Slot Models for Schedulers Enhanced by Planning Capabilities

Roman Barták*

Charles University, Faculty of Mathematics and Physics
Department of Theoretical Computer Science
Malostranske namesti 2/25, 118 00 Praha 1, Czech Republic
bartak@kti.mff.cuni.cz
http://kti.mff.cuni.cz/~bartak

Abstract. Scheduling is one of the most successful application areas of constraint programming and, recently, many scheduling problems were modelled and solved by means of constraints. Most of these models confine to a conventional formulation of a constraint satisfaction problem that requires all the variables and the constraints to be specified in advance. However, many application areas like complex process environments require a dynamic model where new activities are introduced during scheduling. In the paper we propose a framework for constraint modelling of dynamic scheduling problems with activities generated during scheduling. We also show how some typical scheduling sub-problems like alternatives, set-ups and processing of by-products can be modelled in this framework.

1 Introduction

Planning and scheduling are closely related areas that attract a continuous attention of optimisation community for many years. Planning deals with finding plans to achieve some goal or, more precisely, with finding a sequence of activities that will transfer the initial world into one in which the goal description is true. Scheduling solves a bit different task of allocating the activities to available resource over time. Briefly speaking, planning helps answer the question, "What should I do?" while scheduling helps with the question "How should I do it?"

Due to above described differences, it is a common practice that planning and scheduling tasks are solved separately using rather different methods. There is also additional reason for such separation and that is the static formulation of the scheduling problem. We mean that both Constraint Programming (CP) and Operations Research (OR), which are the most widely used technologies in scheduling, are confined to a static formulation of the problem where all the variables and constraints are required to be known in advance. This restriction is perhaps the main reason why CP and OR are still rarely used in planning applications where it is impossible to predict which actions will be used in which combination [9].

* Supported by the Grant Agency of the Czech Republic under the contract no. 201/99/D057.

Despite the above differences, there exists a demand for integrating planning and scheduling into a single framework [1]. We may observe trends to convergence on both sides. Planning community tries to improve quality of plans by considering resource constraints [7] while scheduling community is exposed to problems that require adding some planning capabilities to schedulers [2]. We believe that Constraint Programming may play the integration here thanks to its nice modelling capabilities and despite the static formulation of a CSP [6,9]. Recent development of Dynamic CSP [11] or Structural CSP [8] shows that the constraint community is aware of dynamic real-world problems.

In the paper we deal with the scheduling problems that require an introduction of new activities during scheduling. Such situation may occur when the appearance of the activity depends on allocation of other activities. Examples of such behaviour may be identified in complex-process environments, where set-ups and transition activities must be considered as well as activities for processing by-products, i.e., products that are not ordered.

We propose a general framework for modelling scheduling problems with dynamic activities here. Because activities' generation is typical for planning, we speak about schedulers enhanced by planning capabilities. The proposed framework is based on the notion of activity slot that is a shell being filled by an activity during scheduling. Slot models are not new to the scheduling community but till now they were used in static form mainly in timetabling problems. We propose to extend the slot notion by allowing the slot to move over time or over resources. This gives us more flexibility when modelling problems where the time slots do not appear in the original problem formulation.

The rest of the paper describes the proposed framework in detail. In Section 2 we summarise the reasons for a dynamic problem formulation using the problems from complex-process environments. In Section 3 we overview the mixed planning and scheduling framework proposed in [1] and we describe its instance for schedulers enhanced by planning capabilities. Section 4 is dedicated to description of the slot model. We introduce the slot chains, and we classify the variables and constraints here. In Section 5 we sketch how to solve several problems that appear during scheduling with slot models. We conclude with summary of the results and future steps.

2 Static vs. Dynamic Problem Formulation

Conventional Constraint Satisfaction Problem (CSP) is defined statically, i.e., all the variables and constraints are expected to be known in advance. Therefore, most methods for solving CSP follow the schema:

1) introduce all the variables,
2) post all the constraints,
3) label all the variables respecting the constraints.

Naturally, this schema is reflected in constraint models of real-world problems, i.e., we expect all the entities (variables) and relations among the entities to be specified before we start solving the problem. Because neither the variables nor the constraints are changing during solving the problem, we speak about the *static problem formulation*.

A pure scheduling problem, i.e., the allocation of known activities to available resources over time, can be formulated statically in above sense. Each activity is described by a set of parameters/variables and the scheduling task is to find values of these parameters, typically to find when and in which resource the activity is processed. Before the scheduling starts, the parameters are bound by posting constraints restricting the possible combinations of parameters' values.

Naturally, the complexity of the constraints depends on the problem to be solved. Typically, there are precedence constraints between activities of the same task describing the order in which the activities must be processed (job-shop scheduling):

$$start(act1)+duration(act1) \leq start(act2)$$

We may also describe when the activity is allowed to be processed (scheduling with time windows):

$$min\_time \leq start(activity)$$
$$max\_time \geq start(activity)+duration(activity)$$

Usually, there are also resource capacity constraints specifying how many activities can be processed in parallel by single resource (resource-constrained project scheduling [5]). It may seem that these constraints are not static because their appearance depends on allocation of the activities to the resources. Nevertheless, by adding a "trigger" we may express the constraint statically:[1]

$$resource(act1)=resource(act2) \Rightarrow$$
$$start(act1)+duration(act1) \leq start(act2)$$
$$\vee$$
$$start(act2)+duration(act2) \leq start(act2)$$

Using triggers allows expressing rather complicated constraints, for example by extending the above capacity constraint we may describe the minimal transition duration between activities scheduled to given resource.

There are two main advantages of static formulation of the problem. First, because the static formulation confines the conventional CSP solving process, it is possible to use almost all methods developed for solving CSP including popular local search. This is important especially in large-scale scheduling problems where local search and similar methods proved themselves to be very efficient.

The second advantage of static formulation is the possibility to exploit fully the power of constraint propagation. For example, if we know that there is an overlap between two activities and all the resources have capacity 1 then we may deduce immediately that these two activities cannot be scheduled to a single resource. This is "discovered" using the propagation via above described resource capacity constraint. For scheduling applications several special propagation techniques were proposed to prune the domains even more, like edge finding.

We see one big drawback of static description of constraints with triggers. Some complex real-life relations must be described using very complicated "triggered" constraints. This decreases the readability of the model and it also eliminates the advantage of constraint propagation. Triggered constraints correspond in fact to

[1] The example constraint describes a single capacity resource.

disjunctive constraints and it is a known wisdom that the propagation through disjunction is not very powerful.

To simplify expressing of triggered constraints, it is more appropriate to introduce such constraints during scheduling. If we introduce the constraint as soon as possible we may preserve the advantage of constraint propagation while keeping the constraint in simple form in comparison to equivalent triggered constraints. For example, the above described capacity constraint may be simplified to the form:

$$\texttt{start(act1)+duration(act1)} \leq \texttt{start(act2)}$$
$$\vee$$
$$\texttt{start(act2)+duration(act2)} \leq \texttt{start(act2)}$$

and this constraint is introduced as soon as we know that both activities are allocated to a single resource.

In a *dynamic problem formulation* we allow adding new constraints as well as new variables during scheduling. In particular we mean adding new activities that corresponds to adding chunks of variables and new constraints binding these variables and connecting them to other variables (activities) already in the problem specification. This approach should not be confused with the dynamic constraint satisfaction [11] that tries to revise a current variable assignment with given changes in the constraint graph. In the slot models described further we are trying to refine the given constraint graph by adding more variables describing the activities. From this point of view, our approach is closer to structural constraint satisfaction [8].

2.1 When the dynamic problem formulation is appropriate?

We study the dynamic problem formulation mainly because of real-life scheduling problems in complex-process environments that cannot be formulated statically or the static formulation is too complicated and inefficient.

First typical problem in complex-process environment is existence of many alternative activity chains. Typically, one of these alternatives is chosen during planning so the scheduler gets a “fixed’ sequence of activities to be allocated. However, as noted in [1] this approach requires the planner to have information about possible activity allocation because otherwise, the plans could be too tighten or too relaxed. There is an attempt to postpone choosing the alternative activities to the scheduling stage. In [3], a method of scheduling alternative activities is proposed. This method requires all the alternative activities to be generated in the form of a process plan. Unfortunately, when the number of alternatives is very high, like in complex-process environments, then the process plan is huge. Therefore we believe that generating activities during scheduling is more appropriate in such cases.

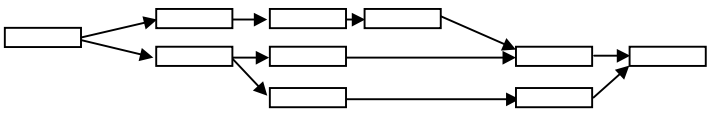

Fig. 1. A process plan with alternatives. Each rectangle corresponds to an activity and the task is to choose the path from the leftmost to the rightmost activity.

Another problem that can be identified in chemical, food or steelmaking industries, is necessity to insert special set-up, cleaning, re-heat [10] or transition activities between production activities. In most current scheduling systems, this is modelled by using transition times between two consecutive activities. Unfortunately, this model cannot be used when these set-up and transition activities produce some low-quality products that must be stored or consumed by other resources[2]. Another example is inserting re-heat or cleaning activity that depends on allocation of other activities. Note that such situation cannot be identified during planning stage that generates the activities because the appearance of the activity depends on the allocation of other activities. In such case, we really need to introduce a new activity during scheduling.

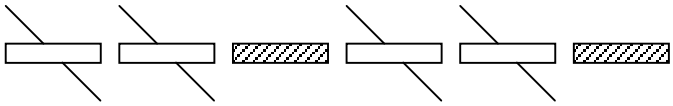

Fig. 2. Cleaning activity (stripped) is inserted after each pair of production activities even if it is not part of any activity chain. The appearance of the cleaning activity depends on the allocation of other activities to the resource.

Last but not least there is a non-ordered production. In complex-process environments there exist plants where the ratio of ordered production is very low, say less than 20%, and the remaining production is driven by a marketing forecast. It is possible to introduce virtual orders describing the marketing forecast so the planner generates activities according to these orders. However, there is a danger that the marketing plan is too tighten and it cannot be scheduled together with given set of orders or the production cost is too high (e.g. due to many expensive set-ups). Therefore it seems to be more appropriate to postpone decision about non-ordered production until the scheduling stage when we may choose from several alternative marketing process plans. For example we may schedule continuation of a production of some item even if there is no demand for the item because it is less expensive to continue in the production than stopping the machine.

3 Integrated Planning and Scheduling

In above paragraphs, we described several examples when generating new activities is appropriate to model some problems and to get better schedules in terms of production cost (details can be found in [2]). Because generating activities is a typical planning task we are speaking about integrated planning and scheduling or, more precisely, about enhancing schedulers by some planning capabilities.

---

[2] Typically, these so-called by-products are not assumed in scheduling systems, which may cause problems with storing if a big quantity of by-products appear. Moreover, the by-products can be used in further production as raw material that may decrease the production cost.

In [1] we proposed a general framework for mixing planning and scheduling. The basic idea of this framework is not very complicated, the planning module generates activities and these activities are immediately passed on the scheduling module that tries to allocate them. The scheduling module uses the constraint propagation to prune the domains of activity variables so the planner can exploit this additional information about partial allocation of activities during generating new activities. Moreover, the scheduler may ask explicitly the planner, to introduce new activities.

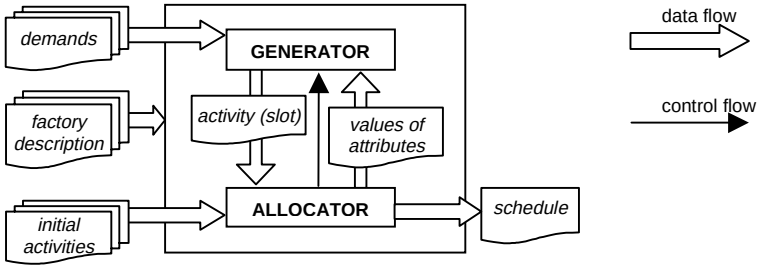

Fig. 3. The structure of mixed planning and scheduling system

The schema of mixed planning and scheduling system described in previous paragraphs is very general and it can be used both for planning under resource constraints and for scheduling enhanced by planning capabilities. By scheduling enhanced by planning capabilities we mean the schedulers where it is possible to introduce new activities during scheduling. If we use a slot model (see later) then we need not a special activity generator; the activity is introduced to the system simply by choosing the value for the Activity variable in each slot. Consequently, we may use the same solving mechanism for the activity allocator and for the activity generator, in particular constraint programming technology.

4 Slot Models for Dynamic Activities

Slot models are not new to the scheduling community, however their usage is restricted to problem areas containing slots in the original problem formulation like timetabling and personnel rostering applications. In these applications, the slot corresponds to a box in a resource and in a fixed time period. The slot is being filled by an activity, person etc. during scheduling. For example in school timetabling, the slot describes some time period, say 9:00-10:00 in the classroom and it is being filled by a particular lecture and lecturer during scheduling. For these applications, it is typical that there are no connections/relations between the slots but the relations are defined between the activities only.

We propose to extend the flexibility of slot models by unsticking the slot from a particular resource and from a time period. This allows us to model wider class of scheduling problems, in particular problems with dynamic activities introduced during scheduling. It may seem that after the proposed generalisation the slot becomes

equivalent to an activity, but this is true only to some extent. Slot is still a box that is being filled by an activity during scheduling. Now, we are allocating the slots to resources over time together with filling the slots by activities.

By using the slots instead of activities we can generate the slots in advance before the scheduling starts. This gives some static character to the dynamic problem so we may exploit advantages of the static problem formulation like constraint propagation. Naturally, we need know how many slots should be generated.

4.1 Slot Chains

In a typical scheduling problem the activities are grouped in a sequence, for example in a process plan describing the sequence of activities necessary to produce an ordered item. We may group the slots in the same way so we are working with slot chains instead of individual slots that are not connected. The important thing about the slot chain is that we may estimate its length or more precisely, upper bound of the length.

In [4] two different activity groupings are identified, grouping per task and grouping per resource. In [2] we described the criteria for choosing the right grouping for particular problem using the classification of constraints. The slot models proposed in this paper can be mapped to both groupings as we show below.

When the activities are grouped per task (a task-centric model) we may have several alternative process plans, but all these plans are finite. Thus, we simply take the length of the longest process plan as the number of slots to be generated. In the task-centric model, the number of tasks is known in advance so we also know the number of slot chains to be introduced because each slot chain corresponds to the task. Note also that the slot chains may have different lengths.

If the activities are grouped per resource (a resource-centric model) then we may calculate the upper bound of the number of slots by dividing the scheduled period by the minimal activity duration. Note also that the knowledge about the initial activity (the activity processed by the resource at the beginning of the schedule) and about the activity transitions can be used to further decrease the upper bound. The number of slot chains in the resource centric model corresponds to the number of resources[3].

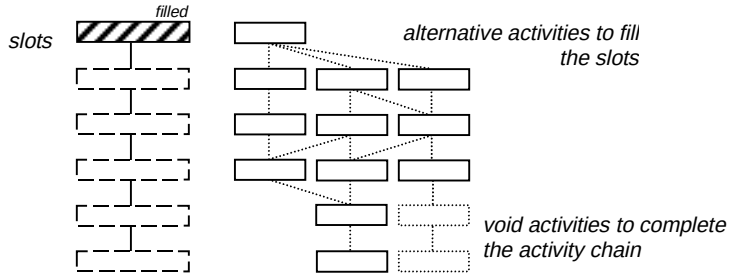

Fig. 4. The slot chain corresponds to the longest chain of activities in the production plan.

[3] Resources with capacity greater than one may be modelled either by several slot chains (the number of chains per resource corresponds to the capacity) or by longer single chain (the length is multiplied by the capacity) or by allowing slots to be filled by more activities (batch processing). Currently we are using the batch-processing model.

Naturally, because the number of slots in the chain equals to the longest activity chain it may happen that after the scheduling, some slots remain empty (if a shorter activity chain is chosen). We require these empty slots to be gathered at the end of the chain so the slot chain is filled uniquely by a given activity chain. This reduces the number of combinations how to fill the slots by activities. Extending each activity chain to the length of the longest activity chain by adding void activities at the end can solve the difficulty with activity chains of different length. The void activity has a zero cost and as soon as a slot is filled by a void activity, all remaining slots are filled by void activities too. We do not need any special mechanism to ensure filling slots by void activities; this is done automatically using propagation via chain constraints (see later).

4.2 Slot Parameters

One of the main reasons why we propose to work with slots instead of with activities directly is that we can move some activity parameters to the slot. This is very important because in the dynamic problem formulation we do not know the activities in advance so we cannot post constraints among the activity parameters until the activity is introduced. However, when (some of) these parameters are moved to the slot, we may post the constraints in advance because all the slots are generated before the scheduling starts. The question is which parameters should be moved to the slot and which parameters should stay in the activity.

The answer is not so complicated. We prefer as many as possible activity parameters to be expressed in the slot because we can post the constraints among them immediately. Consequently, we propose to take an intersection of parameters in all activities as the parameters for the slots. Moreover, each slot has a special parameter Activity whose domain corresponds to identifications of the activities that can fill in the slot. The remaining parameters of activities are introduced when the Activity parameter becomes ground (when the value is chosen for this parameter). There is a special dynamic section for these parameters in each slot.

Fig. 5. The decomposition to static and dynamic parameters of the activities

To summarise it, we classify all the activity parameters into static and dynamic ones. The static parameters are present in each activity; they typically include the resource to which the activity is allocated, the start time and the duration of the activity. These static parameters are represented in each slot. The dynamic parameters are unique for each activity and they are filled in the slot when the activity in the slot is known. The structure of items processed by the activity is a typical example of a dynamic

parameter because this structure can be different in different activities. By the structure of items we mean a list of item quantities with identification of each item.

It should be noted that we could represent the dynamic parameters statically too, if we take a union of all parameters in all activities as the slot parameters. Then, if the activity in the slot does not use some of the parameters then these parameters are ignored. However, if the set of dynamic parameters is too big then their static representation is much larger than the dynamic representation and, thus, the static representation is inefficient. For example if there are thousand items processed by several activities in different combinations then we need thousand parameters to describe the item quantities statically.

Finally, it should be said that if all the activities share the same parameters then we might have an empty dynamic section in the slot. Still, the slot is different from the activity because we do not know which activity is in the slot till the domain of Activity variable becomes singleton.

4.3 Constraints

When the variables in slots are defined we may introduce the constraints among them. We propose to classify the constraints in three groups according to the function of a particular constraint in the slot model. This classification helps us to map particular problem to the slot model and it simplifies understanding the differences between posting the constraints.

Slot constraints (intra-slot constraints). First, there are constraints binding variables in a single slot. Usually, these constraints take care of filling the slot by an activity so they bind the Activity variable with remaining variables in the slot. For example if different sets of resources are defined for different activities then we need to bind Activity and Resource variables. Similarly, if the processing duration of the activity depends on the resource to which the activity is allocated then we need to bind Activity, Resource and Duration variables. Notice that this constraint still includes the Activity variable because we do not know which activity will be filled in the slot. We may define the slot constraints in a simple triggered form:

```
Activity = a1 ⇒ Resource in {r1,r2,r3}
Activity = a2 ⇒ Resource in {r3,r4,r5}
```

or we may use a general (global) relation:

Resource Activity	r1	r2	r3	r4	r5
a1	1	1	1	0	0
a2	0	0	1	1	1

Naturally, the slot constraint may also describe the relations that do not involve the Activity variable like the following example shows:

```
Start + Duration = End
```

Chain constraints (inter-slot constraints or intra-chain constraints). The second group of constraints corresponds to relations between variables from different slots of a single slot chain. Typically, these constraints bind two successive slots in the slot chain and they describe the precedence relations:

$$\texttt{slot(i).Start + slot(i).Duration} \leq \texttt{slot(i+1).Start}$$

or the transition patterns/process plans:

$$\texttt{slot(i).Activity = a1} \Rightarrow \texttt{slot(i+1).Activity in \{a3,a4\}}$$

We may expect that these constraints bind static variables mainly so the constraints can be posted before the scheduling starts. Consequently, the propagation through slots in the chain is ensured. This is very important because we can fill the slots in arbitrary order and the propagation guarantees that we can still fill the remaining slots. Finally note that we do not restrict the chain constraints to bind successive slots in the chain only but arbitrary slots may be connected. This allows us to define more complicated chain structures (not only the linear structure). Again, if the chain constraint binds any dynamic variable then this constraint can be posted after the dynamic variable is created in the slot.

Inter-chain constraints. The last group contains the constraints binding slots from different chains. These constraints express the dependencies between different slot chains, i.e. between different tasks, if the slot chain represents a task, or between different resources, if the slot chain corresponds to a resource. For example if we schedule an activity to two different resources (in two slot chains) then we may require the activity to be processed at the same time by both resources:

$$\texttt{chain(i).slot(k).Activity = chain(j).slot(l).Activity} \Rightarrow \texttt{chain(i).slot(k).Start = chain(j).slot(l).Start}$$

To define such inter-chain constraint we must first identify the slots to be connected. In the timetabling applications, where the slots are fixed to a resource and to time, there is no problem to find the connected slots. For example, it is easy to define the constraint forbidding the lecturer to teach in two different rooms at the same time slot. However, in our slot model, the slots are much more variable so we cannot post the inter-chain constraints until the slots are at least partially filled. Thus, the inter-chain constraints are introduced during scheduling and they are dynamic. We sketch the mechanism of posting such constraints in the next section.

In [2] we proposed a general grouping of constraints for scheduling problems, in particular we distinguished between:

- *resource constraints* describing limitations of single resource in given time, e.g. restricted capacity,
- *transition constraints* describing relations between states of single resource in different time points, e.g. transition patterns,
- *dependency constraints* describing relations between different resources, e.g. supplier-consumer relation.

We can now define the mapping between the classification in resource-centric and task-centric representation and the constraint groups in the slot models.

Table 1. Mapping between the slot model and resource-centric and task-centric representations

Slot models	Resource-centric representation	Task-centric representation
Slot constraints	Resource constraints	Resource constraints
Chain constraints	Transition constraints	Dependencies
Inter-chain constraints	Dependencies	Transition constraints

5 Scheduling with the slot models

Having a declarative model of the real-life problem is only the first step; we also need an efficient solving mechanism. Because we are modelling dynamic problems with changes in the constraint network we choose the traditional CP technique where the constraint propagation interleaves with labelling (it is not clear how the local search and similar methods can be applied to such dynamic problems). We do not describe all the details concerning the propagation techniques and the labelling strategies here (this will be a subject of a separate paper). We concentrate on two specific features of dynamic slot models: how and when to create the dynamic variables and how and when to post the dynamic constraints.

Generating dynamic variables: Because the set of dynamic variables may differ from activity to activity we decided to introduce dynamic variables as soon as the Activity variable in the slot becomes ground (its domain becomes singleton). It is natural to use event-based programming to implement introduction of dynamic variables using the same mechanism as constraint propagation is implemented (typically, the propagation is waked-up when the domain of some constrained variable is changed). Note also, that the introduction of new variables during scheduling changes the constraint network.

Posting the constraints: Because of constraint propagation, that reduces domains of variables and prunes the search space, we prefer to post the constraints as soon as possible. Naturally, we can post all the constraints binding the static variables in advance (if we know which variables are bounded). However, we would also like to keep the constraints in a simple form (as discussed in Section 2) which may lead to postponing the constraint posting. We can identify two main reasons leading to postponing the constraint introduction:

- First, the constraint includes some dynamic variables. Such constraint must be postponed until the dynamic variables are introduced to the problem and we may post this constraint together (immediately after) with creating the dynamic variables.

- Second, it is not clear which variables the constraint should bind. This is a typical case of inter-chain constraints where we do not know which slots should be connected by the constraint. There are two extreme methods for posting such constraints (and many variants in-between): eager and lazy method.

- ✓ *Eager method* corresponds to the static formulation of triggered constraints and it recommends posting the constraints immediately, i.e., we post the constraint between all the possibly connected slots and we use the trigger to activate the constraint. As discussed in Section 2, if the trigger is not too complicated then this method ensures good constraint propagation like the following inter-chain constraint:

```
chain(i).slot(k).Resource = chain(j).slot(l).Resource
   ⇒
    chain(i).slot(k).(Start+Duration) ≤ chain(j).slot(l).Start
       ∨
    chain(j).slot(l).(Start+Duration) ≤ chain(i).slot(k).Start
```

- ✓ *Lazy method* recommends to wait until we know which slots should be connected, i.e., until we have enough information about the values of variables. The advantage of the lazy method is that it generates only necessary constraints (it needs no disjunction), unfortunately, the constraints are generated too late (in reality we must wait until the variables are ground) and, thus, they do not contribute to pruning the search space. In fact, the constraints posted by lazy method behave like a test only. Lazy method is implemented using the event-based programming, where the event corresponds to the condition when the constraint should be posted.

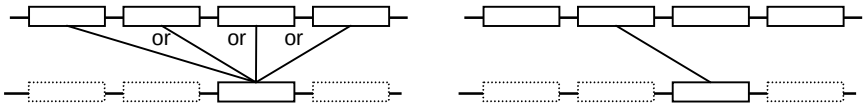

Fig. 6. From eager (left) to lazy (right) method of posting the constraints

The choice of the method depends on a particular problem to be solved. It is even possible to mix both lazy and eager methods via encoding a part of the constraint trigger into the event and thus simplifying the constraint.

6 Conclusions

In real-life industrial scheduling there exists problems that require adding new activities during scheduling. Because the conventional static constraint models are not able to handle such problems we proposed slot models for scheduling problems that require some planning capabilities. This enables us to formulate and solve wider range of real-life problems using the constraint programming technology.

The proposed slot models are based on generalisation of slot models used in timetabling applications. We use the slot as a shell that can be filled by an activity and that encapsulates common parameters of activities. We also propose the classification of constraints using their function in the slot models. Using dynamic variables and constraints simplifies expressing of some constraints in comparison with their static formulation. However, we preserve the advantage of constraint propagation by

posting the constraints as soon as possible. This is the advantage of the slot model over the fully dynamic model where activities are generated completely dynamically.

The slot model makes the core of the scheduling engine implemented as part of the VisOpt scheduling project. It proved itself to be enough general to cover various real-life scheduling problems in complex-process environments.

We are currently working on improving the efficiency of the implementation mainly by using special labelling strategies, by more tighten co-operation between constraint propagation and labelling and by more sophisticate definition of the constraint triggers that decrease the number of disjunctive constraints.

Acknowledgements

Author's work is supported by the Grant Agency of the Czech Republic under the contract number 201/99/D057 and by InSol Ltd. I would like to thank Yossi Rissin and the team of InSol for introducing me to the problem and for interesting and encouraging discussions concerning real-life problems of industrial planning and scheduling. I am also grateful to Helmut Simonis from Cosytec for useful discussions concerning constraint scheduling.

References

1. Barták, R.: On the Boundary of Planning and Scheduling: A Study. *Proceedings of the Eighteenth Workshop of the UK Planning and Scheduling Special Interest Group*, Manchester, UK (1999) 28-39
2. Barták, R.: Dynamic Constraint Models for Planning and Scheduling Problems. *Proceedings of the ERCIM/CompulogNet Workshop on Constraint Programming*, LNAI Series, Springer Verlag (2000), to appear
3. Beck, J.Ch. and Fox, M.S.: Scheduling Alternative Activities. *Proceedings of AAAI'99*, USA (1999) 680-687
4. Brusoni, V., Console, L., Lamma. E., Mello, P., Milano, M., Terenziani, P.: Resource-based vs. Task-based Approaches for Scheduling Problems. *Proceedings of the 9th ISMIS96*, LNCS Series, Springer Verlag (1996)
5. Caseau, Y., Laburthe, F.: A Constraint based approach to the RCPSP. *Proceedings of the CP97 Workshop on Industrial Constraint-Directed Scheduling*, Schloss Hagenberg, Austria (1997)
6. Joslin, D. and Pollack M.E.: Passive and Active Decision Postponement in Plan Generation. *Proceedings of the Third European Conference on Planning* (1995)
7. Koehler, J.: Planning under Resource Constraints. *Proceedings of 13th European Conference on Artificial Intelligence*, Brighton, UK (1998) 489-493
8. Nareyek, A.: Structural Constraint Satisfaction. *Proceedings of AAAI-99 Workshop on Configuration*, 1999
9. Nareyek, A.: AI Planning in a Constraint Programming Framework. *Proceedings of the Third International Workshop on Communication-Based Systems* (2000), to appear
10. Pegman, M.: Short Term Liquid Metal Scheduling. *Proceedings of PAPPACT98 Conference*, London (1998) 91-99
11. Verfaillie, G. and Schiex, T.: Solution Reuse in Dynamic Constraint Satisfaction Problems. *Proceedings of the 12th National Conference on Artificial Intelligence AAAI-94*, USA (1994), 307-312