Agile Production Planning and Control System

Graduate School of Systems and Information Engineering

University of Tsukuba

March, 2005

Tunglun Tsai

Acknowledgements

I wish to express my sincere gratitude to all those who helped me to get to this point of life. A very special acknowledgement goes first to my academic advisor, Professor Ryo Sato, Dean of College of Policy and Planning Sciences, for providing me personal growth and academic advice during the period of my doctoral program in Japan. Without his strict discipline and warm encouragement, I have already given up my dream of being a researcher. Even though he was so busy in teaching and doing research, he never refuses to write commendation letters for me to apply scholarship and tuition deduction, and he even actively recommended me as a research assistant (RA) of the graduate college twice. When I was taking a carefree attitude toward studying, he gave me a firm warning to get back on the track. When I was lacking confidence in my ability to submit a paper, make a presentation, or take a degree, his composed attitude always rebuilds my confidence.

My special appreciation goes next to Professor Ushio Sumita who gave me a valuable comment on the direction of my research in a SIM seminar in 2001, then to Professors Masato Koda and Suzuki Hideo who gave me some great advice on my paper during the summer camp in Kusazu, and finally to Professor Takao Terano who suggested me to introduce minimum generation gap (MGG), a model of generation alternation, into my genetic algorithm (GA) by which the performance and efficiency of GA have been greatly improved.

I would also like to thank my former advisor Professor Ryosuke Hotaka for his encouragement during the period of my first year as a research student in University of Tsukuba. I am very grateful to Professors Ryoh Fujihara, Akihiro Hashimoto, Yasuhiro Monden, Mika Sato-Ilic, Maiko Shigeno, Hideaki Takagi, Yoshitsugu Yamamoto, Akiko Yoshise, Yongbing Zhang for their teaching of the introduction courses from which I have a chance to explore the broad fields of policy and planning sciences.

A deep gratitude goes to Professor Jeremy Kuo who strongly recommended that I should study abroad to have more opportunities for growth and self-mastery, instead of entering local graduate school in Taiwan. He came to Japan many times to visit me, encouraged me to overcome the culture shock, and even offered me the partial living cost of my first year in Tokyo.

I greatly appreciate the Japanese government and University of Tsukuba for granting scholarships and a bounty to me. Besides, I would like to express my heartfelt thanks to Kanto Gakuen University for employing me as an instructor and allowing me to complete my

doctorate at University of Tsukuba continually.

My greatest appreciation goes to my parents in Taiwan and grandparents in heaven who have supported and educated me, and finally to my wife Wendy Fu who gave up her good job in Taiwan to accompany me to Japan.

Abstract

The purpose of this dissertation is to provide a new conceptual scheme for the integrated production planning and control (PPC), called agile production planning and control system (APPCS), along with development of a novel genetic algorithm (GA) for optimal scheduling, called genetic algorithm with minimal generation gap and demand crossover (MDGA), which is quite useful for implementing APPCS. PPC is a system that has functions of planning, execution, and control on elements of material and resource. This dissertation consists of three parts.

In the first part, the concept of APPCS is developed. It has the following three unique features:

1. Production planning and scheduling is made once a planning cycle. Customer orders accumulated during the previous cycle are incorporated at a time in the planning and scheduling for the current cycle. APPCS uses a time bucket as a planning cycle.
2. Scheduling and capacity planning are integrated to produce a feasible production plan.
3. When customers change their request and/or suppliers cannot maintain planned supply, with respect to date or quantity, they give advance notification before it happens. Upon the arrival of such information, APPCS immediately updates the production plan.

While different functions in PPC closely interact against each other, they tend to be managed separately. For example, Yeh (1997) pointed out that the calculation of requirements such as master production scheduling (MPS) and material requirements planning (MRP) is separated from the phase of capacity requirements planning (CRP) in the production planning with MRP/MRP II. They are not integrated. The second point is achieved by the use of so-called advanced planning system (APS) software. For a supporting software package, SyteAPS^{TM} from SYMIX Japan is employed. Notice that an APS alone does not assure successful scheduling. APPCS provides an effective way for scheduling with APS. The third point is the primary feature of APPCS. Suppliers and customers inevitably make changes after the purchase/customer orders are placed. This is a practical situation, since an important machine or person in a supplier could not be available as scheduled, and alternatively, a customer wants to change the requirements even with additional payment. In such a case, advance notification is usually possible, and also such notification seems to bring better results if it could be used properly. Neither a time bucket nor advance notification is incorporated in Yeh's scheduling method.

In the second part, a model of agile production planning and control (APPCM) is described. A model for product data management and planning had been available in Scheer (1994). The formulation of APPCS shows an augmentation so that APPCS is possible. The formulation of APPCS is described by the universal modeling language (UML) and illustrated by class diagram, statechart diagram, and sequence diagram.

In the third part, MDGA is proposed to solve dynamic flexible scheduling (DFS) problem and is developed for optimal scheduling to replace SyteAPS in APPCS. A DFS problem is to achieve a specified goal for given product data, possible resource flexibility, and work-in-process (WIP) inventory. In application of GA to optimization problems, the performance of the algorithms is largely affected, in general, by how to represent genes and how to reproduce a new generation of genes. In MDGA, the gene representation of a DFS problem is defined as an operation for a part with specified resource and WIP inventories. A chromosome is a sequence of such genes, which can be transformed into a set of tasks deployed in a Gantt chart. In other words, a chromosome corresponds to a feasible schedule for the DFS problem. For the reproduction process, minimal generation gap (MGG) approach (Yamamura et al., 1996) is employed. Through the combined use of MGG and demand-wise crossover called demand crossover, MDGA enables one to find a better and feasible schedule for the problem. For a set of DFS problems of reasonable size, the exact optimal solutions are generated by exhaustive search of complete enumeration. These exact solutions are then compared with the corresponding results obtained by MDGA, demonstrating the accuracy of the MDGA approach. Numerical results for applying MDGA to usual job shop scheduling problems are also exhibited, showing the effectiveness of MDGA at a satisfactory level. Some suggestions concerning the use of MDGA for solving DFS problems are provided in the end.

Contents

Acronyms

APPCM	Model of Agile Production Planning and Control
APPCS	Agile Production Planning and Control System
APS	Advanced Planning and Scheduling
BOM	Bill-of-Materials
BOMfr	Bill-of-Manufacturer
CRP	Capacity Requirements Planning
DFS	Dynamic Flexible Scheduling
EDD	Earliest Due Date
EOQ	Economic Order Quantity
ERP	Enterprise Resource Planning
FILO	First-in-Last-out
JIT	Just-in-Time
JSS	Job Shop Scheduling
LFL	Lot-for-Lot
LOX	Linear Order Crossover
MDGA	Genetic Algorithm with minimal MGG and Demand crossover
MGG	Minimal Generation Gap
MRP	Material Requirement Planning
MRP II	Manufacturing Resource Planning
MRPM	Model of Material Requirement Planning
MPS	Master Production Scheduling
NDC	N-Demand Crossover
POM	Production and Operations Management
PPC	Production Planning and Control
PPCM	Model of Production Planning and Control
rMGG	roulette Minimal Generation Gap
SGA	Simple (Standard) Genetic Algorithm
SOP	Sales and Operation Planning
SPT	Shorted Processing Time
TIC	Traditional Inventory Control
TICM	Model of Traditional Inventory Control

TOC	Theory of Constraints
UML	Universal Modeling Language
WIP	Work-in-Process

List of Figures

Figures

Tables

1 Introduction

The purpose of this dissertation is to provide a new conceptual scheme for the integrated production planning and control (PPC), called agile production planning and control system (APPCS), along with development of a novel genetic algorithm (GA) for optimal scheduling, called genetic algorithm with minimal generation gap and demand crossover (MDGA), which is quite useful for implementing APPCS.

PPC is a system that has functions of planning, execution, and control on elements of material and resource. The functions of PPC and its trends are shown in section 1.1. APPCS should implement the basic functions of PPC to be a system of it. We focus on five requirements to accomplish the purpose of this dissertation. The five requirements are enumerated and explained in section 1.2.

The contents of this dissertation are organized into 3 parts to meet the five requirements. In the first part, we propose the concept of APPCS, which has three characteristics.

1. Production planning and scheduling is made once a planning cycle. Customer orders accumulated during the previous cycle are incorporated at a time in the planning and scheduling for the current cycle.
2. Scheduling and capacity planning are integrated as a function to produce a feasible production plan.
3. Advance notification of changes is possible. Once a change is reported, APPCS immediately updates the production plan.

Secondly, in order to formulate the concept and mechanism of APPCS, we describe a model of agile production planning and control (APPCM) by using universal modeling language (UML). Thirdly, since the current advanced planning system (APS) software seems to have a large room in improvement, we propose a novel genetic algorithm to have a quasi-optimal production plan for APPCS. Outlines of the dissertation that contains these parts are shown in section 1.3.

1.1. Production Planning and Control

The definition of production planning and control (PPC) varies across the ages. The science of manufacturing planning and control began in the 1880s with the studies of Federick Taylor and other engineers. The definition of PPC might be the improvement of efficiency and moral of human labor in industry. In the 1960s, better customer satisfaction, higher profits, and lower capital investment are identified as the important objectives of manufacturing, and the definition of PPC became the proper management of inventory to optimize those objectives.

In the 1970s, the development of manufacturing planning and control systems was progressing rapidly, and computers and MRP systems were employed to provide a more accurate date on schedules. These focused some managers on the need for accurate data, valid master plans, and the value of sound execution. Vollman (1997) defines production planning and control as the system provides information to efficiently manage the flow of materials, effectively utilize people and equipment, coordinate internal activities with those suppliers, and communicate with customers about market requirements.

"Materials", "resources", and "time" are internal elements, "customers" and "suppliers" are external roles, and "planning", "execution", and "control" are internal functions that are hidden in the definition. In other words, PPC is a system that has functions of planning, execution, and control on elements of material, resource, and time to communicate with customers and suppliers.

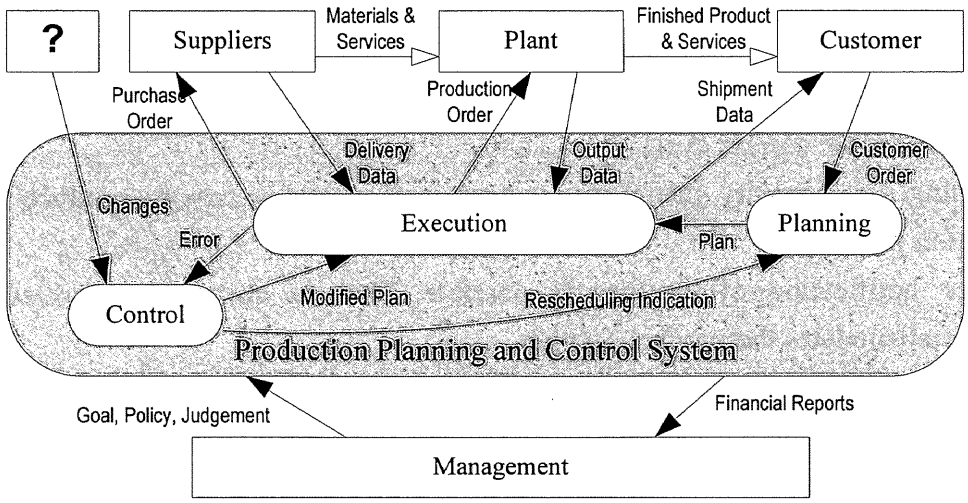


Fig. 1.1: Flows of production planning and control system

Fig. 1.1 shows the flows of PPC system, in which a hollow-arrow line denotes a material flow, a solid-arrow line an information flow, a rectangle a role, and an ellipse a function. Planning is the activity of transforming customers' orders into plans, each of which is a

requirement for a material that is planned to be manufactured by a resource for a period of time before a finish time. Execution is the activity of converting plans into reality. It releases production orders to shop floor and purchase orders to suppliers, accepts the reports of production and delivery, and reports the production result to customers. Control is achieved by comparing execution to plans, detecting significant deviations, reporting them to the proper persons and taking prompt corrective actions.

By applying the PPC system, however, problems happened in the 1980s that many manufacturing companies had much more capital tied up in inventory than they needed. The extra stock is used as safety stock to protect from changes. The source and reason of the changes are the saturation and globalization that make the increasingly fierce global-scale competition. To keep the extra stock against an unpredictable change is not wise. The importance of short leadtime and flexibility of manufacturing became clear. For example, just-in-time (JIT) system that asks a plant to always kaizen itself against changes was successful in shortening safety leadtime and decreasing safety stock while satisfying its customers.

Everything is changeable except for the change itself. The capability of handling changes is a prerequisite of a PPC system as illustrated in Fig. 1.1 with a '?' mark. Accordingly, the definition of PPC might be modified as a system that has functions of planning, execution, and control on elements of material, resource, and time to communicate with customers and suppliers, and deal with "change".

1.2. Problem Formulation

In this dissertation, we focus on the following five requirements to formulate the problem.

1. To capture both practical and detailed complexity of "real PPC" by using the structurally same product data with a modern enterprise resource planning (ERP) package.
2. To invoke production planning by a set of customer orders once a planning cycle (time-bucket approach).
3. To take into consideration work-in-process (WIP) while making a production plan.
4. To incorporate the changes from customers and suppliers as well as WIP in re-planning the production.
5. To provide a goal-oriented optimal production plan.

The product data that is actually used in commercially available ERP packages such as SAP R/3 and SyteAPS^{TM} contains parts, bills-of-materials (BOM), routings, work centers, and resources. Finished product, assembly, and raw material are the types of part. BOM is a term used to define a request-supply relation between parts. A routing is a sequence of operations

to make a part. A work center that enrolls some resources is assigned to an operation, and the operation can be processed by one of the resources in the work center. The first requirement indicates that the product data must be used to produce a feasible production plan. In other words, a feasible plan can only be generated on a basis that product data is well defined and estimated.

In the second requirement, time-bucket approach and real-time approach are the two alternatives considered in production planning. By applying real-time approach, which invokes production planning as soon as a customer order is placed, a customer order is opened earlier and, thus, the more flexible production plan can be prepared for the customer order. However, more purchasing cost due to some additional orders must be paid and more discounts for ordering in bulk cannot be earned if real-time approach is applied in place of time-bucket approach that invokes production planning once a planning cycle for a set of customer orders accumulated during the previous cycle. Both approaches should be provided in the production planning and control system to deal with the various requirements.

The third requirement stresses the importance of considering WIP in producing a feasible production plan. A scheduled receipt that has not yet arrived at an inventory location is considered as a WIP. As soon as the scheduled receipt is completed, the stock on hand is updated to reflect the amount of completed part. So, in other words, a WIP will be the stock on hand at a future point of time, and consequently it can be scheduled to be a material input after the time point. Since some necessary workloads in a production plan can be marked as executed if some appropriate WIP is used instead, getting advantage of WIP is regarded as a way to produce a flexible production plan.

The fourth requirement demands for the mechanism of dealing with changes. An unpredictable change can make a feasible schedule infeasible. Suppliers and customers inevitably make changes after the purchase/customer orders are placed. This is a practical situation. An important machine or person in a supplier could not be available as scheduled. Alternatively, a customer wants to change the requirements even with additional payment. We have mentioned in the previous section that other than material, resource, and time, change has became another important factor of making and evaluating a production plan. In case of any change, the production plan in execution should be modified according to the change. Furthermore, some follow-up actions should be taken with regard to the new production plan.

The last requirement is natural and straightforward. To maximize the profit and minimize the loss are the goals of running a business. For example, maximization of service level is the goal specified to maximize of the profit, and the minimization of makespan and tardiness are the goals of production to partially minimize the loss. To provide a goal-oriented optimal production plan is the responsibility of a production planning and control system.

1.3. Outline of the Dissertation

This dissertation has the following structure.

[Chapter 2] Model of production planning and control system

This chapter provides general issues of production planning and control (PPC) system. PPC has evolved many systems, such as traditional inventory control, material requirement planning (MRP), just-in-time (JIT) system, and theory of constraints (TOC), etc. The system has its strengths and weaknesses. To understand the essence of PPC helps choosing and establishing successful production management system. This chapter provides an abstract model of production planning and control (PPCM) which is described by universal modeling language (UML) to define the core of PPC. PPC has functions of planning, execution, and control on elements of material, resource, time, and change to communicate with customers and suppliers. In the abstract model, the functions are defined on a set of data classes and their associations. The system that implements PPC must provide a realization of the modeling elements of PPCM to ensure that the basic functions of PPC are satisfied. If the modeling elements in PPC are completely satisfied by a system that implements PPCM, then the system is called a complete PPC system. Traditional inventory control system is not a complete PPC system. MRP system is shown to be a complete PPC system. PPCM can be viewed as a description, reference and verification for a system that implements PPC

[Chapter 3] Agile Production Planning and Control System

An agile production planning and control system (APPCS) is proposed. It allows us agile re-scheduling upon unexpected events. It executes schedule-based planning in the sense that scheduling is tightly connected with net requirement planning and capacity requirements planning. APPCS does precise scheduling with so-called an advanced planning system for the set of demands that are accepted in the planning cycle. It produces a feasible production plan on a basis of product data. After releasing the production or purchase orders, notification to change the schedule may arrive from suppliers or customers in advance of due date. Based on the notification, APPCS does re-scheduling in accordance with the then situation. Safety leadtime in purchase orders and safety stock in raw material are used in APPCS to absorb the effects caused by sudden changes. As an example of implementation, it is applied to a fictitious and typical factory with variety of products. The simulation result shows that planning with notification is effective, and that notification has the similar effect with safety leadtime.

[Chapter 4] Model of Agile Production Planning and Control

This chapter provides a UML model of APPCS. The proposed model of APPCS satisfy abstract model of production planning and control (PPCM) is thus a complete PPC system. Beside that, it makes the immediate planning and scheduling possible. In order to testify the model, both instance of the model and its implementation in a simulator are shown.

[Chapter 5] APPCS Optimization

A novel genetic algorithm with minimal generation gap and demand crossover (MDGA) is invented to replace SyteAPS and as an augmentation of APPCS to search for a production plan that meets the goals. The scheduling problem is defined as a dynamic flexible scheduling (DFS) problem, in which the concept of operation requirement and task are used. An operation requirement is a manufacturing requirement for an operation of a part. Customer orders are converted to operation requirements based on product data during production planning. A task is a scheduling result of operation requirements. An operation requirement becomes a task if the responsible resource is specified, and both planned start date and finish date are set.

An operation requirement is represented by a gene in MDGA, and a sequence of the requirements forms a chromosome. MDGA reproduces a population of chromosomes with the principle of minimum generation gap (Yamamura et al., 1996) instead of simple tournament selection used in standard genetic algorithm (GA). In general, a sophisticated reproduction (crossover) method is needed to solve an optimization problem with many constraints by GA. Some proposed reproduction processes in standard GA, such as one-point crossover and two-point crossover, are inefficient because they tends to produce infeasible offspring. Thus by considering the structure of constraints, MDGA uses newly proposed specific crossover, called demand crossover, so that the crossover process produces only feasible offspring.

For a set of DFS problems of reasonable size, the exact optimal solutions are generated by exhaustive search of complete enumeration. These exact solutions are then compared with the corresponding results obtained by MDGA, demonstrating the accuracy of the MDGA approach. The exhaustive search also shows the difficulty in solving DFS problem. Numerical results for applying MDGA to usual job shop scheduling problems are also exhibited, showing the effectiveness of MDGA at a satisfactory level. Finally, since MDGA has many parameters, it is examined how they effect on solution-search process. Some suggestions concerning the use of MDGA for solving DFS problems are provided in the end.

[Chapter 6] Conclusion and Future Research

The results of this dissertation are summarized. Furthermore, some future research topics

are discussed.

2 Production Planning and Control System

2.1. Introduction

In this chapter, a model of production planning and control (PPC) is proposed to be an abstract model of all PPC systems including traditional inventory control (TIC) and material requirement planning (MRP), etc. The abstract model that defines the basic functions on the data classes and associations is simple and easily understandable.

PPC has been developed from the traditional inventory control, and evolved a lot of applications to respond to the various product mixes and process patterns. The evolution began with the scientific management developed to increase the efficiency of human labor in industry in the 1880s.

Before MRP, which was first developed by Joseph Orlicky in IBM, got a tremendous boost in 1972, when some techniques of TIC were widely adopted by many manufacturing companies. The techniques such as economic order quantity (EOQ) and statistical reorder point help determining when and how many parts to make to achieve the lowest cost by considering the factors of cost and demand rate. Orlicky and the other originators of MRP recognized that the TIC is much better suited to final products than components.

Rather than a production planning and control system, TIC is better to be positioned as a forecasting system, since it predicts the future independent requirement, but neglects planning dependent requirements and resource capacity. While on the other hand, MRP has been growing from an algorithm of calculating net requirement to a hierarchy of modules (MRP II) of long-term forecast, aggregate production planning, master production scheduling, material requirements planning, capacity requirements planning, and shop floor scheduling and control.

Just-in-time (JIT), a distinctive style of manufacturing considered by Taiichi Ohno and

successfully applied to Toyota from 1945, places emphasis more on "control" than "planning". In MRP, releases into the production line are triggered by the schedules. As soon as a job is completed at a work center, it is pushed to the next work center. Adversely, in JIT, production is triggered by a demand. When a part is removed from the inventory, the last work center producing the part is given authorization to replenish the part. MRP plans customer due dates, but has to respond to a change. JIT directly responds to changes, but must accommodate customer due date by setting more kanbans or using overtime. JIT system is well suited to repetitive manufacturing environment in which discrete parts are produced at a fairly steady flow rate (Hopp and Spearman, 2000).

MRP plans material requirements first and then resource capacity under the assumption that a resource has infinite capacity. The assumption may lead to bottlenecks during actual production. Due to that the bottleneck determines the throughput of the whole production system, theory of constraints (TOC) prefers to plan first the resource that causes bottleneck during production planning and scheduling.

Every system has its success story, but it also bears some criticisms of its weakness and inadequacy. To choose a proper production planning and control system is vital to the success of production management. In order to make such decision, one should know the essence of production planning and control. There are so many systems and applications developed so far for the purpose of production planning and control. It is easy to get lost in those systems.

No matter how the system evolves, the basic concept and elements of production planning and control remains the same. In the next section, an abstract model of production planning and control (PPCM) is presented by applying universal modeling language (UML), in which the interface, data structure, and functions are defined.

PPCM is the abstract model of TIC, MRP, JIT, TOC, and agile production planning and control system (APPCS). In other words, these systems provide a realization of the abstract model. Section 2.3 and 2.4 show that how functions and data classes are realized by TIC and MRP, respectively. A brief summary is given in section 2.5.

2.2. Production Planning and Control Model

Production planning and control model (PPCM), an abstract model of all production planning and control (PPC) systems, is represented by a data model and a behavior model. A package in UML is a grouping of model elements. The description and relationship among the model elements are explained in Appendix A.1. PPC package is a sub-package of the logistics package, which is again a sub-package of, let's say, the enterprise package. All the packages are arranged in a hierarchy. For example, as shown in Fig. 2.1, the packages including production planning and control package, inbound package, outbound package, etc., together with their associations are used to express the logistics package.

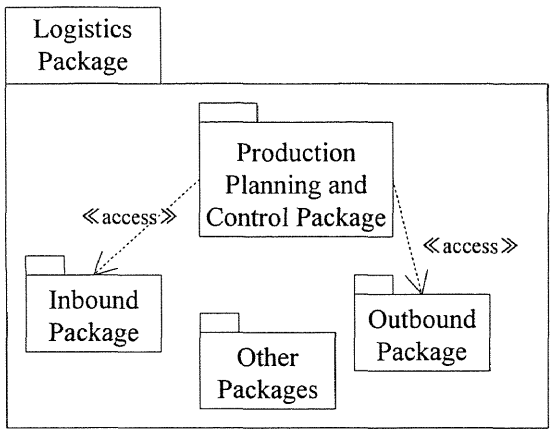


Fig. 2.1: A logistics package and the contained sub-packages

Fig. 2.2 shows that production planning and control model (PPCM), traditional inventory control model (TICM), material requirement planning model (MRPM), and agile production planning and control model (APPCM) are included in production planning and control package. These models together with their associations describe a view of the package.

The core of production planning and control is described in PPCM, and it is an abstraction of TICM, MRPM, and APPCM. In UML, an abstraction is a relationship that relates two models that represent the same concept at different levels of abstraction. The realization abstraction relationships indicate that TICM, MRPM, APPCM, and the future models of production planning and control are required to support all of the properties that PPCM declares.

A model captures a view of a physical system. A model contains data elements, behavior elements, and their mutual relationships. The model of a complex system itself can be very complicated and hard to understand. To solve the difficulty, a diagram is used to show some

of the elements to describe a concept, a function, or an interface with the external elements. In UML, class diagram presents only data elements, use case diagram shows only behavior elements and their associations with external elements, and collaboration diagram contains some data elements and behavior elements together with their links to explain a function.

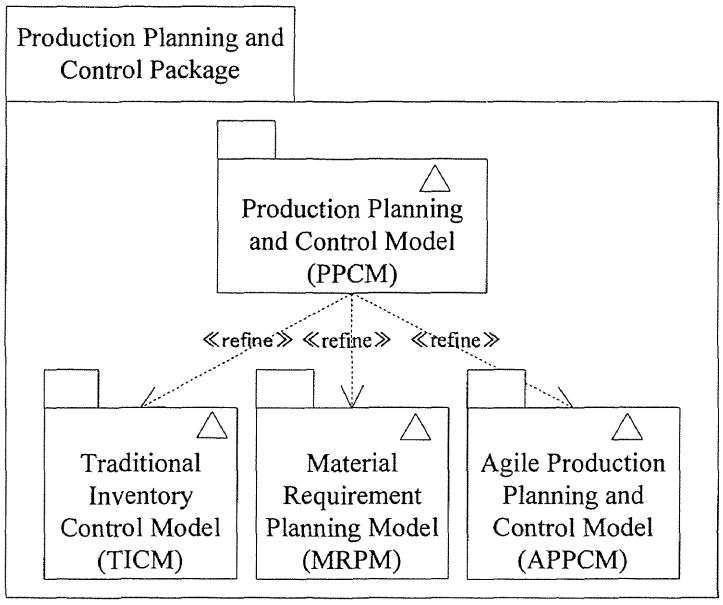


Fig. 2.2: The production planning and control package and its contained models

A class diagram shown in Fig. 2.3 and a use case diagram shown in Fig. 2.4 are used to depict PPCM. The notations and descriptions about the elements used in a use case diagram and a class diagram are provided in Appendices A.2, and A.3, respectively.

2.2.1 Data Model

PPCM is presented from the data view through a class diagram, which classifies the data used in PPCM. Part, epoch, and resource are the three basic elements which must be determined during production planning. In the class diagram, the *Part* class represents a set of items used in a plant either for production or procurement, the *Epoch* class expresses a set of points in the time-axis, and the *Resource* class is the set of machines and labors that can take part in manufacturing.

The *Demand* class is an abstraction of demands from either customer orders or forecasting. An attribute of *quantity* and links of *part*:*Part* and *dueEpk*:*Epoch* must be specified when creating a demand. A link can be viewed as a reference key pointed to another object of a class. For example, *part* is an attribute of the *Demand* class, and it is also an object in the *Part* class.

The *Task* class, which is also an association among *Part*, *Epoch*, and *Resource* classes,

is to represent a set of tasks generated in PPC. A task (*tsk*: *Task*) is a planned manufacturing requirement (*tsk.quantity*) for a part (*tsk.part*: *Part*) which is processed by a resource (*tsk.resource*: *Resource*) for a length of time (*tsk.processTime*) ending at *tsk.finsihEpk*: *Epoch*. A task can also be a procurement requirement for a raw material without specifying a resource.

The *Satisfy* association between the *Demand* class and the *Task* class indicates that a demand can be supplied by a task, but a task can be generated for supplying more than one demand. The *TaskLink* association with both ends connecting to the *Task* class denotes an abstraction of request-supply relations between successor tasks and predecessor tasks in a hierarchy of tasks.

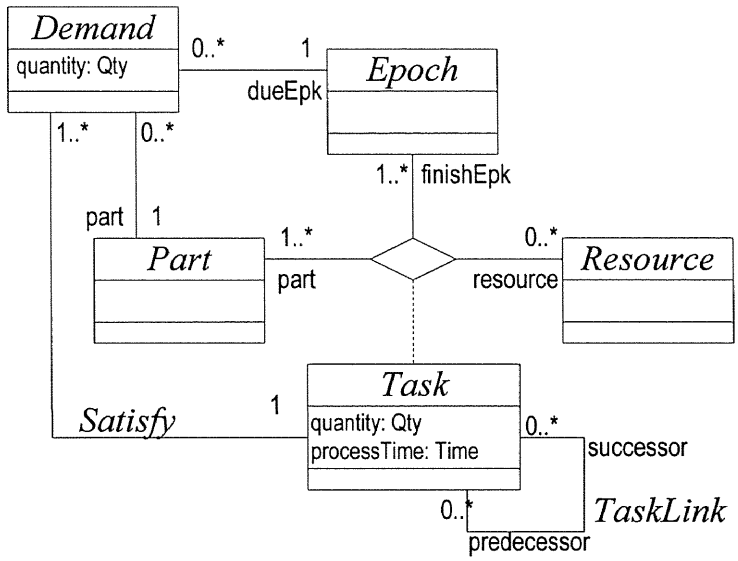


Fig. 2.3: A class diagram of PPCM

2.2.2 Behavior Model

The behavior model of PPCM is depicted by using a use case diagram that is shown in Fig. 2-4. Use cases in the diagram show the main functions of a model. The functions and their associations of PPCM with the actors are discussed in the following paragraphs.

Planning

The planning function transforms a request from a form into another equivalent form. For example, aggregate planning compiles corporate strategies into yearly, seasonal, or monthly requirements for some product families. Master production scheduling converts the aggregate requirements into weekly requirements of specific product. Material planning transforms product level requirements into component-level requirements and, at the same time, into

manufacturing requirements (production orders). The "planner" is responsible for executing the planning function to transform all demands in *Demand* class into tasks in the *Task* class.

Execution

The execution function is to carry the planned requirements (tasks in the *Task* class) into production or procurement orders. Production orders are released to "production manager", and purchasing orders are sent to "purchasing manager" for execution. After the work of production or procurement is completed, they should report the result of their works.

Control

The control function is to check whether or not the execution of production and procurement exactly follows the plan. The differences between them might be caused by some events, such as machine breakdown, quality difficulties with a particular batch, or the supply problem of vendor. Once "production manager" or "purchasing manger" cannot handle a change, the "planner" should be able to assess the impacts and provide a new schedule to incorporate the change.

If there is a change occurred to a demand, the *Satisfy* association helps locating the tasks that are affected by the change. If a task is delayed, the affected successor tasks are known by the links of *TaskLink* association. Once a failure happened to a resource, the affected tasks by the failure can be manifested by links of the *Task* association class.

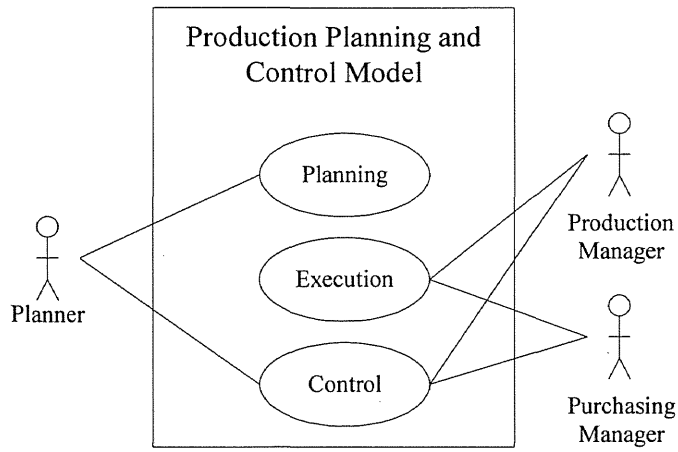


Fig. 2.4: A use case diagram of PPCM

2.3. Model of Traditional Inventory Control

In this section, traditional inventory control is introduced. A model of traditional inventory control (TICM) is proposed, and a mapping from TICM to production planning and control model (PPCM) is provided to show how PPCM is realized by TICM.

Inventory is materials required for production, including raw materials, work-in-process, finished products, and spare parts. The traditional inventory control, also called scientific inventory control, replenishes an item's stock by applying theoretical, mathematical procedures such as economic order quantity (EOQ) and statistical safety stock calculations (Plossl, 1991).

Stock replenishment approach is based on the principle of having inventory items in stock at all times, so as to make them available at the time of need (Oricky, 1965). According to the goal setting of stock replenishment approach, a full stock level of an item is determined and a replenishment order for the demand quantity is placed as soon as a demand arrives.

Another alternative of planning, called reorder point techniques, plans the requirements during replenishment leadtime together with some safety stock to compensate for the fluctuation in demand. A quantity Q of replenishment is made whenever the net stock level drops to the order point (safety stock) s or lower. Many alternatives are proposed to determine Q, such as the statistical order point, min/max, ordering up to level, and economic order quantity, by taking into consideration the demand rate of items, holding cost, and ordering cost, etc.

2.3.1 Data Model

Fig. 2.5 shows a class diagram that describes the static data structure of TICM. The notations and descriptions related to the elements of class diagram are provided in Appendix A.3.

The *Demand* class inherits all of its attributes and the definition from the respective class in PPCM.

Two approaches, stock replenishment and reorder point, are applied to items in the traditional inventory control. An item is replenished by one of the approaches. The items applying different approach have different attributes and behaviors. Consequently, two classes *SRItem* and *RPItem* are used to represent the items applying stock replenishment approach and reorder point approach respectively. Some attributes and methods related to items are in common. The *Item* class is the super class of the two to represent the common attributes and methods. The three item classes with two generation relationships are implemented to refine the *Part* class defined in PPCM.

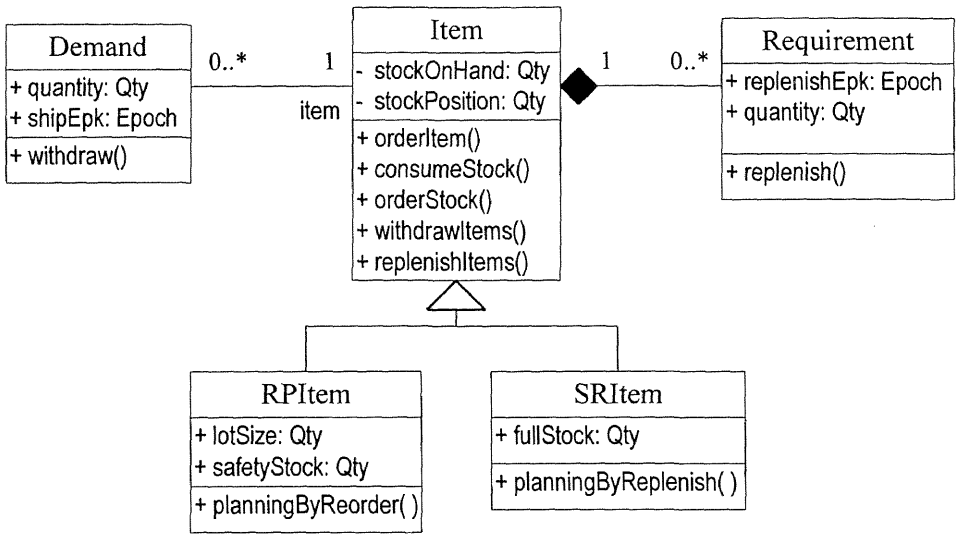


Fig. 2.5: A class diagram of TICM

The stock on hand is the stock that is physically on the shelf; it can never be negative. The stock position of an item is defined by the relation

$$\text{Stock position} = (\text{Stock on hand}) + (\text{Stock on order}) - (\text{Stock on shipment}) \quad (2.1)$$

The stock on order is stock that has been requisitioned but not yet received. The stock on shipment is the stock that has been committed but not yet shipped. Stock position can become negative (namely, if there are backorders). Both stock on hand (*stockOnHand*) and stock position (*stockPosition*) are attributes of the *Item* class.

Since resource is not considered in the traditional inventory control system, the abstract *Resource* class is not implemented in TICM, and the *Requirement* class in TICM partially realizes the *Task* class in PPCM.

The association that is starting from the *Item* class to the *Demand* class indicates that an item can be replenished by invoking many demands, and the inverse navigation reveals that a demand can only request for one and only one item. Another association exists between the *Requirement* class and the *Item* class. Multiplicities of the association explain that many requirements can be planned for an item, but there is one and only one item for a requirement. Since all requirements are generated for an item, the association is a composite.

2.3.2 Behavior Model

The behavior of TICM is described by a collaboration diagram of UML as shown in Fig. 2.6. The diagram that contains instances, relationships, and interactions among the instances helps

understanding the behavior of an operation declared in a class. The notations and descriptions related to the elements of collaboration diagram are described in Appendix A.4.

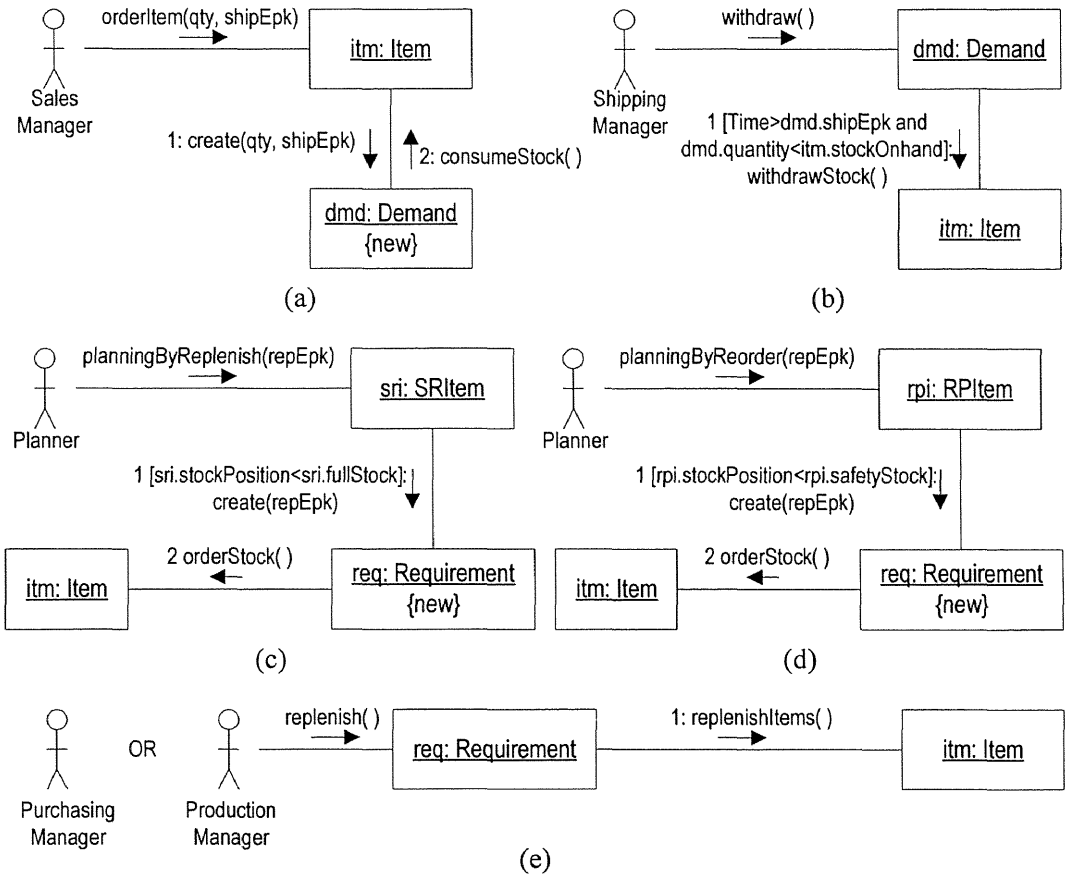


Fig. 2.6: Collaboration diagrams of traditional inventory control model

When a customer order for an item (*itm*: *Item*) arrives, a sales manager send a message to invoke the operation *itm.orderItem*( ) of the item as shown in Fig. 2.6a. The operation creates a new demand (*dmd*: *Demand*{*new*}) for the customer order and connects *itm* and *dmd* with a link. The demand invokes the operation *itm.consumeStock*( ) through the link from *dmd* to *itm* to deduct the requested quantity (*dmd.quantity*) from stock position of the item (*itm.stockPosition*).

If it is time to ship the item (*itm*: *Item*) ordered by a demand (*dmd*: *Demand*) and stock on hand for the item (*itm.stockOnHand*) is enough for the requested quantity (*dmd.quantity*), then the shipping manager invokes the operation *dmd.withdraw*( ), which triggers the operation *itm.withdrawStock*( ) through a link from *dmd* to *itm* as shown in Fig. 2.6b. This operation takes away the shipped quantity (*dmd.quantity*) from stock on hand of the item (*itm.stockOnHand*).

A planner sends a message to trigger the operation *sri.planningByReplenish*( ) of an item (*sri*: *SRItem*) applying stock replenishment as shown in Fig. 2.6c, after the planner found that

stock position of the item (*sri.stockPosition*) is lower than the full stock level (*sri.fullStock*) that is determined beforehand. The operation generates a requirement *req*:*Requirement*{*new*} to replenish the shortage (*req.quantity* =: *sri.fullStock* – *sri.stockPosition*) before an estimated time (*req.replenishEpk*).

A planner can be a device that can detect the shortage and automatically send a signal to invoke the replenishment. As shown in Fig. 2.6d, a planner sends a message to trigger the operation *rpi.planningByReorder*( ) of an item (*rpi*:*RPItem*) applying reorder point approach after the planner found that the stock position (*rpi.stockPosition*) is lower than the safety stock level (*rpi.safetyStock*) which is estimated for the item. The operation generates a requirement *req*:*Requirement*{*new*} to replenish the quantity (*req.quantity* =: *rpi.lotSize*) before an estimated time (*req.replenishEpk*).

After either approach creates a requirement (*req*:*Requirement*) for an item (*itm*:*Item*), through a link from *req* to *itm* the operation *itm.orderStock*( ) of the item is invoked to reflect the stock on order on the stock position by adding *req.quantity* to *itm.stockPosition*.

As soon as a requirement for an item is generated, it is released to production manager or purchasing manager. They are called on to replenish a requirement (*req*:*Requirement*) for an item (*itm*:*Item*) before *req.replenishTime*. As shown in Fig. 2.6e, They report their work by invoking *req.relenish*( ) of the requirement, which immediately lunches *itm.replenishItems*( ) of the item to reflect the replenished quantity on the stock on hand by adding *req.quantity* to *itm.stockOnHand*.

2.3.3 Mapping from TICM to PPCM

Table 2.1 shows how TICM realizes PPCM by a correspondence of classes and functions between the two models. Since resource is not implemented and no control is invoked to respond to changes as shown in the table by the shaded regions, so traditional inventory control is not a complete system of production planning and control.

Table 2.1: Model elements mapping from TICM to PPCM

	PPCM	Implementation of TICM
Classes	*Part*	*Item*; *SRItem*; *RPItem*
	Epoch	It is implemented as a data type (Epoch) in TICM.
	Resource	Resource is not planned in traditional inventory control; hence the detail production execution is not modeled.
	Demand	*Demand*
	Task	*Requirement*: A task in PPCM without considering resource becomes a requirement in TICM.
	Satisfy	No link exists between demands and requirements (Make-to-stock).
	TaskLink	Dependent requirements are not planned.
Functions	Planning	*SRItem.planningByReplenish*(); *RPItem.planningByReorder*()
	Execution	*Demand.withdraw*(); *Requirement.relenish*()
	Control	Backorders might be caused by changes in demand and delay on replenishing the requirements. No action is taken against the changes under the assumption of traditional inventory control.

2.4. Model of Material Requirement Planning

Material requirement planning (MRP) has been evolved from the initial version of Joseph Orlicky (1975), whom many authorities regard as the father of modern MRP, through a version called closed loop MRP (Vollmann et al., 1997, Silver et al., 1998), into enterprise resource planning (ERP), in which MRP is embedded. Closed loop MRP contains capacity planning and ranges widely from master production scheduling (MPS) to shop floor control. It has been implemented successfully in many software system, and acts as a basic structure for some new models, such as manufacturing resource planning (MRP II) by Oliver Wight (1984), and manufacturing planning and control (Vollmann et al., 1997) .The MRP we mentioned in this section is closed loop MRP.

The logic of MRP is simple as shown in Table 2.2, where Gr_t, Sr_t, Sk_t, Nr_t, Po_t, and Pr_t present the values for the tth time bucket. Stock on hand provided by (2.2) shows the stock when no replenishment is planned. Net requirement, shown by (2.3), is planned for the shortage occurred when the planned order is not enough for the gross requirements.

Table 2.2: Material requirements plan for a part

Part P
Leadtime = 1 buckets
Ordering policy: Lot-for-lot

Time buckets (t)	0	1	2	3	4	5	6	7	8
Gross requirements (Gr_t)				30	30	20	0	70	30
Scheduled receipts (Sr_t)	0	30	20						
Stock on hand (Sk_t)		30	50	20	-10	-30	-30	-100	-130
Net requirements (Nr_t)		0	0	0	10	20	0	70	30
Planned order receipts (Po_t)					10	20	0	70	30
Planned order releases (Pr_t)				10	20		70	30	
Supplying part of P (rate=2)									
Gross requirements				20	40		140	60	

A planned order is either a production order or a purchase order planned to satisfy the net requirements, and the scheduled receipts are the planned orders that have been released but not yet finished or received. To plan the net requirements without backorder, various replenishment policies are provided to determine the replenishment lot under the constraint (2.4). For example, (2.5) shows a lot-for-lot strategy. A planned orders planned to be arrived

on t should be released no later than $t-l$ as shown by (2.6). As shown in Table 2.2, the released planned orders become gross requirements of the supplying parts.

$$Sk_t = Sk_{t-1} + Sr_t - Gr_t, \text{ for all } t \geq 1 \tag{2.2}$$

$$Nr_t = \max\{0,\ Gr_t - \max\{Sk_{t-1}, 0\}\}, \text{ for all } t \geq 1 \tag{2.3}$$

$$\sum_{i=1}^{t} Po_i \geq \sum_{i=1}^{t} Nr_i, \text{ for all } t \geq 1 \tag{2.4}$$

$$Po_t = Nr_t, \text{ for all } t \geq 1 \tag{2.5}$$

$$Pr_t = Po_{t+l}, \text{ for all } t \geq 1, \text{ where } l \text{ indicates the leadtime} \tag{2.6}$$

A model of MRP (MRPM) is proposed and presented according to the following sections of master production scheduling, material requirement planning, capacity requirement planning, short term scheduling, and production control.

2.4.1 Master Production Scheduling

The input of MRP is from master production schedule (MPS), which is a requirement for end items. Master production scheduling is a work to generate MPS by integrating customer orders and forecasts demand management with sales plans from sales and operation planning (SOP), and to utilize plant capacity effectively by rough-cut capacity planning. Demand management makes customer delivery promises and resolves trade-offs between manufacturing and market. SOP provides the best trade-offs among incomes (sales marketing objectives), cost (manufacturing objectives), and investments (inventory/financial objectives).

A class diagram and a collaboration diagram of MRPM are shown in Fig. 2.7, and Fig. 2.8, respectively. In the class diagram, the *Demand* class is an abstraction of a group of demands which are the outputs of demand management and SOP. A demand is a requirement (*quantity*) for a part (*part*:*Part*) before a due time (*dueTime*). The due time here is represented by a time bucket.

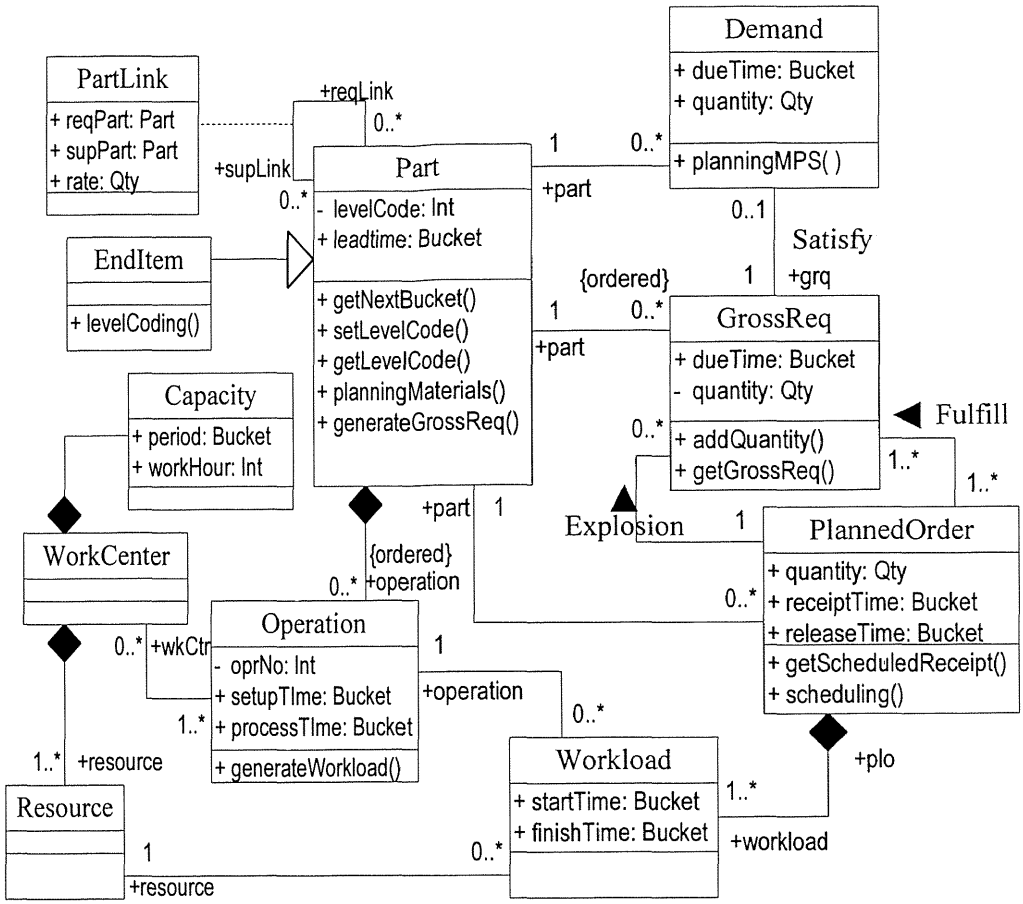


Fig. 2.7: A class diagram of MRPM

Fig. 2.8a shows how the demands are transformed into a set of master production schedules, each of which is viewed as a gross requirement for a finished product in MRP. A gross requirement is represented by a quantity (*quantity*) of a part (*part*:*Part*) required on a time bucket (*dueTime*). The gross requirements are grouped into a *GrossReq* class. A planner planning MPS sends a message to trigger the operation (1*)*planningMPS*() of a demand (*dmd*:*Demand*), if an MPS (*mps*:*GrossReq*) has been created for other demands that have the same values of *part* and *dueTime* with the demand *dmd*, then *dmd.quantity* is added to *mps.quantity* by triggering the operation (i.1)*mps.addQuantity*(), and a new link between *dmd* and *mps* is created (i.2). Otherwise a new gross requirement is created for the demand *dmd*.

The association between the *Demand* class and the *GrossReq* class is very important for the planner to trace the affected gross requirement when a change occurred to a demand, and to show the affected demands when an MPS is delayed due to some internal changes caused by, for example, machine breakdown.

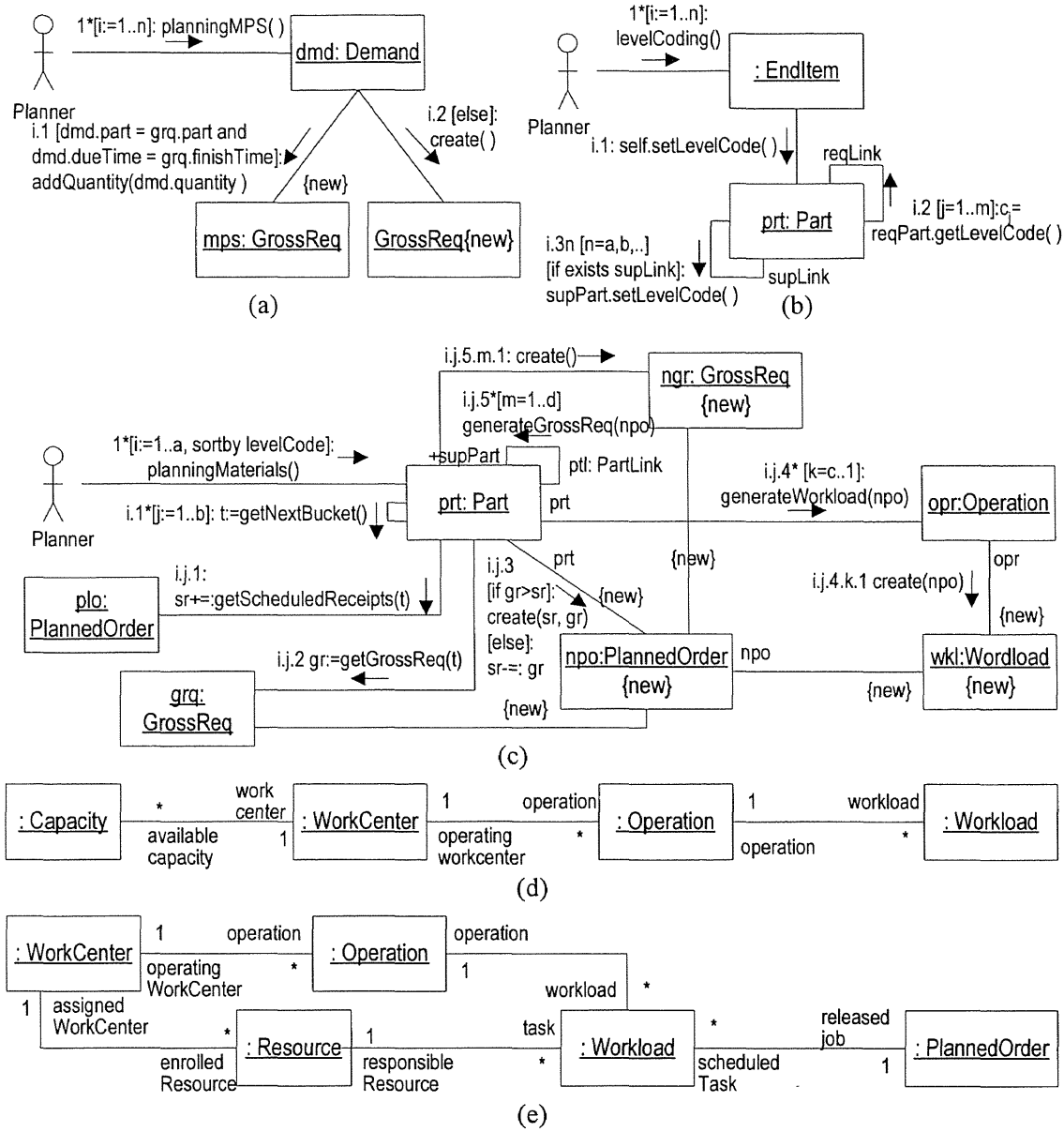


Fig. 2.8: Collaboration diagrams of material requirement planning model

2.4.2 Material Requirement Planning

MRP recursively plans the dependent requirements of MPS. The relation between the requirements is based on a set of request-supply relations between parts, which are modeled as an association class (*PartLink*) as shown in Fig. 2.7. The attribute *rate* of a link *ptl*:*PartLink* from *reqPart*:*Part* to *supPart*:*Part* shows the number of supplying part (*supPart*) required for one requesting part (*reqPart*). A set of such links starting from a requesting part defines a bill of material (BOM) of the part.

There are two different ways to determine the leadtime of processing a planned order. One way, which is called "leadtime scheduling" in SAP™, explores the structure of a routing,

estimates processing time of each operation by taking into consideration the lot size, and reserves the capacity of a resource for the processing time. Another way, which is called "basic date scheduling", regards the estimated leadtime of the part requested by a planned order as the leadtime of the planned order without considering lot size. MRPM supports both ways of scheduling. The two ways and their respective times are shown in Fig. 2.9, in which part B is the supplying part of part A, and part A has two operations and part B has only one operation.

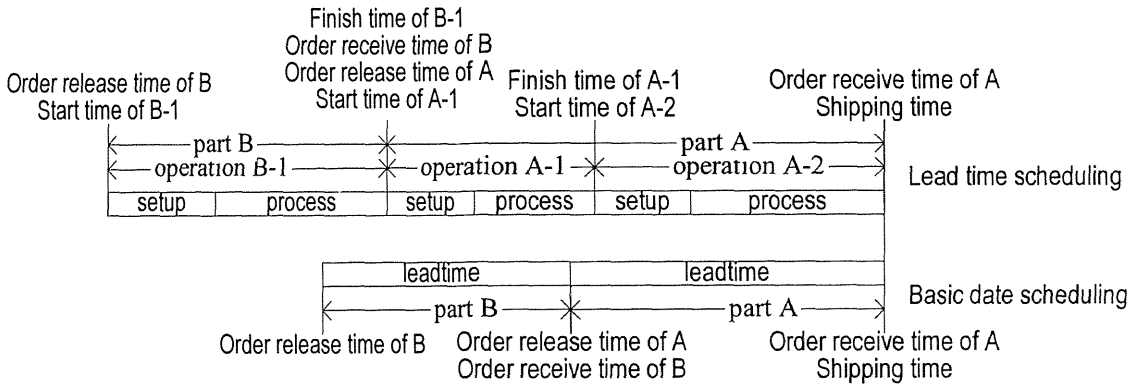


Fig. 2.9: Two ways of determining leadtime for a planned order in MRP

As shown in Fig. 2.7, *leadtime* is an attribute of the *Part* class which is used in basic date scheduling. Moreover, the *Operation* class is an abstraction of operations needed to manufacture a part. That the *Part* class is a composite of the *Operation* class indicates any operation is included in one and only one part. An operation (*opr*:*Operation*) has the attributes of operation number (*oprNo*), setup time (*setupTime*), unit processing time (*processTime*) for calculating the total processing time, and processing work center (*wkCtr*:*WorkCenter*). The attributes of the *Operation* class provides the information needed to do leadtime scheduling. Operation number indicates the processing precedence of the operations in a part. Usually, the operation with a smaller operation number must be processed before the operation with a larger number.

A prerequisite for planning material requirements of a part is that all gross requirements of the part must be known before the planning starts. To plan the parts in a top-down sequence is a way not to violate the prerequisite. A level code is used to identify the position of a part in the hierarchy of parts. The *levelCode* attribute of the *Part* class is to keep the level code for a part. The end item that is not used as a component of any other part is assigned a level code 0 as indicated by (2.7). The level code of the part except for the end item equals one plus the maximum level code among its requesting parts as shown by (2.8).

$$(\forall prt \in Part)\ \text{if}\ prt.reqLink = \varnothing\ \text{then}\ prt.levelCode = 0 \tag{2.7}$$

$$(\forall prt \in Part)\ prt.levelCode = \max_{rl \in prt.reqLink} rl.reqPart.levelCode + 1 \tag{2.8}$$

Fig. 2.8b shows the sequence and objects used in a collaboration diagram to assign a level code to each part. The level coding process starts from assigning level 0 to end items. As shown in Fig. 2.7, the *Part* class is a super class of the *EndItem* class.

The operation (1*)*edi.levelCoding*( ) of an end item *edi*: *EndItem* is triggered to lunch an operation (i.1)*prt.setlevelCode*( ) of its super object *prt*: *Part* through the generation link from *edi* to *prt*. Then the operation (i.2)*prt.getlevelCode*( ) of the part *prt* is invoked to determine level code of the part by adding 1 to the maximum level code among its requesting parts or setting 0 if there is no requesting part. Finally, the (i.3n)*supPart.setlevelCode*( ) operation is invoked for each supplying link *spl* in *prt.SupLink*. This operation is a recursive process in which many of such operations might be called through the links from a requesting part to its supplying parts.

Fig. 2.8c shows the sequence and related objects to plan material requirements for each part *prt*: *Part* in order of its level code with lot-for-lot ordering policy by invoking operation (1*)*prt.planningMaterials*( ), which transforms a set of gross requirements of the part *prt* into (1) a set of planned orders for the part *prt*, (2) a set of workloads for each of the planned order, and (3) a set of gross requirement for the parts supply to the part *prt*. The *plannedOrder* class is an abstraction of the generated planned orders with attributes of order quantity (*quantity*), receive time (*receiptTime*), and release time (*releaseTime*). A set of workloads is generated for a planned order as a proclamation of the required capacities. The *Workload* class is an abstraction of the workloads generated for all planned orders. A workload is an interval of time, starting from *startTime* and ending at *finishTime*, for processing an operation (*operation*) of the requested part of a planned order. Since all workloads must be generated for a planned order, the *PlannedOrder* class is the composite of the *Workload* class.

The operation invokes (i.1*)*prt.getNextBucket*( ) to loop each time bucket. In the *t*-th loop, the operation invokes (i.j.1) *getScheudledReceipts*( ) of a planned order *plo*: *PlannedOrder* whose planned release period is *t* to sum up the scheduled receipts *sr*, and lunches (i.j.2)*GrossReq*( ) of a gross requirement *grq*: *GrossReq* to get the gross requirement *gr* at time *t*. When $gr > sr$, a new planned order *npo*: *PlannedOrder*{*new*} supplement the shortage *npo.quantity* ($= gr - sr$) by time *npo.receiptTime* ($= t$), and new links from the new planned order *npo* to the gross requirement *grq* are created (i.j.3). The *Fulfill* association is an abstraction of the links from planned orders to gross requirements such that the planned

orders are generated to meet the gross requirements.

If "basic time scheduling" is adopted, release time the planned order (*npo.releaseTime*) will be (*npo.receiptTime* – *prt.leadtime*). If "leadtime scheduling" is performed, the processing time is calculated by recursively invoking (i.j.4*)*generateWorkload*( ) of an operation *opr*: *Operation* from the last operation to the first one through the links from *prt* to *opr* with a parameter *npo*. The operation sends a message (i.j.4.k.1) to create a new workload *wkl*: *Workload*{*new*}, a new link from *wkl* to *opr*, and a new link from *wkl* to *npo*. Let wkl_i be the workload of a planned order *npo* generated for the *i*th operation opr_i of the part requested by the planned order. The finish time (*finishTime*) of workload wkl_i is set to be the start time (*startTime*) of the latter workload wkl_{i+1}. If there is no that workload, then the finish time will be (*receiptTime*) of the planned order *npo*. The start time (*startTime*) of the workload wkl_i is calculated by

$$wkl_i.startTime = wkl_i.finishTime - (opr_i.setupTime + npo.quantity \times opr_i.processTime). \quad (2.9)$$

Finally, the start time of the first workload wkl_1 becomes the release time (*releaseTime*) of the planned order *npo*.

A planned order *npo* generated for a part *prt* becomes a source of gross requirement to its supplying parts. The operation (i.j.5*) *ptl.supPart.generateGrossReq*( ) is invoked repeatedly through a link *ptl*: *PartLink* from the part *prt* to *ptl.supPart* to (i.j.5.m.1) create a new gross requirement (*ngr*: *GrossReq*{*new*}) whose due time (*dueTime*) is *npo.releaseTime*, and gross requirement (*quantity*) is (*npo.quantity* × *ptl.rate*), and a new link from the gross requirement *ngr* to the planned order *npo*. The *Explosion* association is an abstraction of the links from gross requirements to planned orders such that the gross requirements are generated as inputs to the planned orders.

2.4.3 Capacity Requirement Planning

Capacity requirements planning (CRP) determines the required capacity through time at each work center. No CRP is necessary for the parts running basic date scheduling. In the usual approach of CRP, known as infinite loading, capacity constraints are ignored when developing capacity profile. Fig. 2.10 shows a capacity profile of work center X with actual capacity and required capacity.

Fig. 2.8d shows a collaboration diagram at specification level to generate a capacity profile where both available capacity and required capacity are investigated. The available capacity of a work center is estimated according to number of available resources and their available work hours in the work center. The *Capacity* class is an abstraction of the available

work hours (*workHour*) of all work centers at all time buckets (*period*). The association between the whole class (*WorkCenter*) and the part class (*Capacity*) provides the necessary data to plot the available capacity on a capacity profile.

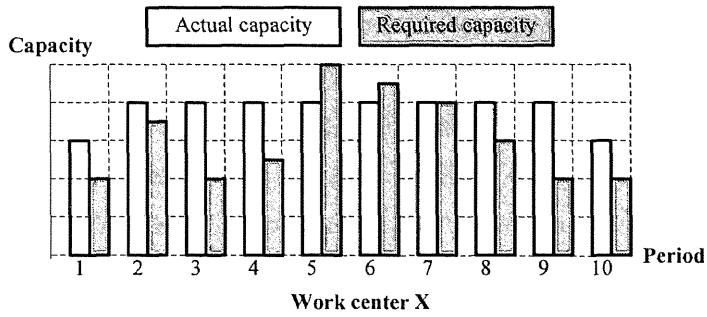


Fig. 2.10: Capacity profiles of work center X

The association between the *WorkCenter* class and the *Operation* class indicates that a work center can process many operations. The workloads planned to perform these operations are known by the association between the *Operation* class and the *Workload* class. The two associations among the three classes provide the needed data to plot the required capacity on a capacity profile.

Fig. 2.10 shows that capacity problems occurred in period 5 and 6. All the workloads planned at a period can be shifted forwardly or backwardly if the shift won't break down the balance of other work centers. For period 5, the solution is simply shifting some workloads ahead to period 4. However, the influence of the shift should be investigated.

The association between the *Workload* class and the *PlannedOrder* class shows the planned order of a shifted workload, and other workloads of the planned order that also have to be moved due to the shifted workload. Also the other planned orders which are either the requesting or supplying planned orders related to the planned order needed to be adjusted. Those planned orders affected are know by the two associations *Explosion* and *Fulfill* between the *PlannedOrder* class and the *GrossReq* class.

The possible solution for period 6 are (1) working over time, (2) using subcontracting, or (3) transferring resources from underutilized work center. If any of these solutions is not possible, then master schedule may be revised and MRP run again until all the work centers have a feasible capacity profile.

2.4.4 Short-term Scheduling

Short-term scheduling is to provide an actual start time and finish time for each planned order. The planned orders are released to shop floor for in-house production and to suppliers for purchasing according to the planned release time. A planned order is called a job after it was

released to the shop floor. Within house, a dispatch list contains the scheduled jobs is sent to shop floor in the morning. The list suggests the sequence in which jobs are to be run on each machine. There are many resources (machines) in a work center. Short-term scheduling chooses a workable resource for each workload of a job and finally the actual start time and actual finish time of the job are known.

To schedule a set of jobs is to fit workloads of the jobs in a Gantt chart. There are thousands of ways to schedule those jobs. The most common and simplest way is to schedule the jobs by a sequence, which is usually determined by some priority rules. The rules to choose a job among the jobs in the dispatch list are EDD (earliest due date), SPT (shortest processing time), etc. A result of short-term scheduling is shown in Fig. 2.11. No job can be fitted in the occupied area of resources in a Gantt chart. Such way of scheduling is called finite scheduling, which sets a detailed schedule for each job through each work center, and explicitly accounts for limited resource capacity at each work center.

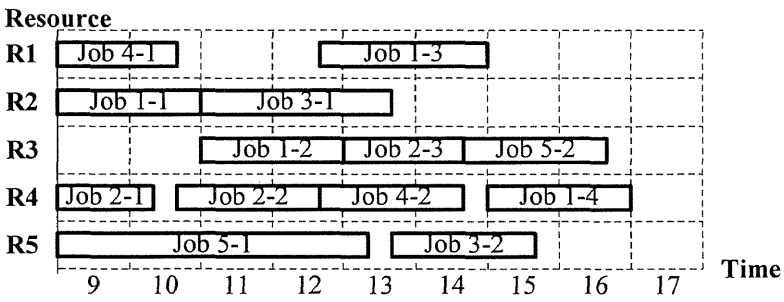


Fig. 2.11: A set of jobs fitted in a Gantt chart forms a schedule

A released job has a set of workloads that are generated in the stage of material requirement planning, and adjusted in capacity requirement planning. The short-term scheduling of a released job is to choose a resource and reserve an interval of time of the resource for each workload of the job. That the workloads scheduled from the first operation to that of the last operation of a job is called forward scheduling.

In Fig. 2.7, *PlannedOrder*, *Workload*, *Operation*, *WorkCenter*, and *Resource* are the classes that used in modeling the short-term scheduling of a released job. Roles of there classes are shown in Fig. 2.8e by a collaboration diagram to demonstrate the process of scheduling. A workload (*wkl*: *Workload*) is planned based on an operation (*opr*: *Operation*), the operation *opr* is assigned a work center (*wkc*: *WorkCenter*), and a resource (*res*: *Resource*) enrolled in the work center *wkc* is selected to be responsible for processing the workload *wkl*. The actual start time (*startTime*) and finish time (*finishTime*) of the workload *wkl* is known after it is fitted in the area belongs to the resource *res* in a Gantt chart, in which some tasks have been scheduled. A scheduled workload is called a task that will be realized in shop floor.

2.4.5 Production Control

The entire MRP process is carried out once per period (typically the period is one week). All changes that have taken place since the previous regeneration are incorporated in the new run. However, some changes such as additional order of an important customer or the delay in arrival of raw material might have the potential for tremendous impact on the plant. In such circumstances, a net change approach is used to incorporate such change on essentially a continuous time basis. In the net change, the modification of the previous schedule is limited to the affected part of schedules.

Assume a large customer order is completely canceled and net change is applied to the change. As shown in Fig. 2.7, the demand is an instance of the *Demand* class. Through the *Satisfy* association between the *Demand* class and the *GrossReq* class, the gross requirement affected by the change can be identified. Subsequently, planned orders for the gross requirement are known through the *Fulfill* association from the *GrossReq* class to the *PlannedOrder* class. The gross requirements, moreover, supply to a planned order can be traced by the *Explosion* association from the *PlannedOrder* class to the *GrossReq* class. If a planned order has been released to shop floor, the in-processing job can be pegged and stopped to avoid waste. Once a delay in the arrival of important raw material or machine breakdown occurred, the affected demands can also be identified on the inverse direction of classes and associations. Some action can be taken immediately to alleviate the damage caused by those changes.

2.4.6 Mapping from MRPM to PPCM

Table 2.3 shows that how classes and function of PPCM are realized by classes and functions of MRPM. All the modeling elements of PPCM are implemented, hence material requirement planning (MRP) is regarded as a complete production planning and control system.

Table 2.3: Model elements mapping from MRPM to PPCM

	PPCM	Mapping by MRPM
Classes	*Part*	*Part*; *PartLink*; *Operation*; *EndItem*;
	Epoch	It becomes a data type (Bucket) in MRPM.
	Resource	*WorkCenter*; *Resource*; *Capacity*;
	Demand	*Demand*
	Task	*GrossReq*; *PlannedOrder*; *Workload*;
	Satisfy	*Satisfy*
	TaskLink	*Fulfill*; *Explosion*;
Functions	Planning	Material requirement planning is achieved by *EndItem.levelCoding*( ) and *Part.planningMaterials*( ). Capacity requirement planning is achieved by *Operation.generateWorkload*( ). The planned orders in *PlannedOrder* is released to shop floor when it is release time. The released planned order (job) is transformed to a set of tasks by *PlannedOrder.scheduling*( ) at the shop floor.
	Execution	A task is processed by the specified resource at the assigned work center during the scheduled interval. After all tasks of a scheduled job are finished, the finished parts are stocked and then supplied to other planned orders.
	Control	*Satisfy*, *Fulfill*, and *Explosion* are associations used to identify the affected demands, gross requirements, planned orders, and tasks by a change.

2.5. Summary

Production planning and control model (PPCM) was defined as an abstract model of traditional inventory control model (TICM) and material requirement planning model (MRPM). The system that implements PPCM must provide a realization of the modeling elements to ensure that the basic functions of production planning and control (PPC) are satisfied. The system that completely implements PPCM is called a complete PPC system. We have shown that TIC is not, but MRP is, a complete PPC system.

Demands are transformed into tasks by planning function. The tasks are executed by releasing to production or procurement. Any change during execution is reported and controlled by first identifying the possible damages brought about by the change and then taking some effective actions to incorporate the change into the production plan.

Part, time (epoch), and resource are the three factors determined in production planning and scheduling. Five classes (*Part*, *Epoch*, *Resource*, *Task*, and *Demand*) and two associations (*Satisfy* and *TaskLink*) are used to define the basic factors and functions of PPC system.

In TICM, resource is not planned; furthermore, it is extremely difficult to identify the impact caused by a change and to revise a schedule because no link between demands and requirements is maintained. Therefore, the traditional inventory control is not a complete PPC system. In distinction from TICM, MRPM implements all elements of PPCM; MRP is thus regarded as a complete PPC system.

No matter how production planning and control systems evolve, PPCM remains unchanged. A new production planning and control system, agile production planning and control (APPCS), is proposed in chapter 3 and a model of APPCS (APPCM) is presented in chapter 4. We will show there how PPCM is realized by APPCM.

3 Agile Production Planning and Control System

3.1. Introduction

This chapter proposes a production planning and control system, called agile production planning and control system (APPCS), which has the following three characteristics:

1. A set of demands from customers is planned in a scheduling cycle. That is, a production planning is made once a planning cycle.
2. Scheduling and capacity planning are integrated to produce a feasible production plan.
3. When customers change their request and/or suppliers cannot maintain planned supply, with respect to date or quantity, they give advance notification before it happens. Upon the arrival of such information, APPCS immediately updates the production plan.

The first point allows us to use a time bucket as a planning cycle. Since sales and operations planning (SOP) and demand management are widely used with materials requirements planning/manufacturing resource planning (MRP/MRP II), and since they usually use the concept of a time bucket (Vollmann et al. 1997), so far developed expertise in such planning activities is incorporated with APPCS.

MRP/MRP II is a widely used production planning and control system implemented in commercially available enterprise resource planning packages so that production schedule at work centers is also calculated. A bill of materials (BOM) of a part is used in the calculation of net requirements and inventory. A routing of a part specifies necessary operations to make the part from component parts, and then is used to calculate necessary time to do the operations. Since the set-up and processing times of operations, together with possible parallel operations, are specified, the total necessary time consumed by all operations can be precisely calculated. Those planned operating times are deployed on the factory's calendar to make production schedule.

As Yeh (1997) pointed out, in the production planning with MRP/MRP II, the calculation of requirements such as master production scheduling (MPS) and MRP is separated from the phase of capacity requirements planning (CRP). They are not integrated. BOMs and routings of a part have complex hierarchical structures, which are mutually related. Therefore, in the case that a resulted production plan should be changed to meet a condition on capacity, production or purchase leadtime, or any kind of feasibility, it is difficult to see what the effect will be if some portion of the resulted plan changes. That is, the separation between production planning and CRP often leads to an unachievable situation.

Vollmann et al. (1997) also pointed out a similar drawback of MRP/MRP II. Then they called for so-called a real-time production and planning system. Yeh (1997) proposed a "schedule based scheduling", which is a kind of a real-time system. It uses a bill of manufacturer (BOMfr) structure that is composed of the combined data of BOM, work centers and routing, and then makes a production plan through "job-oriented finite capacity scheduling". Two processes exist in the scheduling cycle. In phase 1, the priority order among production jobs is decided, and a job to be scheduled is selected. In phase 2, the job is scheduled in a backward and/or forward loading sequence, referring to operations in the corresponding routing (Yeh 1997).

Yeh's scheduling method certainly provides a way of integrating scheduling and capacity planning, and produces a feasible production plan. Nevertheless, a schedule needs to be changed in response to a request from customers, suppliers, or both.

The second point is achieved by the use of so-called advanced planning system software. We used SyteAPS^{TM} from SYMIX Japan.

The third point is the primary character of APPCS. Suppliers and customers inevitably make changes after the purchase/production orders are released. This is a practical situation. An important machine or person in a supplier could not be available as scheduled. Alternatively, a customer wants to change the requirements even with additional payment. In such a case, advance notification is usually possible, and also such notification seems to bring better results if it could be used properly. Neither a time bucket nor advance notification is incorporated in Yeh's scheduling method.

When APPCS is used, the target service level of the whole production process can be decided. Hegedus and Hopp (2001) proposed an optimal parameter setting algorithm for purchase planning in production process. Based on the observation that recent large manufacturers of electronics operate their production process stable with almost no work-in-process (WIP) inventories, they modeled the whole production process as a single machine. The main source of schedule disruption is suppliers in the research. APPCS also uses a safety buffer only in purchase planning. Safety leadtime and safety stock for purchased materials are possible safety buffers in this research. In order to decide the sufficient amount of safety buffer to keep customers' due dates, simulation will be undertaken for various buffer

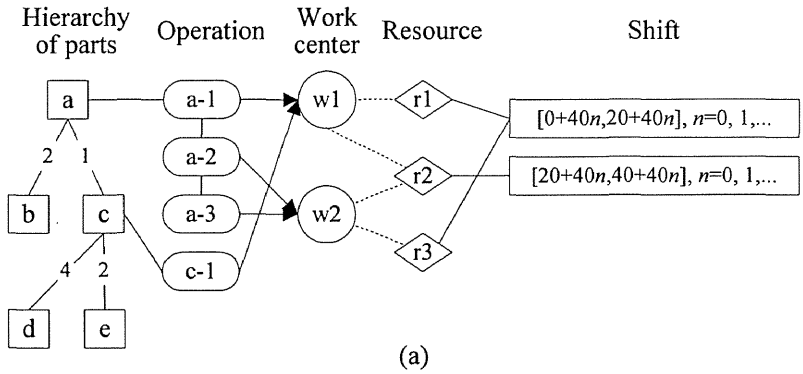
setting. Based on the graph that shows the relationship between buffer levels in purchase orders and service levels, users of APPCS can find a suitable buffer level for target service level.

The rest of the chapter has the following sections. In Section 3.2, the architecture of APPCS is shown. In Section 3.3, the rescheduling of APPCS against advance notification from customers and/or suppliers is proposed. A simulation method to decide suitable buffer level for purchase orders is illustrated in Section 3.4. Conclusion is in Section 3.5.

3.2. Architecture of APPCS

3.2.1 Product Data

In a production planning and control system, parts are classified into three types: finished products, assemblies and raw materials. Raw materials are procured externally and used to manufacture assemblies. For a raw material, a procurement leadtime is estimated. Assemblies are further devoted to manufacture a finished product or another assembly. Such vertical relations each of which specifies a required number stem from a finished product or an assembly to its component parts is called a bill of materials (BOM). A hierarchy of parts can be constructed by applying recursively the request-supply relations defined in BOM.



(a)

part	operation	lead time	setup time	process time	work center
a	1	–	1	0.5	w1
	2	–	0	0.8	w2
	3	–	2	0.6	w2
b		12	–	–	–
c	1	–	3	0.5	w1
d		20	–	–	–
e		16	–	–	–

(b)

Fig. 3.1: An example of product data containing part, operation, work center, resource, and shift

A routing of a part is a sequence of necessary operations to produce the part from the components defined in its BOM. Setup time and processing time per unit are estimated for each operation, which will be processed at a work center in which resources are enrolled. Usual resources are human workers and machines with specific skills. The capacity of a resource is defined as a set of finite time intervals that are called shifts. The shift defines a

start epoch and an end epoch of the working interval. A work center enrolls a set of resources, and one of the resources will be assigned to an operation in scheduling. A hierarchy of parts, called part structure,

An example of a product data is shown in Fig. 3.1. A finished product 'a' is made of input parts {b, c} through a sequence of operations {a-1, a-2, a-3}. The operation 'a-1' is processed at a work center 'w1', and operations 'a-2' and 'a-3' at another work center 'w2'. Resources 'r1' and 'r2' are enrolled in work center 'w1'. Resource 'r2' works for both work centers 'w1' and 'w2'. The shifts of resource 'r1' are described by a set of time intervals. As an example, let the shifts be {[0, 20], [40, 60], ...}, each of which corresponds to available work hours on a factory calendar. When scheduling is undertaken, a resource in {r1, r2} is assigned to operation 'a-1'.

3.2.2 Net Requirement Planning and Scheduling

Production planning starts with given independent requirements for finished products, and finally outputs executable schedules for manufacturing necessary assemblies and requests for procurement of raw materials. Forecasting and customer orders form independent requirements, each of which has a required quantity and due date.

APPCS uses planning cycle. A time interval, e.g., one week, is used as a planning cycle. Production planning will be made regularly for a set of demands for finished products accepted within the planning cycle. For each demand, finished product, required quantity and due date are specified. Required quantity of a demand is the gross requirement of a finished product.

Since scheduling is used to produce a feasible production plan in APPCS, the planning method of APPCS is called a scheduling-based production planning. The outline of the schedule based planning of APPCS is shown in Fig. 3.2.

```
for each demanded finished product {
  for each part of a finished product {
    net requirements planning of the part;
    scheduling the operations for the part;
  }
}
```

Fig. 3.2: Schedule-based production planning of APPCS

In APPCS, the production planning of a part consists of two phases. In the first phase, net requirements are calculated for the component parts of the part. Gross requirement can be supplemented by available stock on hand and/or released planned orders on condition that the planned finish epoch of the orders is in time for due date of the gross requirement. The

unsatisfied portion of gross requirement is called net requirement, which should be filled by manufacturing or procurement. The first phase is called net requirement planning.

In the second phase, capacities of resources are reserved for the planned net requirement. This phase is called scheduling. During scheduling, the time required to process the amount of parts is calculated, capacity of the proper resources equivalent to the time is determined and reserved, and a job with a start epoch and a finish epoch is generated corresponding to the scheduling result. Scheduled finish epoch cannot be later than due time of the net requirement. A job will be released to a work center for production or to a supplier for procurement, which is respectively called a manufacturing job and a procurement job.

The product data shown in Fig. 3.1 will be used to illustrate the planning and scheduling logic of APPCS. Assume that a set of demands $\{d1, d2\}$ has been generated and released at epoch 0 that $d1$ requests 15 pieces of part 'a' before epoch 50, $d2$ requests 20 pieces of 'a' before 90. At epoch 0, stock on hand of part 'a' is 5 pieces, part 'b' is 30, and no stock is for parts {c, d, e}. Besides, a planned purchase order has been released and 50 pieces of part 'd' will be delivered at epoch 10.

Using 5 pieces of finished product 'a' from stock on hand, a manufacturing job $j1$ of which net requirement is 10 pieces is generated for demand $d1$ as shown in Fig. 3.3a. A demand in the figure is represented by a rectangle, a job by an oval, and stock on hand by a filled oval. A line with solid arrow shows the assignment from stock on hand to a job. A line with hollow arrow shows a request-supply relation between a demand and a job.

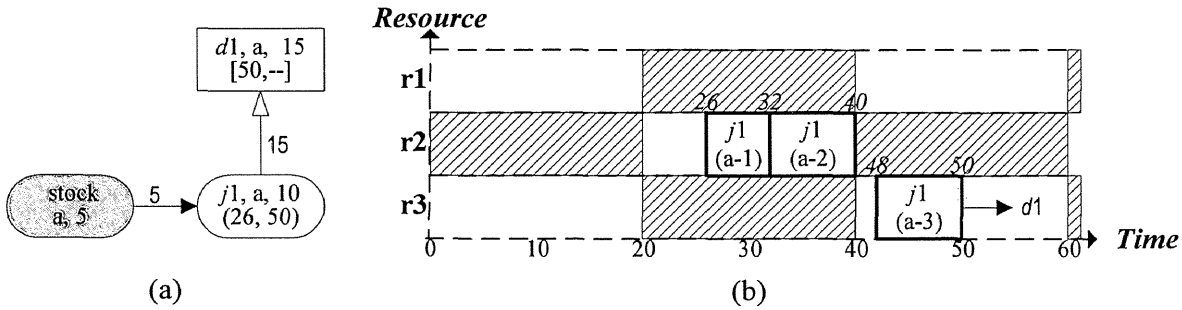


Fig. 3.3: Result of planning and scheduling of job $j1$

In the scheduling phase, the actual start and finish epochs of a job are determined. For a procurement job, the procurement leadtime of raw material is the only factor that should be considered. For a manufacturing job, all operations of a finished product must be scheduled. Total processing time of an operation is calculated by

(Setup time)+(Processing time per item)×(Net requirement). (3.1)

The processing time must be offered by one of the resources in the specified work center, and it can be covered by fragmented intervals or a continuous one on that resource. To schedule a job from the last operation to the first one is called "backward scheduling". On the contrary, "forward scheduling" is to schedule from the first operation to the last one.

Fig. 3.3b shows a scheduling result of job $j1$ in a Gantt chart, in which a white area shows an available (unoccupied) shift of a resource, and a bar with a job is an occupied shift. Resource 'r2' is assigned to operations 'a-1' and 'a-2', and resource 'r3' is assigned to operation 'a-3'. There are 3 shifts occupied by job $j1$. Actual start epoch of the job is the start epoch (26) of the first operation and finish epoch of the job is the finish epoch (50) of the last operation.

So far, production plan for finished product 'a' is completed. We continue the work of planning and scheduling of parts 'b' and 'c'. Since net requirements of 'a' is 10 pieces, gross requirement of parts 'b' and 'c' will be 20 and 10 pieces, respectively. Net requirement of part 'b' is zero and no further scheduling is invoked, because all the required 'b' items are fully supported by the stock on hand. Net requirement of part 'c' equals to its gross requirement. Resource 'r1' is selected among resources enrolled in work center 'w1' for the only operation of part 'c', since it provides a latest finish time (20) for the generated job $j3$.

Subsequently, gross requirements to net requirements of parts 'b' and 'c' are calculated in the following planning. Because part 'b' is a raw material; moreover, its net requirement is zero, no gross requirement is planned for the part. Gross requirements of part 'd' and part 'e' to supply net requirement (20) of 'c' are 40 pieces and 20 pieces, respectively.

Net requirement of part 'd' is again zero, since there is a released planned order that supplies 50 pieces of part 'd' and it arrives in time for the starting of job $j3$. Part 'e' is a raw material whose replenishment leadtime is 16, as shown in Fig. 3.1b, which is independent of the net requirement (20) and consumes no resource. The planning and scheduling continues until there is no gross requirement. In this manner, backward scheduling explodes a demand into jobs in a top-down sequence, and schedules backwardly from due epoch of the demand to a start epoch.

Fig. 3.4 shows the result of net requirement planning and backward scheduling of demand $d1$. The result of net requirement planning is presented by a hierarchy of jobs; while the result of scheduling is by a Gantt chart. Since job $j4$ is scheduled to start from a past epoch (-4), the result of backward scheduling is infeasible.

All of the jobs spanning from demand $d1$ are canceled, and taking away from the Gantt chart. Then, forward scheduling takes place to generate a schedule starting after epoch 0. The procurement job $j4$ is first scheduled from epoch 0 to 16. Subsequently, operation 'c-1'of job $j3$ reserves capacity of 'r2' instead of 'r1' from 20 to 28. Finally, capacities [28, 34] of resource 'r2', [40, 48] of 'r3', and [48, 56] of 'r3' are scheduled for operations {a-1, a-2, a-3}, respectively.

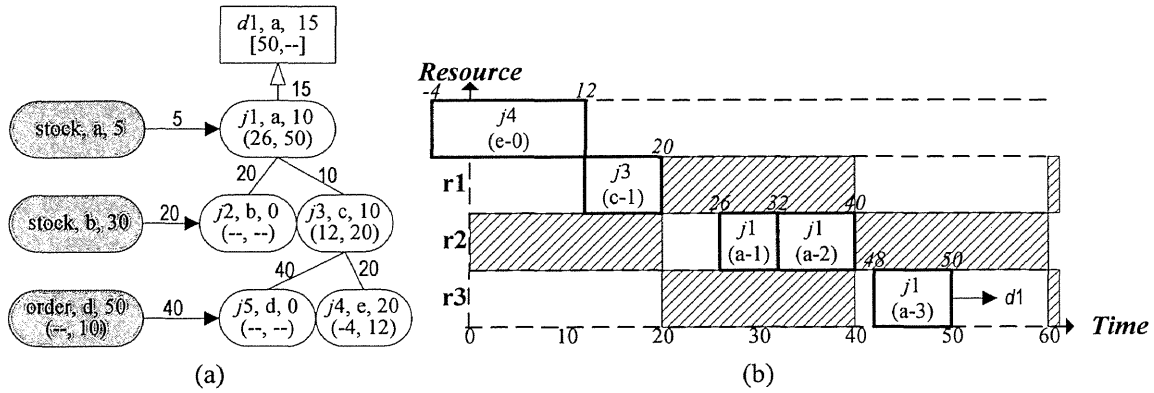


Fig. 3.4: Result of net requirement planning and backward scheduling of demand $d1$

Forward scheduling usually generates some holes on Gantt chart. That is, there may exist fragments of unoccupied intervals. In order to remove such holes, shifting occupied intervals to the right will be tried so that more continuous and wider intervals are hopefully remained for other jobs, which will be called back shifting. Fig. 3.5 shows a result of forward scheduling with shifting back the job $j4$, $j3$ (c-1), and $j1$ (a-1). In this case, the scheduling failed to fill the required due date, 50, of the demand n. Thus, we need to negotiate with the customer about delaying the due date, or consider other possibilities.

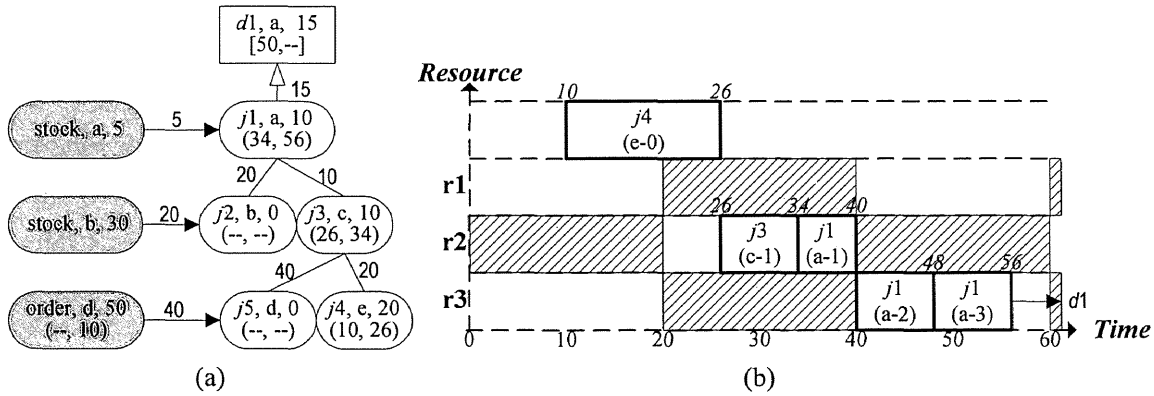


Fig. 3.5: Result of net requirement planning and forward scheduling of demand $d1$

In the scheduling of APPCS, demands are scheduled according to priority. Demand with the highest priority together with its dependent requirements can first consume available stock on hand, released planned orders in net requirement planning phase, and first reserve the free capacity of resources in scheduling phase. Earliest due date (EDD) rule is adopted by granting priority to the demand that has the earliest due date. For example, demand $d1$ is scheduled prior to demand $d2$ if EDD rule is applied. In this manner, demands are scheduled independently and in a sequential order in the scheduling of APPCS.

3.2.3 Procurement

A set of procurement jobs is generated at the end of net requirement planning and scheduling. Each job represents a dependent requirement of raw material. To meet the procurement requirement with respect to due epoch, purchase orders are generated and released to vendors.

If the lot-for-lot (LFL) policy is adopted, a purchase order, or we say supply from the view point of vender, is generated for each procurement job. If more than one procurement job requires the same material, then they can be aggregated into a purchase order. In this instance, quantity of a purchase order is the summation of the net requirements, and planned release epoch and arrival epoch are set to the start epoch and finish epoch of the earliest job among the jobs.

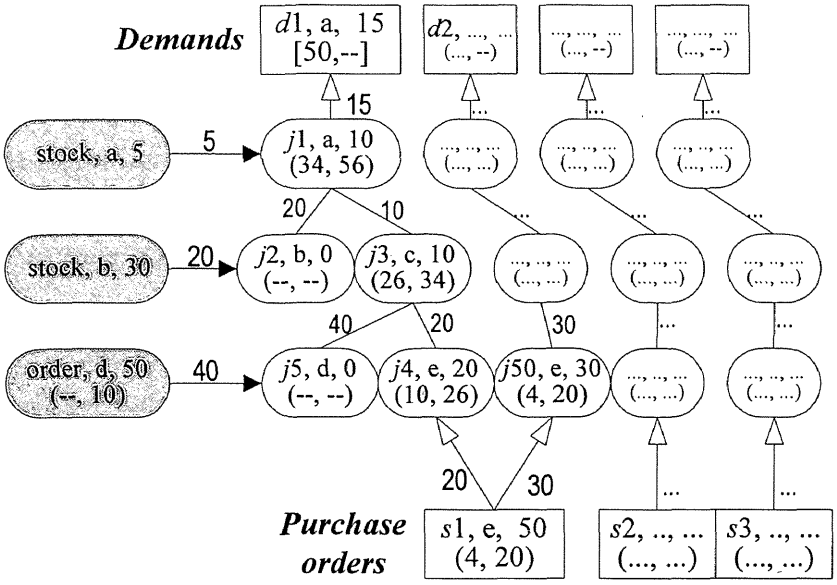


Fig. 3.6: A result of procuring raw material e by purchase order s1

Job $j4$ shown in Fig. 3.5a requests 20 pieces of raw material 'e' by epoch 26. Assume that demand $d2$ is scheduled and there is a job $j50$ that requests 30 units of the same raw material 'e' by epoch 20. Purchase order $s1$ is generated to meet the requirement of the two jobs (see Fig. 3.6).

A set of purchase orders is generated to fill the set of procurement jobs. For a purchase order, raw material to be purchased, quantity, and delivery epoch are specified. A purchase order should be released no later than (due epoch) – lt, where lt denotes the procurement leadtime estimated for the raw material.

3.2.4 Production Execution and Control

Production execution is the realization of the scheduled production and procurement plan. Under the architecture of APPCS, production control monitors the execution of manufacturing jobs and procurement jobs. As shown in Fig. 3.6, purchase order $s1$ will be released to a vender at epoch 4, and it will be delivered at epoch 20 to supply jobs $j4$ and $j50$. The delay of purchase order $s1$ will have impact on both of the jobs, and finally on at least demands $d1$ and $d2$.

A manufacturing job is controlled by verifying that if all of the supplying jobs can be finished before the start epoch, and that for all scheduled operations the assigned resources are available during the assigned interval. For example, as shown in Fig. 3.6, jobs $j4$ and $j50$ are required to start job $j3$, and as shown in Fig. 3.5b, resource 'r2' is responsible for the processing of operation 'c-1' of the job. After a job is finished, it is supplied to other jobs or becomes stock on hand. The delay of the supplying jobs or resource breakdown will delay demand $d1$ and worse the service level.

Production uncertainty is caused by unpredictable event that could make the production plan invalid. A demand change caused by a customer, forecasting error, and supply change caused by a vender are such events. If the production system does not learn to deal with it, it is prone to cause the production system an inconceivable loss.

APPCS responds to the events by invoking the net requirement planning and scheduling again. The demand whose schedule becomes infeasible or unattainable will be added to the rerun list. Silver et al. (1998) indicated that the frequent updates from the planning process leads to a poor communication between the planning department and the shop floor. To avoid such a dispute, once a manufacturing job is released to the shop floor, it cannot be cancelled or stopped. Except for the released in-processing jobs, the previously planned jobs and scheduled tasks belong to the rerun demands will be abandoned. The reserved capacity of the resources by those tasks is freed, and the assigned stock on hand is cancelled. The processing and scheduling run again for the demands according to priority based on the in-processing jobs, inventory, and released purchased orders.

Fig. 3.7 depicts how state (S_n, n=0,1, ...) of planned jobs of a demand is changed by the unpredicted events (E_n, n=1,2, ...) during production execution. Invoking planning and scheduling PS() at notification epoch t_n remedies the change. In the figure, a dot line with its start point in the time axis and end arrow pointing to a job shows an event, and a bold line with an arrow indicates a state change. In each state, an oval shows a job, a filled oval a released job, a square of dot line a finished job.

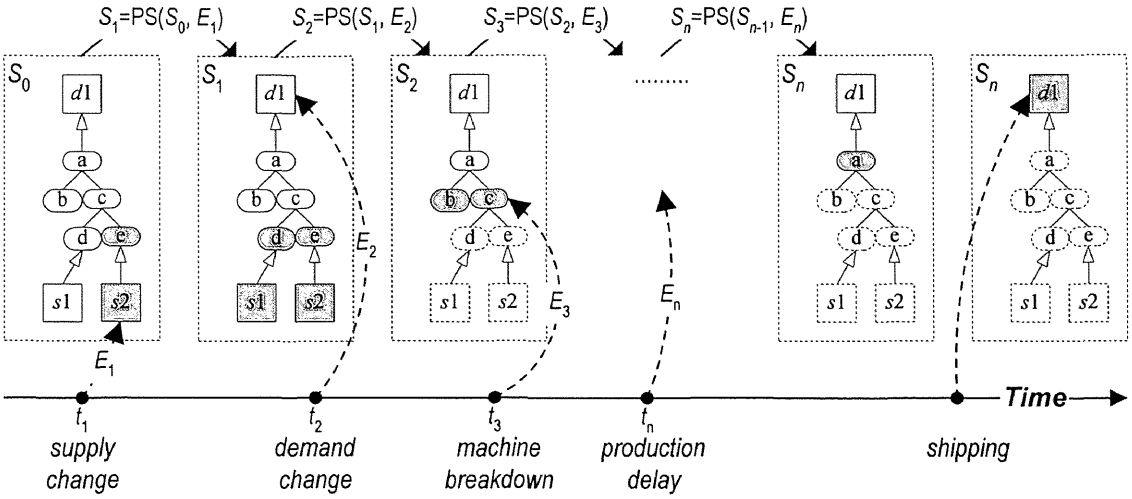


Fig. 3.7: State changes of jobs related to demand d1 by some events

3.3. Dealing with Changes

Scheduling described in section 3.2 is executed by so-called advanced planning system software (like SyteAPS^{TM}). After a production schedule was made, it sometimes faces uncertainty caused by unexpected events such as machine breakdown or change in demanded requirements or supply. The arising uncertainty during procurement and manufacturing can make the schedule infeasible. In this research, we focus on the uncertainty caused by supply and demand. That is, as Hegedus and Hopp (2001) formulated, it is assumed that production process is in good operation without any malfunction or failure, and that suppliers and customers are sources of unexpected events. Since customers want their changes to be attained with minimum possible delay and vendors hope to reduce the effect of their changes to a minimum, they usually send advance notification of their change before it really happens. If a manufacturer could use notification effectively with an appropriate mechanism to respond to such uncertainty, it is strategically beneficial and brings reduction of the damage caused by the uncertainty.

APPCS provides the mechanism against uncertainty through rescheduling as follows. First, the set of planned demands is investigated to find out demands affected by the uncertainty. Let D_c be the set of such demands. When rescheduling is started, some jobs might be still in-processing. Such jobs will not be updated in rescheduling. The other planned jobs except for the in-processing ones are cancelled, and the reserved stock on hand and planned purchase orders by the jobs are freed. The tasks scheduled for the cancelled jobs are abandoned, and the reserved capacity of related resources is freed.

Employing the (earliest due date) EDD rule, the demands in D_c are planned and scheduled once again based on the updated available stocks on hand, released purchase order, and in-processing jobs. After the rescheduling, some demands in D_c may be delayed. The related department or customers should be notified of the delay if no other solutions can be found.

3.3.1 Supply Uncertainty

There are two types of uncertainty with respect to procurement: supply time uncertainty and supply quantity uncertainty. The former refers to any delay of supply beyond that is scheduled. The latter means there is less quantity than is scheduled. The rescheduling algorithm against supply uncertainty is shown in Fig. 3.8.

As shown in Fig. 3.6, purchase order $s1$ is released to a vendor at epoch 4 and it is scheduled to be delivered at epoch 20. Assume that during [4, 20] the vendor notifies of

supply time uncertainty that $s1$ will be delayed for time $dly(s1)$. If $dly(s1) \leq 6$, then $D_c=\{d2\}$ holds. Otherwise, we have $D_c=\{d1, d2\}$. The demands in D_c will be rescheduled by following the EDD rule.

```
// Reschedule against supply uncertainty
x := the procurement job that is notified of change from a supplier;
SWITCH (type of supply uncertainty)
  CASE supply time uncertainty:
    IF a new purchase order will be delivered earlier than the notified time,
    THEN immediately generate and release a new purchase order;
    ELSE wait for the arrival of x;
  CASE supply quantity uncertainty:
    immediately generate and release a new purchase order to cover the shortage;
```

Fig. 3.8: Procedure of rescheduling against supply uncertainty

Since purchase order $s1$ will be delayed for $dly(s1)$, it is supposed to be delivered around $arv(s1) + dly(s1)$, where $arv(s)$ denotes the planned arrival epoch of purchase order s. If notification of the purchase order's delay is at epoch $ntf(s1)$, and if a new purchase order of the raw material 'e' were immediately issued at $ntf(s1)$ that will be delivered earlier than the suggested delivery time, i.e. $(arv(s1) + dly(s1)) > (ntf(s1) + lt('e'))$ holds, where $lt(p)$ denotes purchase leadtime for raw material p, then a new purchase order is created.

Assume the delay time $dly(s1)=5$ for purchase order $s1$ shown in Fig. 3.6 is known at $ntf(s1)=6$. The delayed order $s1$, whose new arrival epoch is 25, has an affect on job $j50$. As shown in Fig. 3.9a, a new purchase order $s4$ that requests 30 pieces of material 'e' was lunched to another vendor at epoch 6, and the new delivery epoch is 22 (6+16), which is earlier than the arrival epoch (25) of the original order $s1$. If $ntf(s1) \geq 9$ or $dly(s1) \leq 2$ holds, the best policy is to wait for the arrival of the delayed purchase order $s1$.

If the vendor notifies of supply quantity uncertainty that less item will be delivered on time than promised for a purchase order $s1$, then we have $D_c=\{d1, d2\}$. The demands in D_c will be scheduled again according to earliest due date (EDD) rule based on the released purchase order $s1$. The demand with higher priority can reserve quantity of the released order. As a result, another purchase order $s4$ must be released immediately (at epoch $ntf(s1)$) to cover the shortage $stg(s1)$ of purchase order $s1$. The new purchase order is expected to be arrived at $(ntf(s1) + lt('e'))$. Fig. 3.9b shows the result of the rescheduling for the supply quantity change on D_c, where priority of $d1$ is higher than $d2$ and the shortage 10 is reported at epoch 12.

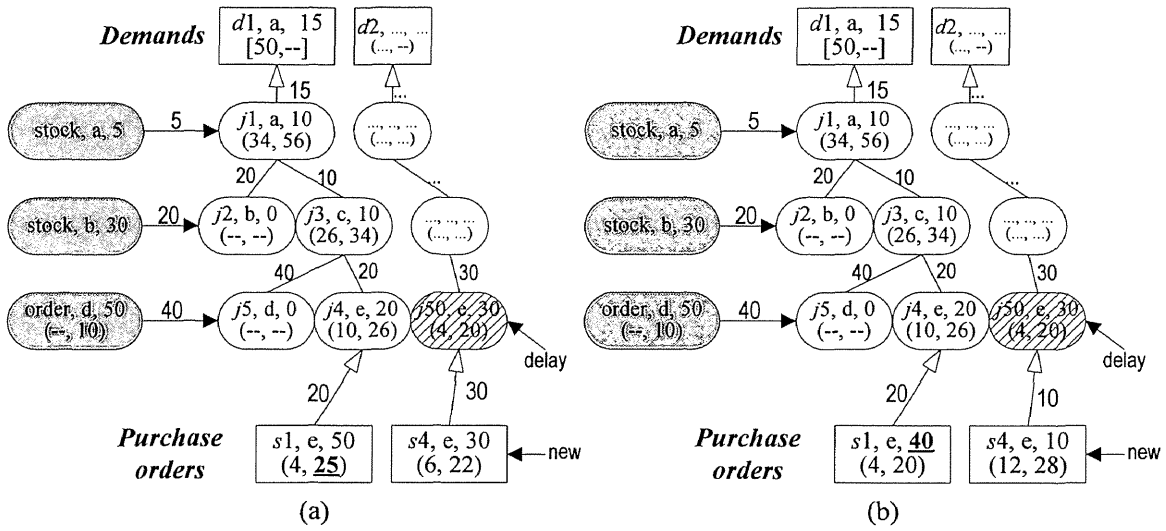


Fig. 3.9: The rescheduling results for (a) supply time uncertainties and (b) supply quantity uncertainty

3.3.2 Demand Uncertainty

Demand uncertainty might happen after a demand is accepted and before it is delivered. There are two types of demand uncertainty. If a customer asked for earlier delivery than committed due time, then it is called demand time uncertainty. If a customer asked for more products than the promised quantity, then it is called demand quantity uncertainty.

As shown in Fig. 3.6, demand $d1$ is scheduled to start from epoch 4 and to end at 56. Demand uncertainty of the demand can happen during [4, 56]. Suppose demand time uncertainty happened. Since the current schedule for demand $d1$ is the result of forward scheduling as shown in Fig. 3.5, and since there is no space for early processing on resource 'r3', it is impossible to achieve the request.

With regard to Fig. 3.5, assume that the customer of demand $d1$ made an additional order of 2 pieces of finish product 'a' at epoch 30. The customer wants to know when the changed request can be completed. As shown in Fig. 3.10a, at epoch 30, purchase order $s1$ has been delivered, job $j4$ is completed and job $j3$ is still in processing. Because job $j1$ is not started, it is cancelled and its planned tasks are freed. Moreover, the assigned but not used stocks on hand are cancelled.

Forward scheduling of the updated demand $d1$ is invoked by further using the in-processing job $j3$. Fig. 3.10b shows a new schedule of demand $d1$ after the uncertainty is handled by rescheduling. Job $j3$ is in time to used by job $j1$, so all its quantity is assigned to job $j6$, and net requirement of job $j6$ is reduced to 2 pieces. An additional purchase order $s4$ is released at epoch 33, and the additional request will be offered at epoch 79.

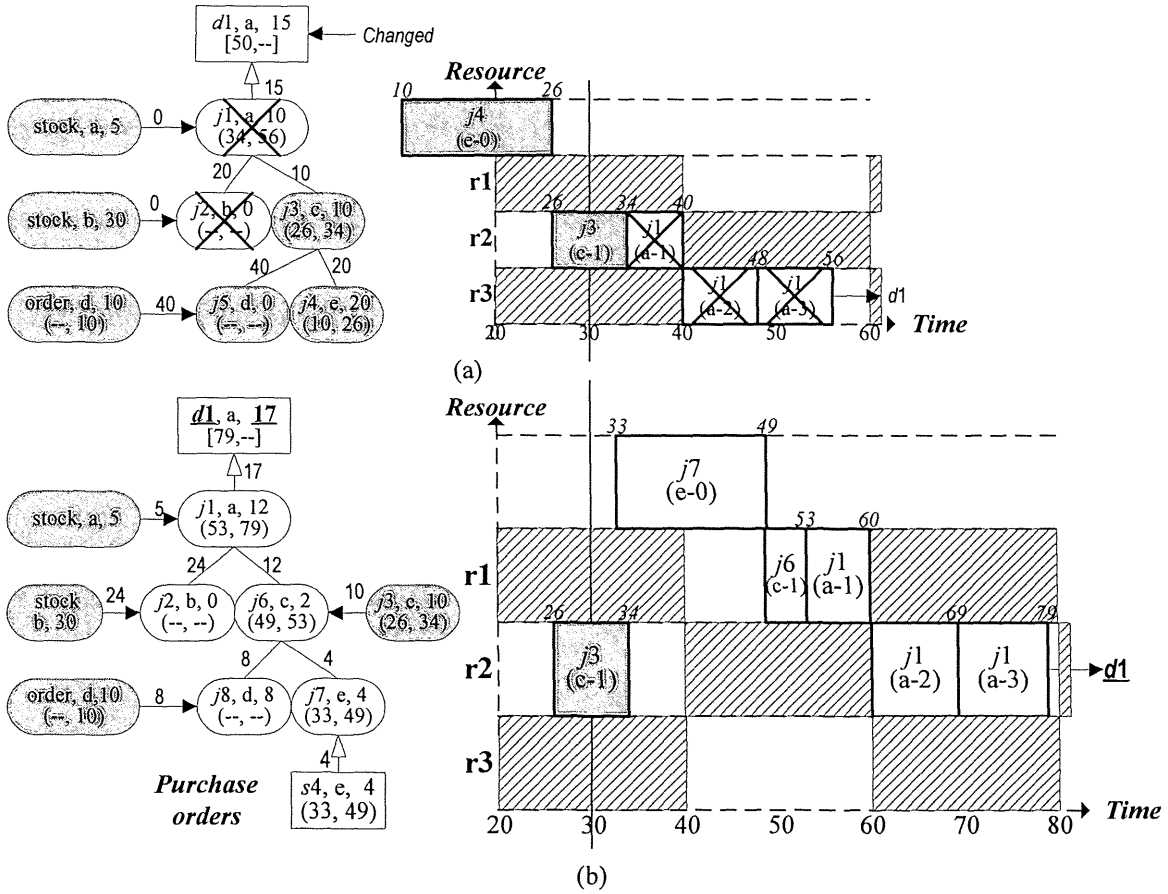


Fig. 3.10: (a) The jobs affected and cancelled by a demand quantity uncertainty, (b) rescheduling result for the uncertainty

3.3.3 Combining Advance Notification with Safety Buffer

Sudden changes in a schedule with respect to quantity or due time will finally affect the service level of the whole production. The service level is defined as the ratio of the number of customer orders filled within their due time to that of delayed customer orders. Safety stock and safety leadtime are commonly used buffering remedies, which are seemingly effective against such sudden changes, i.e. uncertain at the time schedule was made. Since APPCS uses advance notification about changes, the combination of notification with either or both buffers is possible. The advance notification of uncertainty, which is used in APPCS, has a mutual effect with each of both kinds of safety buffers. Several cases of mutual effect are illustrated in Fig. 3.11.

When we reserve safety leadtime buffer against supply time uncertainty (upper left of Fig. 3.11), cases A and B are possible according to the delay time. Case A shows that delayed purchase order arrives before starting the epoch of manufacturing and reserved safety leadtime works successfully in protecting the manufacturing schedules. Case B shows that the delay time is greater than reserved safety leadtime. Depending on the notification epoch of

uncertainty, a new supply, which will be delivered earlier than the original one, is released in time (case B1) or late (case B2) for manufacturing. If notification is too late to have the new supply arrive later than the original supply (case B3), then it is better to wait for the original one.

The cases in which safety leadtime is used against supply shortage are shown in the lower left of Fig. 3.11. A new purchase order must be released to compensate the shortage. If notification comes early enough (case C1), then a new supply is in time for manufacturing. Otherwise, the extra supply will be late (case C2) and we see inevitable delay on schedule.

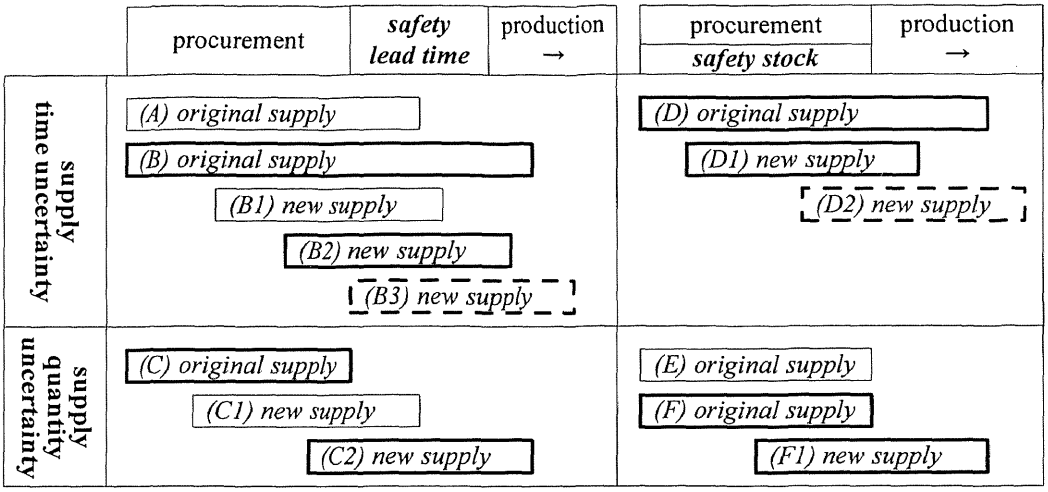


Fig. 3.11: Cases of safety buffers against supply uncertainties

Case D shows two possible situations of safety stock against supply time uncertainty in the upper right of Fig. 3.11. If the notification of uncertainty is early enough (case D1), then the manufacturing can start earlier by releasing a new purchase order. Otherwise, it is wise to wait for the delayed purchase order (case D2).

As shown in the lower right of Fig. 3.11, there are two possibilities of using safety stock buffer against supply quantity uncertainty. Case E depicts that safety stock is enough to meet the additional requirement, and that production is ensured to be on time. Otherwise, extra procurement has to be made immediately for the shortage (case F1) by a new purchase order, but delay of the manufacturing is inevitable.

3.4. Application of the APPCS

This section shows a method to introduce APPCS to achieve a target service level. It uses simulation to see the trade-off among buffer setting and service level under certain amount of demands.

3.4.1 Procedure

The procedure of the method is as follows.

- Determine the planning cycle, e.g. a day, a week or a month.
- Provide a set of demands. We can use forecasting for the planning horizon and/or a breakdown of the company's profit plan.
- Execute scheduling for the demands with various uncertainties to get the relationship between buffer size and a performance index, such as service level.
- Drawing curves of such relationships, find an appropriate safety buffer size that seems to attain target performance.

3.4.2 Simulation Design

Consider a fictitious manufacturer, ABC Plant. Assume that all of the products of ABC Plant have a respective three-level hierarchy of parts. This assumption is not strange because one of the most common enterprise resource planning packages, SAP R/3™, has a sample enterprise, and about half of its 200+ products have a three-level hierarchy of parts (SAP 2000). The low-level code of a finished product is 1, and that of a raw material 3.

ABC Plant has eight products, each of which is highly configurable. For a finished product, its component parts differ among a customer's configuration. The number of assemblies is more than or equal to 1 and less than or equal to 4. The number of component parts per part is more than or equal to 1 and less than or equal to 4. For a part, the number of operations is under 6. For each operation, setup time and processing time per item are under 0.5 hour and 0.2 hour, respectively. For a raw material, the purchase leadtime is under 48 hours. In the simulation, above product data are randomly sampled from the respective uniform distributions with the respective ranges, and the processing work center is assigned randomly. The number of work centers is four, and each work centre has a distinct resource.

The available capacity of a resource is defined by shifts. In the simulation, every resource has the same shift that works from 0:00 to 10:00 hours everyday. Planning cycle is defined as

[01/01, 01/31].

Request quantity of any demand varies according to the exponential distribution with the average of 10. To guarantee the demands to be finished within the planning horizon, the due date of demand is determined by running planning and forward scheduling to get a finish epoch. Then the due date is randomly determined between the finish epoch and 01/31. All the requirements of raw material are integrated and accumulated to become a purchase order and released to a vendor.

For any purchase order, the delay time of supply is smaller than the procurement leadtime. The shortage of supply is smaller than the released quantity. A customer asks for an earlier shipment than the initial due date before the total processing time will pass by. Extra quantity requested by a customer for an initial demand is less than the original request quantity. All of these uncertainties are sampled from suitable uniform distributions in simulation.

- A case in simulation is a combination of the following five parameters:
- Advanced notification is possible or not.
- Source of uncertainty specifies either demand uncertainty or supply uncertainty.
- Type of uncertainty specifies either time or quantity.
- Level of uncertainty: degree of demand uncertainty is high (75%), medium (50%) or low (25%). The degree is defined as the ratio of the changed demands to the total demands. Degree of supply uncertainty is high (50%), medium (25%) or low (12.5%), which is defined similarly.

Rate of buffer: rate of safety leadtime takes a value in {0, 0.1, 0.2, . . ., 1.0}. It means that the rate of safety leadtime that is additionally reserved for the predefined procurement leadtime. Rate of safety stock is a rate of additional quantity to total required quantity.

For example, a case has high-level uncertainty in supply quantity and 20% safety leadtime with advanced notification. Then, APPCS schedules with a procurement leadtime 1.2 times longer than usual, and 75% of purchase orders will change in respective quantities. If a change is reported from a supplier, then rescheduling will be executed. Each case is simulated 40 times to provide a data for the case, and performance indices are calculated for the data.

An average service level is the evaluation variable in our simulation. It is defined as the average ratio of the number of demands finished as it was planned to the number of all demands. A point on a curve in Fig. 3.12 shows the mean of 40 data. The upper and lower vertical line-segments with a point show the second and third quartiles, respectively, indicating what variety the 40 data has. The hollow curve shows cases of using degrees of safety leadtime, and the solid curve shows ones using safety stock against high degree of supply time uncertainty with advanced notification. In the following graphs, only means are shown to draw curves.

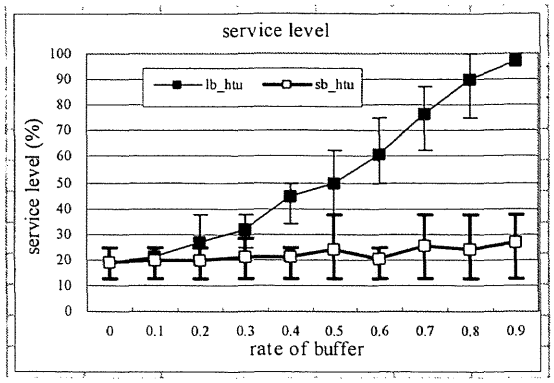


Fig. 3.12: Distribution of evaluation variables of the simulation result (lb_htu: using safety leadtime against high time uncertainty, sb_htu: using safety stock against high time uncertainty)

3.4.3 Simulation Result

Supply Uncertainty

Fig. 3.13a shows the relationship between service level and rate of buffering. The hollow curves depict that relationship with safety stock (sb_), while the solid curves depict that with safety leadtime (lb_). For both kinds of lines, there are three cases: under high (_htu), medium (_mtu) and low (_ltu) levels of supply time uncertainty. The higher the rate of uncertainty is, the lower the service level can be achieved. Note that the service level increases much sharply with rate of safety leadtime buffer than that of safety stock buffer.

The reason why safety leadtime is effective against supply time uncertainty is twofold. When the prescribed safety leadtime is longer than delay time, or when the notification time is early enough, the uncertainty will not delay the production. This result suggests that safety leadtime can be adjusted by the APPCS to achieve a target level of service in different degree of uncertainty environments. When safety stock is used against supply time uncertainty, there is an upper limit of service level for each level of uncertainty. Hence, safety leadtime is more flexible than safety stock under supply time uncertainty.

Fig. 3.13b shows the relationship between service level and supply quantity uncertainty. Since each service level goes gradually up to 100% as rate of both buffers increased, both types of buffer are effective against supply quantity uncertainty. When supply quantity uncertainty is informed, some action must be taken to supplement the shortage. If safety leadtime is reserved, releasing another purchase order is inevitable and no production can be started until the arrival of the purchase material. If safety stock is reserved, the shortage is supplemented either from stock or by releasing another purchase order. The result of the experiment shows that both buffering approaches are adjustable to achieve a target service

level under supply quantity uncertainty.

Fig. 3.13c and 3.13d show the result of the same experiment with Fig. 3.13a and 3.13b, respectively, but the plant will not know the change until the promised delivery date. Thus, the effect of notification on service level can be obtained by the difference of the two pairs of figures. By comparing Fig. 3.13a and 3.13c, we observe that safety leadtime is more effective than safety stock. The comparison of Fig. 3.13b and 3.13d brings us to the fact that the service level of using safety leadtime falls dramatically.

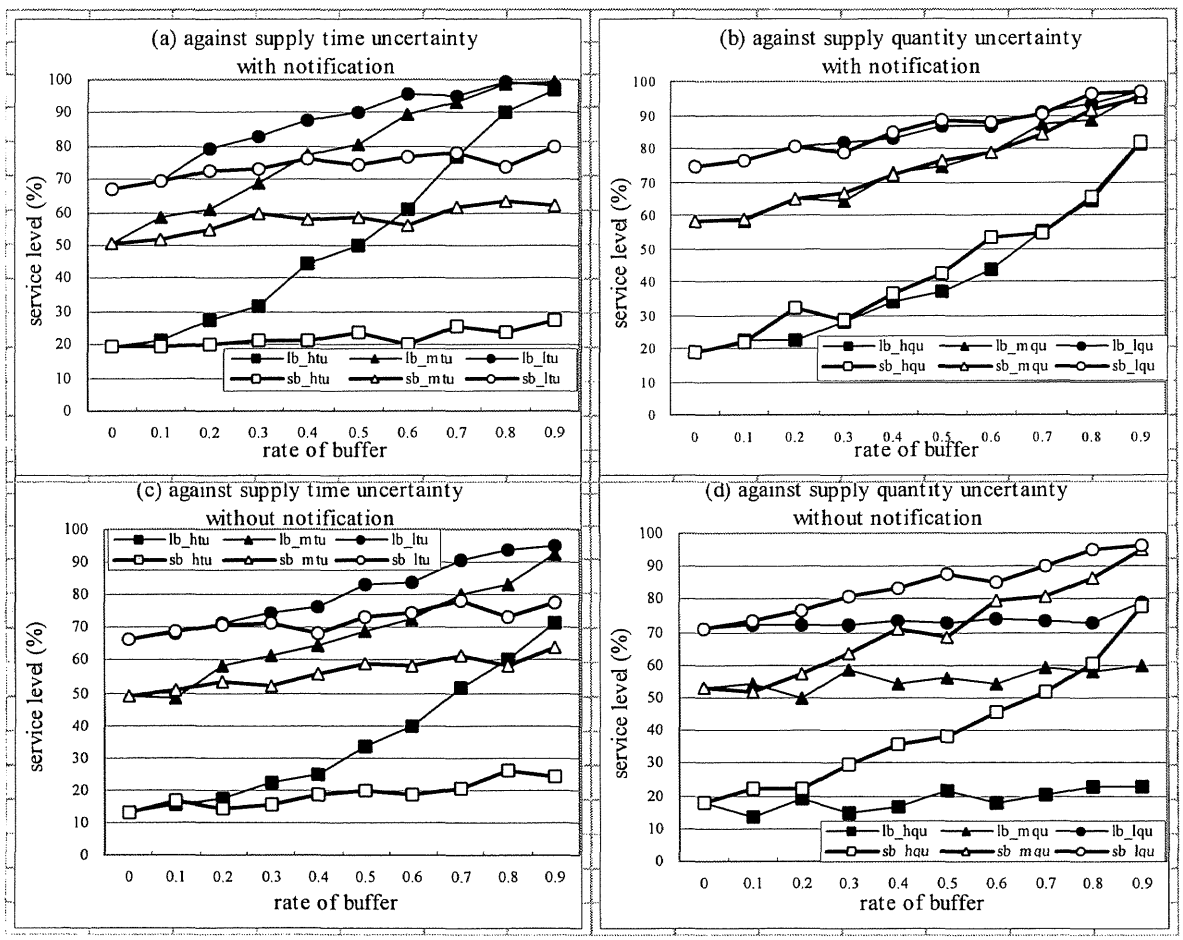


Fig. 3.13: Service level under supply time/quantity uncertainty (xx_yzz: using xx buffer against y level zz uncertainty; xx='lb': safety leadtime, xx='sb': safety stock; y='h': high, y='m': medium, y='l': low; zz='tu': time uncertainty, zz='qu': quantity uncertainty)

Among the related works in comparing safety leadtime and safety stock, a pioneering work was done by Whybark and Williams (1976) who compared the two buffers under both demand and supply uncertainty. They modeled a representative part in an MRP system and concluded that safety leadtime can protect from time uncertainty and the safety stock buffer performs well under quantity uncertainty in either case of demand or supply. If we can use advance notification that informs future change in a schedule, however, then the safety

leadtime gives more flexibility in response to the four types of uncertainty than the safety stock does. Besides, we found that the earlier notification policy, which triggers a release of additional purchase order if rescheduling suggests, can improve the service level and decrease the delay time when demand uncertainty happened. The notification approach can be regarded as another form of safety leadtime that is offered by external vendors and with less effort.

Demand Uncertainty

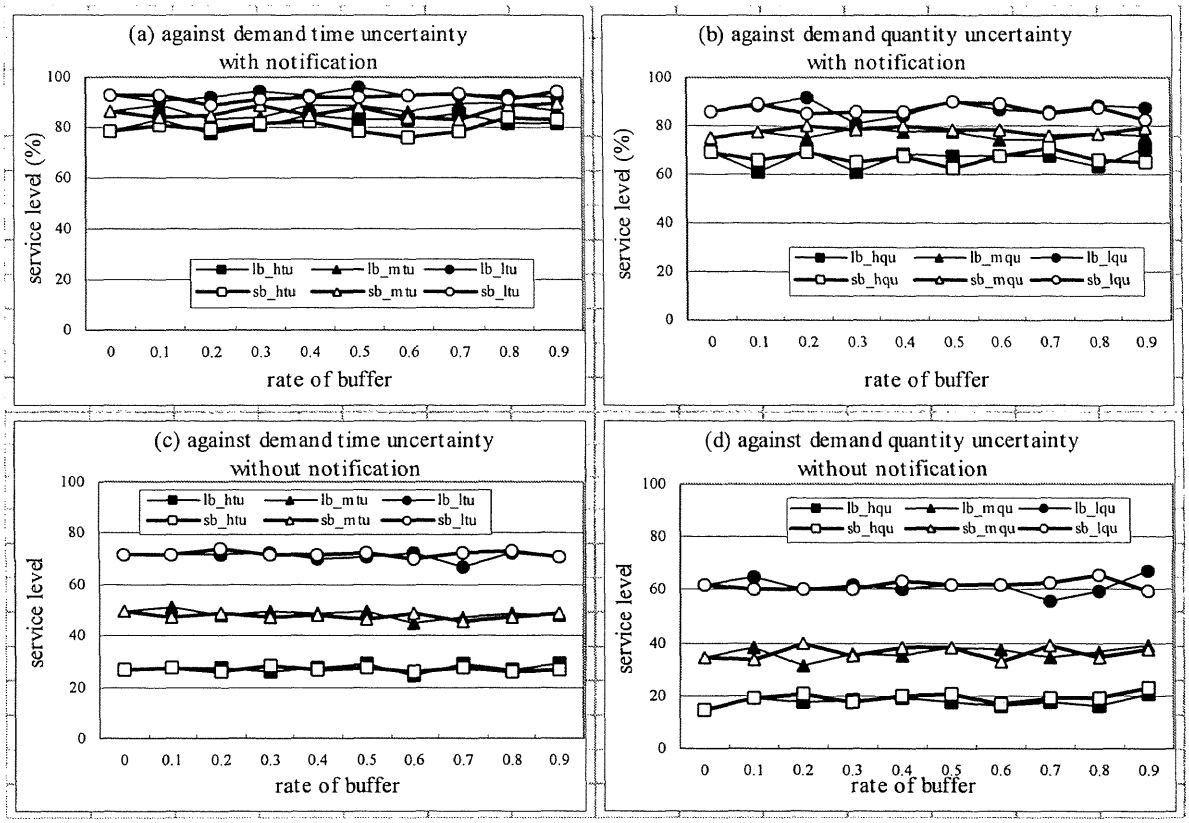


Fig. 3.14: Service level under demand time/quantity uncertainty (xx_yzz: using xx buffer against y level zz uncertainty; xx='lb': safety leadtime, xx='sb': safety stock; y='h': high, y='m': medium, y='l': low; zz='tu': time uncertainty, zz='qu': quantity uncertainty)

Fig. 3.14a and 3.14b are the results of service level under demand time uncertainty (tu) and demand quantity uncertainty (qu) under 0.75 (_h), 0.5 (_m), and 0.25 (_l) degrees of demand uncertainty. Fig. 3.14b and 3.14c show the same experiment without advance notification. It is clear that both safety buffers in procurement are not helpful when increasing the service levels under both types of demand uncertainty. Examining the cases in detail, it was found that if any demand order is changed without advance notification, then it must be delayed. Therefore, as shown in Fig. 3.14c and 3.14d, the service level is almost equal to one minus the rate of uncertainty.

The difference of the service level between Fig. 3.14a and 3.14c or that of Fig. 3.14b and

3.14d shows the effect of advance notification. It improves the service level from 27 to 82% when demand time uncertainty is at high level and from 18 to 67% when demand quantity uncertainty is at high level. It is obvious that the introduction of the advance notification increases the service level. The demand uncertainty is transferred from consumer or retailer through levels of production down to suppliers of the supply chain. Without advance notification, the demand change cannot trigger the rescheduling procedure in real time or cannot be transferred through suppliers. The delay or shortage will be apparent all of a sudden when procured materials will arrive. We conclude that other than supply safety leadtime or safety stock, the notification approach improves the service level under demand uncertainty.

3.5. Summary

Agile production planning and control system (APPCS) with advance notification has been proposed. It uses a planning cycle and builds a feasible production plan for a set of demands. APPCS can enhance the agility of production processes in the sense that it does immediate rescheduling upon advance notification from customers and/or suppliers. The rescheduling of APPCS allows a response in accordance with various situations.

Although APPCS does not use work-in-process (WIP) inventory for assemblies or products, a buffering mechanism against uncertainty can be set in a purchase plan. Users set throughput by the production and purchase plan, and also set the safety leadtime and/or safety stock according to a target service level. As far as the example in section 3.4 is concerned, our simulation analysis showed that safety leadtime is preferable to safety stock in most uncertain situations. Furthermore, advance notification can be regarded as another form of safety leadtime offered by external vendors.

When APPCS is used in a specific firm, a set of demands should be decided. Those demands can be formed from past operation, or from strategically meaningful assumptions such as profit plan for the next year. Then, through simulation, buffering parameters can be decided corresponding to specified service level.

The proposed scheduling method needs lot-for-lot sizing. If the lot-for-lot procedure cannot be employed for some reason or other, rescheduling might bring instability in the schedule. Reduction of such nervousness in rescheduling is out of the scope of this dissertation. Furthermore, APPCS can be used as a real-time system in the sense that it makes a schedule when a customer order arrives and a reschedule when uncertainty happens. It is not clear whether such a real-time APPCS is preferable to the APPCS. This comparison is another topic for future study.

4 Model of Agile Production Planning and Control

4.1. Introduction

This chapter provides a model of agile production planning and control (APPCM). A data model for product data management and planning is available in Scheer (1994). This chapter showed an augmentation of the Sheer's model so that agile production planning and control system (APPCS) is possible. The formulation of APPCS is described by the universal modeling language (UML) and illustrated by class diagram and statechart diagram.

Model of production planning and control (PPCM) described in chapter 2 and system requirements of APPCS specified in chapter 3 act as guidelines of system analysis and design. Data view, constraints, and behavior view of APPCM are shown in section 4.2, 4.3, and 4.4, respectively. The modeling elements of APPCM mapping to that of PPCM is listed in section 4.5 to show APPCM is an implementation of production planning and control (PPC) and APPCS is a complete PPC system. In section 4.6, we provide an example of the APPCM to demonstrate how the APPCS is applied. Finally, a summary is provided in the final section of this chapter.

4.2. Data view

Class diagram of UML is used to describe the data view of APPCM as shown in Fig. 4.1. The notations and descriptions about the elements used in class diagram are provided in Appendix A.3. How the classes used by APPCM to refine the requirement of PPCM is not shown in the figure. The classes of PPCM are listed as subtitle in the data view section, and classes of APPCM are presented under the respective subtitle to show the mapping of the two models.

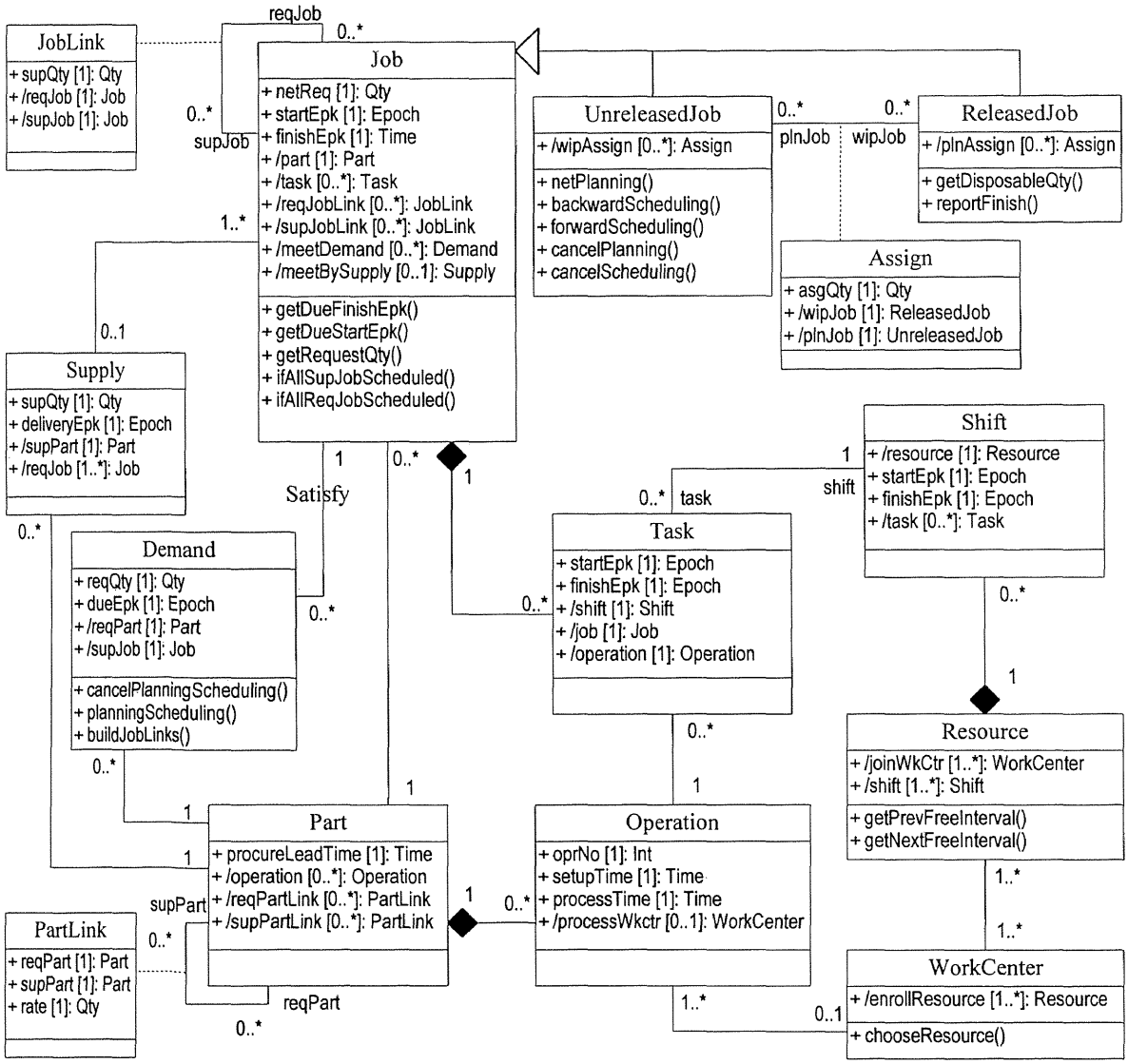


Fig. 4.1: A class diagram of APPCM

Part

The *Part* class is an abstraction of a set of elements such as finished products, assemblies, and raw materials that are used in production or procurement. The attribute *procureLeadTime* of a part represents the time for procuring or outsourcing the part.

A set of operations is specified for a finished product and an assembly. The *Operation* class represents an abstraction of those operations. Since a part owns its operations, there is a composite association from the *Part* class to the *Operation* class. For clarity, an association is also shown in a class by an attribute that begins with a "/" to distinguish it from other attributes. The attributes *setupTime*, and *processTime* of an operation preserve the setup time and processing time per unit of the operation to calculate the total processing time in the scheduling phase. A link *processWkCtr* from an operation to a work center shows that the operation is processed at the work center.

A bill of materials (BOM) of a part can be regarded as a set of links that connect the part to its components with each link specifying the rate (number of components required per part). Association *PartLink* with its both ends attaching to the *Part* class represents the request-supply links between parts. The association is also a class because it has an attribute *rate* that indicates the number of components required per part. A part can be a component of other parts, and a part can be made of other components. For a part, *reqPartLink* attribute indicates links from the part to its requesting parts, and *supPartLink* attribute shows links from the part to its supplying parts.

Resource

A work center is a category of capabilities for processing some operations. Assembly lines, polishing shops, and brushing machines are examples of respective work centers. A resource is either a labor or a machine. The *WorkCenter* class is an abstract entity to represent the work centers in a plant, and the *Resource* class is an abstraction of the resources. APPCS assumes the resource flexibility that a work center can enroll numbers of resources, and a resource can join more than one work center. Accordingly, there is a many to many association between the two classes.

An association between the *Operation* class and the *WorkCenter* class suggests that at most one work center can be assigned to an operation for manufacturing, and many operations can be processed at a work center.

The capacity of a resource is finite and managed by shifts. A shift defines the workable time on the planning horizon, which indicates a time axis toward the future. The *Shift* class with attributes of *startEpk* and *finishEpk* is an abstraction of a group of intervals that show capacities of the resources. Hence, there is a composite association between the *Shift* class and the *Resource* class.

Demand

A demand is used to represent an independent requirement for a finished product before a due epoch. A demand, either the result of forecasting or customer orders, acts as a base unit of net requirement planning and scheduling in APPCS. The *Demand* class is an abstraction of the demands. A demand has attributes of requested quantity (*reqQty*), due epoch (*dueEpk*), and requested part (*reqPart*). The requested part is also a link from the demand to a part.

Task

A demand is transformed into a set of jobs in the net requirement planning phase. A job is defined as a net requirement of a part that is manufactured or procured during a period of time. The start epoch and finish epoch of a job are known after scheduling is completed.

The *Job* class is an abstraction of the jobs generated by net requirement planning. Its attributes are net requirement (*netReq*), start epoch (*startEpk*), finish epoch (*finishEpk*), and requested part (*part*) that is linking to the *Part* class. An association from the *Job* class to the *Demand* class shows that a demand must be satisfied by exactly one job, and a job can be generated to meet many demands.

During production execution, a job is released to the shop floor or to a vender in form of purchase order when it is time to start the job, and its output is input to other jobs after the job is finished. The request-supply relations between the planned jobs form a hierarchy of jobs.

The *JobLink* association class is to describe the request-supply relations between any two jobs. The attribute *supQty* of a link from a job to a supplying job denotes a gross requirement for the supplying job. To view from a job, the attribute *reqJobLink* is an abstraction of the links from the job to its requesting jobs, and *supJobLink* from a job to its supplying jobs. All the jobs are exploded from external demands. We can traverse all jobs explored from a demand if the demand has a link with the root job, which is the job without requesting job in the hierarchy of the jobs. Conversely, the link starting from a root job to a demand provides the information for transferring the finished product. Hence, a bi-directional association is specified between the *Demand* class and the *Job* class.

Stock on hand can be viewed as some jobs that are finished manufacturing or procurement. The WIP can be viewed as an in-processing job not reserved by other jobs. When a change occurred, except for the released jobs, the planned jobs that are affected by the change will be cancelled. During rescheduling, disposable net requirement of the released jobs can be assigned to the planning jobs. For these reasons, the *Job* class has two sub classes, the *ReleasedJob* class and the *UnreleasedJob* class, for identifying the different operations of the two types of job.

The *Assign* association class is an abstraction of the assignments from a released job (*wipJob*) to a planning job (*plnJob*) with an assign quantity (*asgQty*). The multiplicity

suggests that net requirement of a released job can be assign to many planning jobs, and a planning job can take assignments from various released jobs. For an planning job of the *UnreleasedJob* class, the attribute *wipAssign* is an abstraction of the links from assigning jobs to the planning job. For an assigning job of the *ReleasedJob* class, *plnAssign* is an abstraction of the links from planning jobs to the assigning job.

Net requirement of a manufacturing job is transformed to a set of tasks in the scheduling phase. The *Task* class is an entity to represent all the tasks. The processing operation (*operation*: *Operation*) and an unoccupied interval starting from *startEpk* and ending at *finishEpk* within a workable shift (*shift*: *Shift*) of a resource must be specified for a valid task.

To prevent a resource from being assigned a reserved area of its shifts to other tasks, the reserved areas within a shift should be managed. A composite association from the *Shift* class to the *Task* class is for this purpose. The multiplicity reveals that a shift can be shared with many tasks without duplication, and the interval of a task can only be located on a shift.

Procurement

The net requirements of procurement jobs are aggregated and then released to the respective venders with by purchase orders. The *Supply* class acts as an abstraction of the purchase orders. For a purchase order, purchased raw material (*supPart*), promised delivery epoch (*deliveryEpk*), supplied quantity (*supQty*) are specified.

A purchase order can be generated to meet the requirements of one or more than one procurement job. Since there is no need to separate the net requirement into more than two purchase orders purposely, a job is supplied only by a purchase order. An association between the *Supply* class and the *Job* class is an abstraction of such request and supply links.

4.3. Constraints

The model of agile production planning and control (APPCM) should satisfy certain conditions. The conditions are specified to restrict some actions of the model elements. A constraint is a semantic relationship that must be maintained; otherwise, the system described by the model is invalid (OMG 2002). The semantics is described in terms of the set theory. Regarding *Part*, *PartLink*, *Operation*, *WorkCenter*, *Resource*, *Shift*, *Demand*, *Job*, *JobLink*, *Assign*, *Task*, and *Supply* as tables, "$e \in A$" means e is an element of a table A, "*e.attr*" and "*e.opr*()" shows attribute *attr* and operation *opr*() of element e, respectively, and "$a=b$" implies a and b are the same elements.

- The part of a job that is generated to meet a demand must be the same with the requested

part of the demand.

$$(\forall d \in Demand)\ (d.reqPart = d.reqJob.part) \tag{4.1}$$

- The part of a planning job must be the same with the part of a WIP (released job) whose net requirement is assigned to the job.

$$(\forall a \in Assign)\ (a.wipJob.part = a.plnJob.part) \tag{4.2}$$

- To assign net requirement of a WIP (released job) to a planning job, the WIP must be finished in time to supply all requesting jobs of the planning job.

$$(\forall a \in Assign)\ (\forall jl \in a.plnJob.reqJobLink)\ (a.wipJob.finishEpk \leq jl.reqJob.startEpk) \tag{4.3}$$

- The total assigned quantity of a WIP among requirements cannot exceed net requirement of the WIP.

$$(\forall rj \in ReleasedJob)\left(\sum_{a \in rj.plnAssign} a.asgQty \leq rj.netReq\right) \tag{4.4}$$

- To meet the net requirement of a manufacturing job, there must be a supplying job created for every component part of the part that the manufacturing job requested.

$$(\forall j \in Job)\ (\forall pl \in j.part.supPartLink)\ (\exists\, jl \in j.supJobLink)\ (pl.supPart = jl.supJob.part) \tag{4.5}$$

- A link between a job and a supplying job represents a gross requirement for the component part of the supplying job. The gross requirement is the product of net requirement of the job with the number of component parts per part of the job.

$$(\forall jl \in JobLink)\ (\exists\, pl=(jl.reqJob.part, jl.supJob.part) \in PartLink)\ (jl.reqJob.netReq \times pl.rate = jl.supQty) \tag{4.6}$$

- Precedence constraints of jobs: A job must be finished before the start epoch of all of its requesting jobs, and started after the finish epoch of all of its supplying jobs.

$(\forall j \in Job)\ [(\forall rj \in j.reqJob)\ (j.finishEpk \leq rj.startEpk)$ and $(\forall sj \in j.supJob)\ (j.startEpk \geq sj.finishEpk)]$ (4.7)

- The minimum level of net requirement of a job equals to the sum of gross requirements from its requesting jobs and independent requirements from demand minus the WIP assignment, if any. The surplus of net requirement might be reserved for the sake of safety stock.

$$(\forall j \in UnreleasedJob)(j.netReq) \geq \sum_{d \in j.meetDemand} d.reqQty + \sum_{jl \in j.reqJobLink} jl.supQty - \sum_{a \in j.wipAssign} a.asgQty) \quad (4.8)$$

- The part of the purchasing order should be the same with the part of the procurement jobs.

$$(\forall s \in Supply)\ (\forall rj \in s.reqJob)\ (s.supPart = rj.part) \quad (4.9)$$

- Quantity of a purchasing order must not be less than the sum of net requirements of all the procurement jobs. The promise (delivery) epoch of a purchasing order must be earlier than the earliest finish epoch among the procurement jobs. The surplus might be reserved for the sake of safety stock, and the earlier delivery than required could be for the reason of safety leadtime.

$$(\forall s \in Supply)\ (s.supQty \geq \sum_{rj \in s.reqJob} rj.netReq \text{ and } s.deliveryEpk \leq \operatorname{Min}_{rj \in s.reqJob} rj.finishEpk) \quad (4.10)$$

- A resource can join at most one shift simultaneously.

$$(\forall r \in Resource)\ (\forall i, j \in r.shift)\ (\ [i.startEpk, i.finishEpk) \cap [j.startEpk, j.finishEpk) = \emptyset) \quad (4.11)$$

- For a job, all of the operations defined in a routing for making the part must be scheduled.

$$(\forall j \in Job)\ (\forall o \in j.part.operation)\ (\exists\ k \in j.task)\ (o = k.operation) \quad (4.12)$$

- The precedence constrains of tasks in a job must be followed.

$(\forall j \in Job)\ (\forall\ m, n \in j.task)$ (if $m.operation.oprNo > n.operation.oprNo$, then $m.startEpk > n.finishEpk$) (4.13)

- When choosing a resource for a task working on an operation, the resource must be one of the resources enrolled in the assigned work center of the operation.

$(\forall j \in Job)\ (\forall k \in j.task)\ (k.shift.resource \in k.operation.processWkCtr.enrollResource)$ (4.14)

- By assuming finite loading, the interval reserved for a task must be a subset of the occupied shift.

$(\forall j \in Job)\ (\forall k \in j.task)\ ([k.startEpk, k.finishEpk) \subseteq [k.shift.startEpk, k.shift.finishEpk)$ (4.15)

- There must be no intersection between intervals reserved by tasks in a shift.

$(\forall s \in Shift)\ (\forall\ m, n \in s.task)\ ([m.startEpk, m.finishEpk) \cap [n.startEpk, n.finishEpk) = \emptyset)$ (4.16)

4.4. Behavior view

4.4.1 State Transition of a Job

A job has 5 states in its life cycle as shown in Fig. 4.2. They are "created", "planned", "scheduled", "released", and "finished". In the figure, the ellipse shows a state, a solid cycle an initial state, a solid cycle with double lines a final state, and the arrow line connecting two states an event. Two types of event are used in the state chart. A change event occurs when an expression becomes true as a result of a change in value of one or more attributes or associations. A call event represents the reception of a request to synchronously invoke a specific operation. A state is driven by an event to jump to another state.

A job is created to represent a net requirement for a part. A set of jobs are created for a demand.

The planned net requirement (*job.netReq*) of a job (*job*: *UnreleasedJob*) is known after executing operation *job.netPlanning*(). The detail procedures of the methods used in this section are listed in appendix B.

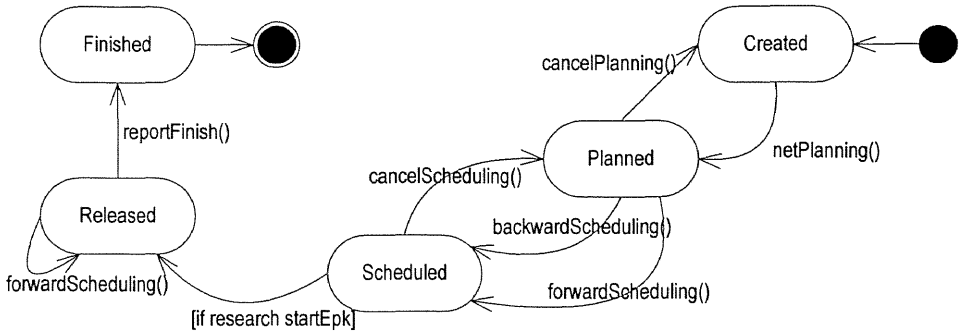


Fig. 4.2: State chart of a job

A "planned" job (*pjb*: *UnreleasedJob*) can go to the scheduled state by scheduling. To satisfy the requirement of a job, the part requested by the job should be processed for all operations. The scheduling process reserves capacity of a resource for each operation of the part. A set of tasks is generated by scheduling for a planned job. The scheduling process is bi-directional. Backward scheduling (*pjb.backwardScheduling*()) of the planned job starts with the last operation and ends by the first operation from a due finish epoch, which is the earliest start epoch among its requesting jobs. Forward scheduling (*pjb.forwardScheduling*()) of the planned job begins with the first operation by starting from a due start epoch, which is the latest finish epoch among the supplying jobs.

A job in the "scheduled" state goes back to the "planned" state when the result of scheduling is infeasible or some changes trigger rescheduling by canceling the scheduled set of tasks, freeing the reserved capacity of resources, and returning the assigned WIP. The operation *sjb.cancelScheduling*() of a scheduled job (*sjb*: *UnreleasedJob*) is invoked to do these works.

A job enters the "released" state when it is the time of *startEpk*. A manufacturing job is released to the shop floor for production. A procurement job is transformed to a purchase order, and released to a vender. We assume that a released job cannot be canceled, i.e. it is unable to go back to the "scheduled" state.

After a job is released, the disposable quantity *rjb.getDisposableQty*() of the job (*rjb*: *ReleasedJob*) can be assigned to other planning jobs in the planning phase. When the job is completed, it goes to the "finished" state by invoking operation *rjb.reportFinish*(). The released job must be finished before the planned finish epoch, or an event is triggered to inform the delay and a new schedule will be generated by APPCS based on the delayed job to incorporate the delay.

A "finished" job is either sent to the requesting jobs, or becomes stock on hand for assigning to other planning jobs in the next planning run. Finally, a finished job enters the final state when all the disposable quantity is used up.

4.4.2 Sequential Flow of a Demand

A demand is a basic unit of planning and scheduling in APPCS. The planning and scheduling of a demand is executed by invoking the operation *dmd.planningScheduling*() of the demand *dmd*: *Demand*. It generates a set of jobs and a set of tasks for the jobs. Fig. 4.3 shows a pseudo code of the operation. Details of the operation and the operations invoked inside are given in appendix B.

```
(1) Job Initialization
(2a) Net requirement planning of jobs
(2b) Backward scheduling of job of jobs
IF the schedule is infeasible
   (3) Canceling the scheduling
   (4) Forward scheduling of jobs
END IF
```

Fig. 4.3: Pseudo code of *Demand.planningScheduling*( ) for net requirement planning and scheduling a demand

The first step of planning and scheduling is to initialize the jobs that will be used later. If it is the first time for a demand to execute planning and scheduling, then *buildJobLinks*() operation of the demand is invoked to build the hierarchy of jobs by following instances of the *Part* class and links of the association *PartLink*. This operation acts as a preparatory work of accumulating all the requirements of such part in a job. If not, there has a hierarchy of parts, and *cancelPlanningScheduling*() is invoked to cancel tasks of the unreleased jobs.

There are two synchronized operations in the second step of planning and scheduling. One is net requirement planning (NP) that plans net requirement of a job. Another one is backward scheduling (BS) that generates a set of tasks for a job. A job can run NP only when the due finish epoch is determined. In other words, the requesting jobs of the planning job, if any, must be in the "scheduled" state. Because the due finish epoch is necessary for the planning job to know whether some WIP assignments are usable or not. A job can run BS only when it is "planned".

PSet and *BSet* are the two sets of jobs that are used to prevent the execution of NP process and BS process from violating the precedence constraints. The *netPlanning*() operation is invoked for any job waiting in *PSet*. The *backwardScheduling*() operation is called for any job waiting in *BSet*. After a job executes NP, if net requirement of the job is greater than 0, then it is appended to *BSet*. After a job runs BS, the supplying jobs that are capable of NP will be appended to *PSet*. A job is capable of NP if it has no requesting job or all of its requesting jobs are scheduled. In APPCS, whether a job is capable of NP is checked by operation *ifAllReqJobScheduled*().

The planning and scheduling of a demand first performs in a top-down direction from the finished product level to the raw material level by a combination of NP and BS processes until *PSet* and *BSet* are empty. The job with the highest priority to run NP can first reserve the disposable quantity of the released jobs (WIP). The job with the first priority of running BS can reserve the available capacity of resources. The sequence of jobs running NP and BS processes determines a schedule for a demand.

Some rules help determining the schedule. For example, a breadth-first rule assumes that NP is invoked when there is no job in *BSet*, and BS is triggered when there is no job in *PSet*. A depth-first rule assumes that *PSet* is a first-in-last-out (FILO) stock, and a job runs BS as soon as it enters *BSet*. In addition, selecting the job that has the earliest finish epoch (EDD) or the shortest processing time (SPT) within *PSet* or *BSet* is the rule that usually used.

Fig. 4.4 shows 3 sequences of planning and scheduling of a demand $d1$ by assuming depth-first rule, breadth-first rule, and no rule. In the figure, *B* indicates the *BSet*, *P* the *PSet*, NP(y) net requirement planning of job y, BS(x) backward scheduling of job x, and element a within the sets means the job of a part a.

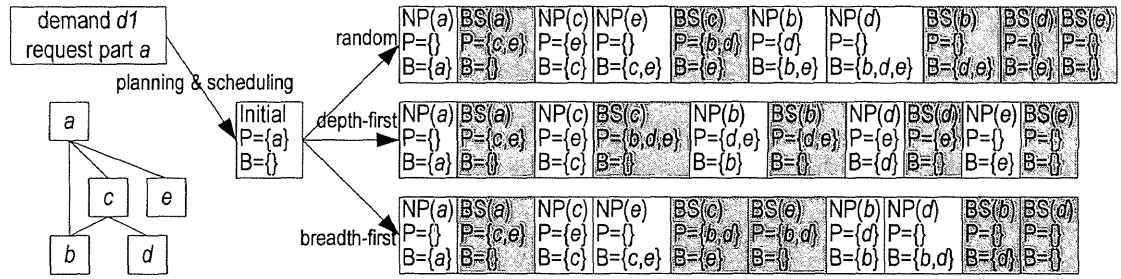


Fig. 4.4: Three possible sequences of executing net requirement planning and backward scheduling

The planning and backward scheduling generates a set of jobs and each job possesses a set of tasks. The jobs together with the request-supply relations form a hierarchy of jobs. If the result shows that the start epoch of any job is located in the past, then the result is infeasible and should be abandoned. Instead, the bottom-up approach that performs forward scheduling from the leaf jobs to the root job is adopted.

As shown in Fig. 4.1, APPCM provides no data structure to memorize the leaf jobs in the hierarchy of jobs that is constructed for a demand. To search for the leaf jobs of a demand, searching first top-down and then bottom-up is inevitable. During top-down searching, *cancelScheduling*() is invoked to cancel the tasks generated by BS for all unreleased jobs.

FSet is a set of jobs that is used to prevent the execution of forward scheduling (FS) process from violating the precedence constraints. After searching top-down, *FSet* contains only the leaf jobs. The method *forwardScheduling*() is invoked to execute FS for the job waiting in *FSet*. The job capable of FS is the job with no supplying job or all of its supplying

jobs are scheduled. In APPCS, whether a job is capable of FS is checked by operation *ifAllSupJobScheduled*().

After a job finish FS, its requesting jobs that are capable of FS are appended to *FSet*. The sequence of FS is determined by choosing the next job in *FSet*. The priority rule, for instance, can be the shortest processing time, earliest due epoch, and smallest net requirement, etc.

A demand begins when any of its jobs is released. During production, if any uncertainty occurs to a demand, then the demand must execute *planningScheduling*() again to remedy the change. After the root job of a demand finishes manufacturing, the demand is finished and the finished product will be shipped to the customer.

4.4.3 State Transition of Procurement Job

After all the demands finish planning and scheduling, purchase orders are generated to meet the net requirements of the procurement jobs. The procurement jobs that request the same part (usually raw material) and whose start epochs are within a period of time are usually integrated as a purchase order.

As soon as a purchase order *sup*:*Supply* is created, it is released to a vender, and all the requesting procurement jobs (*sup.reqJob*) are marked with "released" automatically, and they enter the "finished" state after the supply is delivered. If supply uncertainty occurs to a purchase order, then the demands that directly or indirectly use the part supplied from the purchase order must execute *planningScheduling*() again to remedy the change.

4.5. Mapping to PPCM and Comparing with MRPM

APPCS is a complete production planning and control system, since the modeling elements of production planning and control model (PPCM) are implemented by the classes and functions of APPCM as shown in Table 4.1. Both material requirement planning model (MRPM) and APPCM are both complete systems. The difference of the two models is shown in Table 4.2. In the table, (:*C*) and (:*A*) denote instances of the *C* class, and links of the *A* association, respectively.

Table 4.1: Model elements mapping from APPCM to PPCM

	PPCM	Mapping by APPCM
Classes	*Part*	*Part*; *PartLink*; *Operation*;
	Epoch	It becomes a data type (Epoch) in APPCM.
	Resource	*WorkCenter*; *Resource*; *Shift*;
	Demand	*Demand*
	Task	*Job*; *UnreleasedJob*; *ReleasedJob*; *JobLink*; *Task*;
	Satisfy	*Satisfy*
	TaskLink	*JobLink*;
Functions	Planning	Net requirement planning, and scheduling are executed together by *Demand.planningScheduling*( ).
	Execution	The task in *Task* is released when it is time to start.
	Control	*Satisfy* and *JobLink* are associations used to identify the affected demands and jobs by a change.

Table 4.2: A comparison between APPCM and MRPM

	APPCM	MRPM
Epoch	Real time;	Discrete time (bucket);
Resource	Assume finite capacity; Capacity is specified in a resource by shifts (:*Shift*), the actual workable durations; Capacity of a work center is the union of all shifts of the resources enrolled in the work center;	Assume infinite capacity; Capacity (:*Capacity*) is specified in a work center by work hours for each time bucket;
Task	A task (:*Task*) has the facets of part, epoch, and resource;	A planned order (:*PlannedOrder*) has the facets of part and epoch; A workload (:*Workload*) extends a planned order has the facets of part, epoch, and resource;
TaskLink	The request-supply relations of jobs are defined in *JobLink* class;	Gross requirements (:*GrossReq*) are fulfilled by planned orders (:*PlannedOrder*) through the *Fulfill* association; A planned order (:*PlannedOrder*) is exploded to a set of gross requirements (:*GrossReq*) through the *Explosion* association;
Planning	Net requirement planning and scheduling are executed in parallel; A job (:*Job*) is the result of net requirement planning, a task (:*Task*) is the result of scheduling;	Material requirement planning is first executed, planned orders (:*PlannedOrder*) are the planning result; Short-term scheduling is executed at shop floor after a planned order is released, workloads (:*Workload*) are the scheduling result;
Execution	A task (:*Task*) is released to shop floor for production;	A workload (:*Workload*) is scheduled and directly executed at shop floor;
Control	Links (:*Satisfy*) between demands and jobs, and links (:*JobLink*) between requesting jobs and supplying jobs are used to identify the affected demands and jobs by a change;	Links (:*Satisfy*) between demands and gross requirements, links (:*Fullfill*) from planned orders to gross requirements, and links (:*Explosion*) from gross requirements to planned orders are used to identify the affected demands, planned orders, and gross requirements by a change;

4.6. Testifying APPCM

In order to testify the proposed UML model of APPCS, we discuss two issues: a simulator and an instance.

4.6.1 Simulator

We implemented it as part of a simulator (Sato and Tsai, 2004). The simulator is used in investigating the setting of safety stock or safety leadtime to protect against unexpected events by rescheduling. It is implemented by integrating VC++™ and MSSQL™ database.

4.6.2 Instance of APPCS

To give a whole image of the APPCS and to testify the model, we show an instance of the model in this subsection. Fig. 4.5a shows the instances of product data and initial data for the example. The instances are contained in tables: *Part*, *Operation*, *PartLink*, *WorkCenter*, *Resource*, *Shift*, *Task*, and *Demand*. Elements of the tables follow the definition of class diagram in Fig. 4.1 and the constraints in section 4.2. Initially, a task from epoch 100 to 200 of resource y is scheduled for maintenance.

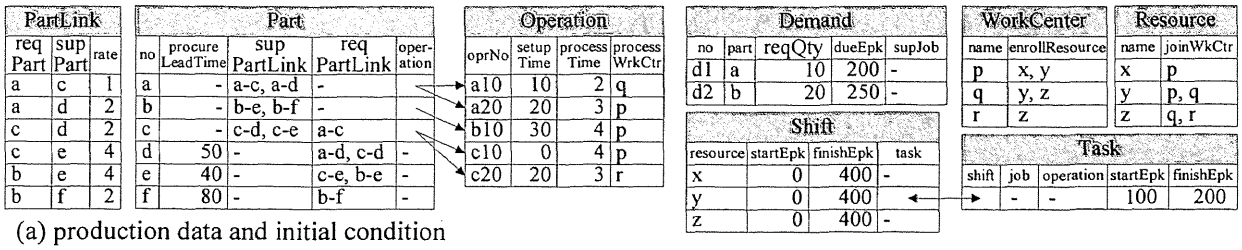
Two instances in the Demand table run *planningScheduling*() in epoch 0 by following the earliest due date (EDD) rule. In case of backward scheduling of a job, the *chooseResource*() obeys the "latest start epoch" rule, and in case of forward scheduling, the "earliest finish epoch" rule is adopted. A result of planning and backward scheduling of demand d1 is shown in Fig. 4.5b by tables of *Job*, *Task*, and *Shift*. The job sequence of planning and scheduling is 1, 2, 3, and 4.

Since the results show that job 3 and job 4 start from the past, hence they are abandoned. Forward scheduling is used alternatively for demand d1 by the job sequence 3, 4, 2, and 1. The result of planning, scheduling, and procuring of demand d1 and d2 is shown in Fig. 4.5c by tables of *Demand*, *Job*, *JobLink*, *Assign*, *Task*, and *Supply*. The procurement jobs 4 and 6 are integrated to form a purchase order s2. Then the production activity begins according to jobs and supplies in Fig. 4.5c.

- Epoch 0: Purchase orders s1 and s2 are released suppliers, and the corresponding procurement jobs 3, 4, 6 enter the "released" state.
- Epoch 10: Purchase order s3 is released, and job 7 becomes released.
- Epoch 40: Supply s2 arrives, and jobs 4 and 6 enter the "finished" state.
- Epoch 50: Supply s1 arrives, and job 3 enters the "finished" state. Operation c10 of job 2 begins with consuming 20 units of part d from job 3 and 40 units of part e from job 4.

Disposable quantity of job 3 becomes 20, and job 4 is used up and destroyed.

- Epoch 60: The supplier informs that the arrival of supply s3 will be delayed for 20 time units, i.e. becomes epoch 110. Since demand d2 is affected by the change, so the not-started job (job 5) of the demand and its schedules are cancelled. Demand 2 runs *planningScheduling*() again to remedy the change based on jobs 6 and 7. In Fig. 4.5d, the result shows that the supply change delays demand d2 for 20 time units.
- Epoch 110: The arrival of the delayed supply s3 makes job 7 enter "finished" state. Job 8 starts and consumes jobs 9 and 10. Job 6, 7, 9, and 10 are used up and thus destroyed.
- Epoch 140: Job 2 is finished. Operation a10 of job 1 starts with consuming 10 of part c from job 2 and 20 of part d from job 3. Job 2 and 3 are used up and thus destroyed.
- Epoch 200: The customer requests 2 more units of demand 1 and desires to know the earliest possible delivery date. Since the only job that belongs to demand 1 is ongoing, it cannot be cancelled. Demand 1 runs *planningScheduling*() again based on job 1. Fig. 4.5e shows that the additional demand can be offered at epoch 324 if purchase orders s4 and s5 can be released immediately.
- After epoch 220: Job 1 finished at epoch 220. Supply s5 arrives at epoch 240, and s4 at 250. Jobs 13 and 14 enter finished state and provide the necessary materials for job 12, which starts at epoch 250. At epoch 270, job 8 is finished and offered to demand d2. At last, job 11 is finished and supplies to demand d1 at epoch 324.



PartLink		
req Part	sup Part	rate
a	c	1
a	d	2
c	d	2
c	e	4
b	e	4
b	f	2

Part				
no	procure LeadTime	sup PartLink	req PartLink	operation
a	-	a-c, a-d	-	
b	-	b-e, b-f	-	
c	-	c-d, c-e	a-c	
d	50	-	a-d, c-d	-
e	40	-	c-e, b-e	-
f	80	-	b-f	-

Operation			
oprNo	setup Time	process Time	process WrkCtr
a10	10	2	q
a20	20	3	p
b10	30	4	p
c10	0	4	p
c20	20	3	r

Demand				
no	part	reqQty	dueEpk	supJob
d1	a	10	200	-
d2	b	20	250	-

Shift			
resource	startEpk	finishEpk	task
x	0	400	-
y	0	400	
z	0	400	-

WorkCenter	
name	enrollResource
p	x, y
q	y, z
r	z

Resource	
name	joinWkCtr
x	p
y	p, q
z	q, r

Task				
shift	job	operation	startEpk	finishEpk
-	-	-	100	200

(a) production data and initial condition

Job											
no	part	net Req	start Epk	finish Epk	supJobLink	reqJobLink	meet Demand	wipAssign	plnAssign	meetBy Supply	task
1	a	10	120	200	1-2, 1-3	-	d1	-	-	-	
2	c	10	30	120	2-3, 2-4	1-2	-	-	-	-	
3	d	40	-20	30	-	1-3, 2-3	-	-	-	-	-
4	e	40	-10	30	-	2-4	-	-	-	-	-

Task				
job	oper-ation	start Epk	finish Epk	shift
-	-	100	200	
1	a20	150	200	
1	a10	120	150	
2	c20	70	120	
2	c10	30	70	

Shift			
task	reso-urce	start Epk	finish Epk
	x	0	400
	y	0	400
	z	0	400

(b) planning and backward scheduling of demand d1

Job											
no	part	net Req	start Epk	finish Epk	supJobLink	reqJobLink	meet Demand	wipAssign	plnAssign	meetBy Supply	task
3	d	40	0	50	-	1-3, 2-3	-	-	-	s1	-
4	e	40	0	40	-	2-4	-	-	-	s2	-
2	c	10	50	140	2-3, 2-4	1-2	-	-	-	-	
1	a	10	140	220	1-2, 1-3	-	d1	-	-	-	
5	b	20	90	250	5-6, 5-7	-	d2	-	-	-	
6	e	80	50	90	-	5-6	-	-	-	s2	-
7	f	40	10	90	-	5-7	-	-	-	s3	-

Task					
job	operation	startEpk	finishEpk	shift	
-	-	100	200		
2	c10	50	90		
2	c20	90	140		
1	a10	140	170		
1	a20	170	220		
5	b10	220	250		
5	b10	90	170		

Shift			
task	reso-urce	start Epk	finish Epk
	x	0	400
	y	0	400
	z	0	400

JobLink		
reqJob	supJob	supQty
1	2	10
1	3	20
2	3	20
2	4	40
5	6	80
5	7	40

Demand				
no	part	reqQty	dueEpk	supJob
d1	a	10	220	1
d2	b	20	250	5

Assign		
plnJob	wipJob	asgQty

Supply				
no	part	supQty	deliveryEpk	reqJob
s1	d	40	50	3
s2	e	120	40	4, 6
s3	f	40	90	7

(c) planning, scheduling and procuring of demand d1, d2

Job												
state	no	part	net Req	start Epk	finish Epk	supJobLink	reqJobLink	meet Demand	wip Assign	pln Assign	meetBy Supply	task
finished	3	d	20	0	50	-	1-3	-	-	-	s1	-
released	2	c	10	50	140	-	1-2	-	-	-	-	
scheduled	1	a	10	140	220	1-2, 1-3	-	d1	-	-	-	
finished	6	e	80	0	40	-		-	-	6-9	s2	-
released	7	f	40	10	110	-		-	-	7-10	s3	-
scheduled	8	b	20	110	270	8-9, 8-10	-	d2	-	-	-	
-	9	e	0	-	40	-	8-9	-	6-9	-	-	-
-	10	f	0	-	110	-	8-10	-	7-10	-	-	-

Task				
job	oper-ation	start Epk	finish Epk	shift
-	-	100	200	
2	c10	50	90	
2	c20	90	140	
1	a10	140	170	
1	a20	170	220	
8	b10	220	270	
8	b10	110	170	

Shift			
task	reso-urce	start Epk	finish Epk
	x	0	400
	y	0	400
	z	0	400

JobLink		
reqJob	sup Job	sup Qty
1	2	10
1	3	20
8	9	80
8	10	40

Demand				
no	part	reqQty	dueEpk	supJob
d1	a	10	220	1
d2	b	20	270	8

Supply				
no	part	supplyQty	deliveryEpk	reqJob
s3	f	40	110	7

Assign		
plnJob	wipJob	asgQty
6	9	80
7	10	40

(d) rescheduling demand d2 at epoch 60 triggered by supply change (s3)

Job												
state	no	part	net Req	start Epk	finish Epk	supLobLink	reqJobLink	meet Demand	wip Assign	pln Assign	meetBy Supply	task
released	1	a	10	140	220	-	-		-	1-11	-	
released	8	b	20	110	270	-	-	d2	-	-	-	
scheduled	13	d	8	200	250	-	11-13, 12-13		-	-	s4	-
scheduled	14	e	8	200	240	-	12-14		-	-	s5	-
scheduled	12	c	2	250	284	12-13, 12-14	11-12		-	-	-	
scheduled	11	a	2	284	324	11-13, 11-12	-	d1	1-11	-	-	

Task				
job	oper-ation	start Epk	finish Epk	shift
1	a20	170	220	
8	b10	220	270	
12	c10	250	258	
12	c20	258	284	
11	a10	284	298	
11	a20	298	324	

Shift			
task	reso-urce	start Epk	finish Epk
	x	0	400
	y	0	400
	z	0	400

JobLink		
reqJob	supJob	sup Qty
11	12	2
11	13	4
12	13	4
12	14	8

Demand				
no	part	reqQty	dueEpk	supJob
d1	a	12	324	11
d2	b	20	270	8

Supply				
no	part	supQty	deliveryEpk	reqJob
s4	d	8	250	13
s5	e	8	240	14

Assign		
plnJob	wipJob	asgQty
1	11	10

(e) rescheduling demand d1 at epoch 200 triggered by demand change (d1)

Fig. 4.5: The instances of APPCM for demonstrating how APPCS works

4.7. Summary

We have proposed a model of agile production planning and control (APPCM) by using universal modeling language (UML). In the model, *Part*, *PartLink*, *Operation*, *WorkCenter*, *Resource*, *Shift*, *Demand*, *Job*, *JobLink*, *Assign*, *Task*, and *Supply* classes and their associations are defined. *Job* and *JobLink* classes and their associations are abstractions of the hierarchy of jobs. The *Task* class is an abstraction of the feasible tasks, each of which describes who, where, when, and how long of an operation. The *Assign* class is an abstraction of the assignments from WIP (released jobs) to the planning jobs when planning and scheduling is executed again because of uncertainties.

If a priority rule and the corresponding parameters, such as lot size, leadtime, or safety stock are specified, then a feasible production plan can be generated and regenerated against uncertainties by first backward scheduling and then forward scheduling. The APPCM has been applied to an implementation of a simulator successfully. Furthermore, the APPCM is demonstrated and testified by an example, which shows how a demand runs planning and scheduling, and how the demand reacts to the supply and demand change during production control.

Hatchuel et al. (1997) pointed out that during the last four decades, continuing development of industrial technology has generated a new make-to-order industrial type. Under such a competitive environment in the industries, customers are usually allowed to change not only the quantity, but also the specification or the style of the product after they made an order. The immediate planning and scheduling can respond to the change with higher service level, better resource utilization, and less material loss. The proposed APPCM makes the immediate planning and scheduling possible.

In order to control a business process with high quality of performance, qualitative analysis of dynamic property such as Sato (1999) is not sufficient. The design of dynamics of a business process is necessary. If we could bring planning components into the design of business processes, then the whole control mechanism can be explicitly managed. The result of this chapter also plays a basic role for that purpose.

5 APPCS Optimization

5.1. Introduction

In this chapter, a novel genetic algorithm, called MDGA, is proposed to solve dynamic flexible scheduling (DFS) problem. MDGA has the following two characteristics.

1. It is integrated with minimal generation gap (MGG) instead of standard genetic algorithm (SGA). MGG is a generation alternation model proposed by Yamamura et al. (1996), which keeps variety of chromosomes in a population while preventing the search process from local optima.
2. It uses newly proposed specific crossover, called demand crossover, which only produces feasible offspring. Unlike one-point crossover (Nearchou, 2004; Lee and Dagli, 1997; Dagli and Sittisathanchai, 1995), valid two-point crossover (Nearchou, 2004), or invalid two-point crossover (Croce et al., 1995), demand crossover exchanges the genes that are related to some demands without violating the precedence constraints.

In the competitive market, for example, it is almost impossible to begin a production after the actual demand is known. Based on information of the present, decision makers always forecast the future. Hopp and Spearman (2000) pointed out three laws of forecasting: (1) forecasts are always wrong, (2) detailed forecasts are worse than aggregate forecasts, and (3) the further into the future, the less reliable the forecast will be. The second law explains the reason why production planning begins with master production schedule (MPS), which plans the long-term requirements of the product family. Subsequently, material requirements planning (MRP) is used to plans the short-term requirements of an individual product. The third law reveals that the less reliable forecast should be revised by some new information.

In the field of production management, researches try to build a model to predict the future demand. The first law does not disparage the activity of forecasting, but call attention to the importance of forecast revision. Sato and Tsai (2004) proposed agile production planning and control system (APPCS) to incorporate a change into production system and

provided a methodology to respond to the change agilely and simultaneously. Once there is a notification of change, APPCS generates another feasible schedule based on work-in-process (WIP). Tsai and Sato (2004a) gave a model of APPCS (APPCM) by using universal modeling language (UML) to show the realizability of APPCS. The schedule developed by APPCS is both practical and feasible, because it is compatible with the product data that has the same structure detail with a commercially available enterprise resource planning (ERP) package and an advanced planning and scheduling (APS) system.

APPCS provides a feasible schedule, but the schedule is not necessarily good. For a plant, a good schedule is the schedule that achieves its own goal and reflects requirements of the market. The goal varies among problems and researches. For most of the scheduling problems, it is difficult to meet all the goals. There may be conflict among different goals. Kacem et al. (2002) proposed an approach to minimize makespan and total processing time (workload) for a flexible job shop schedule problem. The problem is different from the general job shop scheduling problem because it assumes the performance of the machines in a work center is different. Assigning a fast machine to an operation minimizes both makespan and workload at first. However as the capacity of the fast machines approaches to full, the optimization faces a dilemma of continuously choosing a fast machine to increase makespan, or choosing a slow machine to increase workload.

Two approaches are possible among the studies that try to achieve multiple goals. The lexicographic approach searches for the schedule that meets the goals in a lexicographic order. The weighted-sum approach seeks for the schedule that achieves the highest scores of a linear combination of the goals.

A measure of the schedule varies from plant to plant, from single goal to multiple goals and from lexicographic approach to weighted-sum approach. This research aims to solve an optimization problem that achieves various goals subject to a set of feasible schedules that are generated for a set of demands on the basis of product data with resource flexibility and some WIP. The problem is called dynamic flexible scheduling (DFS) problem. It is flexible because it assigns resources in a work center to an operation, and because it responds to various goals. It is dynamic because it is requested to respond to any change in real time. The DFS problem is practical because it adopts the product data that is actually used in commercially available manufacturing planning software (such as SAP R/3 and SyteAPS).

NP-hard problems are problems for which there is no known polynomial algorithm, so that the time to find a solution grows exponentially in problem size (Hopp and Spearman, 2000). Job shop scheduling (JSS) problem is a simplified DFS problem, which will be shown in Section 5.3.2. JSS problem has been shown to be NP-hard by Croce et al. (1995) and Al-Hakim (2001), hence DFS is also an NP-hard problem.

Aytug et al. (2003) provided a review of the use of genetic algorithms to solve the production and operations management (POM) problems. The scheduling problem is one of

them, but the optimization of DFS problem is not in those reviewed researches. In this sense, DFS problem is new. In addition, practical and suitable situation in responding to changes agilely.

GA exhibits parallelism, contains certain redundancy and historical information of the past solutions. It is suitable for implementation on massively parallel architecture (Wang and Zheng, 2001), and it has been applied to a large number of complex search problems (Nearchou, 2004). GA does not rely on analytical properties of the function to be optimized, which makes them well suited to a wide class of optimization problems (Al-Hakim, 2001). However, in view of the randomness property of GA, there is no guarantee of reaching optimum solutions for most scheduling problems. In order to see the power of MDGA in solving DFS problems, it would be proper to compare it with other methods. Since DFS problem is a new class of problems, we cannot find such methods. Therefore, we apply MDGA to JSS problems which are simplified DFS problems, and then compare MDGA with other methods for JSS problems. Notice that the reason of this comparison is not to show the superiority of MDGA for JSS problem, but to show that MDGA is not a bad method to find a good schedule in a DFS problem. (The comparison will be conducted in Section 5.4.)

The rest of this chapter is organized as follows. Section 5.2 introduces APPCS and how it responds to a change by an example. Section 5.3 provides a definition of DFS problem, and a formulation of the problem. Section 5.4 presents MDGA, and gives exhaustive search and a comparison with other GAs to demonstrate its correctness and effectiveness. Section 5.5 provides an insight into the performance of MDGA through an experiment and gives some advice on applying MDGA to solve DFS problem. We give a discussion in section 5.6 about a comparison of GA performance between MGG and SGA, and between demand crossover and other operators. Finally, a summary in the final section is provided.

5.2. Rescheduling Capability of APPCS

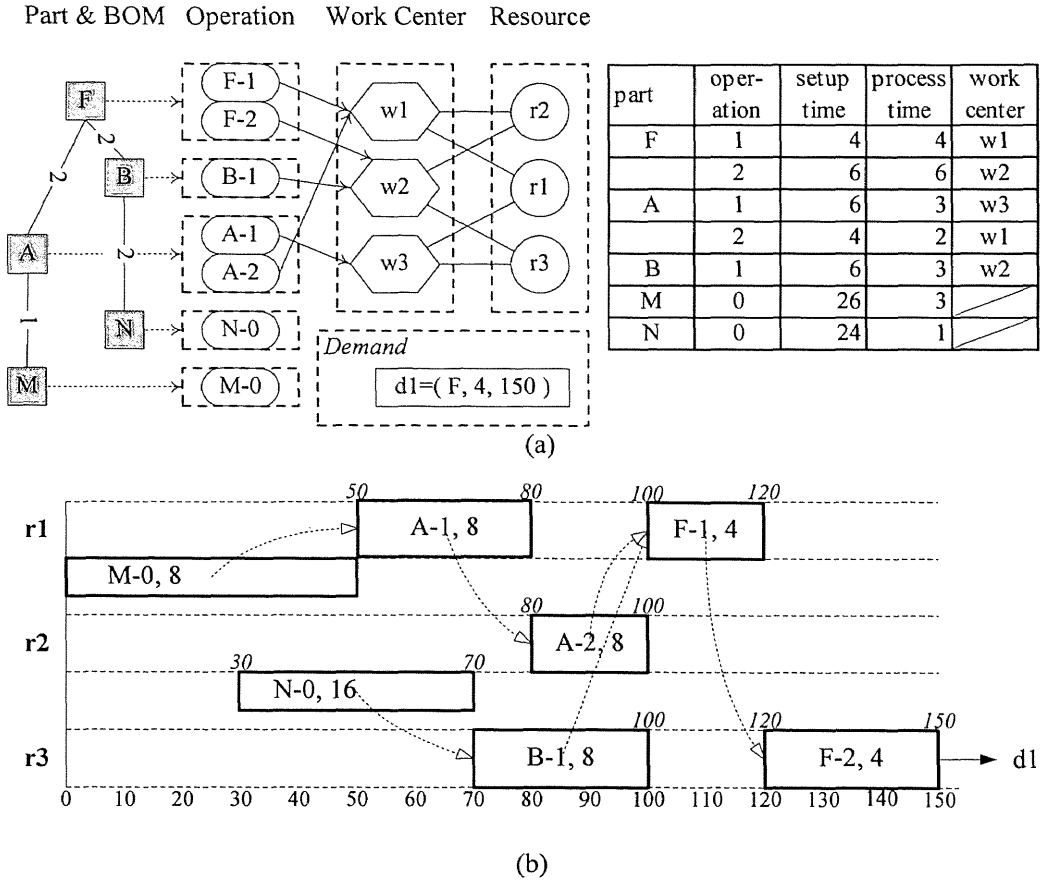
Agile production planning and control system (APPCS) is proposed by Sato and Tsai (2004) to provide a methodology to respond to the change agilely and simultaneously. Once there is a notification of change, APPCS generates another feasible production plan based on work-in-process (WIP) to solve the change. We show the rescheduling capability of APPCS by an example in this section.

A product data is the data related to product design and manufacturing. The product data for APPCS contains part, bill-of-materials (BOM), routing, work center, and resource. Those terms are illustrated with a product part shown in Fig. 5.1a. A gray square in the product data shows a part. Finished product, assembly, and raw material are the types of part. This figure shows that it needs two pieces of assembly 'A' and two pieces of assembly 'B' to make one finished product 'F'. BOM is a term used to define such request-supply relations.

A routing is a sequence of operations to make a part. A work center that enrolls some resources is assigned to an operation, and the operation will be processed by one of the resources. It takes some time to set up a resource before starting an operation. Setup time and processing time for processing a piece of part are estimated for an operation. However, the operation for procurement is processed without specifying a work center. As shown in Fig. 5.1a, finished part 'F' has two operations – 'F-1' and 'F-2'. Operation 'F-2' is processed at work center 'w2' in which resources 'r2' and 'r3' are stationed. It takes 6 time units to set up either resource, and 6 time units to process a part.

A demand is either a customer order or the result of forecasting. In Fig. 5.1b, a schedule generated for a demand 'd1' is illustrated with a Gantt chart. The demand requests 2 pieces of finished product 'F' before due time 150. A rectangular bar in the chart shows a task. In a sense, a schedule is a set of tasks that are fitted in a Gantt chart. The dot line with a white arrow in the chart shows the precedence constraints that regulate the manufacturing sequence of tasks. According to the schedule, a task is released to the shop floor or it is passed on to the purchasing department for subsequent processing. According to Gantt chart shown in Fig. 5.1b, the first task (M-0, 8) should be lunched to a vender at time 0.

Assume a new demand 'd2', which requests 2 pieces of 'F' by time 180, arrives at time 30. According to APPCS, once an event causes any changes in a schedule to happen, all the planned tasks except for the in-processing ones are canceled and a new feasible schedule is plotted out again based on the in-processing tasks according to the updated conditions. When the in-processing task is finished, its output becomes a work-in-process (WIP).



part	oper-ation	setup time	process time	work center
F	1	4	4	w1
	2	6	6	w2
A	1	6	3	w3
	2	4	2	w1
B	1	6	3	w2
M	0	26	3	
N	0	24	1	

Fig. 5.1: (a) Product data of part 'F' and (b) a schedule of demand 'd1' shown by Gantt chart

Fig. 5.2a gives a new schedule for demands 'd1' and 'd2' based on the WIP (M-0, 8) whose planned finish time is 50. A dot line with a solid arrow shows that 8 pieces of the WIP are input to a downstream task (A-1, 8), which starts from time 50. Consequently, it is not necessary to generate a task for operation 'M-0'. This schedule achieves minimal makespan. Some plants may not satisfy with this schedule, because it indirectly causes a long processing time (318) and tardiness (38).

Fig. 5.2b shows another schedule that achieves minimum total processing time (268), and the schedule in Fig. 5.2c attains maximum service level (100%), minimum earliness (0), and minimum tardiness (0). It is reasonable to process a group of identical operations together, to cut down on setup time of processing or on ordering costs of purchasing. The minimum total processing time of the schedule in Fig. 5.2b is achieved at the cost of service level (0%), tardiness (86) and makespan (178). Fig. 5.2d shows a schedule that compromise a goal with makespan of 140 and total processing time of 288 by applying the weighted-sum approach, where the minimum makespan and total processing time are 128 and 268, respectively.

In this manner, any change to a schedule will trigger APPCS to generate an improved, goal-oriented schedule recursively.

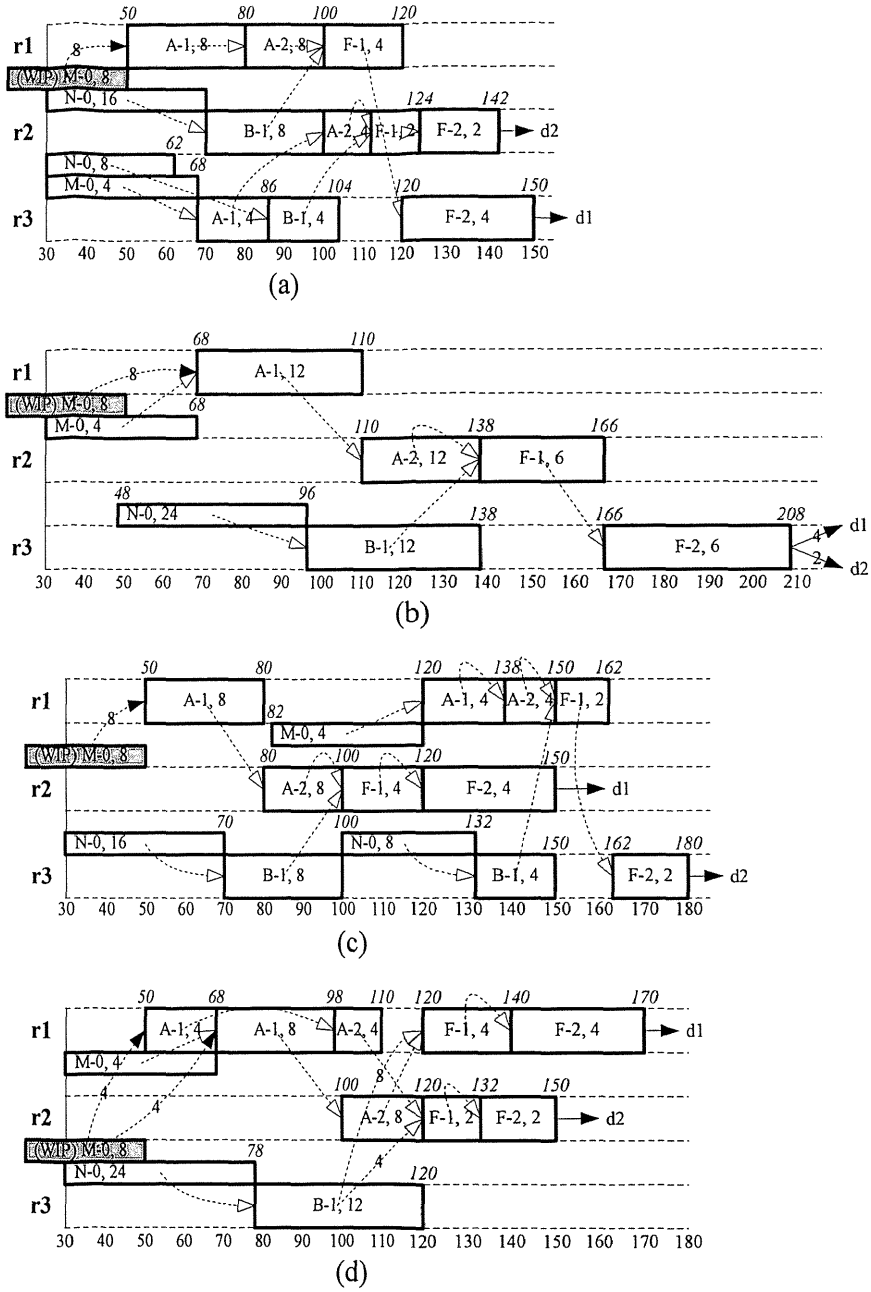


Fig. 5.2: The schedules that achieve (a) minimum makespan, (b) minimum processing time, (c) maximum service level, and (d) a weighted-sum

5.3. Dynamic Flexible Scheduling Problem

A dynamic flexible scheduling (DFS) problem is defined as an optimization problem of minimizing makespan, maximizing service level, and minimizing tardiness subjects to some constraints of scheduling based on product data, operation requirements, tasks, and work-in-process (WIP). Subsequently, a formulation of DFS problem in a systematic way is provided for the encoding of genetic algorithm with MGG and demand crossover (MDGA) in the next section.

5.3.1 Definitions and Notations

Product Data

The following notations are used in defining product data. An instance of the notation is provided following the description. The instance is drawn from the product data shown in Fig. 5.1a.

p_i	A part
$P=\{p_i\}$	The set of parts concerned; $P=\{$F, A, B, M, N$\}$
$bm_{ij}=(p_i, p_j)$	An ordered pair indicates that part p_j is an immediate component of part p_i
$Bm=\{bm_{ij}\}$	The set of the ordered pairs bm_{ij} among parts in P; $Bm=\{$(F, A), (F, B), (A, M), (B, N)$\}$
$Qty(bm_{ij})$	Quantity of p_j per unit of p_i; Qty(F, A)=2
$op_{ij}=(p_i, j)$	An ordered pair, called an operation, represents the jth processing step to make part p_i
$Nop(p_i)$	The number of operations of part p_i; Nop(F)=2
$Spt(op_{ij})$	Setup time of operation op_{ij}; Spt(F, 1)=4
$Pct(op_{ij})$	Unit processing time of operation op_{ij}; Pct(A, 2)=2
$Pcw(op_{ij})$	Processing work center of operation op_{ij}; Pcw(B, 1)=$w2$
w_i	A work center
$W=\{w_i\}$	The set of work centers; $W=\{w1, w2, w3\}$
r_i	A resource
$R=\{r_i\}$	The set of resources; $R=\{r1, r2, r3\}$
$En(w_i)$	The set of resources enrolled in a work center w_i; $En(w_2)=\{r2, r3\}$

Recursively applying the request-supply relations defined in Bm, a hierarchy of parts is constructed. If each part in the hierarchy is replaced with its operations, then we get a

hierarchy of operations.

$Rat(op_{im}, op_{jn})$ Number of parts necessary to be processed by an operation op_{jn} for a part by the successor operation op_{im} defined in the hierarchy of operations; it is defined as

$$Rat(op_{im}, op_{jn}) = \begin{cases} 1 & \text{if } p_i = p_j, m = n+1 \\ Qty(p_i, p_j) & \text{if } (p_i, p_j) \in Bm,\ m = 1,\ n = Nop(p_i) \end{cases} \quad (5.1)$$

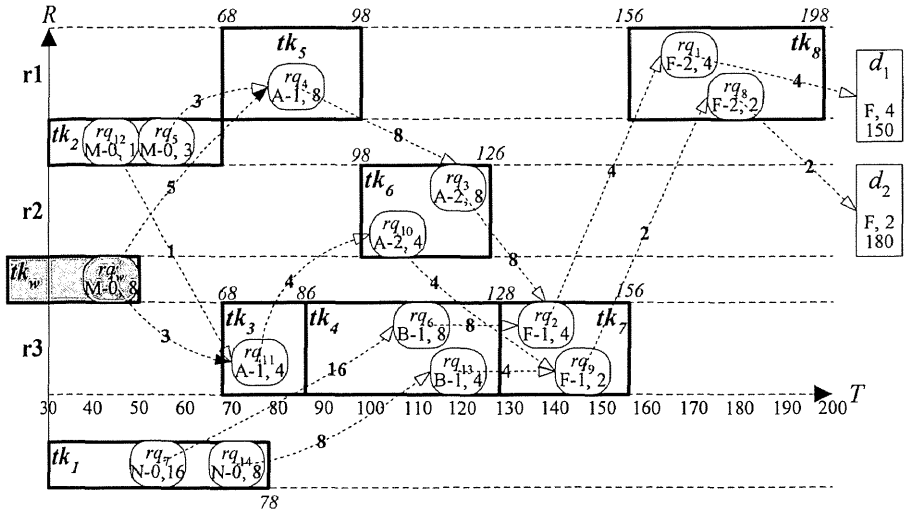


Fig. 5.3: A schedule in terms of terminologies of DFS problem

Demands

The following notations are used to denote the demands. The instance provided after the explanation is from Fig. 5.3, which shows a schedule in terms of the terminologies of the problem.

d_i	A demand
$D=\{d_i\}$	A set of demands; $D=\{d_1, d_2\}$
$Rqq(d_i)$	Requested quantity of demand d_i; $Rqq(d_1)=4$
$Rqp(d_i)$	Requested part of demand d_i; $Rqp(d_2)=$F
$Dut(d_i)$	Due time of demand d_i; $Dut(d_1)=150$

Operation Requirements

An operation requirement is a request for an operation. The request is exploded either independently or dependently from a demand. 'Requirement' is used instead of 'operation requirement' for simplification. The following notations are used in defining requirements and their relations.

rq_i	A requirement

$Rq=\{rq_i\}$	A set of requirements exploded from demands D; $Rq=\{rq_i\}_{i=1..14}$
$Opr(rq_i)$	Operation of requirement rq_i; $Opr(rq_6)=(B, 1)$
$Net(rq_i)$	Quantity (net requirement) of requirement rq_i; $Net(rq_{10})=4$
$e_{ij}=(rq_i, rq_j)$	An ordered pair showing precedence relation that rq_j must be processed before rq_i
$E_{rq}=\{e_{ij}\}$	A set of directed edges representing precedence among requirements Rq; $E_{rq}=\{(rq_1, rq_2), (rq_2, rq_3), (rq_3, rq_4), (rq_4, rq_5), (rq_2, rq_6), (rq_6, rq_7),\ldots, (rq_{13}, rq_{14})\}$
$G_{rq}=(Rq, E_{rq})$	A directed acyclic graph of the requirements Rq
$Pd(rq_i)$	The set of immediate predecessors of rq_i; $Pd(rq_9)=\{rq_{10}, rq_{13}\}$
$Sc(rq_i)$	The immediate successor of rq_i; $Pd(rq_4)=rq_3$
$Met(d_i)$	The requirement planned to meet demand d_i; $Met(d_2)=rq_8$

The requirements in Rq must be generated according to the hierarchy of operations. Let $(rq_i, rq_j) \in E_{rq}$ be arbitrary, and assume that operation $Opr(rq_i)=(p_i, m)$ and $Opr(rq_j)=(p_j, n)$. If $p_i=p_j$, then $m=n+1$; otherwise $(p_i, p_j)\in Bm$, $m=1$, and $n=Nop(p_j)$. Requirements are generated to meet the request of all demands. For a demand d_i, there must be one and only one requirement, $rq_i\in Rq$, such that $Sc(rq_i)=\varnothing$, $Opr(rq_i)=(Rqp(d_i), Nop(Rqp(d_i)))$ and $Rqq(d_i)=Net(rq_i)$ hold.

Tasks

A task is an operation processed within a period of time by a resource, which is generated to meet one or more requirements. The following notations are used for defining tasks and the relation with requirements. Notice that for a set A, $|A|$ represents the number of elements in A.

tk_i	A task
$Tk=\{tk_i\}$	A set of tasks scheduled to meet requirements Rq, $\|Tk\|\leq\|Rq\|$; $Tk=\{tk_i\}_{i=1..8}$
$Rsc(tk_i)$	The resource assigned to process task tk_i; $Rsc(tk_3)=r3$
$Sta(tk_i)$	The start time of task tk_i; $Sta(tk_3)=68$
$Fin(tk_i)$	The finish time of task tk_i; $Fin(tk_3)=86$
$Tq(tk_i)$	A set of requirements scheduled to form a task tk_i; $Tq(tk_4)=\{rq_6, rq_{13}\}$
$Qt(rq_i)$	The task of requirement rq_i; $Qt(rq_6)=tk_4$
Sst	Scheduling start time; $Sst=30$
$Est(rq_i)$	The earliest time at which rq_i can be started; $Est(rq_3)=98$, $Est(rq_{10})=86$
$Lft(rq_i)$	The latest time which rq_i must be completed; $Lft(rq_3)=128$, $Lft(rq_{10})=128$

The earliest and latest times for a requirement are defined as

$$Est(rq_i) = \begin{cases} Sst & \text{if } Pd(rq_i) = \varnothing \\ \max\limits_{rq_j \in Pd(rq_i)} Fin(Qt(rq_j)) & \text{otherwise} \end{cases} \quad (5.2)$$

and

$$Lft(rq_i) = \begin{cases} \infty & \text{if } Sc(rq_i) = \varnothing \\ Sta(Qt(Sc(rq_i))) & \text{otherwise} \end{cases}. \quad (5.3)$$

WIP

A rigorous definition of 'work-in-process' that a requirement whose successor is canceled by some changes is used in this chapter, instead of the general one that a requirement whose task is in-processing. This is because that scheduling start time *Sst* might be far later than notification time of a change, and some in-processing tasks might have been completed.

Wp	The set of work-in-processes; $Wp = \{rq_w\}$
$Qw(rq_i)$	A set of WIP that was allocated to a requirement rq_i; $Qw(rq_4) = \{rq_w\}$
$Wq(rq_w)$	A set of requirements to which a WIP rq_w is allocated; $Wq(rq_w) = \{rq_4, rq_{11}\}$
$Alq(rq_i, rq_w)$	The quantity of WIP rq_w allocated to requirement rq_i; $Alq(rq_4, rq_w) = 5$

Constraints on Scheduling

[C1] Operation consistency for a task: The requirements with identical operation can be combined to form a task. That is, the following constraint should hold.

$$(\forall tk_i \in Tk)\ (\forall rq_i, rq_j \in Tq(tk_i))\ \ Opr(rq_i) = Opr(rq_j) \quad (5.4)$$

[C2] Total processing time: The total processing time of a task, i.e. the difference between finish time and start time, equals to the sum of setup time and the product of unit processing time and the sum of quantities of the contained requirements. It is

$$(\forall tk_i \in Tk)\ (\exists rq_i \in Tq(tk_i))\ \ Fin(tk_i) - Sta(tk_i) = Spt(Opr(rq_i)) + Pct(Opr(rq_i))$$

$$\times \sum_{rq_j \in Tq(tk_i)} Net(rq_j). \quad (5.5)$$

[C3] Resource flexibility: If there is a resource assigned to a task, then it must be enrolled in the work center that is assigned to the operation of requirements satisfied by the task. That is,

$$(\forall tk_i \in Tk)\ (\exists rq_i \in Tq(tk_i))\ \ Rsc(tk_i) \in En(Pcw(Opr(rq_i))). \tag{5.6}$$

[C4] Precedence of tasks: The constraint that a task cannot be scheduled to start earlier than the latest earliest start time and to finish later than the earliest latest finish time among its requirements is denoted by

$$(\forall tk_i \in Tk)\ \ Sta(tk_i) \geq \max_{rq_j \in Tq(tk_i)} Est(rq_i) \text{ and } Fin(tk_i) \leq \min_{rq_i \in Tq(tk_i)} Lft(rq_i). \tag{5.7}$$

[C5] Finite loading on resource: The finite loading constraint of a resource is denoted by

$$(\forall tk_i, tk_j \in Tk) \text{ if } Rsc(tk_i) = Rsc(tk_j), \text{ then } [Sta(tk_i), Fin(tk_i)) \cap [Sta(tk_j), Fin(tk_j)) = \varnothing,$$

where $[t_1, t_2)$ means an interval of time from t_1 to t_2. (5.8)

[C6] WIP allocation: A WIP (planned requirement) can be used by a requirement if it has the same operation with the WIP, and if the WIP is completed before the successor of the requirement starts. That is,

$$(\forall rq_i \in Rq)(\forall rq_w \in Qw(rq_i))\ \ Opr(rq_i) = Opr(rq_w) \text{ and } Fin(Qt(rq_w)) \leq Sta(Qt(Sc(rq_i))). \tag{5.9}$$

[C7] Total quantity of WIP: The total allocated quantity of a WIP among requirements cannot exceed quantity of the WIP. That is,

$$(\forall rq_w \in Wp)\ Net(rq_w) \geq \sum_{rq_i \in Wq(rq_w)} Alq(rq_i, rq_w). \tag{5.10}$$

[C8] Net requirement: For any $(rq_i, rq_j) \in E_{rq}$, quantity of rq_j is calculated by subtracting WIP allocations from the gross requirement requested by rq_i. That is,

$$(\forall (rq_i, rq_j) \in E_{rq})\ Net(rq_j) = Net(rq_i) \times Rat(Opr(rq_i), Opr(rq_j)) - \sum_{rq_w \in Qw(rq_j)} Alq(rq_j, rq_w). \tag{5.11}$$

Dynamic Flexible Scheduling Problem

Denote evaluation functions of makespan by EV_{mks}, service level by EV_{svc}, and tardiness by EV_{tds} for a schedule. Dynamic flexible scheduling (DFS) problem is defined as:

$$\text{Minimize } EV_{mks} = \max_{tk_i \in Tk}(Fin(tk_i)) - \min_{tk_i \in Tk}(Sta(tk_i)), \tag{5.12}$$

$$\text{Maximize } EV_{svc} = (\sum_{d_i \in D} I(Fin(Qt(Met(d_i))) \le Dut(d_i)))/|D|, \text{ where } I: \{T, F\} \rightarrow \{1, 0\}, \text{ or} \tag{5.13}$$

$$\text{Minimize } EV_{tds} = \sum_{d_i \in D} \max\{Fin(Qt(Met(d_i))) - Dut(d_i), 0\}, \tag{5.14}$$

Subject to the following eight constraints.
[C1] Operation consistency for a task
[C2] Total processing time
[C3] Resource flexibility
[C4] Precedence of tasks
[C5] Finite loading on resource
[C6] WIP allocation
[C7] Total quantity of WIP
[C8] Net requirement

5.3.2 Problem Formulation

Requirement arrangement, requirement aggregation, resource assignment, WIP allocation, and scheduling alternatives are the steps to formulate DFS problem in a systematic way.

Requirement Arrangement

Production planning and scheduling assigns available capacity (a time interval) of a resource to requirements in the set of operation requirements Rq. To solve the conflict caused when more than one requirement requests for the same period of time from a resource, these requirements are arranged to a sequence $\langle rq_i \rangle_{i=1..|Rq|}$ $(rq_i \in Rq)$ and the capacity of resource is assigned to the requirements in order of the sequence. Let $SQ = \{sq_1, sq_2, \ldots, sq_k\}$ be a set of all legal sequences on the requirements in Rq, where a sequence of the requirements is said to be legal if the order of the requirements doesn't violate the precedence constraints E_{rq}. For the sample described in Fig. 5.3, the set SQ is shown in Fig. 5.4a, in which a circle with a

number i represents an operation requirement rq_i. The ways to calculate the total number of cases in requirement arrangement is shown in Appendix A.

Requirement Aggregation

The requirements with the same operation that are located adjacent to each other in a sequence of operation requirements can be grouped together into a basket. A basket is a basic unit of scheduling and the requirements in a basket will be scheduled together to form a task. A basket is for a requirement in a sequence of requirements if either side of the requirement does not have any requirement with identical operation.

The aggregation of adjacent requirements without shifting their position in a sequence of requirements complies with the precedence constraints E_{rq}. Denote a set of sequences of baskets by $QA_i = \{ qa_{i1}, qa_{i2}, \ldots, qa_{im} \}$ on a sequence of requirements $sq_i \in SQ$. Fig. 5.4b shows the possible cases of requirement aggregation (QA_i) for a $sq_i \in SQ$ shown in Fig. 5.4a. The ways to calculate the total number of cases in requirement aggregation is shown in Appendix B.

Resource Assignment

A work center is assigned to an operation except operations that need to be planned leadtime for procurement. To keep it simple, we assume that such an operation is assigned to a dummy work center. A basket, including at least one requirement, inherits work center from the requirements, and one of the resources in the work center is assigned to the basket for scheduling.

For a sequence of requirements $sq_i \in SQ$, and for a sequence of baskets $qa_{ij} \in QA_i$, let $RA_{ij} = \{ ra_{ij1}, ra_{ij2}, \ldots, ra_{ijk} \}$ be a set of sequences of resource-assigned baskets. Fig. 5.4c shows some instances of resource assignment for a sequence of baskets shown in Fig. 5.4b. Dummy resource is assigned to baskets k1 and k3, because they are the aggregation of procurement requirements.

WIP Allocation

Quantity of WIP can be allocated to the requirements of the same operation, as shown in [C7], in the new scheduling run. Fig. 5.4d shows the alternative ways to allocate 8 units of WIP rq_w to rq_5 and rq_{12}. Let WA be a set of the possible WIP assignments from WIP in Wp to requirements in Rq. The ways to calculate the total number of cases in WIP allocation is shown in Appendix C.

Scheduling Alternatives

$SA=\{fs, bs\}$ is a function set of two scheduling alternatives - forward scheduling and backward scheduling. Backward scheduling generates a schedule backwardly from due time of a demand, while forward scheduling does it forwardly from the scheduling start time *Sst*.

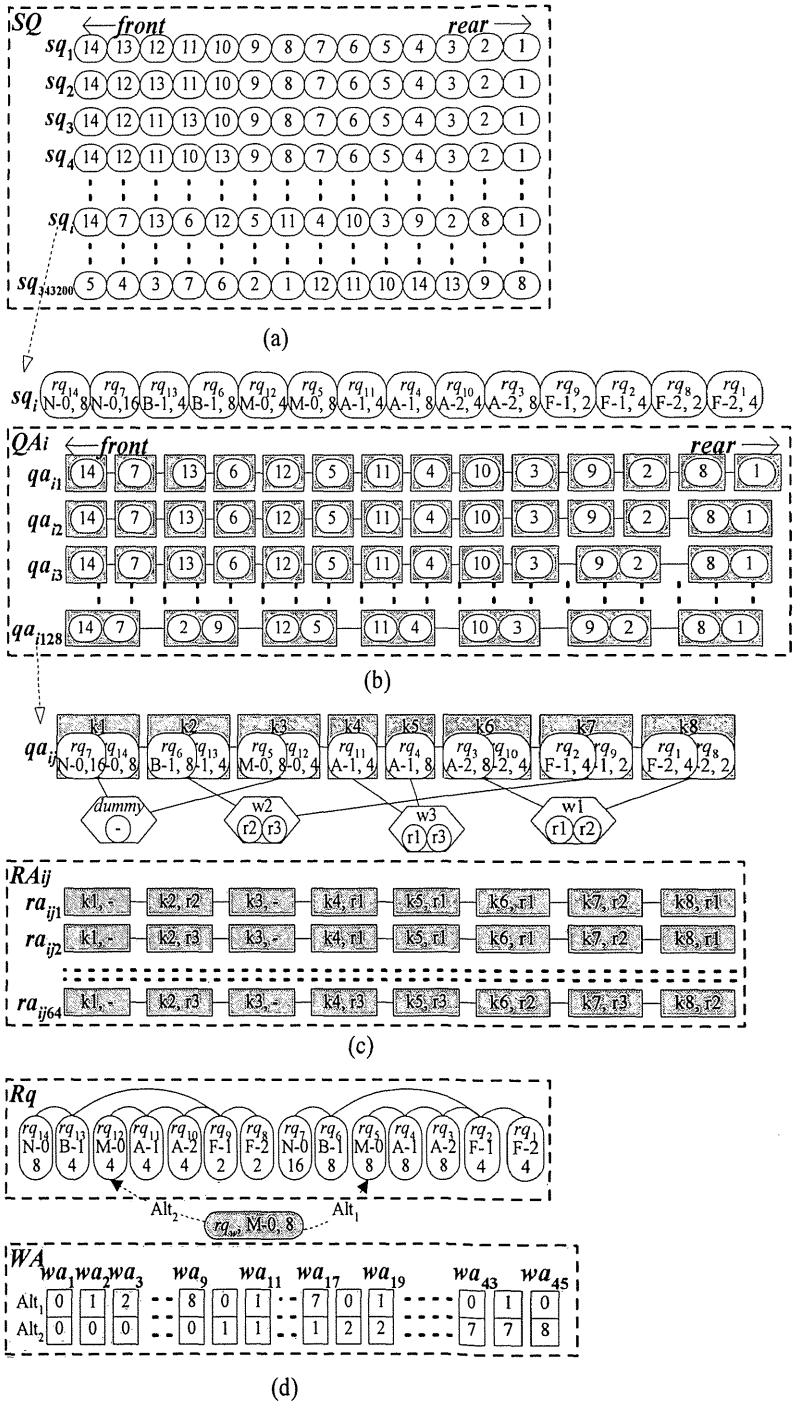


Fig. 5.4: (a) Legal sequences of requirements, (b) sequences of baskets, (c) possible resource assignments for a sequence of baskets, (d) possible WIP allocations for a set of requirements

If we choose a WIP assignment from *WA*, a sequence of resource-assigned baskets from $\bigcup_{i=1}^{|SQ|}\bigcup_{j=1}^{|QA_i|} RA_{ij}$, and determine a scheduling alternative among *SA*, then we get a unique schedule by production planning and scheduling. Domain of DFS problem formulated in a systematic way is thus denoted by

$$SA \times WA \times \bigcup_{i=1}^{|SQ|}\bigcup_{j=1}^{|QA_i|} RA_{ij} . \tag{5.15}$$

Production planning and scheduling is to transform a sequence of resource-assigned baskets with respective WIP allocation and the specification of a scheduling alternative into a set of tasks, which can be fitted in a Gantt chart. One basket, including a set of requirements, is converted to a task. The procedure of production planning and scheduling is shown in Fig. 5.5. Lines from (01) to (03) show that a sequence of resource-assigned baskets running forward scheduling or backward scheduling is determined by the scheduling alternative.

Lines from (04) to (11) show backward scheduling runs net requirement planning together with scheduling in a sequence one by one from the rear basket back to the front one. In line (05), quantity (net requirement) of each requirement in a basket is planned by deducting effective quantity of WIP allocation (due to [C6]) from gross requirement of the successor according to [C8]. Lines (06) and (07) show that a task is generated for a basket and the resource for the task is brought from the basket. Referring to [C2], total processing time of a task is calculated in line (08). In line (09), available intervals of the resource enough and in time for the processing time are gathered. Finish time of the intervals cannot be later than the start time of the successor requirements. The interval with the latest finish time among the intervals is selected and occupied with the processing time of the task as denoted in line (10). Finally, as shown by line (12), if the schedule by backward scheduling starts before scheduling start time *Sst*, then forward scheduling is triggered to generate a feasible schedule from *Sst*.

Forward scheduling plans net requirement from line (13) to (15), then runs scheduling from the front basket to the rear one as listed from line (16) to (21). The net requirement planning is similar to backward scheduling with the exception that all allocated WIP is forced to be used in offsetting the gross requirement as shown in line (17). However, finish times of the WIP must be taken into consideration in determining the earliest start time of the task. As shown in line (20), the earliest start time forces a new task to start after not only the finish times of the predecessor requirements but also the allocated WIP.

The result of production planning and scheduling of a sequence of resource-assigned baskets in Fig. 5.4 is shown in Fig. 5.6, in which '*qty*', '*tpt*', '*lft*', and '*est*' represent total net requirements, total processing time, latest finish time, and earliest start time of a basket,

respectively. As shown in Fig. 5.6a, backward scheduling is executed in a backward sequence from basket k8 to basket k1.

/* *Terms denote baskets together with their relations with requirements*
$Bk=\{ bk_i \}$ A set of the resource-assigned baskets
$\langle bk_i \rangle_{i=1..|Bk|}$ A sequence of baskets on Bk
$Bq(bk_i)$ A set of requirements in a basket
/* *Production planning and scheduling*
Production_ Planning_and_Scheduling (Sequence of baskets: $\langle bk_i \rangle$)
(01) IF scheduling alternatives = 'forward scheduling'
(02) THEN DO Forward_Scheduling (Sequence of baskets: $\langle bk_i \rangle$);
(03) ELSE DO Backward_Scheduling (Sequence of baskets: $\langle bk_i \rangle$);

/* *Backward scheduling of a sequence of baskets*
Backward_Scheduling (Sequence of baskets: $\langle bk_i \rangle$)
(04) FOR each basket $bk_i \in Bk$ in a reverse order of $\langle bk_i \rangle$
(05) Calculate quantity of each requirement in bk_i by [C8] in the confines of [C6];
(06) Generate a task tk_n for all requirements in bk_i due to [C1];
(07) Get resource rs assigned to basket bk_i due to [C3];
(08) Calculate total processing time tpt of tk_n by [C2];
(09) Get a set of intervals Itv of resource rs whose length $\geq tpt$
and finish time $\leq \min\{ Lft(rq) \mid rq \in Bq(bk_i) \}$ due to [C4];
(10) Pick an interval of the latest finish time fin from Itv, and
reserve capacity $[fin-tpt, fin)$ of rs for tk_n due to [C2].
(11) ENDFOR
(12) IF any task in the Gantt chart starts before Sst
THEN DO Forward_Scheduling (Sequence of baskets: $\langle bk_i \rangle$);

/* *forward scheduling of a sequence of baskets*
Forward_Scheduling (Sequence of baskets: $\langle bk_i \rangle$)
(13) FOR each basket $bk_i \in Bk$ in a reverse order of $\langle bk_i \rangle$
(14) Calculate quantity of each requirement in bk_i by [C8];
(15) ENDFOR
(16) FOR each basket $bk_i \in Bk$ in the order of $\langle bk_i \rangle$
(17) Generate a task tk_n for all requirements in bk_i due to [C1];
(18) Get resource rs assigned to basket bk_i due to [C3];
(19) Calculate total processing time tpt of tk_n by [C2];
(20) Get a set of intervals Itv of resource rs whose length $\geq tpt$,
start time $\geq \max\{ Est(rq) \mid rq \in Bq(bk_i) \}$ due to [C4], and
start time $\geq \max\{ Fin(Qt(rq_w)) \mid rq_w \in Qw(Bq(bk_i)) \}$ due to [C6];
(21) Pick an interval of the earliest start time sta from Itv, and
reserve capacity $[sta, sta+tpt)$ of rs for tk_n due to [C2];
(22) ENDFOR

Fig. 5.5: Procedures of production planning and scheduling

A WIP rq_w that ends in time 50 is allocated to requirements rq_5 and rq_{12}, but the WIP is not in time for tasks tk_3 and tk_5, hence the allocation is unusable. Ultimately, the schedule by

executing backward scheduling is not feasible because it is planned to start before the scheduling start time *Sst*. Forward scheduling is done, accordingly. Fig. 5.6b shows the resultant feasible schedule executed by forward scheduling that is starting from *Sst* = 30. The allocated WIP is used, hence less material needs to be purchased.

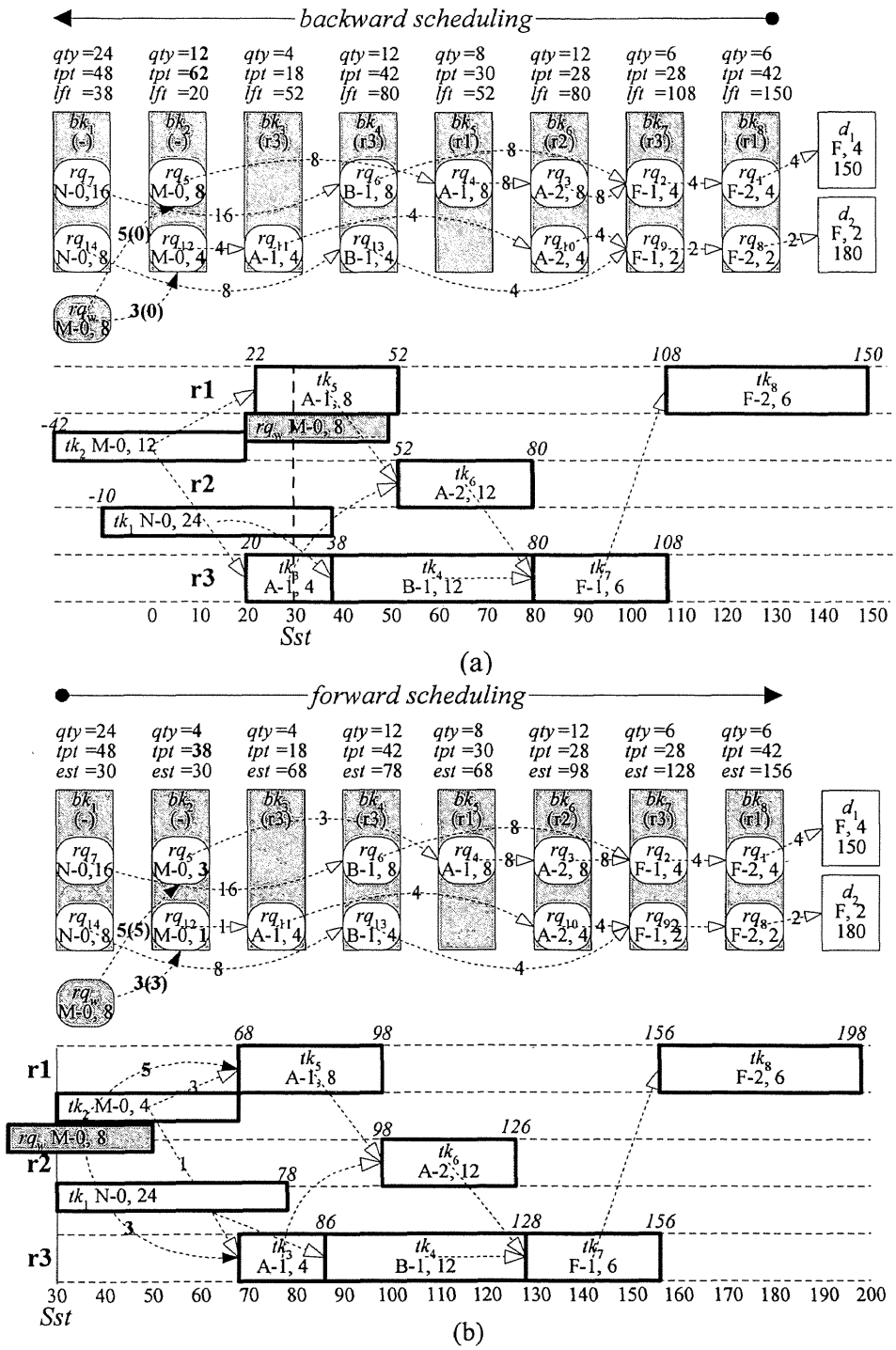


Fig. 5.6: Results of (a) backward scheduling, and (b) forward scheduling

5.4. Genetic Algorithm with MGG and Demand Crossover

This chapter proposes a specific genetic algorithm, called MDGA, to solve DFS problems. A chromosome acts as information carrier through the processes of MDGA. It joins the reproduction process to propagate its offspring by demand crossover and mutation. Then, the offspring's fitness values are measured to compete with those of other chromosomes by minimal generation gap (MGG), a generation alternation model, to decide whether they can be promoted to the next generation. If lost, it is abandoned to have more room for a new chromosome. The processes of reproduction and selection are repeated until all termination conditions are satisfied. The correctness and effectiveness of MDGA will be examined by an exhaustive search and a comparison with other GAs in solving some job shop scheduling problems.

5.4.1 Encoding

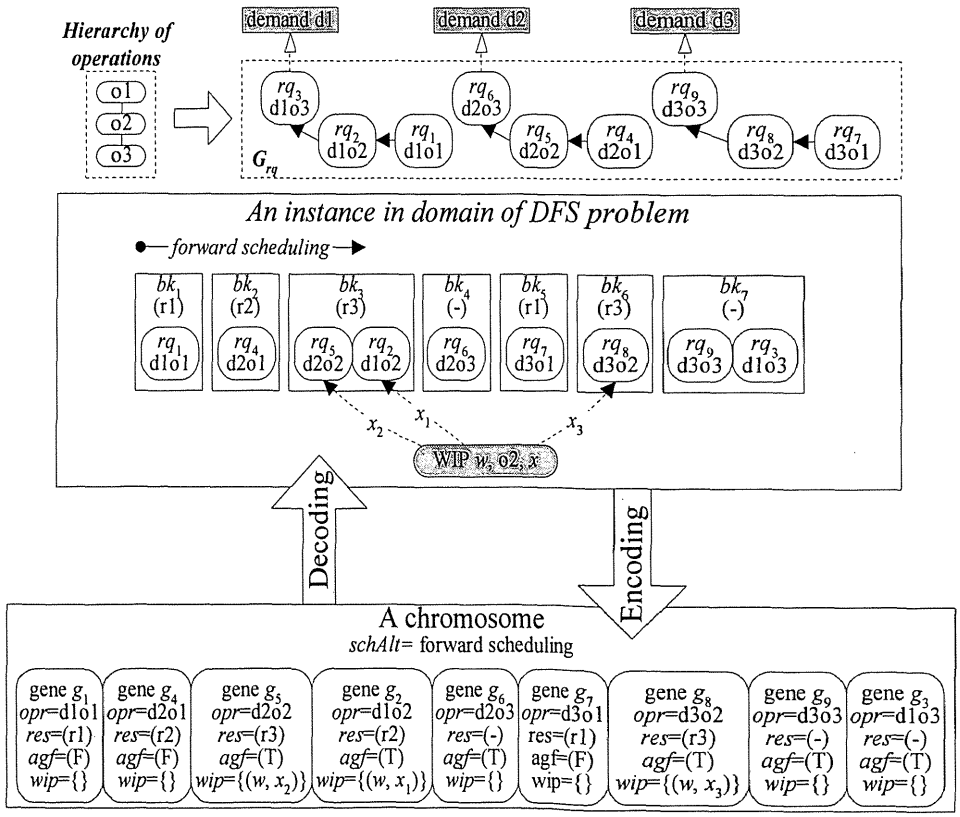


Fig. 5.7: The mapping of DFS problem to the GA encoding

To encode a chromosome is to represent an instance of domain of DFS problem, which is expressed in (5.15). A chromosome is a combination of components, called genes. We encode a gene with a requirement, and a chromosome with a sequence of requirements.

A gene has four attributes: operation, aggregation flag, resource, and WIP allocation. Aggregation flag and operation advise whether the gene is capable of aggregating with other genes or not. A chromosome has an attribute of scheduling alternative, which suggests whether the chromosome should run forward scheduling or backward scheduling. Fig. 5.7 shows the encoding corresponding to the DFS problem, where '*opr*' indicates the operation, '*res*' the assigned resource, '*agf*' the aggregation flag, '*wip*' the WIP allocations of gene, and '*schAlt*' the scheduling alternative.

If several genes have the same operation, 'True' aggregation flags, and are adjacent each other in a chromosome, then they are grouped together in a basket. Subsequently, the responsible resource for a group of genes is randomly selected among resources of the genes. As a result, a chromosome can be decoded back to a sequence of baskets.

5.4.2 Initialization

The first step of MDGA is to generate a set of chromosomes randomly to be the initial population of MDGA. An initialization procedure is proposed in Fig. 5.8 to generate a legal chromosome for the initial population.

```
/* Generate an initial population of chromosomes
Initialization ( )
(01)   Put the last requirement in Rq into a queue Q;
(02)   WHILE Q is not empty
(03)      Remove any requirement rq_i from Q;
(04)      Add predecessor requirements Pd(rq_i) to Q;
(05)      Create a gene gn for rq_i;
(06)      Choose a resource in En(Pcw(Opr(rq_i)))
       for gn due to [C3];
(07)      Set aggregation flag of gn to be 'True' or 'False';
(08)      Append gn to a queue of genes G;
(09)   END WHILE
(10)   Allocate quantity of WIP to genes in G randomly according to [C6], [C7], [C9];
(11)   Designate a chromosome C contains G;
(12)   Assign a scheduling function in {'fs', 'bs'} to C;
```

Fig. 5.8: Procedures of MDGA initialization

The lines from (02) to (09) are a loop for sampling a sequence of genes without violating the precedence constraints. The key to that is appending the predecessor requirements to the sampling pool Q in line (04) immediately after a requirement is removed from Q as shown in

line (03). For a gene, a resource is randomly skimmed off among the resources selected by applying [C3] in line (06), and the aggregation flag is set in line (07). At the end, the quantity of WIP, if any, is distributed to suitable genes randomly according to rules [C6], [C7], and [C9] in line (10). From line (11) to (12), a chromosome is generated to contain the sequence of genes, and finally the scheduling alternative of the chromosome is set to be either forward scheduling or backward scheduling.

5.4.3 Reproduction

Two methods, crossover and mutation, are used in MDGA to reproduce new offspring. In general, crossover operator randomly selects two chromosomes from a population, exchanges some genes of them, and reproduces two new chromosomes. Mutation operator randomly selects a chromosome from the population, reverses some data, and then puts it back to the population. For a DFS problem, both operators must comply with the precedence constraint when reproducing a new sequence of genes. The crossover and mutation of MDGA are explained as follows.

N-demand Crossover

The genes in a chromosome with common attributes form a sub-chromosome. The genes sharing the same resource are competitors, while the genes belonging to the same demand are partners. Besides, the genes reside in a sub-chromosome with their positions in the chromosome might provide some valuable information on solving DFS problem.

After selecting 2 chromosomes from the population, N-demand crossover begins with choosing N demands from D randomly, and then identifies the genes belonging to those demands in both chromosomes. Finally, it exchanges the genes from N demands in a chromosome with the genes in another chromosome. Fig. 5.9a shows how one-demand crossover swaps genes $\{g_4, g_5, g_6\}$, which belong to demand d2 as shown in Fig. 5.7, between parent chromosomes $\{x, y\}$ to make a child chromosome c. Similarly, two-demand crossover swaps genes from demands $\{\mathrm{d1}, \mathrm{d3}\}$ in a chromosome with the genes from the same demands in another chromosome, as shown in Fig. 5.9b.

The crossover operator exchanges not only the sequence of genes in a chromosome but also the embedded information including aggregation flag, resource, and WIP allocations. If the sum of the WIP allocations violates the constraint [C7] after crossover, deduct the surplus or replenish the shortage from the WIP allocations.

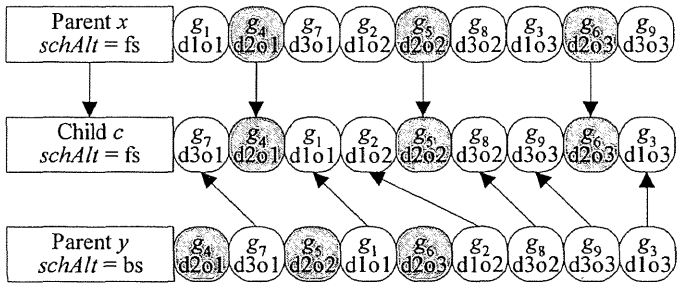
In order to consider the general cases, assume that the selected N demands contain a set of genes $G=\{g_1, g_2, \ldots, g_n\}$ in a chromosome. The chromosome is a sequence of genes denoted by $x=G_1 <\{g_1\}<G_2 <\ldots<\{g_n\}<G_{n+1}$, where '<' means the precedence relation in a

chromosome, and G_1 , G_2 ,..., G_{n+1} are sets of sub-chromosomes; on the same assumption, the same genes in chromosome y can be denoted by $y=H_1<\{g_1\}<H_2<\ldots<\{g_n\}<H_{n+1}$.

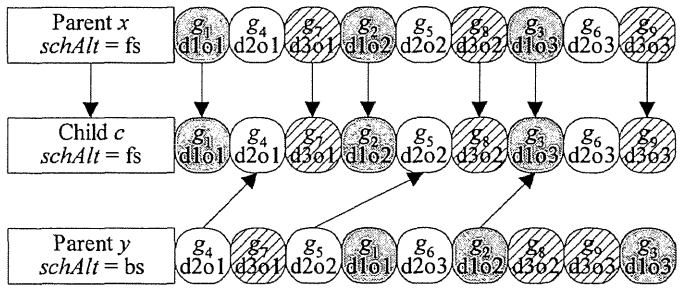
Define the crossover of x and y as

$$x \otimes y = (x \downarrow (\bigcup_{i=1}^{n+1} G_i)) \uparrow (y \downarrow G), \tag{5.16}$$

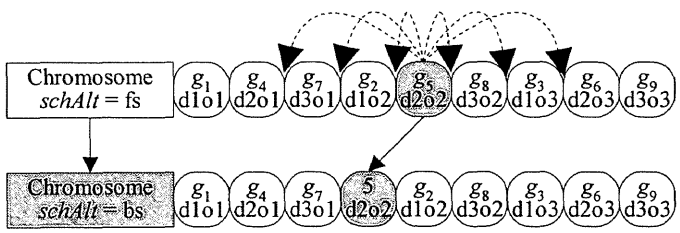
where '$Q\downarrow S$' means taking genes in set S away from sequence Q but remaining the position of other genes unchanged, and '$Q\uparrow P$' means filling up the empty position in sequence Q with genes in a sequence P. The crossover of x and y, $x\otimes y$, is not necessarily equal to $y\otimes x$. The N-demand crossover operator abides by precedence constraints, because the genes in sequences $\langle g_i \rangle_{i=1}^{n}$ and ($H_1<H_2<\ldots<H_{n+1}$) belong to different demands.



(a) one-demand crossover (demand d2 is preserved)



(b) two-demand crossover
(demands d1 and d3 are preserved)



(c) shift mutation

Fig. 5.9: The reproduction operators of MDGA

Shift Mutation

A single gene is chosen randomly from a chromosome, and then inserted into a random position after its preceding genes and before its succeeding gene as shown in Fig. 5.9c. Besides, resource and aggregation flag of the gene, and scheduling alternative of the chromosome are randomly given a new value. The WIP allocation to the gene, if any, is set to 0, and the mismatch caused by shift mutation invokes a process to redistribute WIP quantity among those genes sharing the WIP.

5.4.4 Generation Alternation

Generation alternation models are important to provide controls on searching process. Existing works have often used SGA (simple or standard GA) for a fixed generation alternation model. Yamamura et al. (1996) proposed a roulette minimal generation gap (rMGG) as an extension of MGG. The MDGA applies rMGG to generation alternation.

Fig. 5.10 lists the steps of generation alternation of MDGA. Line (01) shows the generation of initial population; from line (03) to (08), the N-demand crossover operator; and from line (10) to (11), the shift mutation operator.

The crossover operator selects two chromosomes from the population as shown in line (03), reproduces two descendants in line (04), evaluates them in line (05), chooses the best one among the four chromosomes in line (06) and any one from the remains in line (07), and puts the two chromosomes back to the population with replacement in line (08). In this figure, m denotes the population size, n the number of generations, and k the mutation rate.

```
/* Generation alternation with rMGG
rMGG (m, n, k)
(01)   Generate m chromosomes as initial population P;
(02)   FOR each generation UNTIL nth generation
(03)     Remove 2 chromosomes {x, y} from P;
(04)     Apply N-demand crossover on {x, y}, and get {x', y'};
(05)     Evaluate x' and y';
(06)     Choose the best fit from {x, y, x', y'} as b;
(07)     Choose any one from {x, y, x', y'}−{b} as a;
(08)     Put a and b back to P;
(09)     IF random number < k
(10)       Select a chromosome ch from P;
(11)       Apply shift mutation to ch, and evaluate it;
(12)     END IF
(13)   END FOR
```

Fig. 5.10: Procedures of generation alternation with rMGG

The main difference between SGA and MDGA is that SGA reproduces all the offspring at

a generation then carries out the tournament selection from the whole population, while MDGA executes roulette selection from the 4 chromosomes immediately after reproduction. MDGA keeps variety of chromosomes in a population, prevents the search process from rushing into local optima.

5.4.5 Correctness and Effectiveness

The correctness and effectiveness of MDGA is demonstrated by an exhaustive search (ES) and a benchmark. The purpose of ES is to verify the correctness of MDGA by examining all the possible elements in domain of some simple DFS problems to find the best solution, and comparing the solution with the result of MDGA. In the benchmark, MDGA is applied to solve JSS problems which are simplified DFS problems, and then compare MDGA with other methods for the JSS problems.

Exhaustive Search

As shown in Table 5.1, according to 4 sets of demands, four problems based on the product data shown in Fig. 5.1a, and 8 units of WIP M-0 are prepared. These problems and their results by ES and MDGA are shown in Table 5.1.

We accomplished the exhaustive search of problem 1, 2, and 3. Problem 4 had been tried for two months on a PC, while the best value found by ES during the search is even worse than that of MDGA. The correctness of MDGA is proved by that it achieves the optimum value identical to the result of ES. The computer executing ES and MDGA can process about 2000 chromosomes per second. The fact that MDGA reaches the optimum value in less than a second gives an account of the efficiency.

Table 5.1: Result of the exhaustive search

No	Demand set (part, qty, due)	DFS problem Domain size	Time	Result	E_{mks}	E_{prt}	E_{tds}	E_{svc}
1	d1=(F, 4, 150)	5,760	3 sec	ES	120	170	0	100
				MDGA (sn[1])	120 (1)	170 (3)	0 (1)	100 (1)
2	d1=(F, 4, 150) d2=(A, 4, 120)	25,021,440	3.1 hrs.	ES	120	212	0	100
				MDGA (sn)	120 (44)	212 (42)	0 (579)	100 (35)
3	d1=(F, 4, 150) d2=(A, 4, 180) d3=(B, 4, 80)	4,149,596,160	21.4 days	ES	120	248	0	100
				MDGA (sn)	120 (34)	248 (475)	0 (338)	100 (562)
4	d1=(F, 4, 150) d2=(F, 2, 180)	115,142,123,520	49.5 years	ES[2]	126	268	0	100
				MDGA (sn)	120 (633)	268 (1,162)	0 (25)	100 (39)

1 'sn' means number of times of the scheduling run when the best value is found by MDGA.
2 The result of running exhaustive search for about 2 months.

Benchmark

Job shop scheduling (JSS) problem is a subset of DFS problem. Moreover, JSS problem is a restricted DFS problem. If we do not use bill of materials, routing flexibility, WIP, or setup time, if we specify only forward scheduling as the scheduling alternative, and if we select makespan as the evaluation function, then we have a JSS. In other words, it is hardly to produce a practical schedule by solving JSS where the use of product data is inevitable.

A benchmark of some famous JSS problems is used to compare with the work of Croce (1995), who proposed an encoding based on preference rules and an updating step which speeds up the evolutionary process. The problems whose identification starts with 'MT' are from Muth & Thompson, and with 'LA' are from Lawrence, according to Croce (1995).

It is not appropriate to compare the performance of GA on the basis of time, since the experiments are carried out on different computers with different operating system and implemented by using different programming languages with different skill. As MGG has different definition of a generation with SGA, generation is not adequate for comparison either. Croce measured the performance on the basis of the number of chromosomes generated during a run. In a similar way, we count the times of scheduling as the basis of comparison.

Table 5.2: Comparison of MDGA with Croce's GA

Problem	n	m	OPT[1]	NDC[2]	MDGA						Croce	
					POP=50		POP=100				POP=300	
					SCH=10000		SCH=30000		SCH=60000		SCH=30000	
					Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.
MT06	6	6	55	3	55	55.0	–	–	–	–	55	55.0
MT10	10	10	930	5	955	965.2	939	949.0	939	948.4	946	965.2
MT20	20	5	1165	10	1176	1193.4	1174	1178.0	1165	1172.2	1178	1199.0
LA01	10	5	666	5	666	666.0	–	–	–	–	666	666.0
LA06	15	5	926	7	926	926.0	–	–	–	–	926	926.0
LA11	20	5	1222	10	1222	1222.0	–	–	–	–	1222	1222.0
LA16	10	10	945	5	967	979.0	959	973.6	946	963.0	979	989.0
LA21	15	10	1048	7	1074	1098.8	1066	1077.4	1055	1071.2	1097	1113.6
LA26	20	10	1218	10	1281	1294.8	1220	1230.8	1218	1226.6	1231	1248.0
LA31	30	10	1784	15	1784	1784	–	–	–	–	1784	1784
LA36	15	15	1268	7	1336	1339.4	1305	1312.0	1297	1306	1305	1330.4

1. 'OPT' means the best value found so far by the heuristic researches.
2. 'NDC' means number of demand crossover using in the benchmark.

The number of N-demand crossover (NDC) is set to the maximum degree (half of the number of demands). No mutation is set in the benchmark. Table 5.2 shows the performance of MDGA where the number of scheduling (SCH) is used as the termination parameter. SCHs are set to 10000, 30000, and 60000, respectively. Croce's result is shown for comparison, which uses 30000 chromosomes. Two population sizes (POP) are set: POP=50 in SCH=10000

for faster termination, and POP=100 in SCH=30000 and SCH=60000 for slower termination. If MDGA achieves the best value so far (OPT) by applying POP=50 to some easy problem, then the test for POP=100 is omitted. The best makespan shown in the table is selected over five runs, and so is the average makespan.

By observing Table 5.2, MDGA is not a bad method for solving JSS problem. Furthermore, its ability overcomes simple JSS solvers, in the sense that MDGA provides a way to handle practical product data and then is able to produce feasible schedule.

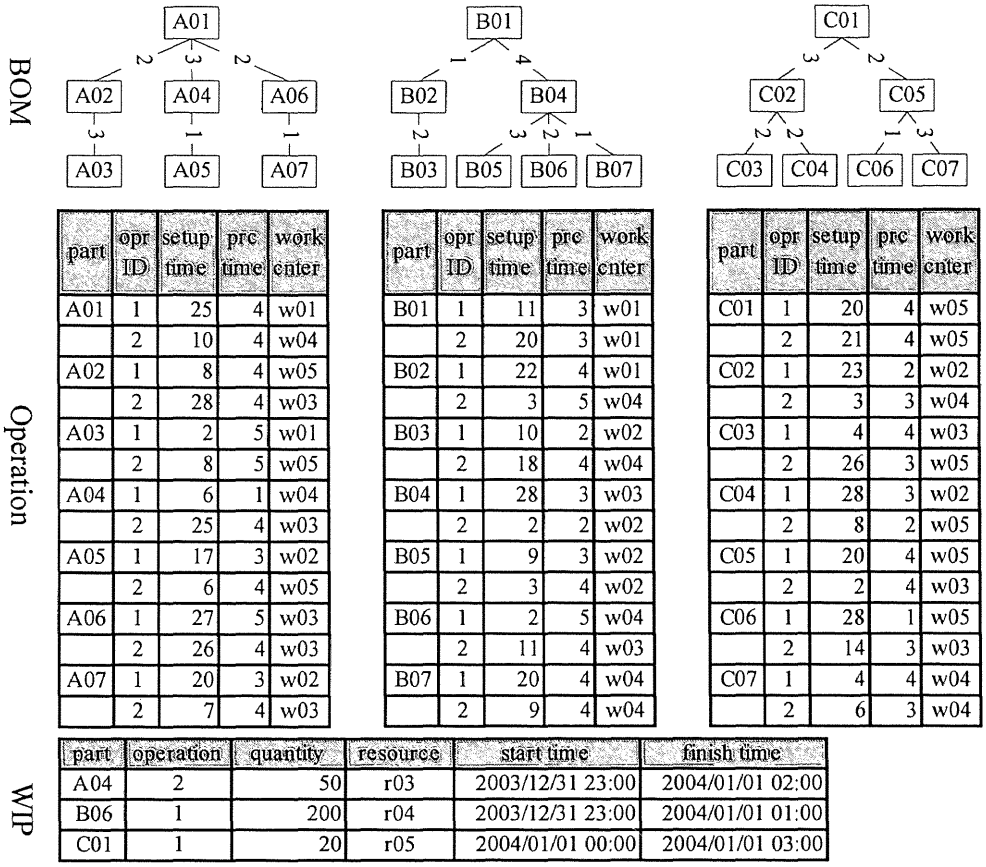
5.5. Experimental Analysis

An experiment is conducted to investigate some factors and their mutual effects on applying genetic algorithms with MGG and demand crossover (MDGA) to dynamic flexible scheduling (DFS) problem. The result shows the limitation of MDGA and also gives some advice on using MDGA smartly. The factors in DFS problem that might have influence on performance of MDGA are the number of demand, routing flexibility, and evaluation functions.

- Number of demands (ND): The number of demands increases exponentially in domain size of DFS problem. Four levels of the factor are set in the experiment as shown in Fig. 5.11b, which are ND=3 (nd03), ND=6 (nd06), ND=12 (nd12), and ND=60 (nd60). The demands in nd06, nd12, and nd60 are generated by splitting quantity of a demand in nd03 into 2, 4, and 20, respectively. The domain of nd03 is a subset of nd06, because requirements exploded from demands in nd03 can be composed by aggregating the requirements from demands in nd06. Similarly, nd06 and nd12 are sub-problems of nd12 and nd60 respectively.
- Resource flexibility (RF): A resource is called flexible, if it works for more than one work centers. Resource flexibility is defined as the average number of work centers that a resource joins. Three levels of the factor, rf01, rf03, and rf05, are well prepared, as shown in Fig. 5.11c, to make the work center - resource pairs in rf01 and rf03 be subset of the pairs in rf03 and rf05, respectively.
- Evaluation function (EF): Three evaluation functions are performed in the experiment, which are makespan, service level, and tardiness.

There are 27 cases composed by 3 NDs, 3 RFs, and 3 EFs in the experiment, and each case runs 10 times. Population size 50 is set, and the number of N-demand crossover (NDC) is set to be $0.5|D|$. The product data used in the experiment is shown in Fig. 5.11a. The result of each run is evaluated when the number of scheduling (SCH) equals 30000.

We run the cases on a laptop computer with Centrino Penitum® M 0.9 Ghz CPU, and the necessary times for running problem nd03, nd06, nd12, and nd60 are 50, 105, 250, and 2610 seconds, respectively. If the numbers of demands are 100, 200, and 300, the necessary times become 200, 1600, and 7500 minutes, respectively. The time needs to run a case grows up exponentially with the length of a chromosome.



Operation

part	opr ID	setup time	prc time	work cnter
A01	1	25	4	w01
	2	10	4	w04
A02	1	8	4	w05
	2	28	4	w03
A03	1	2	5	w01
	2	8	5	w05
A04	1	6	1	w04
	2	25	4	w03
A05	1	17	3	w02
	2	6	4	w05
A06	1	27	5	w03
	2	26	4	w03
A07	1	20	3	w02
	2	7	4	w03

part	opr ID	setup time	prc time	work cnter
B01	1	11	3	w01
	2	20	3	w01
B02	1	22	4	w01
	2	3	5	w04
B03	1	10	2	w02
	2	18	4	w04
B04	1	28	3	w03
	2	2	2	w02
B05	1	9	3	w02
	2	3	4	w02
B06	1	2	5	w04
	2	11	4	w03
B07	1	20	4	w04
	2	9	4	w04

part	opr ID	setup time	prc time	work cnter
C01	1	20	4	w05
	2	21	4	w05
C02	1	23	2	w02
	2	3	3	w04
C03	1	4	4	w03
	2	26	3	w05
C04	1	28	3	w02
	2	8	2	w05
C05	1	20	4	w05
	2	2	4	w03
C06	1	28	1	w05
	2	14	3	w03
C07	1	4	4	w04
	2	6	3	w04

WIP

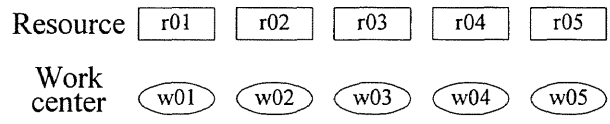
part	operation	quantity	resource	start time	finish time
A04	2	50	r03	2003/12/31 23:00	2004/01/01 02:00
B06	1	200	r04	2003/12/31 23:00	2004/01/01 01:00
C01	1	20	r05	2004/01/01 00:00	2004/01/01 03:00

(a)

No. of demands

nd03			
(A01,	100,	2004/01/07 12:00)	×1
(B01,	100,	2004/01/06 00:00)	×1
(C01,	100,	2004/01/08 10:00)	×1
nd06			
(A01,	50,	2004/01/07 12:00)	×2
(B01,	50,	2004/01/06 00:00)	×2
(C01,	50,	2004/01/08 10:00)	×2
nd12			
(A01,	25,	2004/01/07 12:00)	×4
(B01,	25,	2004/01/06 00:00)	×4
(C01,	25,	2004/01/08 10:00)	×4
nd60			
(A01,	5,	2004/01/07 12:00)	×20
(B01,	5,	2004/01/06 00:00)	×20
(C01,	5,	2004/01/08 10:00)	×20

(b)



Resource Flexibility

level	rf01	rf03	rf05
work center \ resource	w01{r01}	w01{r01,r02,r03}	w01{r01,r02,r03,r04,r05}
	w02{r02}	w02{r02,r03,r04}	w02{r02,r03,r04,r05,r01}
	w03{r03}	w03{r03,r04,r05}	w03{r03,r04,r05,r01,r02}
	w04{r04}	w04{r04,r05,r01}	w04{r04,r05,r01,r02,r03}
	w05{r05}	w05{r05,r04,r03}	w05{r05,r04,r03,r02,r01}

(c)

Fig. 5.11: Experiment data of (a) product data, (b) No. of demands, and (c) resource flexibility

Table 5.3 shows the best value, average value, performance, and variance of the cases. The 'best' values, found so far, of the evaluation functions are discovered by setting parameters to keep MDGA in a divergence status for a long time. The average value is measured over 10 optimal values of a case. Performance of MDGA is defined as the difference between the average value and the best one. The variability of applying MDGA to runs of the cases in the experiment measured by coefficient of variance (CV), denoted by δ/μ,

where δ is the standard deviation and μ the mean of the optimal values.

Table 5.3: The best and statistical results of the experiment

		RF											
		rf01				rf03				rf05			
ND	EF	Best	Avg.	Diff.	CV	Best	Avg.	Diff.	CV	Best	Avg.	Diff.	CV
nd03	Makespan	14338	14338	0	0.000	11073	11254	181	0.013	11073	11352	279	0.023
	Service level	67	53	14	0.306	67	63	4	0.158	67	67	0	0.000
	Tardiness	9033	9164	131	0.036	4658	5712	54	0.097	4067	5381	1314	0.160
nd06	Makespan	13322	13333	11	0.008	10442	10569	127	0.009	10390	10465	75	0.005
	Service level	83	76	7	0.000	83	67	17	0.000	83	67	17	0.000
	Tardiness	6673	6976	303	0.020	1212	3420	2208	0.344	313	3713	3400	0.393
nd12	Makespan	13172	13194	22	0.002	10406	10601	195	0.003	10390	10628	238	0.004
	Service level	83	73	10	0.045	100	74	26	0.060	100	74	26	0.034
	Tardiness	6673	11667	4994	0.016	0	4246	4246	0.180	0	3379	3379	0.479
nd60	Makespan	13112	13587	475	0.003	10406	13498	3092	0.031	10390	11752	1362	0.007
	Service level	83	63	20	0.026	100	66	34	0.043	100	68	32	0.033
	Tardiness	6673	76189	69516	0.046	0	52360	52360	0.140	0	49856	49856	0.139

Since there is an inclusive relationship between levels of ND and RF, the smaller the lot size and the more flexible the resource, the better the best value can be found. However, increasing ND and RF not only provide MDGA with a better chance of optimization, but also enlarge domain size of the problem. The DFS problem with large domain size challenges the limits of MDGA's ability. As shown in the table, increasing ND and RF improves the optimal value at first, but it gets worse when ND and RF continue to increase.

The large ND aggravates the performance of MDGA. RF performs in a similar way with ND except that increasing RF won't delay the response time or severely worsen the performance of MDGA. A plant with high resource flexibility using MDGA against uncertainty is regarded as capable of responding to a change well and efficiently.

In general, the case setting makespan as evaluation function has low variability ($CV<0.1$). Whether or not service level performs stable depends very much on the problem. For some difficult cases like the combination of nd06 and rf03, the performance of service level obtained by setting tardiness as evaluation function is even better than by setting service level itself. There are n+1 degrees of service level if ND equals to n. Having few degree of evaluation makes MDGA easy to converge to a degree and dull to make a step toward a better degree. The cases setting tardiness as evaluation function has high variability ($CV>0.1$) when ND and RF are high. The solution to the high variability of tardiness is to run a case longer or set a larger population.

5.6. Discussion

Tsai and Sato (2004b) showed comparisons of performance on standard job shop scheduling problems between standard genetic algorithm (SGA) and minimal generation gap (MGG), and between demand crossover and other crossover operators such as one-point crossover and two-point crossover. It showed that MGG improves in average 3.5% of the performance of SGA, demand crossover improves in average 3% of the performance of two-point crossover, and MGG plus demand crossover improves in average 8% the performance of SGA plus two-point crossover.

5.7. Summary

A novel genetic algorithm (MDGA) that integrates MGG and demand crossover has been invented to solve dynamic flexible scheduling (DFS) problem. The problem is practical, goal-oriented, resource flexible, and capable of doing rescheduling dynamically. Though MDGA approach to DFS problem has its own value, this research is also an augmentation of agile production planning and control system (APPCS) that only generates a feasible schedule.

The effectiveness and correctness of MDGA have been shown by a benchmark and the exhaustive search. The formulation of DFS problem makes the exhaustive search possible.

The response time of MDGA to DFS problem increases exponentially with the length of a chromosome, which is determined by the shape of BOM, routing, and number of demands. Therefore, when MDGA is applied to a plant, to estimate execution time, it is necessary to calculate the length of a chromosome made from the BOM, routing, and demands. The experiment suggests that if the lengths of a chromosome are 700, 900, 2000, 3000, and 4500, then the response times will be 05 hour, 1 hour, 0.5 day, 1 day, and 5 days, respectively. A more efficient algorithm for a huge DFS problem will be a topic of future research.

A balance between the flexibilities and the ability of MDGA is a key point to get a better optimal value. The experiment for the example indicates that a double or triple flexibility improves about 10% - 25% of optimal value.

Forecasting is always wrong. Reserving safety buffers for a forecasting error is not the only way against unknown uncertainty. Forecasting revision is shown to be possible by APPCS and improved by MDGA.

6 Conclusion and Future Research

6.1. Conclusion

This dissertation is composed of three parts. Agile production planning and control system (APPCS) was proposed in the first part. In the second part, we described a universal modeling language (UML) model of agile production planning and control (APPCM) to formulate the concept and mechanism of APPCS. Finally, in the last part, a novel genetic algorithm called MDGA was proposed to solve dynamic flexible scheduling (DFS) problem to produce a much optimized schedule for APPCS.

The proposed APPCS integrates scheduling with capacity planning to produce a feasible production plan in a planning cycle, and updates the production plan after being informed of a change. APPCS invokes immediate rescheduling upon advance notification from customers and/or suppliers to enhance the agility of production processes. A buffering mechanism against uncertainty can be applied by setting safety leadtime and/or safety stock according to a target service level. A simulation analysis showed that safety leadtime is preferable to safety stock in most uncertain situations, especially when advance notification mechanism is allowable. APPCS can be used as a real-time system in the sense that it makes a schedule when a customer order arrives and reschedules when uncertainty happens.

In the formulated APPCS model, the classes *Part*, *PartLink*, *Operation*, *WorkCenter*, *Resource*, *Shift*, *Demand* are used to define the product data, and classes *Job*, *JobLink*, *Assign*, Task, and *Supply* are used to represent the production plan generate by APPCS. A feasible production plan is generated and regenerated under some constraints that regulate the legality of the production plan by first backward scheduling and then forward scheduling according to a priority rule. A simulator applying APPCS model was implemented and testified by an example. The proposed APPCM was shown to meet the requirements of APPCS.

DFS problem is defined on the basis of product data with resource flexibility and

work-in-process (WIP). The production plan that is obtained by solving DFS problem is thus practical. That WIP is used in the production plan enables rescheduling dynamically in response to changes. The proposed MDGA integrates minimal generation gap (MGG) and demand crossover to solve DFS problem. Though MDGA approach to DFS problem has its own value, this research is also an augmentation of APPCS that only generates a feasible schedule. The correctness of MDGA was partly proved by the exhaustive search, and the formulation of DFS problem makes the exhaustive search possible. Furthermore, a benchmark showed the effectiveness of MDGA. Reserving safety buffers is not the only way against unknown uncertainty. Revision of a production plan is shown to be possible by APPCS and improved by MDGA.

APPCS was proposed and modeled by UML to use the product data of modern ERP packages, to invoke production planning by applying time-bucket approach, to take into consideration WIP, and to incorporate the changes into the production plan. MDGA was proposed to provide a goal-oriented optimal production plan. Therefore, the three parts of this dissertation were shown to achieve the five requirements mentioned in problem formulation (section 1.2).

6.2. Future Research

The topics for future research are as follows.

(1) Applying APPCS to service industry

Since business processes of service companies have something in common with production process, APPCS can also be applied to banks, city halls, or fast food restaurants where customers are waiting for different but well-defined service items by various staffs. Each service item has a sequence of operations that have to be processed by different work groups. There are many staffs in a work group, and each staff in the work group can be responsible for processing an operation. A staff can join in different work groups according to his/her capability.

A customer will be asked and also helped by a staff to order a service item on the arrival of the customer. The priority of customers is determined by their arrival times. A schedule of service is generated and optimized according different goals. The possible one connecting to service level is to minimize customer waiting time, which is defined as the difference between actual waiting time and estimated total processing time. Rescheduling is invoked when a new customer is arriving or for the delay of a staff.

By applying APPCS, customers know the possible waiting time, and thereby they can keep waiting their service without anxiety. The capability and necessity of a staff can be verified by a statistic analysis of the periodical service records. The verification also encourages the staffs to improve their skill and capability. The whole business processes including staff allocation can be improved by a simulation of the past requested services. That is APPCS seems to open a new door to analysis and design of service industries.

(2) Comparison of time-bucket approach and real-time approach

Time-bucket approach and real-time approach are the two alternatives of production planning provided by APPCS. Real-time approach takes advantage of the flexibility to produce a better production plan. Time-bucket approach is preferable if purchasing cost or order changing cost is important. For making a right scheduling policy, it is important to get some beneficial results from the comparison of the two approaches in more detail.

(3) The importance of the quality of quasi-optimization

Genetic algorithm (GA) is a heuristic algorithm that does not guarantee finding any optimal solution. The execution time of GA (or number of scheduling times in MDGA) before

it enters some local optima is a variable to determine the quality of quasi-optimal solution, i.e. the difference between the optimal solution and quasi-optimal solution. Most of the researches are trying to improve quality of quasi-optimization for most of the NP-hard problems. According to APPCS, rescheduling is invoked to produce a new production plan each time a change is reported. It is doubtful that whether or not the quality of quasi-optimization can be improved by setting quality of quasi-optimization to be rather high from the start. For example, a rather high degree of resource utilization achieved by minimizing the makespan might not be flexible enough to produce a new production plan for the next change. Contrarily, the makespan might be shorter if the resource utilization is not so squeezed at first.

If there is an adverse effect on the quality of quasi-optimization under uncertainty by setting a high quality of optimization from the start, it is better to set a lower one to have a better performance and dramatically shorten the execution time. Hence, an experiment is necessary to make clear the relationship between the result quality of quasi-optimization and the setting quality due to the influence of changes.

(4) The optimization of the whole supply chain

The optimization of a node in a supply chain is not necessary the optimization of the whole supply chain. Moreover, no matter how optimized of some nodes in a supply chain, if the whole supply chain is not optimized, the effect of optimization of the nodes might be trivial. Toyota system has been successful in optimizing the whole supply chain by the so called just-in-time approach. Can we say that the optimization processed in all nodes in a supply chain makes the optimization of the whole supply chain? If not, what the optimization of the whole supply chain should be, and what should the nodes respond to the optimization policy of the chain. Together with the impacts brought about by changes in a supply chain, these topics are complicated but important for the nodes to determine their production plans.

(5) Applying APPCS optimization to a plant with large scale scheduling

The time needed to run a case of MDGA grows up exponentially with the length of a chromosome. For example, if the lengths of a chromosome are 700, 900, 2000, 3000, and 4500, then the response times will be 05 hour, 1 hour, 0.5 day, 1 day, and 5 days, respectively, for processing 30,000 chromosomes. The ability of the MDGA is enough for the current research and the future researches (1), (2), and (3). However, if there is a need to apply APPCS to solve a huge DFS problem in a plant with large scale scheduling, a genetic algorithm with a hierarchical structure coding seems to be necessary to replace the current coding.

References

[1] Al-Hakim, L., An analogue genetic algorithm for solving job shop scheduling problems, *International Journal of Production Research*, 39 (7) 1537-1548, 2001.

[2] Aytug, H., Khouja, M., and Vergara, F. E., Use of genetic algorithms to solve production and operations management problems: a review, *International Journal of Production Research*, 41 (17) 3955-4009, 2003.

[3] Croce, F. D., Tadei, R., and Volta, G., A genetic algorithm for the job shop problem, *Computers & Operations Research*, 22 (1) 15-24, 1995.

[4] Dagli, C. H. and Sittisathanchai, S., Genetic neuro-scheduler: A new approach for job shop scheduling, *International Journal of Production Economics*, 41, 135-145, 1995.

[5] Hastings, N.A.J. and Yeh, C.H., Job oriented production scheduling, *European Journal of Operational Research*, 47, 35-48, North-Holland, 1990.

[6] Hatchuel, A., Saidi-Kabeche, D., and Sardas, J.C., Toward a new planning and scheduling approach for multistage production system, *International Journal of Production Research*, 35 (3), 867-886, 1997.

[7] Hegedus, M. G. and Hopp, W. J., Setting procurement safety lead-times for assembly systems, *International Journal of Production Research*, 39, 3459-3478, 2001.

[8] Hopp, W. J. and Spearman, M. L., *Factory Physics – Foundations of Manufacturing Management*, 2nd edition, McGraw-Hill College, 2000.

[9] Kacem, I., Hammadi, S., and Borne, P., Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 32 (1) 1-13, 2002.

[10] Lee, H. C., Dagli, H., A parallel genetic-neuro scheduler for job-shop scheduling problems, *International Journal of Production Economics*, 51, 115-122, 1997.

[11] Nearchou, A. C., The effect of various operators on the genetic search for large scheduling problems, *International Journal of Production Economics*, 88 (2) 191-203,

2004.

[12] OMG, *OMG Unified Modeling Language Specification* (Action Semantics)-V1.4, 2002.

[13] Orlicky, J., *Material Requirements Planning*, New York, McGRAW-HILL BOOK, 1975.

[14] Plossl, G. W., *Managing in the new world of manufacturing*, Englewood Cliffs, NJ: Prentice-Hall, 1991.

[15] Sanchez, L. M., Nagi, R., A review of agile manufacturing system, *International Journal of Production Research*, 39(16), 3561-3600, 2001.

[16] SAP, SAP R/3 IDES 4.6, functions manual, 2000.

[17] SYMIX, OrderLinks 3.5, Integration Guide, 1998.

[18] Sato, R., Meaning of Dataflow Diagram and Entity Life History - A Systems Theoretic Foundation for Information Systems Analysis: Part 2, *IEEE Transactions on Systems, Man, and Cybernetics*, 27, 11-22, 1997.

[19] Sato, R. and Praehofer, H., A discrete event model of business system- A Systems Theoretic Foundation for Information Systems Analysis: Part 1, *IEEE Transactions on Systems, Man, and Cybernetics*, 27, 1-10, 1997.

[20] Sato, R. and Tsai, T., An agile production planning and control with advance notification to change schedule, *International Journal of Production Research*, 42 (2) 321-336, 2004.

[21] Scheer, A.W., *Business Process Engineering*, 2nd edition, Springer, 1994.

[22] Shobrys, D. E., White, D. C., Planning, scheduling, and control system: why cannot they work together, *Computers and Chemical Engineering*, 26, 149-160, 2002.

[23] Silver, E. A., Pyke, D. F., and Peterson, R., *Inventory Management and Production Planning and Scheduling*, 3rd edition, New York, JOHN WILEY & SONS, 1998.

[24] Tsai, T. and Sato, R., A UML model of agile production planning and control system, *Computers in Industry*, 53 (2) 133-152, 2004a.

[25] Tsai, T. and Sato, R., An Agile Genetic Algorithm for Solving Job Shop Scheduling Problem, *Proceedings of the 8th Pacific Asia Conference on Information Systems 2004 (PACIS2004)*, 2004b, Compact Disk.

[26] Tsai, T., Sato, R., and Terano, T., A genetic algorithm with MGG and demand crossover to solve dynamic flexible scheduling problem, *European Journal of Operational*

Research, submitted.

[27] Tu, Y., Production planning and control in a virtual One-of-a-Kind Production company, *Computers in Industry*, 34, 271-283, 1997.

[28] Vollmann, T. E., Berry, W. L., and Whybark, D. C., *Manufacturing Planning & Control Systems*, 4th edition, Boston, Irwin/McGraw-Hill, 1997.

[29] Wang, L., Zheng, D., An effective hybrid optimization strategy for job-shop scheduling problems, Computers & Operations Research, 28 (6) 585-596, 2001.

[30] Whybark, D. C. and Williams, J. G., Material requirements planning under uncertainty, *Decision Sciences*, 7, 595-606, 1976.

[31] Wight, O., *Manufacturing Resource Planning: MRPII*, Essex Junction, Vt.: Oliver Wight Ltd., 1984.

[32] Yamamura, M., Satoh, H., and Kobayashi, S., An analysis on generation alternation models by using the minimal deceptive problems, *Journals of the Japanese Society for Artificial Intelligence*, 13 (5) 746-756, 1996. (in Japanese)

[33] Yeh, C-H, Schedule based production, *International Journal of Production Economics*, 51, 235-242, 1997.

Appendixes

Appendix A: UML Model

The formulation is described by universal modeling language (UML) and illustrated by class diagram, state chart diagram, and sequence diagram. The UML is a language for specifying visualizing, constructing, and documenting the artifacts of software system, as well as for business modeling and other non-software system. It is composed of foundation, behavioral elements, and model management packages. A package is a grouping of model elements. The foundation package defines the constructs to abstract a static model.

A.1: Model Management

Model management is used to manage the modeling of a complex system. A system can be divided into several subsystems according to their functions. If a subsystem is too complicated to be described by some modeling tools or the result of modeling is hard to understand, then it should be subdivided into some smaller systems. Fig. A.2 shows a template of the presentation of model management.

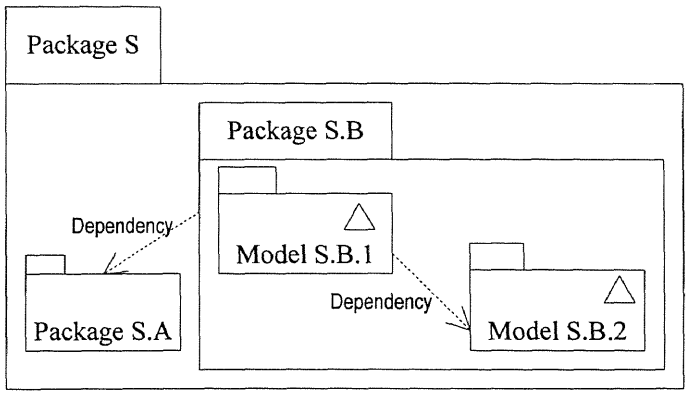


Fig. A. 1: Notations used in model management

Package

A package contains a group of model elements that are used to describe the package. The

model element includes models and packages. A package is shown as a large rectangle with a small rectangle (a tab) attached to the left side of the top of the large rectangle. In UML, a package is used to typify a system or a subsystem, and therefore the package hierarchy is a strict tree.

Model

A model captures a view of a physical system. Different models of the same physical system show different aspects of the system. A model is notated using the ordinary package symbol with a small triangle in the upper right corner of the large rectangle.

Dependency

A dependency indicates a semantic relationship between two model elements including system, package, and model. A dependency is shown as a dashed arrow from

- Access: The granting of permission for one package to reference the public elements owned by another package.
- Refine: A historical or derivation connection between two model elements with a mapping between them.

A.2: Use Case Diagram

A use case diagram shows the relationship within a system and their actors. Fig. A.2 shows a template of the use case diagram.

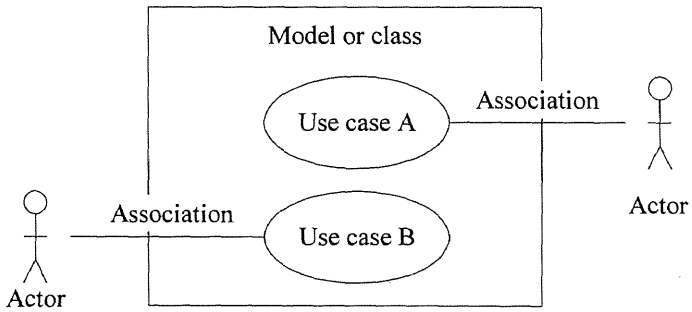


Fig. A. 2: A template of use case diagram

Use case

Use cases represent the functionality provided by a system or a class. A use case is shown as an ellipse containing the name of the use case.

Actor

An actor is a set of roles that users of a system can play when interacting with the system. The standard icon for an actor is a "stick man" with the name of the actor below the icon.

Association

An association is the participation of an actor in a use case. Instance of the actor and instances of the use case communicate with each other. An association between an actor and a use case is shown as a solid line between the actor and the use case. It may have end adornments such as multiplicity.

A.3: Class Diagram

A static data structure is a set of elements including interfaces, packages, classes, associations among the classes, and even instances including objects and links. These elements are used to describe the static structure of a package. A class diagram is a graphic view of the static data structure. The division of the presentation into separate diagrams is for graphical convenience and does not imply a partitioning of the package itself. Fig. A.3 shows the elements in a class diagram.

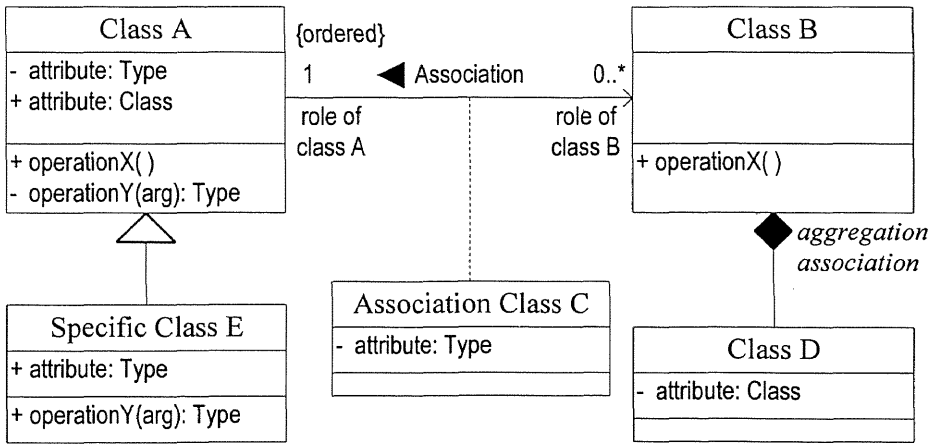


Fig. A. 3: A template of class diagram

Class

A class is an entity for a set of objects with similar property, behavior, and relationships. The property of a class is described by attributes. The behavior of a class is demonstrated by operations. A class is drawn as a solid-outline rectangle with three compartments separated by horizontal lines. The top compartment holds the class name; the middle compartment lists its

attributes; the bottom compartment lists its operations.

Attribute

An attribute is to represent a property of a class. A class has a set of attributes and an object of the class is generated by providing a value to each attribute. For example, name, height, and weight are attributes of human class that is defined for some purpose. A human object has a value for each of those attributes. An attribute is defined by specifying name, data type, visibility, multiplicity, and initial value in UML.

The syntax for an attribute is:

"visibility attributeName: dataType [multiplicity] = initialValue".

- The "visibility" specifies whether the attribute can be seen and referenced by other model elements. It can be '+' public visibility that provides reference for all the model elements, '#' protected visibility for the descendent elements, '-' private visibility for only the object itself, and '~' package visibility for the elements in the same package.
- The "attributeName" is a string used to represent or identify an attribute in a class. Thus, there is no identical attribute name in a class.
- The "dataType" is either a class name or the primitive data type defined in some programming languages. If the data type of an attribute is a class, then the value of the attribute is also an object of the class.
- The "[multiplicity]", represented by "[ower-bound .. upper-bound]", specifies the allowable number of attribute values for a class. For example, multiplicity of 'hand' attribute is "0..2" for a 'human' class. The term may be omitted, in which case the multiplicity is "1..1" (exactly one).
- The "initialValue" is the value assigned to an attribute when an object is created.

Operation

An operation is a procedure that an object of the class may be requested to perform. The procedure being invoked changes values of some attributes or lunches other operations sequentially or synchronously.

An operation has a name, a list of arguments, and a return value. It is denoted by

"visibility operationName (argumentList): returnValueType".

- The "visibility" specifies whether the operation can be seen and invoked by other model

elements. It can be '+' public visibility that limits the invocation to all the model elements, '#' protected visibility to the descendent elements, '-' private visibility to only the object itself, and '~' package visibility to the elements in the same package.

- The "operationName" is a string used to represent or identify an operation in a class. Thus, there is no identical operation name in a class.
- The "argumentList" is a comma-separated list of parameters, which hold values that can be accessed by elements within the procedure of the operation. A parameter can be represented by "parameterName: parameterType", which shows name and data type of the parameter.
- The "returnValueType" specifies the data type of the value returned by an operation. It is either a class name or the primitive data type defined in some programming languages.

Association

A link is an ordered n-tuples of objects (x_1, x_2,..., x_n), where x_1, x_2,..., x_n are instances of respective classes C_1, C_2,..., C_n. An association is an entity to represent a set of links. The classes might not be different. An n-ary association is an association among three or more classes; while a binary association is an association among exactly two classes.

An association can be described by an association name and two or more association ends. An association end is an end of an association where it connects to a class. Role name, multiplicity, ordering, navigability, and aggregation indicator are attributes of an association end.

- Role name: It is a string that indicates the role played by the class connecting to the association end. A class plays different roles in different associations. For example, a human class in a school plays the roles of student, staff, and teacher. The visibility indicator ('+', '#', '-', and '~') is specified in front of a role name to show the visibility of the association traversing in the direction toward the role name.
- Multiplicity: It specifies the allowable number of links that a role can play in an association. It is denoted by "[lower-bound .. upper-bound]". A single start (*) denotes the unlimited nonnegative integer range.
- Ordering: If the multiplicity is greater than one, then the set of related links can be ordered or unordered. Various types of ordering can be specified as a constraint on the association end. If the ordering of links is important for an association, "{ordered}" is specified on the association end.
- Navigability: An arrow may be attached to the association end to indicate that navigation is supported toward the class attached to the arrow. Navigation is suppressed with navigability in both directions, show arrows only for associations with one-way navigability.

- Aggregation indicator: A binary association connecting a 'whole' class with a 'part' class defines a 'whole-part' relationship, which requires an object of the part class be included in at most one object of 'whole' classes at a time. If there is exact one of such 'whole-part' association for a 'part' class, then the association is composite and the 'whole' class is called a composite class. If there is more than one of such association, then the association is an aggregate and the 'whole' class is called an aggregate class. An aggregation indicator is to represent the type of 'whole-part' association of an association on the association end of the 'whole' side. A hollow diamond is attached to the association end to indicate aggregate, and a solid diamond is used to represent composite.

Association class

An association class is an association that also has class properties, such as attributes or operations. An association class is shown as a class symbol attached by a dashed line to an association path. The name in the class symbol should be the same with the name of the association.

Generation

Generation is the taxonomic relationship between a more general class and a more specific class. It is shown as a solid-line path from the more general class to the more specific class with a large hollow triangle at the end of the path where it meets the more general class.

A.4: Collaboration Diagram

A collaboration contains a collection of instances, relationships, and messages among the instances. A collaboration diagram illustrates a collaboration with instances, relationships, and messages of the collaboration to describe the realization of an operation. Fig. A.4 shows the components in collaboration diagram and followed it the description of the components.

Actor

An actor in a collaboration diagram represents the person, software, hardware, or other agent external to the system that is interacting with the system.

Instance

An instance is an object of a class that is charted by a class diagram. An instance playing the role defined by a class is depicted by an object box with the identification string "objectName : className". The 'objectName' may be omitted. In this case, the colon should

be kept with the class name. This represents an anonymous object of the given class given identity by its relationship.

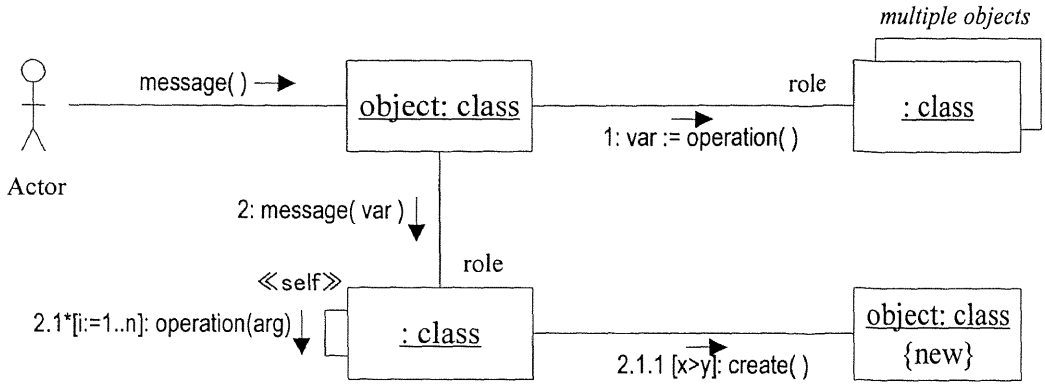


Fig. A. 4: A template of collaboration diagram

Multiobject

A multiobject represents a set of objects on the "many" end of an association. This is used to show operations and messages that address the entire set, rather than a single object in it. A multiobject is shown as two rectangles in which the top rectangle is shifted slightly vertically and horizontally to suggest a stack of rectangles.

Link

A link is a pair of object references. It is also an instance of an association. A link is shown by a line between object boxes. Its name string follows the syntax of an object playing a specific role. A role name may be shown at each end of the link.

Message

A message is a means of communication between two instances connected by a link. The link is used for transportation of the message to the target instance. A message will cause an operation to be invoked, or an instance to be created or destroyed. Message is shown as labeled arrows placed near a link. The arrow points along the line in the direction of the receiving instance. The label has the following syntax: "sequenceTerm: returnValue := messageName (argumentList)".

A 'sequenceTerm' is a dot-separated list of numbers following a condition string denoted by "$n_1.n_2. \ldots .n_i$*[condition]", where n_i is an integer number. Each integer represents a level of procedural nesting within the overall interaction. The sequence of integers represents the sequential order of the massage within the next higher level of procedural calling. For

example, "3.1.4" is after "3.1.3" within the level "3.1". The "*[condition]" indicates an iteration, and "[condition]" shows a branch.

The "messageName" is the name of the operation to be deployed on the receiving instance, or the signal of creating or destroying an instance. The "(argumentList)" is a comma-separated list of arguments enclosed in parentheses. Each argument is a source instance, an attribute of the instance, a link from the instance, or a local variable. The "returnValue" is the value returned at the end of the communication. It can be used as arguments to of the subsequent messages. If the message does not return a value, then it is omitted.

Appendix B

A method of the operations is listed in the appendices for reference. The procedure of the method here is described in terms of programming logic and set theory by assuming *Part*, *PartLink*, *Operation*, *WorkCenter*, *Resource*, *Shift*, *Demand*, *Job*, *JobLink*, *Assign*, *Task*, and *Supply* as existing tables. In some of the procedures, suppose S is a set of elements, *S.add*(i) method adds a job i to the set, *S.delete*(i) removes a job i from the set, *S.get*() gets the next element and remove it from the set, and *S.search*($c1$, $c2$, ...) searches the set for an element with conditions $c1$, $c2$, ...

B.1: Methods of Resource Class

Task Resource.getPrevFreeTask (dueEnd: Epoch, length Time)

// Returns a task represents a free interval of the resource that is located before an epoch (dueEnd) and of a specified length

Task tk:=NEW Task( );
FOR $sf \in \{ s \mid s \in this.shift \,\&\, s.startEpk < dueEnd \}$ DESCENDING BY *sf.startEpk*
  IF $sf.finishEpk < dueEnd$ THEN $dueEnd := sf.finishEpk$;
  $K := \{ k \mid k \in sf.task \,\&\, k.startEpk < dueEnd \}$;
  $tk.finishEpk := \text{Max}\{ t \mid t \in [sf.startEpk, dueEnd] \,\&\, (\forall k \in K)\, t \notin [k.startEpk, k.finishEpk] \}$;
  IF $tk.finishEpk \neq$ Null
    $tk.startEpk := \text{Max}\{ k.finishEpk \mid k \in K \,\&\, k.finishEpk < tk.finishEpk \}$;
    IF $tk.startEpk$=Null THEN $tk.startEpk := sf.startEpk$;
    IF $length < (tk.finishEpk - tk.startEpk)$
      $tk.shift := sf$;
      RETURN tk;
    ENDIF
  ENDIF
ENDFOR
RETURN Null;

Task Resource.getNextFreeInterval (dueStart: Epoch, length Time)

// Returns a resource's free interval that is located after an epoch and of a specified length

Task tk:=NEW Task( );
FOR $sf \in \{ f \mid f \in this.shift \,\&\, f.finishEpk > dueStart \}$ ASCENDING BY *sf.finishEpk*
  IF $sf.startEpk > dueStart$ THEN $dueStart := sf.startEpk$;
  $K := \{ k \mid k \in sf.task \,\&\, k.finishEpk > dueStart \}$;
  $tk.startEpk := \text{Min}\{ t \mid t \in [dueStart, sf.finishEpk] \,\&\, (\forall k \in K)\, t \notin [k.startEpk, k.finishEpk] \}$;
  IF $tk.startEpk \neq$ Null
    $tk.finishEpk := \text{Min}\{ k.startEpk \mid k \in K \,\&\, k.startEpk > tk.startEpk \}$;
    IF $tk.finishEpk$=Null THEN $tk.finishEpk := sf.finishEpk$;
    IF $length < (tk.finishEpk - tk.startEpk)$
      $tk.shift := sf$;

RETURN *sk*;
ENDIF
ENDFOR
RETURN Null;

B.2: Methods of Job Class

Epoch Job.getDueFinishEpk ( )

// Return the due finish epoch of a job, which is the earliest one among its requesting jobs.
Epoch *dueEpkForRequestJob* := Min{ *jl.reqJob.startEpk* | *jl* ∈ *this.reqJobLink* };
Epoch *dueEpkForDemand* := Min{ *d.dueEpk* | *d* ∈ *this.meetDemand* };
RETURN Min (*dueEpkForRequestJob*, *dueEpkForDemand*);

Epoch Job.getDueStartEpk ( )

// Return the due start epoch of a job, which is the latest one among its supplying jobs.
RETURN Max { *jl.supJob.finishEpk* | *jl* ∈ *this.supJobLink* };

Qty Job.getRequestQty ( )

// Return the total independent requirements and dependent requirements of a job
RETURN (∑{ *jl.supQty* | *jl* ∈ *this.reqJobLink* } + ∑{ *d.reqQty* | *d* ∈ *this.meetDemand* });

Qty Released.getDisposableQty ( )

// Return the disposable (assignable) quantity of a job
RETURN (*this.netReq* − *this.getRequestQty*( ) − ∑{ *pa.asgQty* | *pa* ∈ *this.plnAssign*, });

Boolean Job.ifAllSupJobScheduled ( )

// Return true if all the offer jobs of a job is in the "scheduled" state
FOR *sjl* ∈ *this.supJobLink*
  IF *sjl.supJob.netReq* ≠ 0 AND *sjl.supJob.finishEpk* = Null THEN RETURN (False);
ENDFOR
RETURN (True);

Boolean Job.ifAllReqJobScheduled ( )

// Return true if all the request jobs of a job is in the "scheduled" state
FOR ALL *rjl* ∈ *this.reqJobLink*
  IF *rjl.reqJob.netReq* ≠ 0 AND *rjl.reqJob.startEpk* = Null THEN RETURN(False);
ENDFOR
RETURN (True);

void UnreleasedJob.netPlanning ( )

// Net requirement planning a job
Epoch *dueEpk* := *this.getDueFinishEpk*( );
Qty *reqQty* := *this.getRequestQty*( );
AssignableJob := { *rj* | *rj* ∈ *ReleasedJob* & *rj.part* = *this.part* &
  rj.finishEpk ≤ *dueEpk* & *rj.getDisposableQty*( ) > 0 };
FOR *aj* ∈ *AssignableJob* DESCENDING BY *aj.finishEpk*
  Qty *assignQty* := *aj.getDisposableQty*( );

```
    Assign asg:=NEW Assign(plnJob=this, wipJob=aj, asgQty=assignQty);
    this.wipAssign.add(asg);
    aj.plnAssign.add(asg);
    IF reqQty>assignQty
      asg.asgQty:=assignQty;
      reqQty:=reqQty-assignQty;
    ELSE
      asg.asgQty:=reqQty;
      reqQty:=0;
      BREAK;
    ENDIF
  ENDFOR
  this.netReq:=reqQty;
  RETURN
```

void UnreleasedJob.backwardScheduling()

```
// Execute scheduling of a job backwardly from the due finish epoch of the job
Epoch dueFinishEpk:=this.getDueFinishEpk();
IF this.part.operation=Ø
  this.finishEpk:=dueFinishEpk;
  this.startEpk:=dueFinishEpk-this.part.procureLeadTime;
ELSE
  FOR opr ∈ this.part.operation DESCENDING BY opr.oprNo
    Time reqTime:=opr.setupTime+opr.processTime×this.netReq;
    Resource res:=opr.processWkctr.chooseResource(dueFinishEpk, reqTime);
    WHILE reqTime>0
      Task tsk:=res.getPrevFreeTask(dueFinishEpk);
      reqTime:=reqTime-(tsk.finishEpk-tsk.startEpk);
      IF reqTime<0
        tsk.startEpk:=tsk.startEpk-reqTime;
      tsk.job:=this;
      tsk.operation:=opr;
      this.task.add(tsk);
      tsk.shift.task.add(tsk);
      dueFinishEpk:=tsk.startEpk;
    ENDWHILE
  ENDFOR
  this.startEpk:= this.part.backwardScheduling(dueFinishEpk, this.netReq);
  this.finishEpk:=Max{tk.finishEpk|tk∈ this.task};
ENDIF
RETURN
```

void Job.forwardScheduling(now Epoch)

```
// Execute scheduling of a job forwardly from the present time or from the due start epoch of the job.
IF this.part.operation=Ø
  this.startEpk:=now;
  this.finishEpk:=this.startEpk+this.part.procureLeadTime
ELSE
  Epoch dueStartEpk:=this.getDueStartEpk();
```

```
  FOR opr ∈ this.part.operation ASCENDING BY opr.oprNo
    Time reqTime := opr.setupTime + opr.processTime × this.netReq;
    Resource res := opr.processWkCtr.chooseResource ( dueStartEpk, reqTime );
    WHILE reqTime > 0
      Task tsk := res.getNextFreeInterval( dueStartEpk);
      reqTime := reqTime – ( tsk.finishEpk – tsk.startEpk );
      IF reqTime < 0 THEN tsk.finishEpk := tsk.finishEpk + reqTime;
      tsk.job := this;
      tsk.operation := opr;
      this.task.add( tsk );
      tsk.shift.task.add( tsk );
      dueStartEpk := tsk.finishEpk;
    ENDWHILE
  ENDFOR
  this.finishEpk := dueStartEpk;
  this.startEpk := Min{ tk.startEpk | tk ∈ this.task };
ENDIF
RETURN
```

void UnreleasedJob.cancelPlanning ()

```
// Canceling the result of job planning
this.netReq := 0;
DESTROY this.wipAssign;
FOR sjl ∈ this.supJobLink
  sjl.supQty := 0;
RETURN
```

void UnreleasedJob.cancelScheduling ()

```
// Canceling the result of job scheduling, which is either backward scheduling or forward scheduling
this.startEpk := Null;
this.finishEpk := Null;
this.meetBySupply := Null;
DESTROY this.task;
RETURN
```

B.3: Methods of Demand Class

void Demand.buildJobLinks ( )

```
// This method constructs a request-supply relationship between jobs of a demand.
Job jb := NEW Job ( part = this.part );
jb.meetDemand.add( this );
this.supJob := jb;
Job LSet := { jb }, TSet := Ø;
WHILE LSet ≠ Ø
  Job rj := LSet.get( );
  Job k := TSet.search ( part = rj.part );
  IF k = Null
```

```
    TSet.add(j);
    FOR spl∈rj.part.supPartLink
      Job sj:=NEW Job(part=spl.supPart);
      LSet.add(sj);
      JobLink jl:=NEW JobLink(reqJob=rj, supJob=sj);
      rj.supJobLink.add(jl);
      sj.reqJobLink.add(jl);
    ENDFOR
  ELSE
    FOR rjl∈j.reqJobLink
      rjl.supJob:=k;
    DESTROY rj;
  ENDIF
ENDWHILE
RETURN;
```

void Demand.planningScheduling (now: Epoch)

```
// Planning and scheduling of a demand to generate a feasible plan for the demand
IF this.supJob≠Null
  this.cancelPlanningScheduling(now)
ELSE
  this.buildJobLinks();
Job PSet:={ this.supJob }, BSet:=Ø;
Boolean ret:=True;
WHILE PSet≠Ø OR BSet≠Ø
  THREAD
    UnreleasedJob pjb:=PSet.get();
    pjb.planning();
    IF pjb.netReq>0
      BSet.add(pjb);
      FOR sjl∈pjb.supJobLink
        PartLink pl:=PartLink.search(reqPart=pjb.part, supPart=sjl.supJob.part);
        sjl.supQty:=pl.rate×pjb.netReq;
      ENDFOR
    ENDIF
  ENDTHREAD
  THREAD
    UnreleasedJob sjb:=BSet.get();
    sjb.backwardScheduling();
    IF sjb.startEpk<now
      ret:=False;
    FOR sjl∈sjb.supJobLink
      IF sjl.supJob.ifAllReqJobScheduled()=True THEN PSet.add(sjl.supJob);
    ENDFOR
  ENDTHREAD
ENDWHILE
IF ret=False
  Job CSet:={ this.supJob }, FSet:=Ø;
  WHILE CSet≠Ø
```

```
      UnreleasedJob cjb := CSet.get( );
      cjb.cancelScheduling( );
      IF cjb.supJobLink = Ø
        FSet.add( cjb );
      ELSE
        FOR sjl ∈ cjb.supJobLink
          CSet.add( sjl.supJob );
      ENDIF
    ENDWHILE
    WHILE FSet ≠ Ø
      UnreleasedJob sjb := FSet.get( );
      sjb.forwardScheduling( now );
      FOR ALL rjl ∈ sjb.reqJobLink
        IF rjl.reqJob.ifAllSupJobScheduled( ) = True
          FSet.add( rjl.reqJob );
      ENDFOR
    ENDWHILE
ENDIF
RETURN
```

void Demand.cancelPlanningScheduling (now: Epoch)

```
// Canceling the result of planning and scheduling of a demand
Job CSet := { this.supJob };
WHILE CSet ≠ Ø
    UnreleasedJob cjb := CSet.get( );
    IF cjb.startEpk > now
      cjb.cancelScheduling( );
      cjb.cancelPlanning( );
    FOR sjl ∈ cjb.supJobLink
      sjl.supQty := 0;
      IF sjl.supJob.startEpk ≠ Null
        CSet.add( sjl.supJob );
    ENDFOR
ENDWHILE
RETURN
```

Appendix C

C.1: The Total Number of Legal Permutations on Requirements in *Rq*

The size of the set *SQ* depends on the size of the set *Rq*, and the shape of graph G_{rq} on *Rq*. An example of the graph is shown in Fig. 5.3. The graph is split into two branches b_1=⟨ rq_1, rq_2,..., rq_7 ⟩, b_2=⟨ rq_8, rq_9,..., rq_{14} ⟩, and both branches have two sub-branches b_{11}=⟨ rq_3, rq_4, rq_5 ⟩, b_{12}=⟨ rq_6, rq_7 ⟩, and b_{21}=⟨ rq_{10}, rq_{11}, rq_{12} ⟩, b_{22}=⟨ rq_{13}, rq_{14} ⟩, respectively.

The requirements in a branch is regulated by precedence constraints, but no such constraint exists among branches of the same level, e.g. b_{11} and b_{12}. A permutation on elements of the lower-level branches determines a sequence of the higher-level branch.

Let $N=\langle rq_i\rangle_{i=1}^{n}$ and $M=\langle rq_i\rangle_{i=1}^{m}$ be two legal sequences of requirements. A sequence $V=\langle rq_i\rangle_{i=1}^{n+m}$ is a legal permutation on $\{rq_i\}_{i=1}^{m}\cup\{rq_i\}_{i=1}^{n}$ if $V-\{rq_i\}_{i=1}^{m}=N$ and $V-\{rq_i\}_{i=1}^{n}=M$, where '–' is a function removing the elements in a set from a sequence without changing order of the sequence. The various requirements, whose precedence relation within *M* and *N* is unchanged, can be viewed as the same requirements in permutation. Hence, the total number of permutations is $(m+n)!/m!n!$ or C_m^{m+n}.

For example, $C_e^{3+2}=10$ legal permutations of b_1 is determined by permutations on requirements in b_{11} and b_{12}. In a similar manner, b_2 also has 10 permutations. There are $C_7^{7+7}=3{,}432$ legal permutations for a permutation of b_1 and a permutation of b_2. Size of the set of legal sequences $|SQ|$ in Fig. 5.4a is thus 10×10×3432=343200.

C.2: The Number of Requirement Aggregations for a Sequence of Requirements

Assume there are *n* requirements with the same operation that are linked together somewhere in a sequence. The *n* requirements can be put into 1, 2,..., *m* ($m\le n$) baskets with each basket having (c_1, c_2, ..., c_m) requirements. For example, 6 requirements can be put into 3 baskets by ways of (4, 1, 1), (3, 2, 1), and (2, 2, 2). The number of alternatives to distribute *n* requirements into *m* baskets with each basket having (k_1, k_2, ..., k_m) requirements is $(C_{k_1}^{n}C_{k_2}^{n-k_1}C_{k_3}^{n-k_1-k_2}...C_{k_m}^{k_m})/(b_1!b_2!..b_v!)$, where b_i, i=1..*v*, are numbers of baskets whose number of requirements is equal and $\sum_{i=1}^{v}b_i=m$. For example, there are $(C_4^6C_1^2C_1^1)/(2!\times1!)=15$ ways to put 6 requirements into 3 baskets by way of (4, 1, 1). The numbers of ways for the other cases (3, 2, 1) and (2, 2, 2) are 10 and 15 respectively.

There are 7 groups of 2 requirements with the same operation linked together in the sequence sq_i shown in Fig. 5.4b. Each group has 2 ways of aggregation, i.e. either aggregate or not, hence $|QA_i| = 2^7 = 128$.

C.3: The Number of WIP Allocations

Let's first consider the problem of allocating q units of a WIP to n requirements with each requirement having c_i (i=1..n) units such that $\sum_{i=1}^{n} c_i = q$ and c_i is a natural number. This problem can be viewed as permutation of q units of WIP and n different requirements. Let 'o' represent a WIP, r_i (i=1..n) a requirement, and the permutation 'r_1 o o o r_5 r_3 o....' shows c_1=0, c_5=3, and c_3=0, i.e. the number of WIP before a requirement represents the allocated quantity. Since 'r_1 o o o r_5 r_3 o....' and 'r_5 o o o r_3 r_1 o....' represent the same set of WIP allocation, the precedence relation of requirements in a sequence must be fixed to avoid such duplication. Total number of permutations is thus $(q+n)!/q!n!$ or C_n^{q+n}.

Assume $|Wp|=u$, and (q_1, q_2,..., q_u) are the quantities of WIP allocated to number of requirements (n_1, n_2,..., n_u) in Rq, then there are $C_{n_1}^{q_1+n_1} \times C_{n_2}^{q_2+n_2} ... \times C_{n_u}^{q_u+n_u}$ ways of such allocation. The number of possible allocations for the case shown in Fig. 5.4d is $C_2^{8+2} = 45$.

List of Papers

Journal Papers

1. R. Sato, T. L. Tsai, An agile production planning and control with advance notification to change schedule, *International Journal of Production Research*, 2004, VOL. 42, No. 2, 321-336.

2. T. Tsai, R. Sato, A UML model of agile production planning and control system, *Computers in Industry*, 2004, VOL. 53, 133-152.

3. T. Tsai, R. Sato, and T. Terano, A genetic algorithm with MGG and demand crossover to solve dynamic flexible scheduling problem, *European Journal of Operational Research.* (Submitted)

Conference Papers

4. T. Tsai, Ryo Sato, A Formulation of the Iterative Process Prototyping Methodology, *Fifth International Conference Asia-Pacific Region of Decision Sciences Institute 2000 Proceedings*, 2000, Compact Disk.

5. 佐藤 亮, 蔡 東倫, MRP で計画管理するビジネスプロセスの動的特性解析について (On analysis of dynamic property of a business process with MRP), 経営情報学会 2000 年秋季全国研究発表大会予稿集, 2000, 98-101.

6. 佐藤 亮, 蔡 東倫, e-Business とサプライチェーンマネジメントの分析設計のための工学的な基礎概念と展開(Engineering concepts and development for the analysis and design of e-business and SCM), 計測自動制御学会システム・情報部門シンポジウム 2000 講演論文集, 2000, 217-222.

7. TungLun Tsai, Ryo Sato, The Comparison of Safety Lead Time and Safety Stock in MRP System (MRP における安全リードタイムと安全在庫の比較について), 経営情報学会 2001 年春季全国研究発表大会予稿集, 2001, 215-218.

8. 佐藤 亮, 蔡 東倫, 生産とロジスティクスの統合スケジューリング (Integrated scheduling for production and logistics), 経営情報学会 2001 年春季全国研究発表大会予稿集, 2001, 268-271.

9. 蔡 東倫, 佐藤 亮, A Simulator for Agile Production Planning and Control System (APPCS におけるシミュレータの提案), 経営情報学会 2002 年春季全国研究発表大会予稿集, 2002, 242-245.

10. T. Tsai, R. Sato, A UML model for agile production planning and control system, *Proceedings the 6th Pacific Asia Conference on Information Systems 2002 (PACIS2002)*, 2002, Compact Disk.

11. 佐藤 亮, 蔡 東倫, 二村 暢之, スケジューリングによる生産計画：タイムバケットとリアルタイム方式の比較 (Schedule based production planning: comparison of time bucket and real time planning), 経営情報学会 2003 年春季全国研究発表大会予稿集, 2003, 396-399.

12. 佐藤 亮, 蔡 東倫, 二村 暢之, 小野 栄一, 日程計画業務のための ERP を用いる情報システム方法論-quickIPP (Information Systems Methodology for Planning Activity with ERP - quickIPP), 経営情報学会 2003 年秋季全国研究発表大会予稿集, 2003, 174-177.

13. 蔡 東倫, 佐藤 亮, A Scheduler for Agile Production Planning and Control System (APPCS におけるスケジューラの提案), 経営情報学会 2003 年秋季全国研究発表大会予稿集, 2003, 178-181.

14. T. Tsai, R. Sato, An Agile Genetic Algorithm for Solving Job Shop Scheduling Problem, *Proceedings of the 8th Pacific Asia Conference on Information Systems 2004 (PACIS2004)*, 2004, Compact Disk.

15. 蔡 東倫, 佐藤 亮, Applying GA with MGG and Demand Crossover to the Optimization of Production Planning and Scheduling with WIP and Resource Flexibility (WIP と多能工を取り入れた生産スケジューリングを解く遺伝的アルゴリズム：デマンドクロスオーバと MGG を用いる GA), スケジューリング・シンポジュウム 2004 講演論文集, 2004, 111-116.

Discussion Papers

16. Ryo Sato, TungLun Tsai, An agile production planning and control with additional purchase orders, Institute of Policy and Planning Sciences Discussion Paper Series No. 982, University of Tsukuba, 2002, April.

17. T. Tsai, R. Sato, A UML Model of Agile Production Planning and Control System, Institute of Policy and Planning Sciences Discussion Paper Series No. 999, University of Tsukuba, 2002, July.

18. 佐藤 亮, 蔡 東倫, 二村 暢之, 小野 栄一, ERP を用いてビジネスプロセスを作り出すための情報システム方法論：quickIPP, Institute of Policy and Planning Sciences Discussion Paper Series No. 1017, University of Tsukuba, 2003.