleDate: controller time.
workUnits add: aWorkUnit
```

scheduling methods

executableJobs
"Find the combinations of workUnits and requests that can be combined to new jobs and create these."
findRequestFor: aWorkUnit
"Look for requests that can execute aWorkUnit."
formulateJobFor: aWorkUnit and: aRequest
"Formulate a new job from aWorkUnit and aRequest."

JobScheduler - class protocol

instance creation method

in: aBubble for: aResourceType
"Create a new jobScheduler."
```
^self new setController: aBubble setResourceType: aResourceType
```

LateScheduler - instance protocol

Object subclass: LateScheduler
instanceVariableNames: 'controller schedulers transportTaskSteps transportableJobs executableJobs reports'

handling methods

handleReport: aReport
"Update the tasks of which a processUnit has been executed. If the task is finished, then formulate transportUnits for transport to store, else schedule the task for the execution of the next processUnit of the task."
scheduleTask: aTask
```
self scheduleProcessUnit: aTask first
```
handleTransportReport: aTransportReport
"Check if all material of the job has been transported to a resource if this is the case then add the job to executableJobs."

scheduling methods

scheduleProcessUnit: aWorkUnit

```
    | resourceType |
    resourceType := aWorkUnit resourceType.
    (schedulers at: resourceType) scheduleWorkUnit: aWorkUnit
scheduleRequest: aRequest
    | resourceType |
    resourceType := aRequest resourceType.
    (schedulers at: resourceType) scheduleRequest: aRequest
```

accessing methods

executableJobs
"Return all jobs that may be sent to their resources."
transportUnits
"Return all transportUnits of material that has to be transported."

LateScheduler - class protocol

instance creation

for: anArrayOfResourceTypes in: aController
"Create a new scheduler for aController."

EarlyScheduler - instance protocol

LateScheduler subclass: EarlyScheduler
instanceVariableNames: 'destinations'

"The public messages of the class EarlyScheduler are the same as those of LateScheduler only the implementation differs and the sequence in which they are sent."

handling methods

handleReport: aReport
"Update the tasks of which a processUnit has been executed. If the task is finished, then formulate transportUnits for transport to store, else generate the transportUnits for transport of the material to the next resource."
handleTransportReport: aTransportReport
"Try to schedule the processUnits, belonging to transported material, on the resource."

accessing methods

executableJobs
"Return all jobs that may be sent to their resources."
transportUnits
"Return all transportUnits of material that has to be transported."

EarlyScheduler - class protocol

instance creation

for: anArrayOfResourceTypes in: aController
"Create a new scheduler for aController."

C.5 Processors

ProcessorObject - instance protocol

Bubble subclass: ProcessorObject
instanceVariableNames: 'addressTable portNameTable'

"The interactions to different processorObjects are multiplexed via one port. Below is illustrated with the send and receive interaction how the multiplexing works. The instance variable addressTable is a dictionary. It contains associations with a name and the linked address. The portNameTable is a dictionary which contains a name and the portName to which the linked address is connected."

initializing methods

```
initializeTasks
    addressTable := Dictionary new.
    portNameTable := Dictionary new.
    self initializeAddressTable
```

receiving methods

```
receiveFrom: portName
    "Additional code for multiplexing via one port and for time stamping the received
    object."
    | item |
    item := super receiveFrom: portName if: [ :item | item address == self].
    item setArrivalDate: self time.
    ^item
```

Other receiving methods are reimplemented in similar ways.

sending methods

```
send: object to: portName
    "Additional code for multiplexing via one port."
    ^super
        send: (object setAddress: (addressTable at: portName))
        to: (portNameTable at: portName)
```

Other sending methods are reimplemented in similar ways.

ManufacturingController - instance protocol

ProcessorObject subclass: ManufacturingController
instanceVariableNames: 'planner scheduler transportScheduler requestList'

"The planner is an instance of ProcessPlanner. The scheduler is either an instance of LateScheduler or EarlyScheduler. The transportScheduler is an instance of JobScheduler. The sending of requests is done with help of requestList. RequestList contains requests associated with the time the request has to be sent. The requestList is an instance of SortedCollection, the elements are sorted to their time they have to be sent. A request is placed in requestList with the next statement:
requestList add: self formulateRequest -> (timeToSendRequest)
When this is done depends on the request send strategy. For instance with a 'Fixed-

WIP' strategy it is done together with the sending of a report. With 'uniform starts' the sending of a request goes together with the adding of the next request to requestList."

simulation control methods

```
initializeTasks
    ...
    requestList := SortedCollection sortBlock: [:a :b | a value <= b value].
    ...
body
    self
        receiveFromOneOf: #('controller' 'resource' )
        before: self requestSendTime
        do:
            [ :portName :item |
            portName = 'controller'
                ifTrue:
                    [self handleJob: item.
                    self sendAvailableTransportJobs].
            portName = 'resource'
                ifTrue:
                    [item isRequest
                        ifTrue:
                            [self handleSubrequest: item.
                            self sendAvailableTransportJobs].
                    item isReport
                        ifTrue:
                            [self handleSubreport: item.
                            self sendAvailableTransportJobs].
                    item isTransportRequest
                        ifTrue:
                            [self handleTransportRequest: item.
                            self sendAvailableTransportJobs].
                    item isTransportReport
                        ifTrue:
                            [item belongsToFinishedTask
                                ifTrue:
                                    [self handleLastTransportReport: item.
                                    self sendAvailableReports]
                                ifFalse:
                                    [self handleTransportReport: item.
                                    self sendAvailableSubjobs]]]
        ifTimedOut: [self sendRequest]
handleJob: aJob
    | tasks |
    tasks := planner makeProcessPlansFor: aJob.
    tasks do: [:task | scheduler scheduleTask: task].
    self scheduleTransportUnits
handleLastTransportReport: aTransportReport
    planner handleTransportReport: aTransportReport
handleSubreport: aReport
    scheduler handleReport: aReport.
    self scheduleTransportUnits
handleSubrequest: aRequest
    scheduler scheduleRequest: aRequest.
    self scheduleTransportUnits
handleTransportReport: aTransportReport
    scheduler handleTransportReport: aTransportReport
```

```
handleTransportRequest: aTransportRequest
    transportScheduler scheduleRequest: aTransportRequest
scheduleTransportUnits
    scheduler transportUnits do:
        [ :transportUnit |
        transportScheduler scheduleWorkUnit: transportUnit]
requestSendTime
    "Return the time the next request has to be sent."
```

sending methods

```
sendAvailableReports
    | reports |
    reports := planner reports.
    reports do: [ :report | self send: report asynchronousTo: 'controller']
sendAvailableSubjobs
    | subjobs |
    subjobs := scheduler executableJobs.
    subjobs do:
        [ :subjob | self send: subjob asynchronousTo: subjob resourceName]
sendRequest
    | request |
    requestList isEmpty
        ifFalse:
            [request := requestList removeFirst key.
            self send: request asynchronousTo: 'controller']
sendAvailableTransportJobs
    | transportJobs |
    transportJobs := transportScheduler executableJobs.
    transportJobs do:
        [ :transportJob | self send: transportJob asynchronousTo: 'transporter']
```

FactoryController - instance protocol

ManufacturingController subclass: FactoryController
instanceVariableNames: ' '

simulation control methods

```
body
    self receiveFromOneOf: #('consumer' 'supplier' 'resource' ) do:
        [ :portName :item |
        portName = 'consumer'
            ifTrue:
                [item isPotential
                    ifTrue:
                        [(planner hasCapacityFor: item)
                            ifTrue:
                                [self handlePotentialOrder: item.
                                self sendPotentialOrders]
                            ifFalse:
                                [self sendQuotations]].
                item isReal
                    ifTrue:
                        [self handleRealOrder: item.
                        self sendRealOrders]].
        portName = 'supplier'
            ifTrue:
```

```
                [item isQuotation
                    ifTrue:
                        [self handleQuotation: item.
                        self sendQuotations].
                item isInvoice
                    ifTrue:
                        [self handleInvoice: item.
                        self sendAvailableTransportJobs]].
        portName = 'resource'
            ifTrue:
                [item isRequest
                    ifTrue:
                        [self handleSubrequest: item.
                        self sendAvailableTransportJobs].
                item isReport
                    ifTrue:
                        [self handleSubreport: item.
                        self sendAvailableTransportJobs].
                item isTransportRequest
                    ifTrue:
                        [self handleTransportRequest: item.
                        self sendAvailableTransportJobs].
                item isTransportReport
                    ifTrue:
                        [item belongsToFinishedTask
                            ifTrue:
                                [self handleLastTransportReport: item.
                                self sendInvoices]
                            ifFalse:
                                [self handleTransportReport: item.
                                self sendAvailableSubjobs]]]
handleInvoice: anInvoice
    | tasks |
    tasks := planner handleInvoice: anInvoice.
    tasks do: [ :task | scheduler scheduleTask: task].
    self scheduleTransportUnits
handlePotentialOrder: aPotentialOrder
    (planner hasCapacityFor: aPotentialOrder)
        ifTrue: [planner handlePotentialOrder: aPotentialOrder]
handleQuotation: supplierQuotation
    planner handleQuotation: supplierQuotation
handleRealOrder: aRealOrder
    planner handleRealOrder: aRealOrder
```

shipping methods

```
shipProductsFor: anInvoice
    | shipJob |
    shipJob := MaterialRequest destination: 'outside'.
    shipJob addAll: anInvoice material.
    self send: shipJob asynchronousTo: 'store'
```

sending methods

```
sendInvoices
    | invoices |
    invoices := planner invoices.
    invoices do:
        [ :invoice |
        self shipProductsFor: invoice.
```

```
        self send: invoice asynchronousTo: 'consumer']
sendPotentialOrders
    | supplierOrders |
    supplierOrders := planner orders.
    supplierOrders do:
        [ :order | self send: order asynchronousTo: order supplier]
sendQuotations
    | consumerQuotations |
    consumerQuotations := planner quotations.
    consumerQuotations do:
        [ :quotation | self send: quotation asynchronousTo: 'consumer']
sendRealOrders
    | supplierOrders |
    supplierOrders := planner orders.
    supplierOrders do:
        [ :order | self send: order asynchronousTo: order supplier]
```

Resource - instance protocol

ProcessorObject subclass: Resource
    instanceVariableNames: ''

accessing methods

resourceName
resourceType
subresources
controller
store
transporter

Resource - class protocol

Resource class
    instanceVariableNames: 'maxInventoryLevel minBatchSize maxBatchSize recipes operationTypes'

"The variable recipes contains the recipes the resource can execute. Every recipe is associated with the operation that invokes the specific recipe. The operations of the recipe are in fact suboperations of this operation. The variable operationTypes contains the operationTypes the resource is able to execute."

default setting methods

```
setInventoryLevel: anInteger
    maxInventoryLevel := anInteger
setMaxBatchSize: anInteger
    maxBatchSize := anInteger
setMinBatchSize: anInteger
    minBatchSize := anInteger
```

accessing methods

maxInventoryLevel
maxBatchSize
minBatchSize
operationTypes

recipes
resourceType

recipes methods

atOperation: anOperation putRecipe: aRecipe
"Add a recipe to recipes."
initializeRecipeFor: anOperation
"This methods calculates the recipe that belongs to anOperation. It also calculates the parameter and the processTime of the operation. It initializes all the suboperations by sending the message initializeRecipeFor: anOperation to the subresources. The method delivers the initialized anOperation as return value"

LeafResource - instance protocol

Resource subclass: LeafResource
instanceVariableNames: ''

accessing methods

controller
^self
store
^self

Transformer - instance protocol

LeafResource subclass: Transformer
instanceVariableNames: ''

simulation control methods

body
| request rawMaterial job operation finishedMaterial report |
request := self formulateRequest.
self send: request to: 'controller'.
rawMaterial := self receiveMaterialFrom: 'outside'.
job := self receiveFrom: 'controller'.
operation := job operation.
finishedMaterial := self execute: operation on: rawMaterial.
report := self formulateReportFrom: job and: finishedMaterial.
self send: report to: 'controller'.
self sendMaterial: finishedMaterial to: 'outside'
execute: anOperation on: aMaterialCollection
self workDuring: anOperation processTime forReason: 'processing'.
^aMaterialCollection
formulateReportFrom: aJob and: aMaterialCollection
"Return a new report that contains the result of the execution of the job."
formulateRequest
"Return a new request that contains the capabilities of this leafResource."
receiveMaterialFrom: aPortName
| material |
material := self receiveFrom: aPortName.
^material
sendMaterial: aMaterialCollection to: aPortName
| materialRequest material |

```
[aMaterialCollection isEmpty]
    whileFalse:
        [materialRequest := self receiveFrom: aPortName.
        material := Material new.
        materialRequest do:
            [ :aMtl |
            aMaterialCollection remove: aMtl.
            material add: aMtl].
        material setAddress: materialRequest destination.
        self send: material to: aPortName]
```

Transformer - class protocol

recipes methods

initializeRecipeFor: anOperation

"This method calculates the parameter and the processTime of the operation. The method returns the initialized anOperation."

Store - instance protocol

LeafResource subclass: Store
instanceVariableNames: 'buffer'

simulation control method

```
body
    | material |
    self receiveFromOneOf: #('outside' 'inside' 'controller' ) do:
        [:portName :item |
        (item isKindOf: MaterialRequest)
            ifTrue:
                [material := self removeFromBuffer: item.
                self send: material to: item destination].
        (item isKindOf: Material)
            ifTrue:
                [self addToBuffer: item]]
```

adding methods

```
addToBuffer: aMaterialCollection
    buffer addAll: aMaterialCollection
removeFromBuffer: aMaterialRequest
    | material |
    material := Material new.
    aMaterialRequest do:
        [:aMtl |
        buffer remove: aMtl.
        material add: aMtl].
    material setAddress: aMaterialRequest destination.
    ^material
```

Transporter - instance protocol

LeafResource subclass: Transporter
instanceVariableNames: 'position'

simulation control method

```
body
    | request transportJob origination destination materialData material report |
    request := self formulateRequest.
    self send: request to: 'controller'.
    transportJob := self receiveFrom: 'controller'.
    origination := transportJob origination.
    destination := transportJob destination.
    materialData := transportJob material.
    self moveTo: origination.
    material := self pick: materialData from: origination.
    self moveTo: destination.
    self place: material at: destination.
    report := self formulateReportFrom: transportJob and: material.
    self send: report to: 'controller'
formulateMaterialRequestFor: aMaterialCollection from: aPortName
    "Return the material request to receive the material mentioned in
    aMaterialCollection."
formulateReportFrom: aJob and: aMaterialCollection
    "Return a report for the executed aJob."
formulateRequest
    "Return a new request."
moveTo: aPortName
    position = aPortName
        ifFalse:
            [self workDuring: self transportTime forReason: 'moving'.
            position := aPortName]
pick: aMaterialCollection from: aPortName
    | request material |
    request := self formulateMaterialRequestFor: aMaterialCollection from: aPortName.
    self send: request to: aPortName.
    aPortName = 'store'
        ifTrue: [material := self receiveFrom: 'store']
        ifFalse: [material := self receiveFrom: 'resource'].
    ^material
place: aMaterialCollection at: aPortName
    self send: aMaterialCollection to: aPortName.
```

Consumer - instance protocol

ProcessorObject subclass: Consumer
instanceVariableNames: 'orderTimeDistribution orderTime productTypeDistribution productAmountDistribution dueDateDistribution acceptionDistribution'

simulation control methods

```
body
    self
        receiveFrom: 'supplier'
```

```
        before: orderTime
        then: [:anItem | self handleSupplierItem: anItem]
        ifTimedOut: [self generateOrder]
generateOrder
    "Create a new order."
handleInvoice: anInvoice
    "Receive the material mentioned in the invoice and update the administration."
handleQuotation: aQuotation
    "Generate if necessary a new Real order and update the administration."
handleSupplierItem: anItem
    anItem isQuotation
        ifTrue: [self handleQuotation: anItem]
        ifFalse: [self handleInvoice: anItem]
```

Supplier - instance protocol

ProcessorObject subclass: Supplier
    instanceVariableNames: 'acceptionDistribution orders'

simulation control methods

```
body
    self
        receiveFrom: 'consumer'
        before: self nextDeliveryTime
        then: [ :anOrder | self handleOrder: anOrder]
        ifTimedOut: [self deliverNextProducts]
deliverNextProducts
    "Deliver material to the consumer and send an invoice."
handleOrder: anOrder
    "Administrate anOrder, if it is potential, a quotation is sent to the consumer."
nextDeliveryTime
    "Return the time the next order has to be delivered."
```

Market - instance protocol

EnvironmentProcess subclass: Market
    instanceVariableNames: ''

Index

Curriculum Vitae

Henk Smit werd geboren op 29 juli 1959 te Laren (Gld.). Van 1971 tot 1977 volgde hij op de Rijks Scholen Gemeenschap te Lochem een VWO opleiding. Van 1977 tot 1985 studeerde hij Electrotechniek aan de Universiteit Twente. Hij studeerde af bij de vakgroep Besturings-Systemen en Computertechniek. Het onderwerp van zijn afstudeer-opdracht was het ontwikkelen van een robotarm voor het verplaatsen van textiel. Vanaf 1985 is hij werkzaam aan de Technische Universiteit Eindhoven bij de vakgroep Produktietechnologie en -Automatisering (WPA) van de faculteit Werktuigbouwkunde. Hier verzorgt hij onder-wijs en verricht hij onderzoek. De eerste twee jaar had het onderzoek betrekking op het ontwerpen en realizeren van robotbesturingen. Daarna startte zijn promotieonderzoek zoals beschreven in dit proefschrift.

STELLINGEN

behorende bij het proefschrift

A Hierarchical Control Architecture for Job-Shop Manufacturing systems

1. Onderzoek naar scheduling heeft alleen zin als men beschikt over een besturingsarchitectuur waarbinnen het schedulingsalgoritme wordt geïmplementeerd.

Dit proefschrift.

2. De tegenstrijdige uitspraken over "sequencing" regels in de literatuur worden o.a. veroorzaakt doordat de onderzoekers geen eenduidige manier gebruiken om de prestaties van een produktiesysteem te meten.

Montazeri M.,
A modular simulator for design, planning, and control of flexible manufacturing systems,
Dissertation, Katholieke Universiteit Leuven (1987).

Dit proefschrift.

3. Met de keuze van "release" regels kan een grotere invloed op de prestatie van een produktiesysteem worden uitgeoefend dan met de keuze van "sequencing" regels.

Wiendahl H.-P.,
Belastungsorientierte Fertigungssteuerung: Grundlagen, Verfahrungsaufbau, Realisierung,
Carl Hanser Verlag, Munchen (1987).

Wein L.M.,
Scheduling semiconductor wafer fabrication,
IEEE Transactions on Semiconductor Manufacturing 1 (3), 115-130 (1988).

Dit proefschrift.

4. Voor het specificeren van het produktieproces schiet het gebruik van stuklijsten tekort.

Dit proefschrift (Sectie 2.6).

5. Het produceren van verschillende generaties IC's in één en dezelfde fabriek heeft twee belangrijke nadelen: de onrust die ontstaat bij het vervangen van apparatuur en de "job shop" layout.

6. Bij het oplossen van problemen met de computer is de keuze van een goede programmeertaal meer dan het halve werk.

Smit G.H.,
De besturing van waferfabs,
Memorandum, Faculteit Werktuigbouwkunde,
Technische Universiteit Eindhoven (1988).

Dit proefschrift.

7. Veel communicatie leidt tot inflatie van informatie.

8. Met de introductie van krachtigere computers neemt de gemiddelde tijdsduur van een "simulatierun" eerder toe dan af.

9. Gezien de huidige welvaartsverdeling is een beleid dat onderscheid maakt tussen politieke en economische vluchtelingen, onrechtvaardig.

10. Elk vak zou naast de wetenschappelijke inhoud ook de ethische aspecten van het vakgebied moeten behandelen.

11. Het is vreemd dat normale mensen vlees eten.

12. Het feit dat een stelling geen afbreuk mag doen aan de reputatie van de TUE, doet afbreuk aan deze reputatie.

13. De rockgroep Normaal heeft een belangrijke bijdrage geleverd aan het zelfbewustzijn van plattelandsjongeren.

Henk Smit Eindhoven, 10 maart 1992