A planning board generator. Part I: Problem instances and types

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computing Science

Memorandum COSOR 94-42

A Planning Board Generator
Part I: Problem Instances and Types

M. Wennink
M.W.P. Savelsbergh

Eindhoven, December 1994
The Netherlands

A Planning Board Generator
Part I: Problem Instances and Types

Marc Wennink
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands

Martin Savelsbergh
School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205
U.S.A.

Abstract

A planning board is a planning tool that uses the Gantt chart as its main representation mechanism. A planning board generator aims at facilitating the development of planning boards; using a description of the problem type for which a planning board is to be developed and the desired representations and manipulations, the generator automatically creates a prototype of the planning board. In this paper we discuss our view of a planning board generator and introduce a method to describe problem instances and types.

Keywords: planning board, planning board generator, problem type, problem instance, description method, Gantt chart, representation, manipulation.

1 Introduction

A planning board is a planning tool that uses a Gantt chart to represent a plan and provides the means to modify the plan by modifying the Gantt chart. A planning board is a useful tool in many problem situations. Unfortunately, developing an automated planning board requires a lot of time and energy. In addition, all problem situations have their own characteristics and require their own specific planning board.

Our goal is to identify common properties of problem situations for which a planning board is useful and to use these properties to develop a *planning board generator*. Given a description of the problem situation for which a planning board has to be developed and a specification of the desired representations and manipulations, the generator should automatically create an initial version of the planning board.

The purpose of this paper is twofold. First, we will discuss the main characteristics of a planning board generator. Second, we will present description methods for problem instances and types. Representations and manipulations will be discussed in a sequel paper.

In Section 2, we will elaborate on our view of a planning board generator. We will define the notion of a plan more precisely and use this notion to determine problem types for which planning boards are useful tools. In Section 3, we will introduce a method to describe instances of such problem types. This method serves as a basis for our method to describe problem types, which will be presented in Section 4. Examples of descriptions of problem instances and types are given in Appendices A and B respectively.

2 The Planning Board Generator

As said above, a planning board is a tool that uses a Gantt chart to represent a plan and provides the means to modify the plan by modifying the Gantt chart.

Although the Gantt chart is the primary representation, other representations, such as data tables and inventory graphs, can also be incorporated in an automated planning board. Various representations of instances and plans may provide a better understanding of the problem being solved. However, a planning board should also provide the means to manipulate these representations, so as to enable us to create and modify plans. The notion of manipulation should be interpreted broadly. For a machine scheduling problem, for example, manipulations could move a task from one machine to another as well as compute an optimal schedule for a machine given the tasks that are currently assigned to it. In the terminology introduced by Anthonisse et al. [2] with respect to interactive planning systems, manipulations cover the entire spectrum from *assistant* functions to *advisor* functions.

Together, representations and manipulations provide a powerful tool that is useful for a large class of problem types. However, each problem type requires its own sets of representations and manipulations. In fact, each problem type requires its own planning board. A planning board generator (PBG) aims at facilitating the development of planning boards.

2.1 The Planning Board

On a Gantt chart the horizontal axis represents some *time interval*, the vertical axis represents *resources*, and rectangles on the chart represent assignments of *processes* to resources and time intervals.

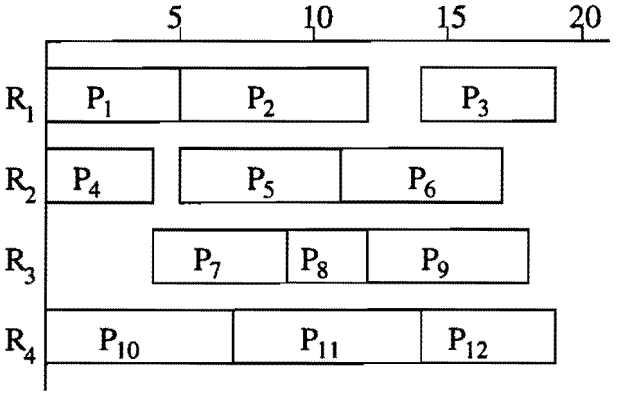

Figure 1: A Gantt chart

An example of a Gantt chart is given in Figure 1. There are twelve processes, P_1,...,P_{12}, and four resources R_1,...,R_4. Each process is assigned to some time interval and to one of the resources. For example, process P_6 is assigned to resource R_2 and the time interval [11,17].

A large part of this paper is concerned with identifying problem types for which a planning board is a useful tool, and with developing a method to describe these problem types. To be able to do so, we start with formalizing the notions of plan and planning board.

A *plan* is an assignment of processes to resources and time intervals.

The notions *resource* and *process* will be discussed in more detail in Section 3.3. For the time being, it is sufficient to relate the term process to the objects represented by the rectangles in a Gantt chart, and the term resource to the objects that are represented on the vertical axis of the chart. Note that the Gantt chart representation plays a role in the definition: we assign processes to resources, rather than resources to processes.

A *planning board* is a tool for developing plans, which uses the Gantt chart as a representation mechanism.

A single Gantt chart cannot represent all aspects of real-life problems, because of the multi-dimensional character of such problems. A planning board should therefore be able to provide several Gantt chart representations, with different resources along the vertical axes, as well as other representation mechanisms, like data tables and inventory graphs. In order to develop a plan, the user of a planning board must be offered the possibility to perform several kinds of manipulations.

2.2 The Planning Board Generator

We aim at the following situation. A *planner* has to develop plans for a number of instances of the same problem type. He thinks that a planning board may be a helpful tool. He therefore asks a planning board *designer* to build a planning board for that specific problem type. The designer asks for the characteristics of the problem type, the desired representations, and the required manipulations. The designer activates the planning board *generator* to process this information and to produce an initial version of the planning board.

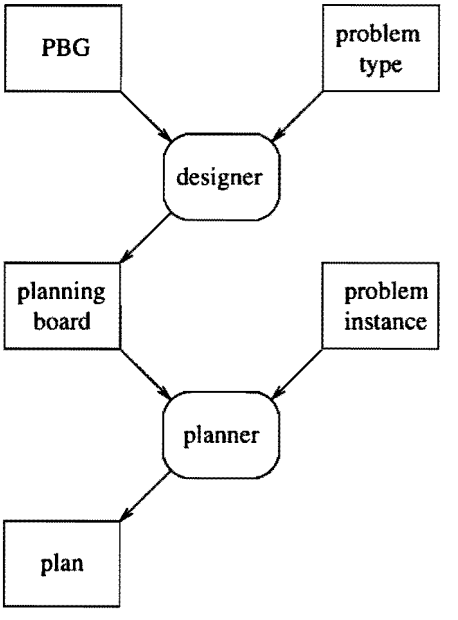

Figure 2: Functionality of the planning board generator

This situation is depicted in Figure 2. The designer uses the PBG to construct a planning

board for a specific problem type. The planner uses this planning board to develop plans for various problem instances.

A planning board can handle all instances of a certain problem type. A PBG can construct planning boards for a set of problem types. This set will be called the *general problem class.* In the next sections, we discuss the description of problem instances and problem types within the general problem class.

2.3 Related Research

The role of decision support systems and, more specifically, graphical interfaces in solving planning problems has been widely discussed in the literature (see e.g., Anthonisse et al. [2], Fisher [4], Hurrion [7]). Furthermore, planning boards have been developed for a wide range of problem situations including course scheduling, ship scheduling, timetabling, job-shop scheduling, and production scheduling (e.g., Fisher et al. [5], Moreira and Oliveira [12], Viviers [13]). In most cases these planning boards are equipped to handle a small and well-specified class of problem instances. More relevant and more closely related to our research are systems that have been developed in order to deal with a broader class of planning situations and offer wider possibilities for problem description. Examples of such systems are the systems developed by Jackson et al. [8], Jones and Maxwell [9], and Woerlee [14]. These systems differ from our PBG in that they are developed for a particular application area; material logistics, manufacturing scheduling, and production scheduling respectively. With our PBG, we do not restrict ourselves to a particular application area, but we focus on the underlying general problem structure: processes must be assigned to resources over time. This general structure allows the use of the same representation mechanism (e.g., Gantt charts) and similar manipulations for problems in completely different application areas. Therefore, we think that this general problem structure can be exploited effectively to facilitate the development and implementation of planning boards for these application areas.

Although it is not concerned with planning tools that use the Gantt chart as main representation mechanism, the graph-based modeling system (GBMS) of Jones [10] is conceptually very similar to our PBG. Both the GBMS and the PBG aim at facilitating the development of user interfaces for interactive systems. In a GBMS, attributed graphs are used as representation mechanism. The created interface is called an *instance editor.* An instance editor enables a modeler to add and delete nodes and edges in order to construct a graph of the appropriate type. In a GBMS, the *designer* specifies a graph type and the manipulations that can be performed on graphs of that type. This specification is sufficient to automatically generate an instance editor for graphs of that type. Using a similar terminology, one can say that a planning board enables a planner to perform manipulations on (representations of) plans in order to construct a solution to a problem of the appropriate type. In our setting, the designer specifies a problem type and the desired representations and manipulations. From this specification, the PBG automatically constructs a planning board for instances of this type.

An important difference between a GBMS and a PBG is the representation mechanism that is used, attributed graphs versus Gantt charts. Another important difference relates to the way in which a graph type or problem type and the corresponding sets of manipulations are described.

Jones's GBMS allows the designer to create his own types of nodes and edges, each with its own set of attributes. Also the manipulations can be developed by the designer. Although

this bottom-up approach gives the designer a lot of freedom, it also requires knowledge of the relatively unfamiliar theory of graph-grammars.

For our PBG, in contrast, we apply a top-down approach. There exists a *general problem class*, of which all problem types that the PBG can handle are special cases. The designer of a planning board can describe his particular problem type by specifying the appropriate subclass of the general problem class. Similarly, a set of possible manipulations is given, from which a planning board designer chooses a subset. The designer in a GBMS, on the other hand, must specify the desired node types and edge types, as well as the desired manipulations, from scratch.

The advantage of this top-down approach is that the implementation of the PBG can be specifically equipped to deal with the allowed problem types only, thus yielding a more effective and efficient system. Furthermore, it has enabled us to develop a description method that exploits the common structural properties of the problem types, in contrast to the use of graph grammars, which are more generally applicable.

A disadvantage is that we cannot obtain full generality. Our PBG is restricted to the general problem class. However, we have tried to keep this class as general as possible indeed, only leaving problem types for which planning boards are not useful out of consideration.

This paper focuses on a description method for problem instances and a description method for problem types. Therefore, it is interesting to compare its capabilities with those of existing classification schemes, for example for machine scheduling problems (e.g. French [6], and Lawler et al. [11]). These classification schemes typically use a fixed number of parameters, each describing a particular problem characteristic. Since such a parameter can only specify a characteristic of all the instances of the problem type, these classification schemes do not offer the flexibility that we need in a description method that is to be used in a PBG that can handle a wide range of application areas. Consider a problem type in which there are two kinds of processes, the first kind possessing deadlines but no release times, the second kind possessing release times but no deadlines. The existing classification schemes only allow us to specify that all processes may possess release times and deadlines. The description methods presented in this paper do offer the possibility to describe such a problem type correctly. It is worth noting that the translation of a problem type description in terms of one of the existing classification schemes into a description using our method can usually be done quite easily.

3 Problem Instances

The designer of a planning board has to provide the PBG with information on the problem type, the desired representations, and the desired manipulations. The user of a planning board has to provide the planning board with a specific problem instance. In this section, we introduce a description method for problem instances. In Section 4, it will serve as a basis for our description method for problem types. Examples of instances described with our method are given in Appendix A.

3.1 An Overview of the Description Method

The instance description method has been developed with the following objectives in mind:

1. It should be possible to describe instances of all problems for which we feel that a planning board provides a useful tool. In fact, the method will implicitly define a general problem class. Note that this class should be fairly large for a PBG to be of any interest.

2. The instance description method must form the basis of a type description method. It should therefore be possible to specify subclasses of the general problem class by identifying the typical properties of problem instances within such subclasses.

3. It should be possible to use the description of an instance to efficiently perform some of the tasks of a planning board, such as verifying feasibility of assignments and providing information on various aspects of a plan.

We have tried to achieve these objectives by using a widely applicable and well-studied paradigm, namely attributed graphs, and by stressing the common structural properties of the problems.

Attributed graphs are widely used as a mechanism for describing many kinds of models in different fields of research, from sociology to chemistry. It is therefore not only a mechanism that many people are familiar with, but also one that has proven to be very flexible. Furthermore, there exists a massive theory on problems related to graphs, which we can use.

All problems for which a planning board is useful deal with assignments of processes to resources and time intervals. Therefore it must be possible to describe the characteristics of the processes and the resources that occur in the instance as well as the time system. Furthermore, it must be possible to describe relations between processes, between resources, and between processes and resources. The most important relation that has to be described is which assignments of processes to resources are feasible and which are not.

The attributed graph associated with an instance will be called an *instance graph*. The instance graph must be able to represent a variety of situations, e.g., a process that can be performed on any resource out of a set of resources, a process that requires a specific combination of resources, and a process that can be performed on any combination of resources out of a set of possible combinations of resources.

The core structure of the instance graph describes the feasible assignments of processes to resources. Each process and each resource are represented by a node. For each possible choice of resources and each possible combination of resources an auxiliary node is introduced that is connected to the objects involved and thus forms a $K_{1,n}$ where n is the number of objects involved.

Several other relations between objects can be described by graph constructs as well. Binary relations, i.e., relations involving exactly two objects, can be modeled by an edge or an arc. For example a precedence relation indicating that one process has to be performed before an other one. n-Ary relations, i.e., relations involving a set of n objects ($n > 2$), can be modeled by a $K_{1,n}$.

Not all aspects of problem instances can be represented in terms of nodes, arcs, and edges. These aspects can be divided into two categories: properties of individual objects, and properties of assignments. Examples of the first category are the sizes of the processes, the capacities of the resources, and the length of the planning period. These properties will be described as attributes of the corresponding nodes, or as attribute of the time system. An example of the second category is the processing time of a process when it is assigned to a particular combination of resources. In order to describe these kinds of properties, attributes of appropriate auxiliary nodes are introduced.

Before we discuss the various elements of the instance graph, we will spend some time on the key elements of plans: time, processes, and resources.

3.2 Time

Time plays an important role in planning boards. The way in which time occurs is different for different problems. For a timetabling problem for schools, the planning period may cover five days consisting of eight hours, with an hour being the smallest time unit, whereas for a machine scheduling problem, the planning period may be one working day of 12 hours, with a minute being the smallest time unit. For flexibility purposes we allow the introduction of a time system for each problem instance. This time system consists of time units on different levels. The smallest time unit is described in the first level, the second smallest time unit in the second level, etc. For the jth time unit also the conversion factor with respect to the (j-1)st time unit is given. For example, a planning period covering one week, with one second being the smallest time unit, is described in the following way.

number of levels: 5

level 1
unit name: second
level 2
unit name: minute
conversion factor: 60
level 3
unit name: hour
conversion factor: 60
level 4
unit name: day
conversion factor: 24
level 5
unit name: week
conversion factor: 7

planning period: 1 week

3.3 Resources and Processes

3.3.1 Resources and their attributes

Depending on the application, a resource can be almost anything. In a production planning application, personnel, money, raw materials, and machines may all be resources. In a time-tabling application for a school, the resources may be classrooms and teachers.

An important concept related to resources is that of a *capability*. A resource may possess several capabilities, i.e., it may be able to perform various tasks. A mechanic may be qualified to change oil and to repair brakes. A teacher may be qualified to teach mathematics as well as physics.

Although a resource may be able to perform different tasks, some set-up or change-over may be necessary before a particular task can be performed. Such a set-up brings the resource in the required *mode*. The corresponding set-up time can be either sequence dependent or sequence independent. In the first case, the set-up time is completely determined by the required mode. In the second case, the set-up time depends on the current mode of the resource and the mode that is required. The concept of mode is different from the concept of capability. A sawing machine may possess only one capability, the capability *sawing*, but several modes, one for each possible size of an object that can be sawn.

Most resources are not free commodities. Their utilization by some process will affect the possibilities for utilization by other processes. Quantities related to the utilization of a resource are *usage* and *consumption*.

The *usage* of resource R by process P at time t is a quantity that indicates to what extent R is occupied by P. The *total usage* of R at time t, i.e. the sum of the usage of all processes assigned to R at time t, is limited by the *capacity* of R at time t.

The *consumption* of resource R by process P during a time interval $[t_1, t_2]$ is a quantity that indicates to what extent R is consumed by P during that interval. The *total consumption* of R up to time t^*, i.e. the sum of the consumption of R up to t^* over all processes assigned to R, is limited by the *supply* of R up to t^*.

Based on the different kinds of utilization the following distinction between resources can be made (see also [3]):

- *Renewable resources*: Resources for which only their total usage at every moment is constrained. An example of such a resource is a painting machine, which can paint all day but no more than one object at the same time.
- *Nonrenewable resources*: Resources for which only their total consumption up to any given moment is constrained. An example of such a resource is finances.
- *Doubly constrained resources*: Resources for which both total usage and total consumption are constrained. An example is personnel. An employee can perform only a limited number of tasks at the same time and can perform these tasks only during a limited number of man hours.

Usage and consumption can both occur in either continuous or discrete quantities. A painting machine may be able to paint up to ten objects at the same time. Its capacity then is ten units, and the assignment of a painting job to this machine implies a usage of one unit. It is impossible to paint half objects, and therefore usage, in this case, is a discrete quantity. The storage capacity of a truck, in contrast, can be used in continuous quantities. With

respect to consumption, fuel will be consumed in continuous quantities, but in assembling a car steering wheels will be consumed in discrete quantities.

Mostly, usage and consumption will be modeled as one-dimensional quantities. Multi-dimensionality, however, is possible. For an employee there may exist a limit on the number of hours per day he may work (e.g. 10 hours) as well as a limit on the number of hours per week (e.g. 40 hours). This cannot be modeled by supplies that replenish the 'hours-inventory' to 10 at the beginning of each day, because the week-limit may then be exceeded. By modeling consumption and supply as two-dimensional quantities we can solve this problem. Another example of two-dimensional usage is found in transportation problems, where the usage of a truck is limited with respect to volume as well as weight.

The capacity and supply of a resource may change during the planning period as a result of instance-dependent and plan-dependent factors. For example, during a certain period the capacity of some machine may be smaller than normal because of maintenance, or the supply of some half-product will increase at the moment that a process that represents the production of that half-product is completed. Instance-dependent changes in capacity and supply will be modeled in terms of attributes of the resources. Plan-dependent changes in the supply of a resource will be modeled as (possibly negative) consumption of that resource. Similarly, the available capacity of a resource is reduced when it is used by a process, and it is increased again as soon as that process is completed.

A resource that possesses several capabilities may have different performance levels with respect to these different capabilities. The aforementioned mechanic may be very fast in changing oil, but he may be very slow when it comes to repairing brakes. In order to handle such differences, resource characteristics related to speed, usage, and consumption have to be specified for each of its capabilities.

A resource node has the following attributes:

name: A name that uniquely identifies the resource.

availability periods: A set of time intervals specifying the time periods during which the resource is available.

category: An indicator that specifies the type of resource, i.e., *Renewable, Nonrenewable*, or *Doubly Constrained.*

number of modes: A natural number. For each mode the following must be specified:

- **name:** A name that uniquely identifies the mode.
- **set-up time:** A function of the status of the resource, i.e., the active mode, that computes the set-up time that is needed to bring the resource into this mode.

usage: This attribute consists of three sub-attributes.

- **divisibility:** An indicator that specifies whether usage of the considered resource is modeled as a continuous or as a discrete quantity. In case usage is continuous the value is 0 and in case usage is discrete the value is the discretization unit.
- **dimension:** A natural number representing the dimension of usage of the considered resource.
- **capacity:** A function of time, describing the plan-independent capacity.

consumption: This attribute consists of three sub-attributes.

- **divisibility**: An indicator that specifies whether consumption of the considered resource is modeled as a continuous or as a discrete quantity. If consumption is continuous the value is 0 and if consumption is discrete the value is the discretization unit.
- **dimension**: A natural number representing the dimension of consumption.
- **supply**: A function of time, describing the plan-independent supply.

number of capabilities: A natural number. For each capability the following must be specified:

- **name**: A name that uniquely identifies the capability. If different resources possess the same capability, the same name must be used when referring to this capability.
- **speed**: A real number that is used to determine the duration of a process when that process is assigned to this resource. It is not necessarily equal to the actual speed of the resource, as will be illustrated in Section 3.4.4.
- **usage factor**: A real number that is used to determine the usage of the resource when a process is assigned to this resource. If this resource is used in combination with other resources, then this attribute is also used to determine the usage of those resources. In Section 3.4.4 we will discuss the use of the usage factor extensively.
- **consumption factor**: A real number that is used to determine the consumption of the resource when a process is assigned to this resource. If this resource is used in combination with other resources, then this attribute is also used to determine the consumption of those resources. In Section 3.4.4 we will discuss the use of the consumption factor extensively.

The **usage** attribute is only specified for renewable and doubly constrained resources, the **consumption** attribute only for non-renewable and doubly constrained resources. These attributes represent plan-independent properties of a resource. Usage and consumption only occur during the time intervals in which processes are assigned to that resource, and the used and consumed amounts will be different for different processes. It is therefore not possible to model usage and consumption in terms of resource attributes. Furthermore, actual usage and consumption may occur only during part of the total processing time. For example, a production run may require a machine for the entire period, whereas it may require an employee to start the machine only for the first five minutes of the period. Similarly, the entire required amount of raw materials may be consumed at the beginning of the process, whereas fuel may be consumed at a constant rate during the processing period. The specification of these usage and consumption patterns will be discussed in Section 3.4.4.

3.3.2 Processes and their attributes

A process can be anything that must be assigned to a set of resources and a time interval in order to obtain a feasible plan. In a timetabling problem for a school, the processes may be the lessons that must be assigned to teachers, class-rooms, and time intervals. In a machine scheduling problem, the processes are the tasks that must be assigned to machines and time intervals.

We distinguish two kinds of processes: *repetitive* and *non-repetitive* processes. A non-repetitive process is performed only once. A repetitive process can be performed an arbitrary

number of times; the number of repetitions is determined by the planner by specification of the processing interval.

A process node has the following attributes:

name: A name that uniquely identifies the process.

type: An indicator that specifies the type of process, i.e., *repetitive* or *non-repetitive.*

mode: An indicator that specifies the mode in which the resource to which the process is asigned has to be.

size: A real number that is used to determine the duration of the process in case of a non-repetitive process, or to determine the length of one repetition in case of a repetitive process.

usage intensity: A vector that is used to determine the usage of the resource(s) that the process is assigned to.

consumption intensity: A vector that is used to determine the consumption of the resource(s) that the process is assigned to.

release time: A point in time before which the process cannot be performed.

deadline: A point in time after which the process cannot be performed.

due date: A point in time by which the process preferably should be completed.

split: An indicator that specifies whether the process may be preempted or not. It has value 0 if the process, once started, cannot be interrupted, 1 if the process may be temporarily stopped and later resumed using the same resources, and 2 if the process can also be resumed using other resources.

3.4 Describing Feasible Assignments

In this section, we discuss *requirement relations.* Requirement relations are relations between processes and resources indicating which resources can be used to perform a process. We take an incremental approach in describing requirement relations. We start with simple requirements, i.e., processes requiring a single capability, and continue with more complicated requirements, i.e., processes requiring combinations of capabilities and processes requiring one of several combinations of capabilities. At the end of this section we will discuss how aspects related to feasible assignments such as duration, set-up times, usage, and consumption are modeled.

3.4.1 Processes requiring a single capability

A process that requires a specific capability can be assigned to any resource that possesses that capability. In fact, a choice has to be made between all resources that possess the required capability. Such a choice is modeled by a $K_{1,n}$, in which the auxiliary node, the capability node, is connected to all nodes representing resources that possess the considered capability. For each capability that is required by some process, a capability node is introduced, and the associated $K_{1,n}$ is formed. The requirement relation is modeled by an edge connecting the

process node and the capability node that represents the required capability.

A capability node has only one attribute:

name: The name uniquely identifies the capability. It must be the same as the name, given as a resource attribute, of the capability it represents.

An example is given in Figure 3. Processes P_1 and P_2 require capability C_1, which is possessed by resources R_1 and R_2. Process P_3 requires capability C_2, which is possessed by resources R_2 and R_3.

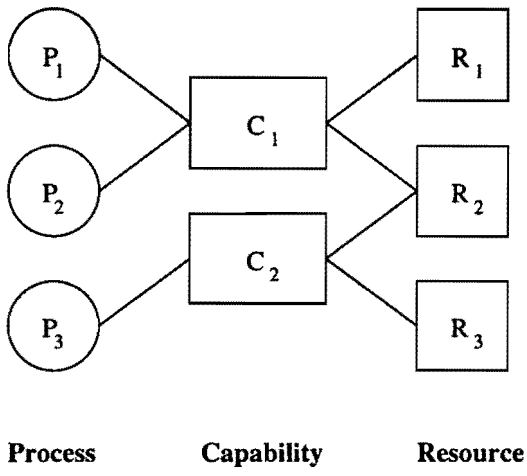

Figure 3: Processes requiring a single capability

3.4.2 Processes requiring a combination of capabilities

In many problems, performing a process requires the use of a combination of resources, or, better, a combination of capabilities. Both paint and a painter are needed to paint an object. Lessons cannot be taught unless a teacher and a classroom are available. Such a combination of capabilities is again modeled by a $K_{1,n}$. The auxiliary node, the *capability set* node, is connected to all capability nodes representing the required capabilities. If a process requires a particular combination of capabilities, this is modeled by introducing an edge connecting the process node and the corresponding capability set node. The process can be performed by any combination of resources that possesses all capabilities in the capability set.

Note that, although any combination of capabilities may be considered a capability set, only those combinations that are actually required by the processes are of interest.

A capability set node has several attributes. However, most of them will be introduced when we discuss duration, consumption, and usage in more detail. Here, we only introduce one attribute:

name: A name that uniquely identifies the capability set.

An example of requirement relations dealing with capability sets is given in Figure 4. Processes P_1 and P_2 both require a resource possessing the capability C_1 *and* a resource possessing capability C_3. The resource combination $\{R_1, R_4\}$ would satisfy these requirements.

Process P_3 requires a resource with the capability C_2 and a resource with the capability C_3. The combination $\{R_3, R_4\}$ is feasible. The combination $\{R_2, R_4\}$ is feasible for all three processes.

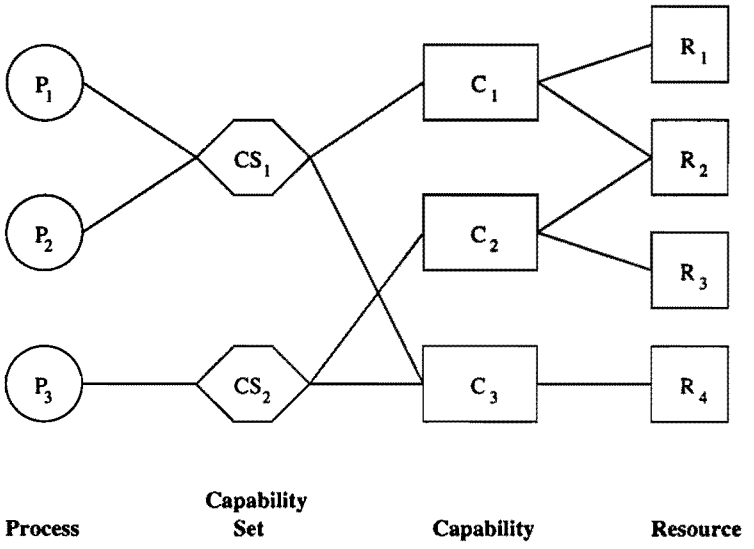

Figure 4: Processes requiring a combination of capabilities

3.4.3 Processes requiring one of several combinations of capabilities

Sometimes, a process does not require a specific combination of capabilities, but one of several alternative capability sets. In such a case, we say that the process requires a particular *function*. This is modeled by a $K_{1,n}$, in which the auxiliary node, the function node, is connected to all capability set nodes between which a choice can be made. If a process requires a function, a choice between the associated capability sets is to be made. The process then can be performed by any set of resources that possesses all capabilities in the chosen capability set.

Functions may be useful for production problems in which different production modes occur, which imply the possible use of totally different resources, e.g. production by hand or by machine.

A function node has only one attribute:

name: A name that uniquely identifies the function.

An example of requirement relations with functions is given in Figure 5. Processes P_1 and P_2 both require function F_1, implying that they must be performed by resources possessing the capabilities in capability set CS_1 *or* by resources possessing the capabilities in CS_2.

In summary, three different types of $K_{1,n}$'s may occur in the subgraph representing the requirement relations. The auxiliary nodes are the capability nodes, the capability set nodes, and the function nodes, respectively. The first $K_{1,n}$ represents a *choice* relation between resources possessing the same capability. The second states that a particular *combination* of capabilities is required. The third again represents a possibility for *choice*, now between

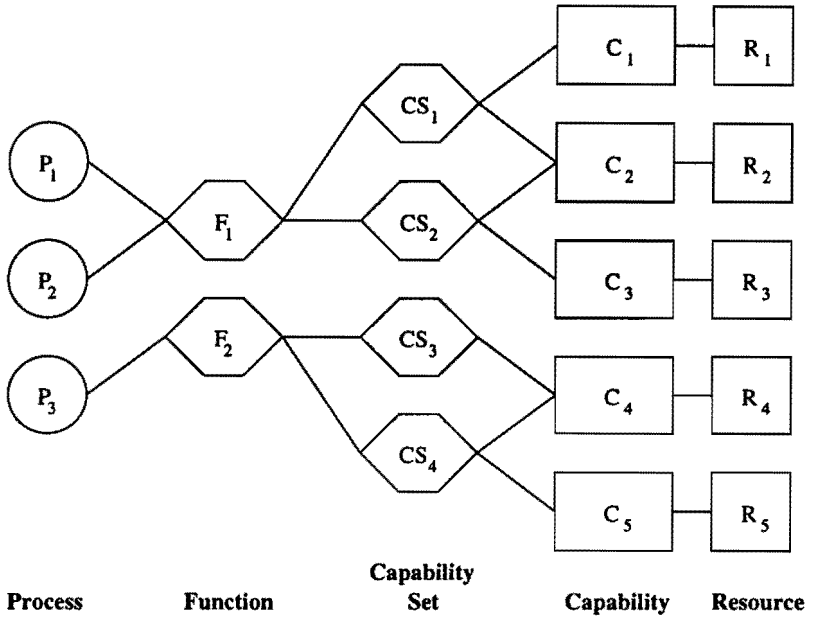

Figure 5: Processes requiring one of several combinations of capabilities

several capability sets. Each process node is connected, via a *requirement edge*, to either a capability node, or a capability set node, or a function node.

3.4.4 Duration, set-up times, usage, consumption

The subgraph representing the requirement relations enables us to identify for each process the resources to which it can be assigned. Given such an assignment, we want to determine the following (assuming that the considered process is non-repetitive):

- The *set-up times* required to bring the resources in the correct modes.
- The *duration* of the process: the time to complete the process. Together with the starting time of the process, the duration determines the *processing interval.*
- The *usage pattern* of each resource that is used: the time interval within the processing interval during which the process occupies the resource and the amount of the resource that is occupied by the process during this time interval.
- The *consumption pattern* of each resource that is consumed: the time interval within the processing interval during which the process consumes the resource and the amount of the resource that is consumed by the process during this time interval. We will assume that consumption takes place at a constant rate. Instantaneous consumption can be modeled by specifying an infinitesimal time interval.

In case of repetitive processes, a set-up is only performed at the beginning of the first repetition, and the duration corresponds to the length of one repetition. We will assume that the usage and consumption patterns will be the same for every repetition. If, for example,

at the beginning of a repetition a certain amount of raw materials is consumed, the same amount will be consumed at the beginning of all other repetitions.

The set-ups are much easier to deal with than duration, usage and consumption, because the set-up time of a particular resource does not depend on the other resources that are used. Given an assignment of a process to a combination of resources, for each of those resources the set-up time can be computed using the **mode** attribute of the process and the **set-up** attribute of the considered resource. The duration of the process, as well as the usage and the consumption of a specific resource, may depend on all the resources that occur in the assignment. For example, when a process requires a machine and fuel, the consumption of fuel will not only depend on the process but also on the machine that is actually used.

When a process requires a single capability, duration, consumption, and usage can easily be determined. The duration of the process is computed as the quotient of the **size** attribute of that process and the **speed** attribute of the resource it is assigned to. The interval during which any consumption or usage occurs is equal to the processing interval. The consumed amount is computed as the product of the **consumption intensity** attribute of the process and the **consumption factor** attribute of the resource, and the used amount is computed as the product of the **usage intensity** and the **usage factor** attributes.

The situation is more complicated when a process requires a combination of capabilities. In order to deal with these complications, we introduce several attributes for the capability set nodes. The first deals with the duration of the process.

duration: A function of the size attribute of the process and the speed attributes of the resources that computes the duration, i.e.,

$$d(s_0, s_1, \ldots, s_n),$$

where s_0 is the value of the size attribute of the process, n is the number of capabilities in the capability set, and s_i $(i = 1, .., n)$ is the value of the speed attribute of the resource possessing capability i.

Suppose the capability set consists of the capabilities *machine, employee*, and *material.* Let s_0 be the size attribute of the process, and let s_1, s_2, and s_3 be the speed attributes of the machine, the employee, and the raw material, respectively. If the speed is completely determined by the speed of the machine, the following duration function is used:

$$d(s_0, s_1, s_2, s_3) = s_0/s_1.$$

If the ability of the employee to work with that machine plays a role, the duration function may be something like

$$d(s_0, s_1, s_2, s_3) = s_0/v_{s_1,s_2},$$

where $V = \{v_{ij}\}$ is some predefined matrix, v_{ij} being the speed at which a process is performed when the value of the speed attribute of the used machine is i and the value of the speed attribute of the used employee is j. Note that in this case the speed attributes do not reflect the actual speed of the resources, but are merely indices used to extract the correct values from a matrix.

For each capability in the capability set, the following two attributes are specified in order to describe the usage pattern.

usage interval: A function of the starting time and the duration of the process that computes the time interval during which usage of the resource possessing the considered capability takes place, i.e.,

$$[t_1, t_2] = t_u(start, duration).$$

usage volume: A function of the usage factor attribute of the process and the usage intensity attributes of the resources that computes the volume used, i.e.,

$$v_u(i^u, f_1^u, \ldots, f_n^u),$$

where i^u is the value of the usage intensity attribute of the process, n is the number of capabilities in the capability set, and f_i^u $(i = 1, .., n)$ is the value of the usage factor attribute of the resource possessing capability i.

Let us again consider the example with the capability set consisting of the capabilities *machine, employee*, and *material*. Some machines require an employee during the first five minutes for starting it up, whereas others do not. This can be modeled by assigning a value of 1 to the usage factor attributes of the machines that do require an employee ($f_1^u = 1$), and a value of 0 to those that do not ($f_1^u = 0$), and by applying the following usage pattern:

$$t_u(start, duration) = [start, start + 5],$$

$$v_u(i^u, f_1^u, f_2^u, f_3^u) = f_1^u.$$

For each capability in the capability set, the following two attributes are specified in order to describe the consumption pattern.

consumption interval: A function of the starting time and the duration of the process that computes the time interval during which consumption of the resource possessing the considered capability takes place, i.e.,

$$[t_1, t_2] = t_c(start, duration),$$

where *start* is the starting time of the process, and *duration* is the value computed by the function found in the **duration** attribute described above.

consumption volume: A function of the consumption factor attribute of the process and the consumption intensity attributes of the resources that computes the volume consumed, i.e.,

$$v_c(i^c, f_1^c, \ldots, f_n^c),$$

where i^c is the value of the consumption intensity attribute of the process, n is the number of capabilities in the capability set, and f_i^c $(i = 1, .., n)$ is the value of the consumption factor attribute of the resource possessing capability i.

Suppose the capability set again consists of the capabilities *machine, employee*, and *material*, and let the *material* capability be the only one that is possessed by non-renewable

resources. If a process consumes a fixed amount of the material at a constant rate during the entire interval, the consumption pattern for the *material* capability is as follows:

$$t_c(start, duration) = [start, start + duration],$$

$$v_c(i^c, f_1^c, f_2^c, f_3^c) = i^c,$$

where the value of the consumption intensity attribute (i^c) equals the fixed consumed amount.

The attributes of the capability sets allow us to describe complex consumption and usage patterns for a large variety of problems. Obviously, assigning the correct values to the different process and resource attributes is important. Sometimes it may be necessary to assign to, for example, the speed attribute of a resource, a value that has little to do with the actual speed of that resource.

Example 3.1 We consider the problem of making coffee, more specifically the capability set *MakeCoffee* consisting of the capabilities *CoffeeMachine, Filter, GroundBeans, Water,* and *Coffee.*

There are two resources that possess the capability *CoffeeMachine.* Machine A can make one liter of coffee in 15 minutes, machine B does it in 10 minutes. Making one liter of coffee requires 0.10 units of *GroundBeans*, consumed at the beginning of the processing interval, and 1 liter of water, consumed gradually during the entire processing interval. Independently of the amount of coffee to be made, one filter is required, consumed at the start.

If we want to perform the process *0.8Coffee*, making 0.8 liter of coffee, we model this by assigning the value 0.8 to the consumption intensity attribute as well as to the size attribute of the process.

Furthermore, we assign the values 4/60 and 6/60 to the speed attributes of resource A and B respectively, and use one minute as time unit.

The attributes of the capability set *MakeCoffee* then are:

duration: *size / CoffeeMachine.speed*

(CoffeeMachine)

usage interval: $[start, start + duration]$

usage volume: 1

(Filter)

consumption interval: $[start, start]$

consumption volume: 1

(GroundBeans)

consumption interval: $[start, start]$

consumption volume: *consumption intensity* $*0.10$

(Water)

consumption interval: $[start, start + duration]$

consumption volume: *consumption intensity*

(Coffee)

consumption interval: $[start, start + duration]$

consumption volume: - *consumption intensity*

□

3.5 Other Relations Between Objects

Specifying the requirement structure, i.e., specifying for each process the combinations of resources it can be assigned to, is only part of the description of a problem instance. Usually various other relations exist between processes and resources. Many of these can easily be described in the instance graph.

3.5.1 Precedence relations

A precedence relation indicates that the set of allowed start and completion times of some process depends on the start or completion time of some other process. Precedence relations are binary relations that can be represented by arcs between process nodes in the instance graph. We distinguish four types:

- *finish-to-start:* Process A must be completed before process B is started;
- *start-to-start:* Process A must be started before process B is started;
- *start-to-finish:* Process A must be started before process B is completed;
- *finish-to-finish:* Process A must be completed before process B is completed.

A *time-lag* may be associated with each of the four types of precedence relations. For example, a finish-to-start relation can be stated as 'process A must be completed at least 10 minutes before process B starts.'

The attributes of precedence arcs are:

type: Either finish-to-start, or start-to-start, or start-to-finish, or finish-to-finish.

time lag: The minimum and maximum allowed time lag.

3.5.2 Common resource relations

A set of processes may be related because they have to be processed on the same set of resources. Such an n-ary relation is described as a $K_{1,n}$; the auxiliary node is the common resource set node.

The attribute of the common resource set node is:

name: A name that uniquely identifies the common resource set.

In case of a common resource relation between several processes, there is no need for individual requirement edges. Instead the requirement edge will connect the common resource set node to a capability Node, a capability set node, or a function node. In this light, the common resource relation can be seen as an *obligation* relation, in contrast to the *choices* represented by functions and capabilities. As soon as one of the processes in a common resource set is assigned to a particular resource combination, all other processes in that set have to be assigned to the same resource combination.

3.5.3 Process group relations

A set of processes may be related for some other reason than common resources. Such an n-ary relation is represented by a $K_{1,n}$; the auxiliary node is the process group node. For instance, the notion of a job in machine scheduling problems can be modeled as a process group.

The attributes of the process group node are:

name: A name that uniquely identifies the process group.

release time: No process in the process group can be performed before its release time.

deadline: No process in the process group may be completed after its deadline.

due date: All processes should preferably be completed by their due date.

exclusion: If this attribute has the value *true*, then no two processes in the process group may be performed at the same time.

Note that release time, deadline, and due date can be specified for all processes in a process group separately, but even when the **exclusion** attribute has the value *false*, the introduction of a process group can be useful for emphasizing characteristic structures in problem instances.

3.5.4 Resource group relations

Similarly to process groups, resource groups can be used to emphasize certain relations between resources. The introduction of resource groups imposes no additional restrictions on the set of feasible plans.

The attribute of the resource group node is:

name: A name that uniquely identifies the resource group.

3.6 Extensions

In the previous sections, we have shown that it is possible to describe many different aspects of the problem instances that we are interested in by using attributed graphs. The combined use of graph elements and attributes has enabled us to deal with such diverse problem characteristics as precedence constraints, resources possessing the same capabilities, and complex consumption patterns. Our general problem class, i.e., the class of problem types of which instances can be described with our method, includes a large variety of problems that are of theoretical or practical interest. However, there still exist many problem instances that currently cannot be described. This is not a consequence of the lack of expressive power of

attributed graphs, but it is a consequence of the selection criteria that we have applied with respect to the problem characteristics that we wanted to be able to describe. These selection criteria have been chosen somewhat arbitrarily. The main goal of our research is to show that it is possible to develop a PBG that can generate planning boards for a relatively large class of problem types. Therefore, our description method must be able to deal with a fairly large class of interesting problem types. Both the notions 'fairly large' and 'interesting' are subjective, but we think that they do apply to the general problem class that is induced by our description method.

3.7 Views of the Instance Graph

In the sections above, we have discussed the various components that can be used to describe a problem instance by an instance graph. The specification of the nodes, arcs, edges, and $K_{1,n}$'s with their associated attributes results in an instance graph that represents all information about the problem instance under consideration. During the planning process, a planner may want to view parts of this information. If he wants to make an assignment for a particular process, he may be interested in the set of resources to which the considered process can be assigned, or he may want to know which other processes have a precedence relation with the considered process. This kind of information can easily be obtained because it corresponds to relatively small subgraphs of the instance graph. Such a subgraph will be called a *view* of the instance graph. Many views can be defined. A few examples follow below.

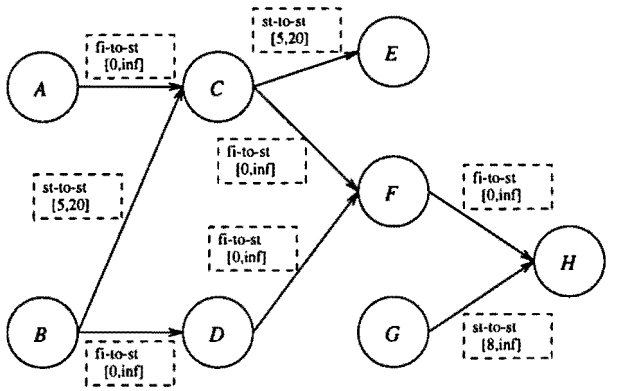

Figure 6: A precedence view

Assignment view: This view presents for a set of processes the resources to which it can be assigned, i.e., all the paths originating from one of the processes and ending at a resource. Examples of this view have already been given in Figures 3 to 5.

Process view: This view presents for a set of processes the process groups and common resource sets to which they belong.

Resource view: This view presents for a set of resources the resource groups and capabilities to which they belong.

Precedence view: This view presents for a set of processes the precedence relations between them. An example is given in Figure 6. The arc from node B to node C implies that process B must be started at least 5 time units and at most 20 time units before process C is started.

A view is a convenient mechanism to present parts of the information embedded in the instance graph. Note that it is possible that a view contains some redundancy. For example, when a particular capability is possessed by only one resource, the corresponding capability node can be considered as redundant. By connecting the process, common resource set, function, and capability set nodes that are connected to that capability node, directly to the corresponding resource node, a view can be *reduced* to a smaller, possibly more insightful one.

4 Problem Types

In general, planning boards are not developed as a tool for solving one particular problem instance, but as a tool for solving a set of problem instances with similar characteristics. We will use the term *problem type* to refer to a set of problem instances satisfying certain conditions. The general problem class, i.e., the set of all problem types of which instances can be described by the instance description method, is a problem type itself.

A PBG requires information about the problem type for which a planning board has to be generated and information about the desired representations and manipulations. Information about the desired manipulations could be something of the following sort: the planning board must support the reassignment of a process. If at a given moment process A is assigned to resource R_1 with start time t_1, the user should be able to reassign process A to resource R_2 with start time t_2. It should be obvious that the implementation of such a manipulation will depend on the problem type. For a problem type in which all resources are available during the whole planning period, checking the feasibility of a reassignment is much easier than for a problem type in which the resources are only available during certain time intervals.

In fact, the main purpose of the description of the problem type is to provide the necessary information to implement the desired manipulations and representations as efficiently and effectively as possible. Consequently, a description of a problem type should provide information on those common characteristics of the various problem instances that are important for the development of manipulations and representations. Therefore, care should be taken to ensure that the description of the problem type is accurate. It should contain all problem instances for which the planning board is to be used, but as few others as possible because they may lead to a planning board that is less efficient and effective than possible.

Problem types will be specified by restrictions on the instance graph, i.e., a problem type is the set of instances that can be described by an instance graph that satisfies the given restrictions.

4.1 Restrictions on the Instance Graph

The characteristics of a problem instance are defined by the structure of the associated instance graph and the values of the attributes of the various graph elements. We can therefore distinguish two types of restrictions: on the graph structure and on the attribute values.

4.1.1 Restrictions on the graph structure

Characteristics of a problem type that can be described in terms of restrictions on the graph structure relate to the presence or absence of the various graph elements and to the way these graph elements are interconnected.

Consider the well-known job shop scheduling problem. The restrictions that have to be imposed on the graph structure are the following:

- The only node types that appear are process, resource, capability, and process group.
- Each capability node is connected to exactly one resource node.
- Each requirement edge connects a process to a capability.

- Precedence relations occur in chains and only involve processes in the same process group.

If the problem type becomes more complex, it is not always possible to describe restrictions that are valid for all graph elements of the same type. For example, consider the extension of the job shop scheduling problem in which each task requires both a machine and some raw material. In an instance description, this extension would be modeled with a capability set node that specifies that a machine and some raw material are required for a task. However, in a type description it is insufficient to specify that capability set nodes exist and that each capability set node is connected to exactly two capability nodes, because also instances in which a task is assigned to two machines would satisfy this restriction.

In order to deal with such problems, we need to be able to distinguish nodes of the same type. The notion of a *node group* is introduced precisely for that reason. A node group is a group of nodes of the same type that are subject to the same restrictions. In the above extension of the job shop scheduling problem, we would introduce two capability node groups, *Machine* and *Inventory*, and impose the following restrictions:

- The only node types that appear are process, resource, capability, capability set, and process group.
- Each capability node is connected to exactly one resource node.
- Each capability set node is connected to exactly one capability node in the node group *Machine* and to exactly one capability node in the node group *Inventory*.
- Each requirement edge connects a process to a capability set.
- Precedence relations occur in chains and only involve processes in the same process group.

Besides restrictions on the presence or absence of the various graph elements and on the the way they are interconnected, it is often necessary to also impose restrictions on the number of them. For example, if the raw material in our extension of the job shop scheduling problem is the same for all tasks, one should be able to specify that the number of nodes in the node group *Inventory* is precisely one.

4.1.2 Restrictions on the attributes

Characteristics of a problem type that can be described in terms of restrictions on the attributes of the graph elements relate to their domains. The domain of an attribute is its set of admissible values. In a problem type description it is possible to reduce a domain by specifying restrictions on the set of admissible values. Attribute restrictions are specified for node groups.

4.2 Describing Problem Types

In this subsection, we present a syntax which can be used to describe the restrictions discussed in Section 4.1 more formally. The syntax can be modified or extended in order to deal with other kinds of restrictions. In Appendix B several problem types are described using this syntax.

4.2.1 Node groups

The first part of the description of the problem type concerns the node groups that will appear in an instance. For each node type the associated node groups and a specification of the number of nodes in these node groups have to be given. For example,

process:	Task1	{1,...,∞}
	Task2	{1,...,∞}
resource:	Machine	{1,...,4}
capability:	Type	{1}

indicates that there are three node types and four node groups, that the node groups Task1 and Task2 may have an arbitrary number of nodes, that the node group Machine may have one up to four nodes, and that the node group Type has precisely one node.

Instances with five or more resources or more than one capability do not belong to this problem type. Also instances with common resource set nodes, function nodes, capability set nodes, process group nodes, or resource group nodes do not belong to this problem type.

In the example above, the number of nodes in a node group is restricted to be in a specified set, e.g., {1}, {1,..,4}, and {1,..,∞}. In addition to specifying a set, it is also possible to relate the cardinality of a node group to the number of nodes in another node group. For example,

resource:	Machine	{2,...,100}
capability:	Type	#(Machine)

would indicate that the number of nodes in the node group Type is equal to the number of nodes in the group Machine.

4.2.2 Graph structure

The second part of the description of the problem type concerns the structure of the instance graphs. It deals with n-ary relations, requirement edges, and precedence arcs.

For each of the $K_{1,n}$'s associated with the node groups, the node groups to which they can be connected and a specification of the cardinality of these node groups have to be given. For example,

capability:	Type	Machine	#(Machine)
capability set:	Transform	Type	{1}
		InputInv	{1}
		OutputInv	{1}

indicates that each node in the node group Type is connected to all nodes in the node group Machine, and that each node in the node group Transform is connected to one node in the node group Type, one in the node group InputInv, and one node in the node group OutputInv.

Requirement edges connect process nodes and common resource set nodes to function nodes, capability set nodes, and capability nodes. Restrictions on the possible positions of requirement edges are specified for all relevant node groups. For example,

requirement edges:	Task	Type
	CRSet1	Mode1
	CRSet2	Mode2

indicates that (process) nodes in the node group Task are connected by a requirement edge to (capability) nodes in the node group Type, and that (common resource set) nodes in the

node groups CRSet1 and CRSet2 are connected by a requirement edge to (function) nodes in the groups Mode1 and Mode2, respectively.

Precedence arcs usually occur between processes in the same node group or in the same process group. Furthermore, the graphs describing precedence relations often have a special structure, such as chains or trees. The structure of the precedence graph is specified for all relevant node groups. For example, suppose Group1,...,Group4 are process node groups, and Job is a node group of process groups. Then,

precedence arcs:	Group1	intree
	Group2	outtree
	Group3 ∪ Group4	general
	Job	chain

indicates, that the subgraph induced by precedence arcs of (process) nodes in the node group Group2 is an intree, that the subgraph induced by precedence arcs of (process) nodes in the node group Group3 is an outtree, and that the subgraph induced by precedence arcs of (process) nodes in the node groups Group3 and Group4 does not have a special structure. Furthermore, for each process group in the node group Job the following must hold: the subgraphs induced by precedence arcs of nodes representing processes in that process group are chains,

4.2.3 Attributes

The third part of the description of the problem type concerns the attributes of the different objects. For each node group as well as for the precedence arcs, a domain is specified for each attribute. For example,

process:	Task	type	non-repetitive
		size	{1,2}
resource:	Machine	category	non-renewable
precedence:	Group1	type	{finish-to-start,start-to-start}
		time lag	[0, 50]

indicates that processes in the node group Task are non-repetitive and have size 1 or 2, that resources in the node group Machine are non-renewable, and that the precedence relations between nodes in the node group Group1 are either finish-to-start or start-to-start with a minimum time lag of 0 and a maximum time lag of 50 time units.

It is also possible to enforce two attributes to take on the same values. For example,

process:	Task	size	{1,...,∞}
		consumption intensity	size

indicates that the size attribute of processes in the node group Task can take on any positive integer value, and that the value of the consumption intensity attribute is equal to the value of the size attribute.

The attributes of capability sets are functions of attributes of other objects. In the problem type description, either these functions are completely specified, or restrictions on their shape are imposed. In referring to attributes of other objects we apply a two-field notation: *object.attribute*, where *object* is a node group. When misinterpretation is impossible, it suffices to only state *attribute*. For example,

capability set:	ResourceSet	duration	size/MacCap.speed
		(MacCap)	
		usage interval	$[start, start + duration]$
		usage volume	1
		(InvCap)	
		consumption interval	$[start, start]$
		consumption volume	$a * consumptionintensity$
			$a \in (0, \infty)$

indicates that for a capability set in the node group ResourceSet the duration function is defined as the quotient of the size of the process and the speed of the resource possessing a capability in the node group MacCap. Furthermore, that resource is used during the entire processing interval, the resource possessing a capability in the node group InvCap is consumed at the beginning of the processing interval, and the function that is used for computing the consumed volume is a linear function of the consumption intensity attribute of the process.

When certain attributes of objects are irrelevant for the problem type we want to describe, for example the due date attribute in case no due dates occur, this can be indicated by 'does not apply'. If no domain is specified for an attribute of an object, we assume that the default domain specifications apply. The default domain specifications for the different attributes are the following:

ATTRIBUTES

process:	type	non-repetitive
	mode	does not apply
	size	$(0,\infty)$
	usage intensity	1
	consumption intensity	does not apply
	release time	does not apply
	deadline	does not apply
	due date	does not apply
	split	0
resource:	availability periods	does not apply
	category	renewable
	number of modes	0
	usage	
	divisibility	1
	dimension	1
	capacity	$c(t) = 1$
	consumption	
	divisibility	1
	dimension	1
	supply	$s(t) \in \{1, ..., \infty\}, t = 0$
		$s(t) = 0, t \neq 0$
	number of capabilities	*no default*
	speed	1
	usage factor	1
	consumption factor	does not apply

capability set:	duration	*no default*
	usage interval	*no default*
	usage volume	*no default*
	consumption interval	*no default*
	consumption volume	*no default*
process group:	release time	does not apply
	deadline	does not apply
	due date	does not apply
precedences:	type	finish-to-start
	time lag	$[0,\infty]$

Time occurs in the problem type description in a similar way as attributes. In the default time system, there is only one level and there is no restriction on the length of the planning period:

TIME	
number of levels	1
planning period	$\{1,\ldots,\infty\}$

5 Concluding Remarks

We have introduced and discussed various aspects of a planning board generator, a tool that facilitates the development of automated planning boards. A planning board generator uses a specification of the characteristics of a problem type, a specification of the desired representations, and a specification of the required manipulations to produce a prototype of an automated planning board.

In this paper, we have concentrated on specifying characteristics of problem instances and problem types. We have developed an instance description methods and a type description method, both of which rely heavily on the concepts of attributed graphs.

In a subsequent paper, we will concentrate on specifying representations and manipulations. In an interactive planning environment informative representations of a problem instance and possible solutions as well as easy-to-use manipulations are extremely important.

Although the Gantt chart has been and will be the main representation mechanism for a planning board, various other representations, such as the 'views' that were discussed briefly in this paper, can enhance its effectiveness considerably. It is a nontrivial task to take concepts such as a Gantt chart and a view and convert them into an actual representation using available graphic primitives.

Although simple manipulations on the Gantt chart, such as reassigning a process to another resource and time interval, have been and will remain important functions of a planning board, more advanced manipulations can be supported in an automated planning board, such as the computation of a schedule for a given subset of the processes. It is a challenging task to design a relatively small set of simple and advanced manipulations that together constitute a powerful yet easy-to-use planning board.

It is prohibitive to develop the most effective and efficient representations and manipulations for all possible problem types. We envision that the planning board generator will incorporate a scheme that exploits the structure of problem types, as defined implicitly by our description method, to customize representations and manipulations.

The preceding remarks provide a preview of some of the aspects that will be discussed in greater detail in our sequel paper.

References

[1] J.M. ANTHONISSE, K.M. VAN HEE, J.K. LENSTRA. 1988. Resource-Constrained Project Scheduling: an International Exercise in DSS Development. *Decision Support Systems* **4**, 249-257.

[2] J.M. ANTHONISSE, J.K. LENSTRA, M.W.P. SAVELSBERGH. 1988. Behind the Screen: DSS from an OR Point of View. *Decision Support Systems* **4**, 413-419.

[3] J. BŁAŻEWICZ, W. CELLARY, R. SŁOWIŃSKI, J. WEGLARZ. 1986. Scheduling under Resource Constraints - Deterministic Models. *Annals of Operations Research* **7**.

[4] M.L. FISHER. 1985. Interactive Optimization. *Annals of Operations Research* **5**, 541-556.

[5] M.L. FISHER, M.B. ROSENWEIN. 1989. An Interactive Optimization System for Bulk-Cargo Ship Scheduling. *Naval Research Logistics* **36**, 27-42.

[6] S. FRENCH. 1982. *Sequencing and Scheduling: an Introduction into the Mathematics of the Job-Shop*, Horwood, Chichester.

[7] R.D. HURRION. 1986. Visual Interactive Modelling. *European Journal of Operational Research* **23**, 281-287.

[8] P. JACKSON, J.A. MUCKSTADT, C.V. JONES. 1989. COSMOS: A Framework for a Computer-aided Logistics System. *Journal of Manufacturing and Operations Management.* **2**, 122-148.

[9] C.V. JONES, W.L. MAXWELL. 1986. A System for Manufacturing Scheduling with Interactive Computer Graphics. *IIE Transactions* **18**, 298-303.

[10] C.V. JONES. 1990. An Introduction to Graph-Based Modeling Systems, Part I: Overview. *ORSA Journal on Computing* **2**, 136-151.

[11] E.L. LAWLER, J.K. LENSTRA, A.H.G. RINNOOY KAN, D.B. SHMOYS. 1993. Sequencing and Scheduling: Algorithms and Complexity, in: S.C. Graves, A.H.G. Rinnooy Kan, P.H. Zipkin (eds.), *Logistics of Production and Inventory*, Handbooks in OR & MS 4, Elsevier Science Publishers B.V., North-Holland, Amsterdam.

[12] N.A. MOREIRA, R.C. OLIVEIRA. 1991. A Decision Support System for Production Planning in an Industrial Unit. *European Journal of Operational Research* **55**, 319-328.

[13] F. VIVIERS. 1983. A Decision Support System for Job Shop Scheduling. *European Journal of Operational Research* **14**, 95-103.

[14] A.P. WOERLEE. 1991.*Decision Support Systems for Production Scheduling.* PhD Thesis, Erasmus University Rotterdam.

A Examples of Problem Instances

In this appendix we describe the instance graphs, or parts of it, of three problem instances. The first example shows what kinds of connections between the different types of nodes are possible. In the second example, we consider a timetabling problem for a school. The third example deals with a factory scheduling problem with a non-trivial consumption pattern.

Example A.1 We consider the following production planning problem. Four products of two different types are to be made. Products *1* and *2* are products of the first type, products *3* and *4* are of the second type. Processes *P1*, *P2*, *P3*, and *P4* represent the production of products *1*, *2*, *3*, and *4*. There is a fifth process, *M*, which represents some maintenance activities. Performing *P1* can be done in two modes, the normal node or the special mode, *P2* requires the special mode. In both modes an employee and a machine of the type *MT1* are used, but some machines of that type can only be used in the special mode and others only in the normal mode. The processes *P3* and *P4* require a machine of the type *MT2* and an employee. The combination that is used for *P3* must also be used for *P4*. The maintenance process is to be performed by one of the employees. The instance graph for this problem is given in Figure 7.

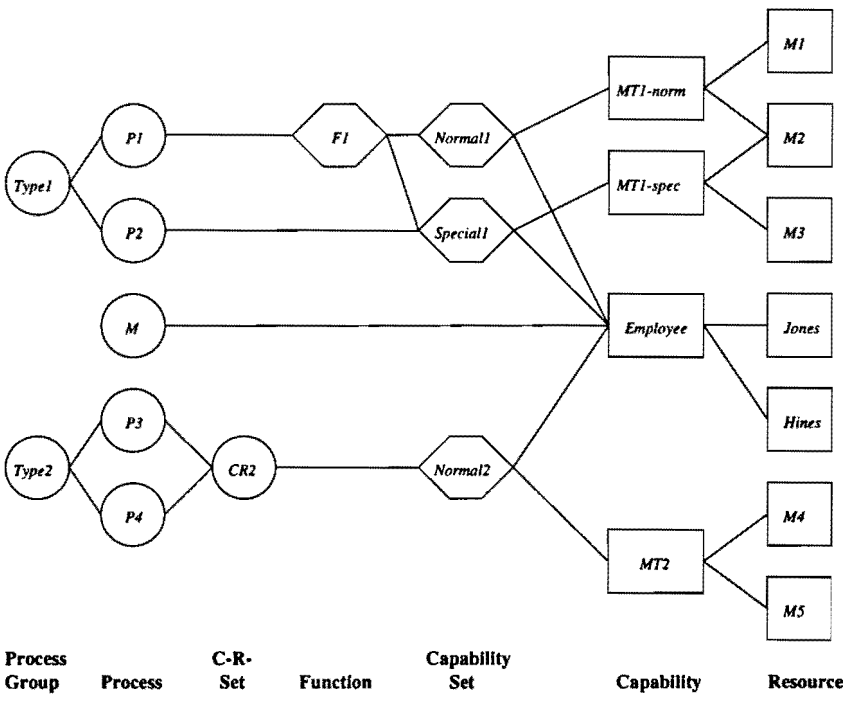

Figure 7: A production planning problem instance

This example illustrates that all different kinds of resource requirements discussed in Section 3.4 can occur in the same problem instance. Process *P1* requires a function, process *P2* requires a capability set, and process *M* requires a capability. Furthermore, since processes *P3* and *P4* must be performed on the same combination of resources, there is an associated requirement edge connecting the corresponding common resource set node with, in this case,

a capability set node. □

Example A.2 We consider the following timetabling problem. Each week, each group of pupils has to get lessons in different subjects during a specified number of hours. Some lessons require teachers with full qualification, others require teachers with partial qualification. Most lessons can be given in normal classrooms, but for some subjects the lessons require special rooms. Chemistry lessons, for example, must be given in rooms in which experiments with chemicals can be performed.

We can describe this situation in the following way. The processes are the lessons given to the different groups. They are non-repetitive. An example is the chemistry lesson for group 3. There are two kinds of resources: teachers and classrooms. A teacher can have several capabilities, representing the subjects he is qualified to teach and the levels of qualification. Since a teacher may be fully qualified for one subject and partially qualified for the other, it is necessary to introduce capability nodes for each possible combination of subject and qualification. A classroom also may have several capabilities. A particular classroom may be suited for chemistry lessons, as well as normal (for example, English) lessons.

Each lesson requires a particular kind of classroom and a particular kind of teacher. These requirements are represented by requirement edges connecting the processes and capability sets. Each capability set consists of two capabilities, one representing the required kind of classroom, the other representing the required kind of teacher. For example, the lesson English for group 4 requires the capability set *E-F*, representing a normal classroom and a teacher with full qualification for the subject English.

A part of an instance is depicted in Figure 8. Only the lessons English and chemistry for groups 3 and 4 and not all teachers and classrooms are considered.

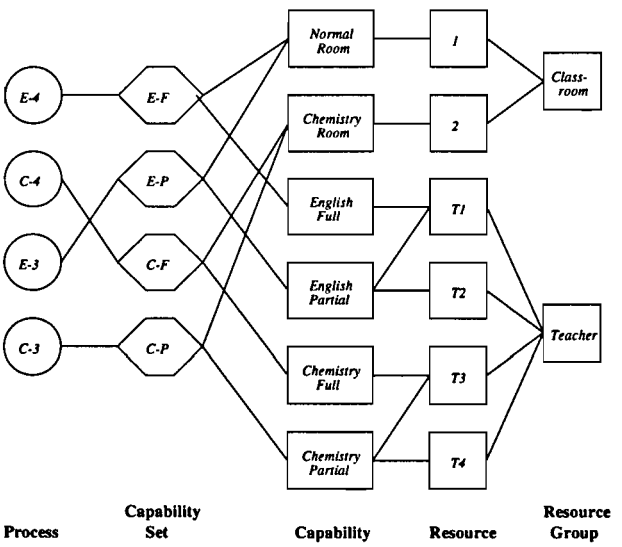

Figure 8: Part of a timetabling instance

The planning period will consist of 5 days of say 8 hours. The resources are renewable resources with capacity 1. A process uses both kinds of resources fully during the entire processing interval, which is always one hour. □

Example A.3 Jones and Maxwell [9] give an example of a factory scheduling problem. There are three processes, *1MSUB*, *2MSUB*, and *ASSEM*. Process *1MSUB* uses a machine of the type *A-MILL* and materials *MAT1* and *MAT2* to create half-products *SUB1*. Process *2MSUB* uses the machine *MILL* and the half-product *SUB3* to create another half-product, *SUB2*. Process *ASSEM* is performed by a worker, who assembles half-products *SUB1* and *SUB2* to create the end-product *WIDGET*. The reduced assignment view of this instance is given in Figure 9.

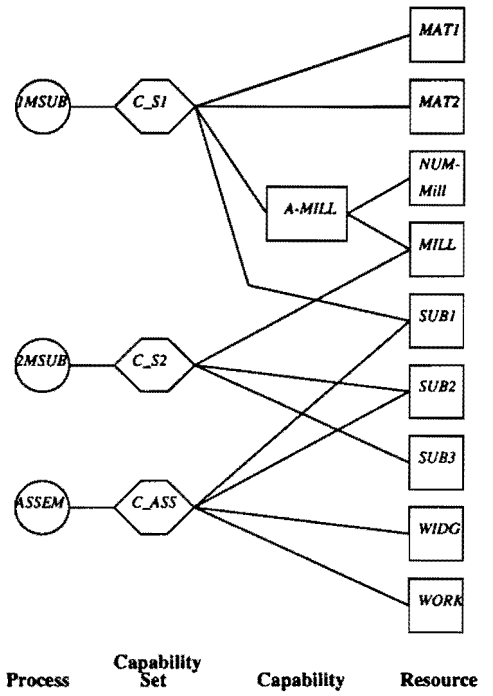

Figure 9: A reduced assignment view of a factory scheduling instance

All processes are repetitive. The machines are renewable resources; the materials, half-products, and end-products are non-renewable resources. The machines have capacity 1 and are always used up to capacity. The materials are consumed at a constant rate during the processing period, and the half products and end products are consumed (produced) at the beginning (end) of each repetition.

We consider the capability set *CS1*. The duration solely depends on the *MILL* that is used, and the consumed volumes are completely determined by the consumption intensity (i^c) of the process and the consumption factor of the considered resource ($f^c(.)$). The attributes of capability set *CS1* then are:

name: *CS1*

duration: $= size\ /\ A\text{-}MILL.speed$

(MAT1)

consumption interval: $[start, start + duration]$

consumption volume: *consumption intensity / MAT1.consumptionfactor*

(MAT2)

consumption interval: $[start, start + duration]$

consumption volume: *consumption intensity / MAT2.consumptionfactor*

(A-MILL)

usage interval: $[start, start + duration]$

usage volume: 1

(SUB1)

consumption interval: $[start + duration, start + duration]$

consumption volume: - *consumption intensity / SUB1.consumptionfactor*

□

B Examples of Problem Types

Example B.1 The simple job shop scheduling problem type discussed in Section 4.1, in which each task requires one specific machine, can be described as follows:

NODE GROUPS

process:	Task	{1,...,∞}
resource:	Machine	{1,...,∞}
capability:	MacCap	#(Machine)
process group:	Job	{1,...,∞}

GRAPH STRUCTURE

capability:	MacCap	Machine	{1}
process group:	Job	Task	{1}
requirement edge:	Task	MacCap	
precedence arcs:	Job	chain	

ATTRIBUTES

process:	Task	type	non-repetitive
resource:	Machine	category	renewable
		number of capabilities	1
		(MacCap, 1)	

Note that although non-repetitive and renewable are the default values for the **type** and **category** attributes, we have explicitly mentioned them in the problem type description for clarity. □

Example B.2 In Appendix A, an example of a timetabling problem has been discussed. Lessons must be assigned to teachers and classrooms, and the feasibility of an assignment depends on the qualification of the teacher (what subject, full or partial qualification) and the properties of the classroom (chemistry lessons require specific facilities for performing experiments). The example can be seen as an instance of the problem type in which each lesson (process) requires a combination (capability set) of a type of classroom and a specific qualification (both capabilities).

In the terminology of our description method, these restrictions can be formulated in the following way. We only give the node groups and the graph structure restrictions that are related to the processes, resources, capabilities, and capability sets.

NODE GROUPS

process:	Lesson	{1,...,∞}
resource:	Teacher	{1,...,∞}
	Classroom	{1,...,∞}
capability:	Qualification	{1,...,∞}
	ClassroomType	{1,...,∞}
capability set:	Room&Qual	{1,...,∞}

GRAPH STRUCTURE

capability:	Qualification	Teacher	{1,...,∞}
	ClassroomType	Classroom	{1,...,∞}
capability set:	Room&Qual	ClassroomType	{1}
		Qualification	{1}
requirement edges:	Lesson	Room&Qual	

□

Example B.3 Anthonisse et al. [1] describe resource-constrained project scheduling (RCPS) as follows.

> A set of tasks is to be processed by a set of resources. For each task, there is a release time and a deadline, which define a time interval in which the task must be processed. Once a task is started it must be completed without interruption.
>
> For any two tasks, there are a lower bound and an upper bound on the length of the time period between the completion of one task and the start of the other. (...)
>
> A function may be performed by various combinations of resources, each with its own speed. In general, each function has a class of feasible resource sets, and the processing time of a task depends on the feasible resource set that is chosen to perform the function it requires. The processing time is the amount of work (i.e., the number of units of the function) required by the task divided by the speed of the feasible resource set.
>
> No resource can be allocated to two tasks at the same time. If a set of resources is allocated to a task, then each of its constituent resources is occupied by that task from its starting time until its completion time. For each resource there is a set of time intervals during which the resource is available.

We will now describe this problem type with our description method.

The time system is the default system. There is only one time unit, and the planning period can be as small or as large as one wants.

Four kinds of objects are distinguished in the description above: tasks, resources, resource sets, and functions. In our terminology, they are processes, resources, capability sets, and functions, respectively. In fact, a fifth kind of object, the *project*, is considered, which is equal to the process group. Beside these five objects, our method requires capability nodes. Each resource has a unique capability. Therefore, the number of capability nodes is equal to the number of resource nodes. There are no restrictions on the number of nodes of any of the other types.

NODE GROUPS

process:	Task	{1,...,∞}
resource:	Resource	{1,...,∞}
capability:	ResCap	#(Resource)
capability set:	ResourceSet	{1,...,∞}
function:	Function	{1,...,∞}
process group:	Project	{1,...,∞}

Each resource possesses a unique capability. Therefore, each capability node (belonging to the node group ResCap) is connected to one resource node. The capability $K_{1,n}$'s are in fact $K_{1,1}$'s. The ResourceSets may consist of any number of resources, or better, of any number of capabilities. Each function node may be connected to any number of ResourceSet nodes (possibly one), and each process requires a function. For each Project, the number of tasks in it is completely instance dependent. Precedence relations are only possible between tasks in the same Project.

GRAPH STRUCTURE

capability:	ResCap	Resource	{1}
capability set:	ResourceSet	ResCap	{1,...,∞}
function:	Function	ResourceSet	{1,...,∞}
process group:	Project	Task	{1,...,∞}
requirement edges:	Task	Function	
precedence arcs:	Project	general	

All processes are non-repetitive. The **usage intensity** does not apply, because it is not required in order to describe usage in the capability set attribute. The **consumption intensity** does not apply because all resources are renewable. The **size** attribute of processes, and the **release time**, **deadline**, and **due date** attributes of Projects and processes may take on any value. The tasks cannot be preempted.

Only renewable resources occur. A resource cannot be used for different tasks at the same time. Hence, the **capacity** is constantly 1, and the usage discretization unit is also 1. The speed is not determined by individual resources, but only by the resource set as a whole. Therefore, the **speed** attribute does not apply. Also, the **usage factor** and the **consumption factor** do not apply.

Since no consumption occurs, the consumption attributes do not apply. The **duration** is determined as the quotient of the size attribute of the process and a number representing the speed of a resource combination that is different for each capability set. Each resource that a process is assigned to is used during the entire processing interval. The used volume is equal to one for each resource.

The precedence relations are always of the finish-to-start type. There are no restrictions on the corresponding time lags.

ATTRIBUTES

process:	Task	type	non-repetitive
		usage intensity	does not apply
		consumption intensity	does not apply
		release time	[0,∞)
		deadline	[0,∞)
		due date	[0,∞)
resource:	Resource	category	renewable
		number of capabilities	1
		(ResCap, 1)	
		speed	does not apply
		usage factor	does not apply
		consumption factor	does not apply

capability set:	ResourceSet	duration	$a * size$
			$a \in (0, \infty)$
		(ResCap)	
		usage interval	$[start, start + duration]$
		usage volume	1
process group:	Project	release time	[0,∞)
		deadline	[0,∞)
		due date	[0,∞)
precedences:	Project	time lag	[[0,∞),[0,∞)]

Anthonisse et al. [1] give an example of a problem instance of the RCPS type. The corresponding instance graph is given in Figure 10.

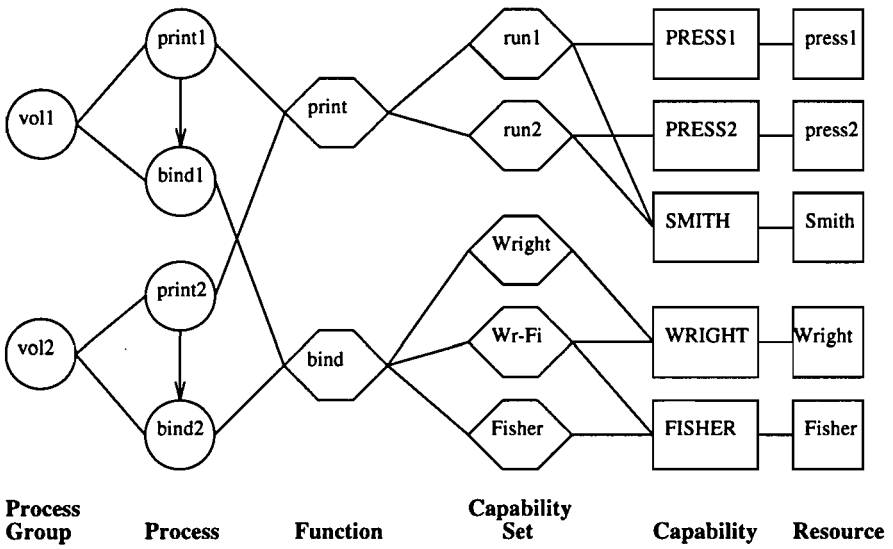

Figure 10: The RCPS instance

□

Example B.4 Jones and Maxwell, [9], discuss factory scheduling problems. In these problems three kinds of objects occur: processes, inventories, and machines. Processes are performed on one machine and consume and produce the contents of inventories.

In their article, Jones and Maxwell introduce a description method that is based on networks. We can describe this problem type with our description method as well.

We introduce three resource node groups. One node group consists of nodes that represent machines, the other two represent continuous inventories and discrete inventories, respectively. The reason for introducing two inventory node groups is that different restrictions on the attribute values can be specified for continuous and discrete inventories.

Since different consumption patterns apply when an inventory serves as input or occurs as output of a process, for each of both types of inventory two capability node groups are introduced. There are no functions, common resource sets, process groups, and resource groups.

NODE GROUPS

process:	Process	{1,...,∞}
resource:	Machine	{1,...,∞}
	ContInventory	{1,...,∞}
	DiscInventory	{1,...,∞}
capability:	MachineType	{1,...,∞}
	ContInInv	{1,...,∞}
	ContOutInv	{1,...,∞}
	DiscInInv	{1,...,∞}
	DiscOutInv	{1,...,∞}
capability set:	ResourceSet	#(Process)

There may be several machines of a particular type. For each process, the actually required inventories are completely specified. Hence, the corresponding $K_{1,n}$'s connecting capabilities and resources are simple edges. Each ResourceSet consists of one machine type and an arbitrary number of inventories, which can be discrete or continuous, and can serve as input or occur as output. There are no precedence relations.

GRAPH STRUCTURE

capability:	MachineType	Machine	{1,...,∞}
	ContInInv	ContInventory	{1}
	ContOutInv	ContInventory	{1}
	DiscInInv	DiscInventory	{1}
	DiscOutInv	DiscInventory	{1}
capability set:	ResourceSet	MachineType	{1}
		ContInInv	{0,...,∞}
		ContOutInv	{0,...,∞}
		DiscInInv	{0,...,∞}
		DiscOutInv	{0,...,∞}
requirement edges:	Process	ResourceSet	

The time system is again simple. There is only one time unit, and the planning period can be as small or as large as one wants.

All processes are repetitive processes. The **size** attribute can take on any value. Jones and Maxwell introduce a process attribute (Time/Lot) that deals with both the duration of one repetition and the consumption. In our approach, we enforce the **size** attribute and the **consumption intensity** to take on the same value. The **usage intensity** does not apply. We assume that a process may not be interrupted during a repetition, hence the **split** attributes have value 0. There are no release times, deadlines, and due dates.

Machines are renewable resources, with capacity equal to 1. They may have several capabilities, all representing a particular MachineType. Since the duration of a process is uniquely determined by the size of that process, the **speed** attribute for machines does not apply. Similarly, **usage factor** and **consumption factor** do not apply.

The inventories are non-renewable resources. The resources of the ContInventory type have continuous divisibility, the resources of the DiscInventory type have discrete divisibility. Again, the **speed**, **usage factor**, and **consumption factor** attributes do not apply. The supply attribute can take on any value.

Consumption patterns for the different inventories depend on whether they are discrete or continuous and on whether they are used as input or as output. The consumed volumes are determined by dividing a number that is different for each ResourceSet by the consumption intensity of the process. This number is positive if the inventory is used as input, and negative if the inventory is used as output. Consumption of the continuous inventories occurs during the entire processing period; consumption of discrete inventories occurs at the beginning (when used as input), or at the end (when used as output), of each repetition.

ATTRIBUTES

process:	Process	type	repetitive
		usage intensity	does not apply
		consumption intensity	size
resource:	Machine	category	renewable
		number of capabilities	$\{1,\ldots,\infty\}$
		(MachineType, $\{1,\ldots,\infty\}$*)*	
		speed	does not apply
		usage factor	does not apply
		consumption factor	does not apply
resource:	ContInventory	category	non-renewable
		consumption	
		divisibility	0
		supply	$s(t) \in [0, \infty)$
		number of capabilities	2
		(ContInInv, 1)	
		speed	does not apply
		usage factor	does not apply
		consumption factor	does not apply
		(ContOutInv, 1)	
		speed	does not apply
		usage factor	does not apply
		consumption factor	does not apply
	DiscInventory:	category	non-renewable
		consumption	
		supply	$s(t) \in \{0, \ldots, \infty\}$
		number of capabilities	2
		(DiscInInv, 1)	
		speed	does not apply
		usage factor	does not apply
		consumption factor	does not apply
		(DiscOutInv, 1)	
		speed	does not apply
		usage factor	does not apply
		consumption factor	does not apply

capability set:	ResourceSet	duration	size
		(Machine)	
		usage interval	$[start, start + duration]$
		usage volume	1
		(ContInInv)	
		consumption interval	$[start, start + duration]$
		consumption volume	$a/consumptionintensity$
			$a \in (0, \infty)$
		(ContOutInv)	
		consumption interval	$[start, start + duration]$
		consumption volume	$a/consumptionintensity$
			$a \in (-\infty, 0)$
		(DiscInInv)	
		consumption interval	$[start, start]$
		consumption volume	$a/consumptionintensity$
			$a \in (0, \infty)$
		(ContOutInv)	
		consumption interval	$[start + duration, start + duration]$
		consumption volume	$a/consumptionintensity$
			$a \in (-\infty, 0)$

In Appendix A we have already discussed an instance of this problem type. The graph given in Figure 9 does not fit in the description of the problem type above since it represents a reduced view. In the complete graph the connections between capability sets and resources are always established via capabilities. The resource *SUB2* would be connected to two capability nodes, one representing the use of *SUB2* as input and one representing the use of *SUB2* as output. *MAT2* on the other hand, although also occurring in two capability sets, would be connected to only one capability node, since it is only used as input. □