Integrated modelling: when time and resources play a role

Roman Barták

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské námestí 2/25
118 00 Praha 1, Czech Republic
bartak@kti.mff.cuni.cz

Abstract

A formal model of the planning or scheduling problem is the first step in the design of a solver for such a problem. In the paper we propose a basic framework for modelling planning and scheduling problems that involve reasoning about time and resources. In this framework we go beyond the traditional definitions of planning and scheduling and, from the beginning, we expect integration of both these areas.

Introduction

Traditional AI planning tackles the problem of sequencing operators to achieve some goal. In STRIPS-like planning, the operator is defined by pre-conditions and effects, i.e., the pre-conditions must be satisfied to use the operator, and the effects hold after using the operator. The task is to find a sequence of operators starting from a given set of pre-conditions and achieving a given set of effects.

There is no explicit usage of time and resources in traditional planning. In fact, there are no numeric values used so planning methods are based mostly on symbolic manipulation. That is the reason why planning is assumed to be an AI problem rather than a number crunching task. Nevertheless, we can find time and resources behind the traditional planning notions. At least relative time must be assumed if speaking about operator sequencing, i.e., the pre-conditions hold just before we execute the operator and the operator's effect will be true since we execute the operator (until another operator annihilates the effect). Still, traditional planning uses instantaneous operators, i.e., no duration of the operator is assumed. This is OK if we are just sequencing the operators, but, it may cause problems when overlaps of operators are allowed. Moreover, in reality executing the operator takes some time so the planning system should assume this time when looking for a valid sequence of operators. The above observations are reflected in so called durative actions that are included in the recent version of PDDL (Fox and Long 2001), a modelling language for planning problems, and that are studied in (Coddington, Fox, and Long 2001).

While time is hidden in semantics of operators, the resources can be encoded in formulas defining pre-conditions and effects. Even one the earliest planning problems - a block world problem - involved a resource, the robot's hand that moves the blocks over the table. Encoding resource in pre-conditions and effects is a standard way of modelling resources in traditional planning. However, this technique covers only a limited number of resources, we can call them *state resources*. Pre-conditions describe a required state of the resource to execute the operator, e.g., an empty hand, and effects describe a state of the resource after executing the operator, e.g. holding a block A.

In reality, the interaction between resources and operators and the integration of time and resources is more complex, e.g. a single resource may execute several operations in parallel. This brings planning to a new level where the quality and feasibility of the plan depends on time and resources too. Planning community is aware of such real-life demand and handling of time and resources is a hot topic in AI planning.

Time and resources play a key role in the areas of scheduling and timetabling too. The scheduling task is to allocate a known set of activities to available resources over time respecting precedence, capacity and other constraints. Timetabling can be seen as a special case of scheduling (Wren 1996) with different view of space-time (slots) and different objectives. Thus, we will not speak about timetabling separately.

The main difference of scheduling (and timetabling) from planning is that in scheduling we know the structure of activities while planning has to construct this structure. Therefore, when solving real-life problems planning and scheduling modules can be kept separated: first, we plan which activities (operators) are necessary to satisfy the demands and, second, we schedule the activities to available resources. This could be useful in some problems due to efficiency issues (Srivastava and Kambhampati 1999) but in other areas, integration of scheduling and planning seems necessary (Barták 1999) or (Smith, Frank, and Jónsson 2000). Note that this integration is not easy because of rather different techniques used to solve problems in planning and scheduling. While planning is

based mainly on symbolic manipulation, scheduling uses number crunching techniques from operations research. Recently, constraint satisfaction seems to provide a bridge between these two different technologies so discussions about integration of planning and scheduling are becoming more realistic now. Constraint programming is a widespread technology in scheduling (Wallace 1994); application of constraint satisfaction techniques to planning problems is described in (Binh Do and Kambhampati 2000), (Laborie 2001), (Nareyk 2000), or (Van Beek and Chen 1999) among others.

When speaking about integration of planning and scheduling, a formal modelling framework to describe such problems is one of the first issues. There exists a de facto standard modelling language PDDL for description of planning problems (Ghallab et al. 1998) and this language is being extended to model time (Fox and Long 2001). Other approaches in planning attempts to model resources (Brenner 2001) or (Koehler 1998). Still, all these approaches have their limitations when describing real-life resources and time.

Surprisingly, there is no system independent language for scheduling problems; at least we are not aware of any such language. There exists a well-known classification of scheduling problems using the triple (machine environment | job characteristics | optimality criterion) by Graham et al. (Brucker 2001). However, this is an academic classification, not a modelling language to describe a particular problem. Some modelling languages, like STTL (Kingston 2001), exist for timetabling problems but these languages can hardly be extended to general scheduling problems or to planning problems.

In this paper, we describe a framework for integrated description of both planning and scheduling problems. This framework is based on our previous works on modelling scheduling problems enhanced by planning capabilities (Barták 1999) and (Barták and Rudová 2001) so time and resources play an important role there. We have abstracted from a particular scheduling problem to cover a wider class of problems including pure planning and pure scheduling problems. This paper is a bit refined version of our proposal from (Barták and Rudová 2001). Here, we concentrate on a basic structure of the framework rather than on particular attributes (even if we mention some attributes to illustrate how the objects are used). This gives us a freedom of designing a generic framework that can be filled by attributes and that way adapted to a particular problem area. We also describe how such formalisation can be used to support planning/scheduling.

The paper is organised as follows. In Section 2, we highlight the main roles of formal models. In Section 3, we describe basic modelling requirements to capture real-life planning and scheduling domains and in Section 4 we specify how to model a particular problem in the given domain. Section 5 is dedicated to pre-scheduling and pre-planning techniques that prepare the formal model for solving.

A Context for Formal Models

The design of a formal model is a crucial step to understand all the details of the problem and to find a solution of the problem. However, having a formal model or more precisely having a modelling language to describe problems has other advantages. Basically, such modelling language serves as an interface (see Figure 1).

Naturally, the modelling language forms an interface between the real problem and the solver. Having such interface brings several advantages. First, the solver is independent from the problem description, i.e., it is possible to exchange the solver for a better one without changing the problem specification. For example, we can use a special solver for a particular domain without changing the user interface of the system or the problem description. Second, we can have several user interfaces for modelling different problems and all these user interfaces may share a common generic solver via the unified interface. In fact, we can use an automated modeller that converts the problem description from an ERP system or, generally, from a database describing the problem to a formal model. The solver does not need to know what is the source of the model. To summarise it, the formal modelling language provides an interface between various modules in a complete planning/scheduling system.

Universal description of planning and scheduling problems brings also the advantage of sharing problem domains and problems between researchers. Thus, it simplifies maintenance of benchmark sets. We sketch some other usages of the formal model later in the paper.

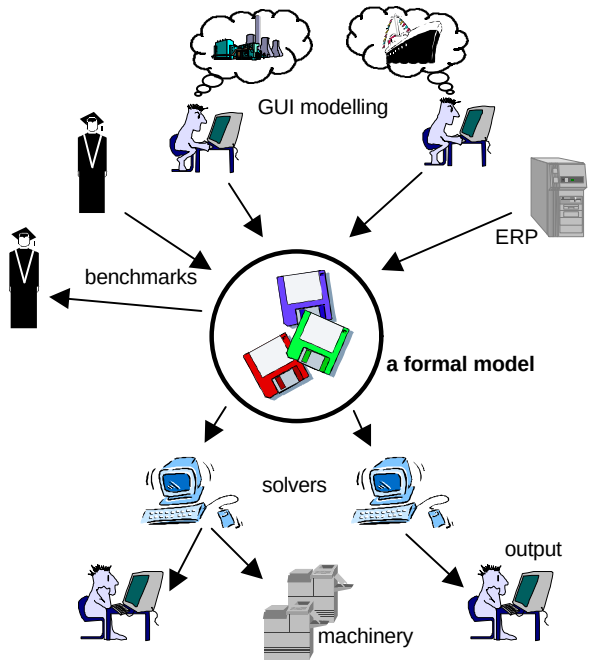

Figure 1. The role of a formal model.

Domain Modelling

When describing a problem, we can start with the description of the problem area - a domain. This makes the model more general, because it simplifies changes of the model. What is it a domain? Let us start with a real-life example of industrial scheduling. When scheduling processes in the factory, the problem description consists of the description of the factory, i.e. machines and processes, and the description of demands (orders). In this case, the domain corresponds to the description of the factory and the particular problem consists of the domain and a set of demands. We can say that the domain is a static part of the whole problem that is not changing or the changes are less frequent.

We propose the model for a domain to consist of three basic elements: activities, resources, and recipes. Activity is a basic scheduled/planned object that usually occupies some time and space. Resources define space for processing the activities and recipes describe direct relations between the activities.

Resources

Resource is an object that defines space for processing the activity. We will speak about connection between resource and activity later, so let us now concentrate on resource-only features.

Life of the resource, i.e., evolution of the resource in time can be described using a sequence of *states*. For example, the resource oven uses four states load - heat - unload - clean and these states are repeating in a cycle. Some resources, e.g. classroom in timetabling, have only one state. We expect that resource is an object (machine, room etc.) so consumable resources like fuel are modelled using a tank etc. The resource appears in a single state at a given time so the schedule for the resource consists of the sequence of non-overlapping states.

Basically, the model of resource consists of the set of states and transitions among the states (see Figure 2). The transition describes how the resource can change a state. Typically, information about timing is included so we can define minimal and maximal duration of the state, working time for the states, and transition time.

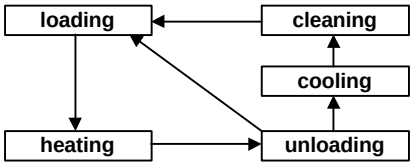

Figure 2. A state transition diagram for the resource.

Because the resource defines a space for activities, we should describe how much space is available in each state - a state capacity. The state capacity restricts the number of activities that can be processed together. We can also restrict the alignment of activities in the state. Basically, we distinguish between parallel processing, where there is no restriction about the alignment of activities, and batch processing, where the overlapping activities must start and complete at identical times (see Figure 3).

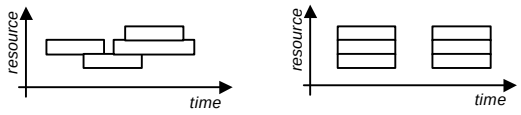

Figure 3. Parallel (left) vs. batch (right) processing.

To summarise the above discussion, the model of resource consists of the states with some attributes and the transitions between the states (see Figure 4).

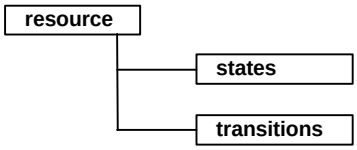

Figure 4. A basic structure of the resource model.

Activities

Activity is a basic scheduled/planned object so when modelling the problem we should specify which activities can be used in the solution. The basic attribute of the activity is its duration, i.e., time occupied by the activity. We can also use time windows to restrict when the activity can be processed.

In many cases, the activity requires some resources for processing. For example, a lecture in timetabling requires a classroom and a teacher, a heating activity in industrial scheduling requires an oven, and a moving activity in transport planning requires fuel. So for each activity we can assign a set of resource requirements. In the *resource requirement* we describe the way of using the resource. Some resources are consumed or produced, we call them consumable resources, and some resources are just used, we call them renewable resources (see Figure 5).

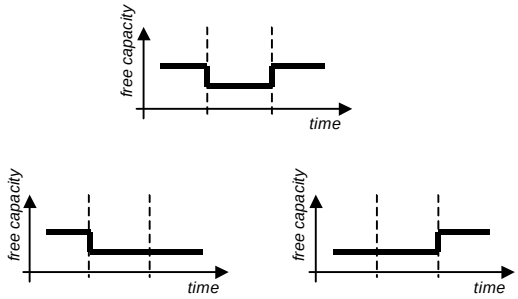

Figure 5. Renewable (top) and consumable (bottom) resources. Dashed lines indicate start and end of the activity.

Naturally, we should also describe what capacity of the resource is consumed/used/produced. We can also describe what state of the resource the activity requires. Note that the states with batch processing are meaningful for renewable usage of the resource only while parallel processing can be used both for renewable and for consumable usage of the resource.
When specifying the resource requirement, we usually have alternative resources that can satisfy the requirement. Thus we attach a list of resources to each requirement (see Figure 6).

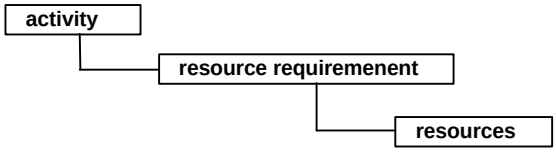

Figure 6. A basic structure of the activity model.

Recipes

The model of activities and resources can describe an indirect relation between the activities only. In particular, the only modelled relation between the activities is via a shared resource, e.g., two activities cannot run in parallel if they share a resource with capacity 1. Such modelling is usually enough for (most) timetabling problems. However, in planning and scheduling we need to model direct relations between the activities (and between the resources), for example a supplier-consumer dependency or a precedence.

Traditional planning uses STRIPS-like rules (Fikes and Nilsson 1971) to model relations between the activities: each activity has some pre-conditions and it generates some effects that may become pre-conditions of another activity. If we add some attributes to the pre-conditions and effects (typically logical terms are used to describe both pre-conditions and effects) we have a general mechanism for information passing between the activities. In HTN (Hierarchical Task Network) Planning (Erol, Hendler, and Nau 1994) the activities are connected into a task graph so more constraints can be expressed over the activities. Moreover, the tasks can be part of another task graph so planning is done via task decomposition and conflict resolution.

To simplify description of relations between the activities we introduce a notion of *event*. Each activity requires some events to precede it, we say that the activity consumes the events, and each activity generates some other events, we say that the activity produces the events. We call a triple (activity, consumed events, produced events) an *activity environment*. Note that we may have several environments for a single activity, e.g., there exists various combinations of input items consumed by the activity that produces another item. Moreover, we can put constrains between the event and the activity, for example to describe the allowed delay between the event and the activity.

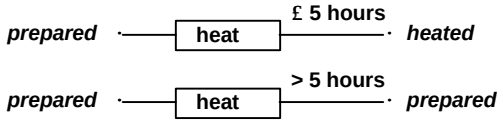

Figure 7. Two activity environments for a single activity; consumed events are on the left side and produced events are on the right side. Notice also the timing constraint between the activity and the produced event.

To provide richer modelling capabilities we propose to combine activity environments into a *recipe*. Basically, a recipe is a DAG (directed acyclic graph) where nodes are marked by activities and events. The edge goes either from an activity to an event produced by the activity or the edge goes from an event to the activity that consumes the event. In particular there are no direct edges between the activities and no direct edges between the events. The activity must be connected to all its produced and consumed events (for a given activity environment). So an activity environment forms a sub-graph in the recipe. If there are more environments for the activity then the activity may appear more times in the recipe (each appearance corresponds to one activity environment). However, there are no duplicate events in the recipe. There is one exception when the event may appear two times in the recipe. If the event is produced by one activity and consumed by another activity and connecting both activities to the same event node forms a cycle in the graph. To break the cycle (we require the recipe to be a DAG) we divide the event into two events, one is used as a consumed event only and the other one is used as a produced event only. Let us call such event a *broken event*. Such situation may appear if we want to model recycling or similar features of the real problem (see Figure 8).

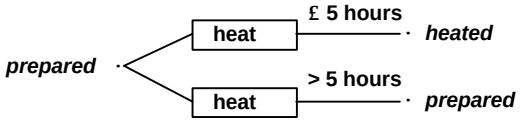

Figure 8. A primitive recipe Heating; the edges goes from left to right. There is also a broken event "prepared".

In the recipe, there exist three types of events: events that are both produced and consumed (by different activities), events that are produced only, and events that are consumed only. In case of recycling described above, the broken event is part of both consumed-only and produced-only sets of events. Together, the recipe behaves like a meta-activity and thus we can use the recipe within another recipe like an activity environment (see Figure 9).

During planning we are decomposing the required recipes to individual activities but we can also connect different recipes via common events (one recipe produces the event and another recipe consumes the event). Still there could be some events that are consumed only (there is no action that consumes such event); these events may

correspond to purchases of raw material etc. Similarly, there could be produced only events, e.g. describing appearance of the final product. We call such produced-only and consumed-only events *one-way events.*

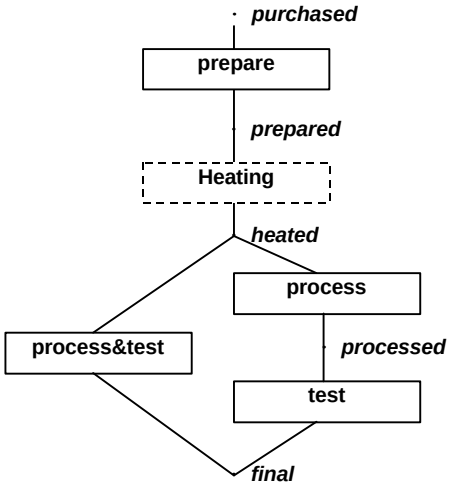

Figure 9. A recipe using another recipe (dashed).

If we expect that all the events have unique names then we can represent the recipe as a set of activity environments and recipes. In such a representation it is clear how the activities and recipes are connected via common events.

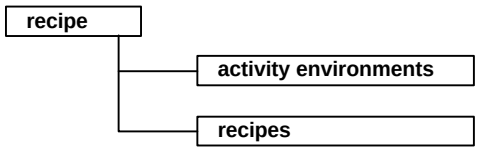

Figure 10. A basic structure of the recipe.

Problem Modelling

A domain model describes the problem area i.e. which resources are available, what activity types can be used, and what are the relations between the activities. To specify a particular problem we need to describe the actual activities. This could be done explicitly, like in traditional scheduling and timetabling, where the set of activities is given as the input and the task is to allocate the activities to resources respecting the resource and recipe (precedence) constraints. In traditional planning, the input consists of some events and the task is to generate the activities in such a way that the events are connected via activities i.e. the activities in the plan are described implicitly via the events. In our framework, we propose to combine both these ways of input specification, i.e., depending on the input we will solve either a pure scheduling (timetabling) problem or a pure planning problem or a mixture of both.

Initial data

If we are using resources in the problem, it is a good manner to describe the initial situation/state of each resource. In timetabling this is useless because there are no states. In pure scheduling this is done via specification of the activity with pre-allocation of the activity to the resource and to initial time.

In our framework we allow description of the initial state(s) of each resource as well as specification of activities that are known before we start scheduling. These activities may be pre-allocated, i.e., some of the parameters of the activity are known (like time and used resources) or the parameters are unknown and the task is to find their value (allocate the activity to resources and time). Using such initial data allows us to model pure scheduling and timetabling problems or to use the system to complete partially known schedules. In the second case, new activities are introduced during scheduling to fill gaps in recipes.

Goals

To further extend the planning features of the framework, we allow specification of known events in the description of the problem. Remind that the events make a connection node between the activities. If there appears an event in the system then this event must be produced by some activity and consumed by another activity. Only the one-way events may have either the consumer or the producer. To start planning, we can put some initial events to the system and the system will try to cover them, i.e., to find an action that produces the event and/or the action that consumes the event. Introduction of the action may cause introduction of new events and the task is to cover all the events. As we said above it means that there must be an action producing the event and an action consuming the event. A missing action (producer or consumer) in a one-way event is substituted by including the event among the initial events. Note that this process is similar to STRIPS planning where we have to find activities generating the final effects using the initial pre-conditions.

It is possible that some one-way events are introduced during the process of planning and these events are not included among the initial events. For example we can introduce an event describing a purchase of raw material. To allow such situation we can mark some one-way events as free events. Then, we can introduce a free event during planning if some activity requires it even if the event is not among the initial events.

To summarise the above paragraphs, the problem is described by specifying the domain (a problem area) and by describing some objects in the final schedule, namely some activities and initial events. The task is to fill the gaps in the schedule following the recipes and respecting the resource constraints (see Figure 11). It means that the resulting plan consists of the activities allocated to resources and connected with other activities via events.

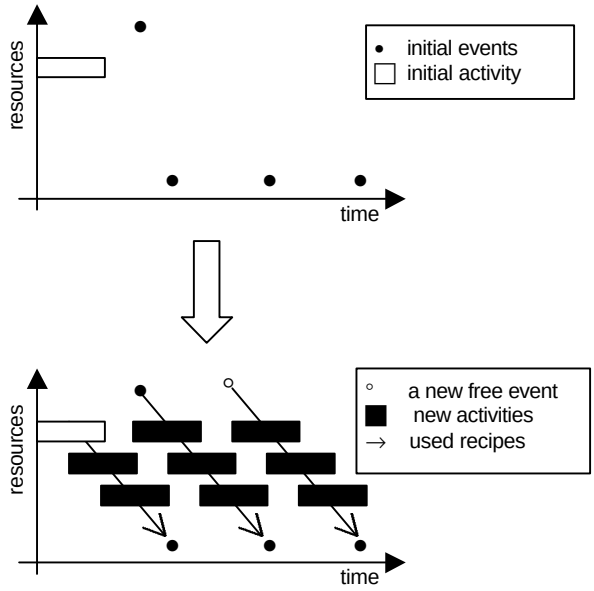

Figure 11. Gantt charts - from the problem description (top) to the solution (bottom).

Before Planning/Scheduling

When the problem is described formally, the next step is to solve the problem. However, we can look in data and insert some addition steps that may simplify the process of problem solving. Basically, we distinguish three steps that could/should be done before we start solving the problem:

- data checking that reports errors in data,
- data pre-processing that simplifies the model,
- data analysing that finds useful information for the solver.

Data Checking

The modelling framework may provide a formal language for problem description. This language is typically based on some underlying language like LISP or Prolog so we can use tools of the underlying language to ensure syntactic soundness of the model. Nevertheless, syntactic soundness does not guarantee that the model is semantically correct, i.e., that there are no bugs prohibiting finding a solution. Especially, when a less-experienced user designs the model we can expect many such bugs. Semantic bugs can be discovered during planning/scheduling but it is often a tough process leading to a very long computation (the system tries to find a solution even if "visibly" no solution exists).

Our and others [personal communication to Helmut Simonis] experience says that it is very important to check data before we start scheduling. Such data checking can be automated in some way, for example to discover (some) clashes in data. Some data-checking can be general, i.e., designed for all models. For example, we can check if resources required by activities are present in the model or if all states of the resource are accessible from the initial state. Note also that the data checker may identify parts of the model that could cause problems, i.e., it is not an error but it could be an error. For example, the graph of states for the resource consists of more components that are not connected etc.

Other data checks may be designed for particular instances of the modelling framework. Assume that time windows are defined for activities and for states of the resources. We can check if the activity can be processed by a given resource in a given state by comparing time windows of the activity and the state.

We mentioned just few data checks, many other checking techniques can be proposed for a particular class of models. Generally, data checking is not a complete technique that guarantees existence of the solution; otherwise complete data checker includes full planner/scheduler which is not the goal of data checking. The point is that as much as possible (polynomial) data checks should be done before we start (exponential) planning or scheduling.

Data Pre-processing

When the developer designs a formal model of a non-trivial problem then he or she should take in account the details of the solving algorithm. Sometimes the modelling language guides the user to design "reasonable" models but if the modelling framework is general like our framework then it is hard to integrate all good modelling skills into the modelling language itself. Moreover, the end users prefer the models that are close to what they know in reality rather than the models with "low-level" tricks that make the model easier for scheduling. Finally, the formal model of the problem may be designed automatically from an ERP system or a database describing the domain. All in all, the pure formal models may contain features that are sound but that make scheduling more complicated.

To remove "bad" features of the pure model we can smooth it out by applying some pre-processing techniques that change the model into a model easier for scheduling. The only requirement about the pre-processed model is that it must be equivalent to the original model. Such equivalence is defined in the following way: the post-processed schedule of the pre-processed model is a schedule of the original model (see Figure 12).

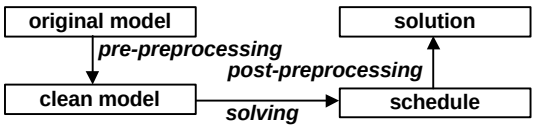

Figure 12. Using pre- and post-processor

Some pre-processing techniques change significantly the model, e.g. by using different structure of activities. For example, when the end user describes a resource using

states, he or she tends to describe all possible states, e.g. loading - processing - unloading. Or all these states are saved in a database describing the resource so it is natural that they appear in the automatically generated model as well. It implies that there must be loading, processing, and unloading activities as well. However, if such sequence is unique, then the experienced modeller abstracts from these states and uses just one abstract state/activity to describe the situation. Visibly, using just one activity is easier for scheduling then using three activities. Moreover, if we substitute such activity by a triple of activities in the final schedule then we get a schedule for the original problem so the conversion is sound.

Other pre-processing techniques are less invasive and they just remove some unfeasibility from the model. Assume that time windows are defined for activities and for resources (their states). If we know that some resource must be selected from the set of alternative resources then we can make an intersection of the time window for the activity with the union of the time windows of these resources to get a new time window for the activity. It means that some values may be removed from the time window for the activity, which decreases the search space.

Note finally that pre-processing is closely related to data checking so both techniques can be applied together.

Data Analysing

When we have a sound and complete formal description of the problem, there remains one "small" step - to solve the problem. There exist many solving algorithms for particular classes of planning, scheduling, and timetabling problems; the hard job is to choose the one that best fits the problem or to extract some information from data that the solver can use.

At top level, by analysing the data we can decide automatically whether the problem belongs to traditional planning (no resources and time), to traditional scheduling (all activities are known), or if it requires both approaches. In case of traditional problems we can further classify the problem. For example, in traditional scheduling there exists a Graham's classification of scheduling problems and a catalogue of efficient algorithms for solving these problems (Brucker 2001). Theoretically, if we classify the problem, which can be done by analysing the data, then we can find a solving algorithm automatically. Unfortunately, Graham's classification is rather academic so we can hardly expect that a real-life problem fits into a category in this classification. Still, it is possible to find sub-problems that can be solved using existing efficient algorithms and the rest of the problem is solved using some generic technique like constraint satisfaction.

Note that data analysis may be used also to find additional information for the solving algorithm. For example, we can go beyond simple activity joining described in the previous section and we can identify some required dependencies between the activities, so called landmarks (Porteous and Sebastia 2000). A planning algorithm can then use information about landmarks to improve its efficiency.

Conclusions

In the paper we describe an integrated framework for modelling planning and scheduling problems. We concentrate on an informal description of such a framework rather than on a precise specification of all the attributes and solving algorithms.

This paper extends the work from (Barták and Rudová 2001) in the way of more precise specification of objects, in particular recipes. We also separated the model of domain from the problem and we put our framework into a context of existing frameworks for planning (like HTN, STRIPS) and scheduling (resources). Finally, we showed how such a formal framework might automate some data processing before we start planning/scheduling.

Acknowledgements

The research is supported by the Grant Agency of the Czech Republic under the contract no. 201/01/0942. The author would like to thank Hana Rudová, Roman Mecl, and the team of VisOpt Ltd. for useful discussions concerning modelling real-life scheduling and timetabling problems. The author is also grateful to Ondrej Cepek for proof-reading of the paper draft.

References

Barták R. 1999. On the Boundary of Planning and Scheduling: A Study. In *Proceedings of the 18th Workshop of the UK Planning and Scheduling SIG*, 28-39, Manchester, UK.

Barták R. and Rudová H. 2001. Integrated Modelling for Planning, Scheduling, and Timetabling Problems. In *Proceedings of the 20th Workshop of the UK Planning and Scheduling SIG*, 19-31, Edinburgh, UK.

Binh Do M. and Kambhampati S. 2000. Solving planning-graph by compiling it into CSP. In *Proceedings of AIPS 2000*, 89-91.

Brenner M. 2001. A Formal Model for Planning with Time and Resources in Concurrent Domains. In *Proceedings of IJCAI-01 Workshop Planning with Resources*, Seattle.

Brucker P. 2001. *Scheduling Algorithms*. Springer Verlag.

Coddington A., Fox M., Long D. 2001. Handling Durative Actions in Classical Planning Frameworks. In *Proceedings of the 20th Workshop of the UK Planning and Scheduling SIG*, 44-58, Edinburgh, UK.

Erol K., Hendler J., and Nau D. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network

Planning. In *Proceedings of 2nd International Conference on AI Planning Systems*, 249-254.

Fikes R. and Nilsson N.J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2: 189-208.

Fox M. and Long L. 2001. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. Technical Report, Department of Computer Science, University of Durham, UK.

Ghallab M., Howe A., Knoblock C., McDermott D., Ram A., Veloso M., Weld D., Wilkins D. 1998. PDDL - The Planning Domain Definition Language, Tech Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Kingston J.H. 2001. Modelling Timetabling Problems with STTL. In *Proceedings of The Practice and Theory of Automated Timetabling*, 309-321, LNCS 2079, Springer Verlag.

Koehler J. 1998. Planning under Resource Constraints. In *Proceedings of 13th European Conference on Artificial Intelligence*, 489-493, Brighton, UK.

Laborie P. 2001. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results. In *Proceedings of 6th European Conference on Planning*, 205-216, Toledo, Spain.

Nareyek A. 2000. AI Planning in a Constraint Programming Framework. In *Proceedings of 3rd International Workshop on Communication-Based Systems*.

Porteous J. and Sebastia L. 2000. Extracting and Ordering Landmarks for Planning. In *Proceedings of the 19th Workshop of the UK Planning and Scheduling SIG*, 161-174, Milton Keynes, UK.

Smith D.E, Frank J., and Jónsson A.K. 2000. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review*, 15(1): 61-94.

Srivastava B. and Kambhampati S. 1999. Scaling up Planning by teasing out Resource Scheduling. Technical Report ASU CSE TR 99-005, Arizona State University.

Van Beek P. and Chen, X. 1999. CPlan: A Constraint Programming Approach to Planning. In *Proceedings of AAAI-99*, 585-590.

Wallace, M. 1994. Applying Constraints for Scheduling, in: *Constraint Programming*, Mayoh B. and Penjaak J. (eds.), NATO ASI Series, Springer Verlag.

Wren A. 1996. Scheduling, Timetabling and Rostering - A Special Relationship. In *Proceedings of The Practice and Theory of Automated Timetabling*, 46-76, LNCS 1153, Springer Verlag.