Modular synthesis of efficient schedules in a timed discrete event plant *

R.K. Boel
SYSTEMS Group,
Electrical Engineering Department,
Universiteit Gent,
Technologiepark Zwijnaarde 9,
B-9052 Gent, Belgium.†

F.J. Montoya
Depto. de Informática y Sistemas,
Facultad de Informática,
Universidad de Murcia,
Campus de Espinado,
E-30071, Murcia, Spain.‡

Abstract

This paper treats optimal scheduling in large timed discrete event systems as a supervisory control problem. Scheduling tasks in a steel plant is treated as a realistically sized case study. A sequence of tasks must be completted in as soon as possible, while satisfying all the constraints in the model. These different constraints are specified via different components in a modular plant representation. Components can be represented as timed Petri nets, leading to a graph of interacting modules. The acyclic nature of the graph consisting of the most critical components is exploited in order to find a heuristic but fast way of searching through the very large set of feasible orderings.

Keywords: Scheduling, supervisory control, timed discrete event systems, timed Petri nets

1 Introduction

Timed discrete event systems describe the evolution of physical plants, usually at a high level of abstraction, by specifying the points in time when certain events are allowed to happen. Whenever an event takes place the state of the system changes to a different value. Examples of such timed discrete event models are timed automata [1] and timed Petri nets [4]. Supervisory control of timed discrete event systems can delay or completely prevent the execution of controllable events (similar to the event disabling in supervisory control of untimed discrete event systems) but in this paper we also assume that for some events the controller can force enabled and controllable events to be executed at some time instant.

Scheduling sequences of tasks is a typical example of supervisory control as introduced by Ramadge and Wonham [6]. Control decisions for scheduling consist in selecting where to execute tasks, and when to start these tasks. The goal of scheduling is the minimization of the time needed for executing a given set of tasks. The sequence of tasks should satisfy logical and temporal constraints. Some of these constraints are local and are encoded in the components of the timed discrete event plant. Other constraints are logical specifications such as precedence rules, bounds on time delays, etc.

It is possible in principle to solve this problem by backward recursion using dynamic programming [2]. However the size of the state space grows very fast for timed discrete event systems, and hence it may not be feasible to store the value function efficiently. One can also try forward recursion to describe the tree of reachable states starting from the present (initial) plant state, and then search by enumeration for the optimal path in this tree. In either case, using backward or forward recursion, experience shows that the problem quickly become computationally intractable because of the explosion of the number of states to be considered.

In this paper we describe an algorithm for finding the optimal schedule which searches the complete state space in such an order that it is very likely that the optimal solution is found quickly. We observed that in the case study of a steel plant the most critical constraints for the schedule are described by three components which form an acyclic graph. The first compoent describes the arrival of a batch of steel at the convertor, the conversion and the first stage of treatment. The scheduling decisions here are the starting time of conversion of a new batch, and which convertor to use. Output of this component then goes to the further stages of metallurgical treatment, where different qualities of steel require different routes through the plant and dif-

*Research supported in part by the TMR project NCN - Research Network ERB FMRXCT-970137, and by the ESPRIT project VHS - Verification of Hybrid Systems. Part of the results in this paper were based on research carried out by the first author within the framework of the Belgian Program on Interuniversity Attraction Poles, Prime Minister's Office, Science Policy programming. The scientific responsibility rests with its authors.

†Senior Research Fellow, F.W.O.-Vlaanderen, E-mail: rene.boel@rug.ac.be.

‡E-mail: fmontoya@dif.um.es

ferent execution times for the operations. In this second stage of operations the scheduler must select which one of the possible positions for treatment will be used, and which batch should receive priority in case of resource conflicts. Finally the output of this second stage of operation is sent to the continuous casting machine, where the speed of casting can be selected by the scheduler within specified bounds. Each of these three components can be represented by a timed automaton or by a timed Petri net. These components form an acyclic graph in the sense that inputs to component n are outputs of components $l < n$.

Our algorithm for finding an optimal schedule works as follows. The variable to be optimized is the time at which the last batch leaves the last component in our plant model - the continuous caster in our case study. Hence we can search for an optimal schedule as follows. First find an optimal schedule, and its associated cost, in the last component, e.g. via an exhaustive search of the reachability tree. Since each component separately is relatively small this exhaustive, forward search may be computationally feasible. The optimal cost which is found depends on the timing of the output events of the preceding component. It can be interpreted as a value function in dynamic programing, and we call it the "value at component n" from now on. Hence it becomes possible to find an optimal schedule in this preceding component by using the same method for the plant model where the last component is deleted, and where the new cost criterion is the "value function". This local optimisation in turn defines a "value function" at the output of stage $n-2$. This is repreated for all components in the acyclic graph. This algorithm combines backward recursion between components and forward search within each component.

In practice there are always some modules which have input from downstream modules and which send output to upstream modules. For example in the steel plant case study we still have to represent constraints on the availability of the transportation system, and on the availability of empty stee ladles. Components of the plant model representing these additional constraints have inputs from and outputs to all of the other components, making the overall plant model cyclic. It is therefore necessary to verify whether the proposed schedule based on the acyclic subgraph, remains feasible for the full, cyclic, model. Iterative improvements, as explained below, are then necessary. The quality of our algorithm depends on how well the search algorithm selects new schedules in case the proposed schedule is not acceptable. In section 3 we explain some examples of techniques that can be used in order to find a valid and optimal schedule in a computationally efficient way.

This paper is structured as follows. Section 2 explains briefly how our modular algorithm for synthesizing an optimal schedule works, using a steel plant as a case study; timed Petri net models are used to represent the different components of the steel plant. In section 3 we use these models to describe in more detail how the schedule is synthesized. In section 4 we draw some conclusions, and state some open problems.

2 Case study: representation of the steel plant

The research presented in this paper is based on a real life case study of scheduling tasks in a steel plant. The layout of the steel plant is represented in figure 1. It consists of two convertors, several positions for metallurgical treatment, a continuous casting machine, two overhead cranes, a transport and storage system for empty steel ladles, and 4 tracks connecting neighbouring positions for metallurgical treatment.

The operations of the steel plant must be scheduled so that it produces in the correct order a prespecified sequence of batches of steel of different qualities (each batch characterised by a recipe and by a minimal and maximal casting times). The optimal schedule should minimize the completion time of the last batch in the sequence, while satisfying all model constraints. The treatment of batches can be subdivided in 3 consecutive phases:

- conversion and initial stage of metallurgical treatment,
- metallurgical treatments specific to the quality of steel (recipe) to be produced,
- continuous casting of the batches.

Each of these phases naturally corresponds to one module in the mathematical model of the next section. The three modules clearly are interconnected in an acyclic way. Besides this acyclic graph of modules, the plant model also consists of two other modules representing the transport of empty steel ladles, and the movement of the two cranes (running on the same rail and hence subject to "no collision" constraints) which transport the steel ladles between different position.

Each of the components of this plant model can be represented as a timed petri net with input and output events. We assume the reader is familiar with Petri nets, a graphical representation of a discrete event system. It consists of places and transitions, connected via arcs from a place to a transition or from a transition to a place. Places can contain tokens, representing in this case the presence of a batch in position corresponding to the place or the location of a crane. The marking of places by tokens represents the state of the plant. A Transition t can fire if all the upstream places

of t contain at least one token. The transition (and the corresponding event) is then said to be enabled. The firing of a transition removes a token from each of its upstream places, thus modelling the event changing the state of the plant. Timed Petri nets associate clocks to each transition, and impose the limitation that an event must be executed within a certain time interval after the corresponding transition becomes enabled. Depending on the case these intervals can be specified by lower bounds only (the event is never forced to happen) or upper bounds (the event is forced to happen if the upper bound is reached) or both.

The first, upstream module in the plant model describes the operation of the two convertors. Convertor $i = 1, 2$ starts producing a batch of raw steel for the k_i-th time at θ^{conv}_{i,k_i}. This is a controllable variable, to be selected by the scheduler. Several constraints have to be met by the convertor operations: there is a minimal time interval between the end of one batch and the start of the next batch on the same convertor; during part of the conversion a resource is needed which cannot be used by both convertors at the same time (mutual exclusion); after n consecutive conversions (with n lying in a specified interval) on convertor i this convertor has to be cleaned. As soon as conversion of batch (i, k_i) is completed the raw steel is poured in an empty ladle (provided the empty ladle module has put an empty ladle on the car on track i by that time). This steel ladle is then transported by the car on track i to position $(i, 1)$ for a metallurgical treatment of duration in $[L_{mt1}, U_{mt1}]$.

This first component can be modelled as a timed Petri net, with input events whenever the conversion of a new batch of steel is started, and with output events each time a batch completes the first stage of metallurgical treatment at the time θ^{first}_{i,k_i}. At that time the scheduler has to decide which recipe $\hat{q}_n \in Q$ will be applied for batch $n = k_1 + k_2$, where k_2 denotes the number of batches produced on convertor 2 before k_1 batches were completed on convertor 1, or vice versa. Note that batches have to reach the continuous casting machine according to a prespecified sequence of recipes $q_n \in Q$, (since no overtaking is possible in the continuous casting component) but that it is possible that $\hat{q}_n \neq q_n$ because batches of different qualities can require very different execution times in module 2, and hence overtaking is possible in module 2. The choice of $\hat{q}_n$ specifies the sequence of metallurgical treatments to be carried out on the n-th batch in positions $(1, 2)$ (machine ♯2 in fig. 1), $(1, 3)$ (machine ♯3 in fig. 1), and $(2, 2)$ (machine ♯5 in fig. 1). The selected recipe q_n also specifies the duration of the operations in each position. Some tasks can be carried out interchangeably on $(1, 2)$ and $(2, 2)$, but $(1, 3)$ carries out specialised treatments. Transfer of a steel ladle from position $(i, 1)$ (machine ♯1, resp. ♯4 in fig.1) to position $(j, 2)$ requires the availability of a crane, whether $i = j$ or not. Transfer from $(1, 2)$ to $(1, 3)$ and vice versa does not require a crane; moreover there cannot simultaneously be a steel ladle in $(1, 2)$ and in $(1, 3)$. The scheduler in module 2 not only must select the recipe to be used for each arriving steel ladle, but it also must select for each of the interchangeable tasks whether it is to be executed on $(1, 2)$ or on $(2, 2)$.

Fig. 2 presents a timed Petri net model for module 2. In order to obtain a more compact model a "colour" is associated with each token in this Petri net. In order to keep track of the recipe selected for each batch, the token corresponding to this batch is assigned a color $\hat{q}_n$, corresponding to the quality of steel which will be produced by this batch. The duration of treatments in the positions in module 2, and the path followed by the n-th token (= the n-th batch) depends on the colour. Each batch entering module 2 at time $\theta^{first}_{i,k}$ is put in a choice place, where it leaves as soon as the selected position for its next metallurgical treatment becomes available. A batch can enter a position only provided the car on the corresponding rail is available in that position. This means that no other batch is in a position along the same track. This requirement is not shown in fig. 2. It must be added via cycles in the Petri net, ensuring a state invariant corresponding to there always being one token (one car) on a track.

This model of module 2 is fairly complicated, containing several choice places and several synchronising transitions (several internal synchronisations representing resource availability requirements, and synchronisation with the crane model, which have not been shown in fig. 2). However physical limitations of the plant guarantee that there can be at most 2 steel ladles simultaneously in module 2. Hence the size of the reachability tree is acceptable, provided the order in which different qualities are produced is fixed.

Next the batch of steel of quality q_n is sent to the holding position of the turret of the continuous casting machine (provided this position is free). The continuous casting machine operates as follows. The empty ladle which contained the $(n-2)$-th batch must be removed (by a crane) from the holding position of the turret, in time for the n-th batch to be transported to this holding position no later than δ time units prior to the completion time θ^{finish}_{n-1} of the casting of the $(n-1)$-th batch. After completion of casting of batch $n-1$ the turret can be turned and casting of batch n starts. The empty ladle used by batch $n-1$ must then be removed from the turret. Casting of the n-th batch takes at least Δ_n time units, but it can be slowed down in order to guarantee that there is no interruption in the casting (the casting machine must operate continuously). We assume that the casting of batch n can take at most Δ^{max}_n time units

to complete. This 3rd module of the plant model can be represented by a timed coloured Petri net, accepting at (input) time θ_n^{second} the n-th batch of steel, of quality q_n. The casting time $d_n \in [\Delta_n, \Delta_n^{max}]$ for the n-th batch is again an important controllable variable to be selected by the scheduler.

The movement of the 2 cranes running on the same track (and hence subject to a no-collision requirement) is modelled by a rather large timed automaton or timed Petri net, representing the (quantised) position of each crane, whether a crane is idle or whether it is lifting or lowering a load, whether it is moving left or right. The crane module must accept as input all the requests for crane allocation received from modules 2 and 3. The crane module is also influenced by the times at which cranes are deallocated by modules 2 and 3. Finally there is a simpler module, representing the transport and the cleaning of empty ladles, which are removed from the holding position of the turret of the continuous caster, and delivered at the convertors. The quality of the proposed schedule will depend critically on whether there exists a schedule for crane movements and for transport of empty ladles which satisfies all the requests by the modules 1, 2, and 3.

3 Modular Synthesis of a Schedule

In this section we describe the synthesis of an optimal schedule in more detail, using the model of section 2. The schedule must ensure that all the constraints and specifications encoded in the components of the model are satisfied. Moreover the schedule must ensure that the n-th batch reaching the continuous casting machine has quality q_n. Moreover for each batch there is a minimal and a maximal time interval for casting. Each batch must also satisfy a maximum residence time in the system (otherwise the batch would cool down too much): if the n-th batch is started on convertor i at time $\theta_{i,k_i(n)}^{conv}$ then θ_n^{finish} should be less than $\theta_{i,k_i(n)}^{conv} + \Theta^{max}$.

The optimal schedule must satisfy all these constraints, and minimize the time θ_N^{finish} when the last batch (of quality q_N) is completed.

In module 3 the schedule tries to minimize θ_N^{finish}. It turns out that the timed Petri net model of module 3 is a timed marked graph, so that the execution times of the events can be described via a linear equation in the $(max, +)$-algebra [3]: (where θ_n^h denotes the time at which the turret is turned for the n-th time, θ_n^{second} is the time when the n-th batch leaves the 2nd module, and where $\Delta_n^{min,max}$ represent resp. the minimal and the maximal time needed for casting the n-th batch of quality q_n.)

$$\theta_n^{finish} = \theta_n^h + [L_{tt} + \Delta_n, U_{tt} + \Delta_n^{max}]$$

$$\begin{aligned} \theta_n^h \;=\; & \max\{\theta_n^{second} + [L_h, U_h], \\ & \theta_{n-1}^h + [L_{tt} + \Delta_{n-1}^{min} + L_{3,a} + L_h, \\ & U_{tt} + \Delta_{n-1}^{max} + U_{3,a} + U_h], \\ & \theta_n^{3,all} + [L_{3,d}, U_{3,d}] \end{aligned}$$

where we generalise the classical (max,+) notation by writing $[L, U]$ to denote a number within this interval (each time this interval appears in an equation this parameter is implicitly assumed to take the same value). Similar equations can be written down for the times θ_n^{all} and θ_n^{deall} when the crane is allocated and deallocated for the n-th time for module 3. Using these variables it is possible to express the time $\theta_n^{finish} - \theta_n^{second}$ which the n-th batch spends in module 3.

The minimal cost

$$\begin{aligned} & c^*(\theta_n^{second}, n = 1, \ldots, N; \text{crane availability}) \\ & \quad = \min\{\theta_N^{finish} \mid \\ & \mid \theta_n^{second}, n = 1, \ldots, N; \text{crane availability}\} \end{aligned}$$

can be evaluated recursively using these (max,+)-linear expressions. This minimal cost function or value function (depending only on a "local information set of output times) is transmitted to module 2. This must be interpreted as the minimal cost $c^*(\theta_n^{second}, n = 1, \ldots, N; \text{crane availability})$ which the system can achieve if departure times $\theta_n^{second}, n = 1, \ldots, N$ are achieved. Moreover it is necessary to transmit to module 2 the maximum time $\Theta^{max} - \theta_n^{finish} + \theta_n^{second}$ that the n-th batch is allowed to spend in component 1 and 2.

Based on this (backward recursive) information sent from module 3 to module 2, module 2 synthesizes its schedule so as to minimize its local cost function: $c^{**}(\theta_n^{first}, n = 1, \ldots, N; \text{crane availability}) = \min[c^*(\theta_n^{second}, n = 1, \ldots, N; \text{crane availability}) \mid \theta_{i,k_i}^{first}, k_1 + k_2 = n = 1, \ldots, N; \text{crane availability}]$ This calculation is more complicated than for module 3 because there are several choices to be made. However once the sequence of positions to be visited by each of the N batches has been selected, then it becomes easy to express $\theta_n^{second}, n = 1, \ldots, N$ as a function of $\theta_n^{first}, n = 1, \ldots, N$. Moreover it becomes easy then to provide a formula for the time spent in module 2 by each batch - providing information on the maximum time that this batch can spend in module 1 - and on the times at which cranes have to be allocated and deallocated for module 2. In general the number of possible selections of sequences is prohibitively large: for most

qualities there are 4 possible routes through module 2 and for $N = 50$ batches in the schedule, this gives 4^{50} sequences to be considered. In practice however most of these choices will be excluded by resource conflicts, or will be unfeasible given the present state of module 2. Heuristic rules for selecting these routes are fairly easy to establish.

The modular scheduler will then pass the value function

$$c^{**}(\theta_{i,k}^{first}, n = 1, \ldots, N; \text{crane availability})$$

to module 1 together with the maximal time that each batch can spend in module 1, given the schedule selected in modules 2 and 3. This allows module 1 to calculate the optimal times $\theta_{i,k}^{conv}$ to start conversion of a new batch. Note that the main decision here is the starting time of the next conversion. The starting times of conversion will usually be taken as late as possible, subject to the requirement that the module 2 should be able to send the batches to module 3 in time to ensure that continuous casting can be guaranteed. The proposed schedule also specifies, via a simple (max,+)-formula, the latest arrival time for each empty ladle.

Given these locally optimal schedules in modules $1, 2$, and 3 the modular algorithm verifies whether the crane model of module 4 and the empty ladle module both can satisfy all the requests for crane allocation, at times defined by the schedules, and the maximum arrival times for empty ladles. This requires verification of the reachability of certain states in the timed automaton representing the cranes and the empty ladle transport. Various tools exist (such as UPPAAL and KRONOS) for verifying reachability in very large timed automata. In case the algorithm finds that the schedule is not feasible in the sense that a crane or an empty ladle cannot reach a position early enough, then another schedule has to be generated from the present one. This requires information of the sensitivity of the cost function (and of the constraint satsifaction) on the times selected by the schedule.

The (max,+)-recursions also can be used in a simple fashion in order to detect constraints that are *active* or *critical* in the sense that the derivative of the final cost criterion θ_N^{finish} with respect to θ_n^{second} is 1 or ∞. The derivative is 0 in the case that sending the n-th batch to the holding position of the turret sooner only implies that the n-th batch has to wait longer because the casting of the previous batch $n-1$ cannot finish sooner anyway; the derivative is 1 when the $n-1$-st batch would have to be delayed (casting of the $n-1$-th batch is forced to take longer than Δ_{n-1}) in order to guarantee continuous casting; the derivative becomes ∞ when delaying θ_n^{second} leads to an infeasible schedule because the maximum delay Δ_{n-1}^{max} would be exceeded, or because the time span of batch n would exceed Θ^{max}. No other values of the derivative can occur.

New schedules can be generated in several ways. The event synchronised with an infeasible request for a crane movement can be delayed until the crane can arrive. In case the event is not active nor critical, its delay will not increase the cost. The algorithm will first try to reschedule inactive event times as late as possible in the schedule, in module 2. Starting the rescheduling near the end of the sequence (for large n) is intuitively best because these delayed events near the end will have the least risk of causing further conflicts. If this does not work, then one has to consider starting conversion of some future batch $n_{present}, n_{present}+1, \ldots, N$ earlier (where $n_{present}$ is the lowest number of a batch which has not left module 2 yet). Indeed by starting these operations earlier it is possible that the crane will be released earlier and that the allocation requests can be met. Given the way the local schedule in module 1 has been obtained it is clear that this modification of θ_{i,k_i}^{conv} will not change the overall cost of the schedule, since it will not advance the increase time θ_n^{finish}. Otherwise module 1 would have chosen a different start time anyway.

In case the above method does not find a feasible rescheduling, then one has to change the paths taken by the batches in module 2, again starting by changing the paths of the latest batches $N, N-1, N-2, \ldots, n$, or the cleaning time of the convertors. This modification of the paths can only increase the overall cost, because otherwise this other path would have been selected by the local schedule of module 2 in the first place. Continuing this search will eventually move through the whole reachability tree, but this will be done in such a way that branches with a low cost are searched first.

In order to guarantee that the algorithm will eventually (and usually quickly) arrive at the optimal schedule (or at an optimal schedule, since the optimum will usually not be unique), it is necessary to guarantee that no cycles can occur during the search through the reachability tree. It must be possible to detect during the search which schedules have already been visited before. Even though the search space is huge, it can be shown that the number of state classes for an interval timed Petri net is finite [Heiner]. Hence it is possible to encode all possible schedules using a hash table, and to check each time whether the proposed new schedule has already been visited. The algorithm will then eventually find an optimal schedule because all possibilities are searched in an efficient order, and because the search will stop as soon as a candidate schedule is found that is optimal in the acyclic graph, and that verifies all the constraints in the cyclical graph.

4 Conclusions and Future Work

This paper proposes an algorithm for finding the optimal schedule in a large plant, by combining the advantages of backward and forward recursion. The method is based on repesenting the plant via a graph of modules, each module described by a timed discrete event model. It is often the case that there exists a largest acyclic graph of modules which describe the most time-critical operations in the plant. Within this acyclic graph of plant modules our algorithm propagates information upstream like in dynamic programming. However within each module we use forward recursion, starting at the present state, in order to avoid having to calculate and store the value function in many unreachable states. Efficient, well strucured methods for representing the states and functions of the state (like the value function) form on etopic of research for making the proposed approach practically viable. The cyclic interactions caused by the other modules is used afterwards in an iterative improvement procedure. Efficient ways of using sensitivity measures in finding good imporvemnets, leading to the optimum as quickly as possible are also a topic of further research.

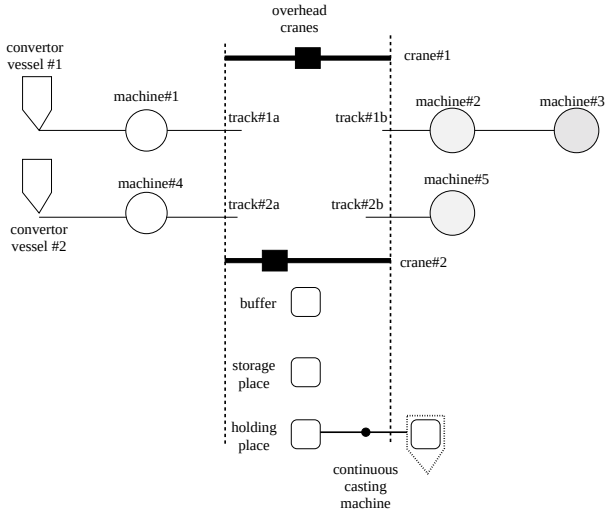

Figure 1: Plant Layout

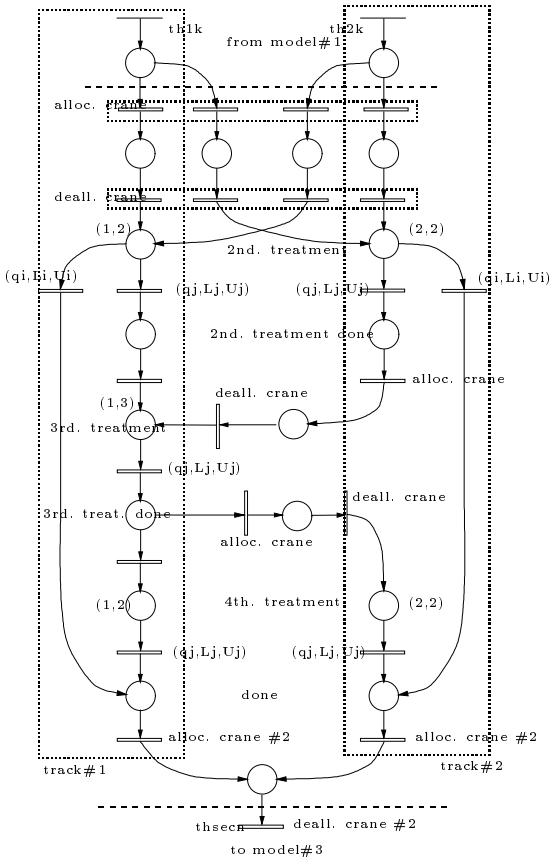

Figure 2: Module#2: Second stage

Bibliography

[1] R. Alurand D. Dill *A theory of timed automata*, Theor. Comp. Sci., vol. 126, pp. 183-235, 1994.

[2] E. Asarin and O. Maler, *As Soon as Possible: Time Optimal Control of Timed Automata*, Proceedings of the 2nd Workshop on Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 1569, Springer, pp. 19-30, 1999.

[3] F. Baccelli, G. Cohen, G. Olsder and J. Quadrat: *Synchronization and Linearity - an algebra for discrete event systems*, Wiley, 1992.

[4] R. Boel, B. Bordbar, and G. Stremersch: *Controlled timed Petri nets: equivalnce relations, model reduction*, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 1998.

[5] M. Heiner and L. Popova-Zeugman, *Worst-case analysis of concurrent Systems with Duration Interval Petri Nets*, in: Entwurf komplexer Automatisierungssysteme, Braunschweig, 1997.

[6] P. Ramadge and W.M. Wonham, *Supervisoy control of a class of dicrete-event systems*, SIAM J. Control and Optimization, vol. 25, Jan. 1987, pp. 206-230.