Verification of a Scheduler in B through a Timed Automata Specification

Odile NASR, Jean-Paul BODEVEIX, Mamoun FILALI, Miloud RACHED
IRIT, Paul Sabatier University
118 Route de Narbonne, 31062 Toulouse FRANCE
{nasr,bodeveix,filali,rached}@irit.fr

ABSTRACT
This paper proposes a methodology for specifying and verifying schedulers using the B method. It is based on the refinement mechanism. The specification must manage time through clocks, whereas the natural modeling of schedulers exploits only stopwatches.

Keywords
Specification, Verification, B method, timed Automata.

1. INTRODUCTION
In a real time system, the different tasks must respect various timing constraints: the execution of these tasks must not violate a maximum delay, no task should finish after its absolute deadline [6, 5]. The principal goal of scheduling is to allow the respect of these timing constraints. In order to verify real time properties of a system in the presence of a scheduler, we must translate this scheduler into basic timed automata of Uppaal [8]. Thus we have to verify the correctness of the timed automaton encoding. For this purpose, we use the B method [4, 1] to develop an implementation of a scheduler. Whereas the abstract specification of a scheduler exploits only stopwatches, the implementation based on timed automata [3, 2] relies on clocks. Here clock progression is done at the same speed and the only possible operations on these clocks are resets and comparisons. The main difficulty of this approach is to encode preemptive schedulers. The proposed method relies on delaying the computation bound of the preempted process.

The rest of the paper is organized as follows. Section 2 introduces the B method. Section 3 presents the different steps needed to model the scheduling in B. The conclusion is presented in section 4.

2. B METHOD
In B [4, 1], a development begins with the construction of a model which expresses the requirement specifications. It specifies, in abstract machines, the main state variables of the system and the properties (invariants) that these variables must satisfy at any time. Operations allow to transform the machine variables. The B model is then refined until obtaining a complete implementation of the software system. Refinements allow to include gradually the problem details in the formal development. The formal specification is then realized gradually and not directly. The following section illustrates the refinement mechanism: first the abstract machine scheduler is refined into untimed machines, then we introduce time through stopwatches and then clocks.

3. B MODELING ARCHITECTURE
The various scheduling algorithms can be modelled in B hierarchically, by an abstract machine and its refinements (Figure 1).

- The abstract machine scheduler is used to model any type of scheduling.
- untimed,nonpreemptive and preemptive,priority allow to model respectively the nonpreemptive and preemptive policies.
- timed,nonpreemptive introduces temporal aspects
- fcfs,policy models the FCFS policy (First Come First Served).
- preemptive,first,_priority adds temporal aspects to preemptive priority using stopwatches.
- preemptive,uppaal fulfills the requirements of the timed automata model. The verification of proof obligations of this refinement guarantees the correctness of the Uppaal modeling for preemptive schedulers.

In order to handle the priorities and pre-emptions, we have defined the machines priority,graph, timed,priority,graph and istacks. They model respectively the priority graph needed to check if there must be pre-emption, the priority graph with nodes having limited life durations and the stack of preempted processes.

Modelling a specific pre-emptive policy like EDF (Earliest Deadline First) [7], consists in refining the timed graph timed,priority,graph. Here, we present a high level specification of a scheduler defined by the machine scheduler.

3.1 Machine scheduler
This abstract machine defines the common behavior to any scheduling policy, pre-emptive or nonpre-emptive. It models in B an atemporal scheduler where we handle the processor allocation.
**Figure 1:** Machines hierarchy

- **State space:** we distinguish between the processes that have requested a processor time (ready) and the current process (running) which holds the processor. This process is also an element of ready unless ready is empty.
  
  \[
  \text{ready} \in \text{FIN(process)} \land \\
  \text{running} \in \text{PROCESS} \land \\
  (\text{running} \neq \text{ready} \Rightarrow \text{ready} = \emptyset)
  \]

- **Operations:** during the scheduling modeling, we have distinguished between several handlings and consequently several operations. In this machine, we do not consider time. Thus the SKIP operation modelling clock progression is specified by the skip substitution which means “do nothing”.

```
SKIP = skip
```

The first request of the process pp is modelled in the operation \( \text{wakeup}(pp) \). pp being the first process, gets directly the processor.

\[
\text{wakeup}(pp) = \begin{cases} 
\text{PRE pp} \in \text{PROCESS} \land \text{ready} = \emptyset \land \text{pp} \in \text{ready} \land \text{running} = \text{pp} \\
\text{END}
\end{cases}
\]

The following requests for the processor are specified by the operation \( \text{request}(pp) \). The set \text{ready} is not empty, since this is not the first request of the system. The choice of the running process depends on the possibility of pre-emption. In the pre-emptive case, \( \text{pp} \) gets the processor if it has the highest priority, otherwise the current process remains running and \( \text{pp} \) becomes ready. At this abstraction stage, these two behaviors are introduced via a non-deterministic choice.

\[
\text{request}(pp) = \begin{cases} 
\text{PRE pp} \in \text{PROCESS} \land \text{ready} \neq \emptyset \land \text{pp} \not\in \text{ready} \land \text{running} = \text{pp} \\
\text{ready} = \text{ready} \cup \text{pp} \\
\text{END}
\end{cases}
\]

When the current process finishes its execution, the election of a new process pp consists in choosing it among all the processes having required the access to the processor. This elected process becomes running.

\[
\text{election}(pp) = \begin{cases} 
\text{PRE pp} \in \text{PROCESS} \land \text{pp} \in \text{ready} \land \text{pp} \in \text{running} \land \text{running} = \text{pp} \\
\text{END}
\end{cases}
\]

The last system call is expressed by the operation \( \text{stop} \) which empties \text{ready}. The execution end of the last running process contributes in stopping the system.

\[
\text{stop} = \begin{cases} 
\text{PRE running} \in \text{ready} \land \text{running} \in \{\text{running}\} \land \text{ready} = \emptyset \\
\text{END}
\end{cases}
\]

### 3.2 Validation and proof

Most of the proof obligations associated with the specification and the refinements have been proved using Atelier B. An essential ingredient when proving a refinement is the gluing invariant describing the relationship between the variables of the abstract and concrete systems. Machines scheduler, isacks, priority_graph, timed_priority_graph, untimed_nonpreemptive and timed_nonpreemptive were proven automatically by Atelier B. The remaining proofs are essentially arithmetic and include a great number of hypotheses. They are especially in \text{preemptiveuppaal}, where we eliminate the stopwatches in order to respect a timed automata specification. They are also in \text{apolicy} where we redefine the priority relation according to requirements of EDF. The proof obligations of the machine \text{preemptiveuppaal} define the validity of the proposed method.

In order to prove many proof obligations, we have enriched the rule database of B with new theorems related to relational operators like \( \text{"or" "and" "or-all" . . .} \) as well as rewriting rules. Some of these rules are obvious, but do not appear in the predefined rules of the Atelier B. Others are less trivial and specific to the selected handling and modeling. Given the difficulty of introducing general proof rules to handle all properties, we have manually checked the rest in order to show the correctness of the method.

### 4. CONCLUSION

In this paper, we have presented the development scheme of a scheduler in B. It handles time through clocks with respect to timed automata, instead of stopwatches. All the proof obligations of our specification were proven automatically and manually. A way to continue this work would be to improve the specification in order to reduce the number of proof obligations related to the graph handling.

### 5. REFERENCES


