ANALYSIS OF OSEK/VDX BASED AUTOMOTIVE APPLICATIONS

Libor Waszniowski, Zdenek Hanzalek

Czech Technical University
Centre for Applied Cybernetics, Department of Control Engineering
Karlovo nám. 13, 121 35 Prague 2, Czech Republic
{hanzalek, xwasznio}@fel.cvut.cz

Abstract: The aim of this article is to show, how an automotive real-time software application running under real-time operating system compliant with OSEK/VDX standard can be modelled by timed automata. The application under consideration consists of several basic or extended tasks, it includes resource sharing and synchronisation by events. For such system, model checking theory based on timed automata and implemented in model checking tools can be used for verifying complex time and logical properties of proposed model. Use of this methodology is demonstrated on the automotive case study. The automated transmission system and its control software running under OSEK compliant operating system are modelled and properties necessary for its proper function are verified by model checker UPPAAL.

Keywords: Real-Time Operating Systems, Validation

1 INTRODUCTION

Several special purpose real-time software analysis methods have been developed in the area of real time scheduling (Buttazzo, 1997; Liu, 2000) and they have been widely used in praxis. These methods dedicated to schedulability analysis, e.g. rate monotonic analysis (RMA) (Sha, and Goodenough, 1991), are very successful for analysis of systems with independent periodic tasks but incorporation of non-periodic tasks and inter-task communication primitives can lead to pessimistic results. To achieve more general and more precise analysis, the model checking method allowing to automatically verify safety and liveness properties of the system model, can be used. The model expressing time and structural properties of the system can be based on timed automata or on hybrid automata.

Corbet (1996) provides a method for constructing models of real time Ada tasking programs based on constant slope linear hybrid automata. The model can be automatically analysed by HyTech verifier. Even thought the author reports that the analysing algorithm does usually terminate in practice, the reachability problem for hybrid automata is undecidable in general.

Hybrid automaton (or some of its subclass e.g. stopwatch automaton (Cassez and Larsen, 2000)) is needed for accurate modelling of preemption (Henzinger, et al., 1998). The continuous variable used to measure the amount of CPU time allocated to each task must progress when the corresponding task is executed and must be stopped when the corresponding task is preempted. Such behaviour cannot be modelled by timed automaton that does not allow stopping of the clock variable (Alur and Dill, 1994; Bouyer, et al., 2000). Therefore only over-approximate model of preemption based on timed automata is known (Fersman, et al., 2003; Waszniowski and Hanzálek, 2003b).

On the other hand, the advantage of timed automata is decidability of reachability problem and the model-checking of timed computation tree logic (TCTL) properties problem. Fersman et al. (2002) and Amnell et al. (2003) extend the timed automata (without loosing of mentioned advantage) by asynchronous processes (i.e. tasks triggered by events) to provide model for event-driven systems. Tasks associated to locations of timed automaton are executable programs characterised by its worst-case and best-case execution time, deadline and other parameters for scheduling (e.g. priority). Transition leading to a location in such automaton denotes an event triggering the task. Released tasks are stored in a queue and they are assumed to be executed according to a given scheduling strategy. Both non-preemptive and preemptive scheduling strategies are allowed but in the case of the preemptive scheduling and different task worst-case and best-case execution time, the model is only over-approximation from the model checking point of view (this approximation is identical to the one made by timed automata model). The results of the schedulability analysis are not affected by this approximation (Waszniowski and Hanzálek, 2003).

Based on these observations we provide model of non-preemptive tasks based on timed automata. Although the model based on timed automata does not allow to model preemption accurately, its advantage is (opposite to hybrid automata that are appropriate for accurate preemption modelling) that the termination of the model checking algorithm is guarantied. Model checkers are available for such model (e.g. Kronos[1] and UPPAAL[2] (Behrmann, at al., 2001)). Preemptive schedulers are known to provide higher utilisation of processor than the non-preemptive ones (Buttazzo, 1997). On the other hand big advantage of non-preemptive scheduling is lower stack requirement that is important for embedded applications where the available RAM is limited. Moreover the non-preemptively scheduled applications are easier to program and to debug. This is significant advantage especially for hard real time applications like the automotive once where the highest reliability is required. This paper presents another important advantage of non-preemptive scheduling that is the possibility to create mathematical model of the application based on timed automata and to verify its safety and liveness properties by full automatic model-checking tool in finite time.

The modelling methodology is briefly described in section 2 and demonstrated on non-trivial automotive case study in section 3.

2 MODELING OF OSEK BASED APPLICATIONS

OSEK compliant operating system provides static priority based scheduling. Each task can be specified as preemptive, non-preemptive or non-preemptive in specified group. Only non-preemptive tasks are considered in the rest of this paper since they can be modelled by timed automata accurately. OSEK compliant operating system provides basic tasks and extended tasks (OSEK, 2003). Basic tasks are created at system generation time as suspended, after activation (by another task or alarm) they become ready and after starting by scheduler they become running. Running task may relinquish the processor by system call *Schedule* and become ready or it may terminate its execution and become suspended. Extended tasks are moreover allowed to use system call *WailEvent*, which may result in waiting state. Waiting task is released by occurrence of at least one event, which the task has waited for. Released task becomes ready.

[1] http://www-verimag.imag.fr/TEMPORISE/kronos/
[2] http://www.uppaal.com

2.1 Tasks Modelling

Task code is supposed to consist of non-preemptive blocks of code called computations (computations do not contain any blocking operation), branching, loops, system calls for communication and synchronisation (e.g. *SetEvent, WaitEvent*) and system call *Schedule* for explicit rescheduling.

Computations (*Comp* for short) representing the lowest modelled granularity are supposed to take non-deterministic execution time $C \in \langle L, U \rangle$ (lover and upper margins allow to involve uncertainty of execution time due to non-modelled code branching inside the computations, interrupt handlers execution, cycle stealing by DMA device, etc.). Execution time of other tasks elements is not considered (it can be involved in precedent computation). Suppose for example simple application consisting of high-priority basic task *Task1* and low-priority extended task *Task2* in Fig. 2.1.

```
Task1() {
  Comp1;
  Schedule();
  Comp2;
  TerminateTask();
};
```

```
Task2() {
  Comp1;
  while(TRUE) {
    WaitEvent(Event1);
    ClearEvent(Event1)
    Comp2;
  };
};
```

Fig. 2.1. Example of tasks pseudocode

Periodic activation of *Task1* provides periodic alarm *Alarm1* modelled by automaton depicted in Fig. 2.2. The second automaton in Fig. 2.2 is model of *Alarm2* providing periodic setting of event *Event1*.

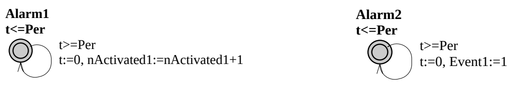


Fig. 2.2. Models of *Alarm1* and *Alarm2*

Each task is represented by one timed automaton in application model. These automata share variables to model the sequential execution of tasks organised by scheduler. Models of tasks from Fig. 2.1 are in Fig. 2.3 and Fig. 2.4.

The modelling notation of the model-checking tool UPPAAL is used. The double circle used for locations *Suspended* specifies an initial location. The location *WaitEvent* with the letter C inside the circle is so called *committed* locations providing atomicity of traversing of in-coming and out-coming transitions (committed location must be left immediately without any interference of other automaton in the model). Above a circle representing a location is specified its name and time invariant representing upper bound of time spend in this location. Above an oriented edge representing a transition is synchronisation through channel (transition marked by *chan!* must be taken synchronously with the transition marked by *chan?* and vice versa) and under synchronisation is guard (all coma separated expressions must hold to enable transition). Under the edge are updates (all coma separated expressions are executed).

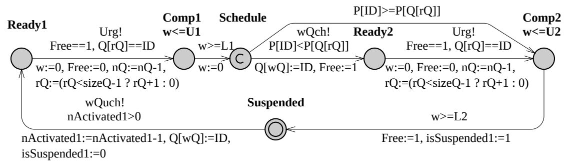


Fig. 2.3. Model of *Task1*

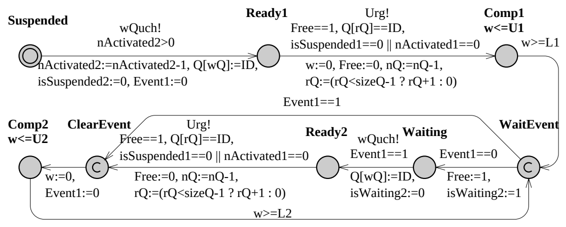


Fig. 2.4 Model of *Task2*

Automaton representing the model of a task consists of locations and variables representing state of the task code execution from the application point of view (*Comp1*, *Comp2*, *Schedule*, *WaitEvent*, *ClearEvent*), state of the task from the scheduling point of view (*Suspended*, *Ready1*, *Ready2*, *Waiting*) and state of the scheduler and objects of the operating system (*Free*, *Q*, *Event1*,...).

Each task is identified by unique integer *ID* (0,1,2,...). Priority of the task is stored in global array *P*, indexed by *ID*. *ID*s of all tasks, which are in *Ready* state, are stored in the queue modelled as global array *Q* representing a circular buffer. The integer *nQ* is the number of elements in the queue. The integer *rQ* is the position for reading of the first element in *Q* and the integer *wQ* is position of the first empty element in *Q* as is depicted in Fig. 2.5. Tasks are ordered in descending order according to their priorities in *Q* (*rQ* points to the ready task with the highest priority).

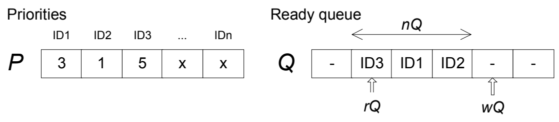
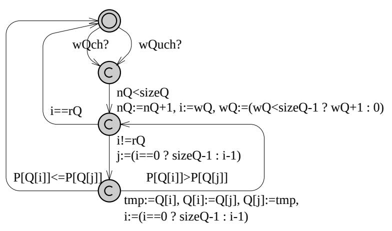


Fig. 2.5. Array of tasks priorities and ready queue

ID of the task leaving the *Ready* state is deleted from the ready queue by decrementing the number of elements in the queue *nQ* and by moving reading pointer *rQ* to the next element in the queue. *ID* of the task entering the *Ready* state is written to the end of ready queue. The ready queue must be reordered after this operation. Ordering according priorities is provided by automaton *wPriorQueue* depicted in Fig. 2.6. Reordering mechanism is started by synchronisation channel *wQch* or *wQuch*. The channel *wQch* is used on transitions from *Comp* to *Ready* where the upper margin of the time of taking this transition is expressed by time invariant of location *Comp* (It is not distinguished *Comp1* and *Comp2* or *Ready1* and *Ready2* respectively when it is not necessary and it is abbreviated by *Comp* or *Ready* respectively). The channel *wQuch* is used on transitions from *Suspended* or *Waiting* to *Ready*. Since no upper margin of the time of taking these transitions is expressed, the channel *wQuch* is declared as *urgent channel* (no time progress is enabled when there are some enabled transitions synchronised through urgent channel). This transition is therefore taken as soon as it is enabled.



wQch?
wQuch?
nQ<sizeQ
nQ:=nQ+1, i:=wQ, wQ:=(wQ<sizeQ-1 ? wQ+1 : 0)
i==rQ
i!=rQ
j:=(i==0 ? sizeQ-1 : i-1)
P[Q[i]]<=P[Q[j]]
P[Q[i]]>P[Q[j]]
tmp:=Q[i], Q[i]:=Q[j], Q[j]:=tmp,
i:=(i==0 ? sizeQ-1 : i-1)

Fig. 2.6. Automaton *wPriorQueue* providing reordering of ready queue *Q*

Mutual exclusive access of the tasks to running state (locations *Comp1* and *Comp2*) is guarded by two-state variable *Free*. Moreover, only the highest priority task (its *ID* is at the top of ready queue) can become running. To prevent processor idling (no task is running even thought it would be possible), the transition from *Ready* to *Comp* location must be taken as soon as it is enabled. This is provided by urgent channel *Urg*, which provides synchronisation with still enabled transition in one location automaton constructed specially for this purpose.

Suspended task becomes ready when it has been activated at least once (*nActivated1>0*). When the conformance class of operating system does not support multiple activations, zero must be assigned to the variable *nActivated1* on transition ending in *Suspended* location.

Each transition beginning in *Ready* location in *Task2* automaton has guard containing expression *isSuspended1==0* || *nActivated1==0* (|| is logical *or* and == is relation *equality*). This expression prevents the low-priority task *Task2* to become running when the high-priority task *Task1* is suspended but can be immediately activated. It can be seen that the transition from *Suspended* to *Ready1* in *Task1* automaton has higher priority than the transitions from *Ready1* to *Comp1* and from *Ready2* to *ClearEvent* in automaton *Task2*. This transition priority assignment reflects the fact that the operating system releases waiting tasks and activates suspended tasks, if it is possible, before scheduling. This problem is studied in (Waszniowski and Hanzálek, 2003a) in more details.

Notice that the proposed model is simplified by assumption that the context switch does not take any time. Context switch time can be simply involved in the computation time of each computation.

2.2 Resource modelling

OSEK compliant operating system provides resource management for coordination of concurrent access of several tasks to shared resources. Since system call *Sechedule* or *WaitEvent* must not be called while a resource is occupied, resources are in non-preemptive tasks protected just by avoiding of using *Sechedule* and *WaitEvent* within a critical section.

Nevertheless if there is requirement to use resources in non-preemptive scheduling, it can be very simply modelled. Due to OSEK priority ceiling resource access protocol (OSEK 2003) (sometimes called Highest Locker or Immediate Inheritance protocol), the only effect of getting and releasing resources is task priority change from its static priority to the priority ceiling of the resources and vice versa. It can be incorporated to the proposed model by changing actual task priority in *P[ID]* to resource priority ceiling at transition representing *GetResource*. The old value of *P[ID]* must be stored in resource dedicated variable *PriorityBeforeRequest* that is assigned back to the actual task priority in *P[ID]* at transition representing *ReleaseResource*.

3 AUTOMATED TRANSMISSION CASE STUDY

This section demonstrates modelling of the OSEK compliant operating system based applications on the example of the automated transmission system. Five-speed gearbox is assumed. Each of three shifting rods is actuated by servo that can shift the collar form neutral to one gear or to the second gear. Automaton representing one shifting rod is depicted in Fig. 3.5. Dry clutch actuated by servo is supposed. It is modelled by automaton depicted in Fig. 3.4. Gearbox and clutch are controlled by computer running software consisting of three non-preemptive tasks executed under OSEK compliant operating system. Task *AutomatedTransmisionTask* (Fig. 3.1) selects according to engine condition appropriate gear (variable *DesiredGear*). It is assigned by the lowest priority 1 and it is periodically activated by alarm *AutTransAlarm* modelled by automaton in Fig. 3.7 b). *AutomatedTransmisionTask* is modelled by automaton in Fig. 3.6. If the currently engaged gear (variable *CurrentGear*) differ from the desired gear, task *GearBoxTask* is activated. The *GearBoxTask* (Fig. 3.2) opens the clutch, disengages current gear, engages desired gear and closes clutch. This task is assigned by the middle priority 2. It is modelled by automaton in Fig. 3.8. When the *GearBoxtask* disengaging or engaging any gear, it specifies which shifting rod servo (variable *RodServo*) and in which direction (variable *MoveDir*) it is necessary to move. Then it activates the highest priority task *RodServoTask* that controls the movement to the desired position in closed loop (see Fig. 3.3 and automaton in Fig. 3.9).

```
AutomatedTransmissionTask() {
  Comp1;
  if (GearBoxReady==0)
    TerminateTask();
  Schedule();
  Comp2;
  if (DesiredGear==CurrentGear)
    TerminateTask();
  ActivateTask (GearBoxTask);
  TerminateTask();
};
```

Fig. 3.1. *AutomatedTransmissionTask* pseudocode

```
GearBoxTask() {
  GearBoxReady:=0;
  ClearEvent (ClutchClosedEvent);
  OpenClutch;
  WaitEvent (ClutchOpenedEvent);
  if (CurrentGear != NEUTRAL) {
    ClearEvent(EngagedEvent);
    MoveDir:=Gear2BackDir (CurrentGear);
    ActivateTask (RodServoTask);
    WaitEvent (EngagedEvent);
  };
  if (DesiredGear!=NEUTRAL) {
    ClearEvent(EngagedEvent);
    MoveDir:=Gear2Dir(DesiredGear);
    RodServo:=Gear2Rod(DesiredGear);
    ActivateTask (RodServoTask);
    WaitEvent (EngagedEvent);
  }
  CurrentGear:=DesiredGear;
  ClearEvent (ClutchOpenedEvent);
  CloseClutch;
  WaitEvent (ClutchClosedEvent);
  GearBoxReady:=1;
  TerminateTask();
};
```

Fig. 3.2. *GearBoxTask* pseudocode

```
RodServoTask() {
  Move (RodServo, MoveDir);
  ClearEvent (EndLimitEvent);
  SetRelAlarm (ServoPerAlarm, PERIOD, PERIOD);
  while (TRUE) {
    ComputeServoPID;
    WaitEvent (ServoPerEvent || EndLimitEvent)
    GetEvent (RodServoTask, refMask)
    if (*refMask & EndLimitEvent) {
      Stop;
      break;
    };
  ClearEvent (ServoPerEvent);
  };
  CancelAlarm (ServoPerAlarm);
  SetEvent(GearBoxTask, EngagedEvent);
  TerminateTask();
};
```

Fig. 3.3. *RodServoTask* pseudocode

The goal of this case study is to create model of the described automated transmission system and its control system and to use the model-checking tool UPPAAL to verify the following properties:

- At most one shifting rod can leave neutral.
- Any shifting rod may move only when clutch is opened.
- The *GearBoxTask* execution is finished within 0.4s after activation.
- Each gear is engaged and clutch is closed 0.5s after the gear was selected.
- Clutch is not opened more than 0.4s.

Theses properties was formalised in TCTL based UPPAAL specification language and successfully verified by UPPAAL verifier.

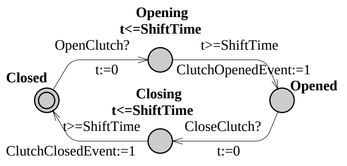
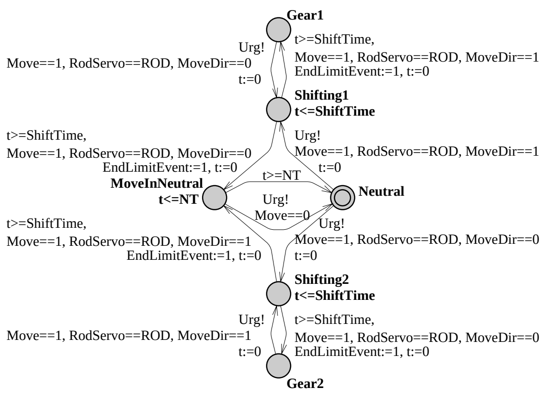


Fig. 3.4. Clutch automaton



Gear1
Urg!
Move==1, RodServo==ROD, MoveDir==0
t:=0
t>=ShiftTime,
Move==1, RodServo==ROD, MoveDir==1
EndLimitEvent:=1, t:=0
Shifting1
t<=ShiftTime
t>=ShiftTime,
Move==1, RodServo==ROD, MoveDir==0
EndLimitEvent:=1, t:=0
Urg!
Move==1, RodServo==ROD, MoveDir==1
t:=0
MoveInNeutral
t<=NT
t>=NT
Urg!
Move==0
Neutral
t>=ShiftTime,
Move==1, RodServo==ROD, MoveDir==1
EndLimitEvent:=1, t:=0
Urg!
Move==1, RodServo==ROD, MoveDir==0
t:=0
Shifting2
t<=ShiftTime
Urg!
Move==1, RodServo==ROD, MoveDir==1
t:=0
t>=ShiftTime,
Move==1, RodServo==ROD, MoveDir==0
EndLimitEvent:=1, t:=0
Gear2

Fig. 3.5. One shifting rod automaton

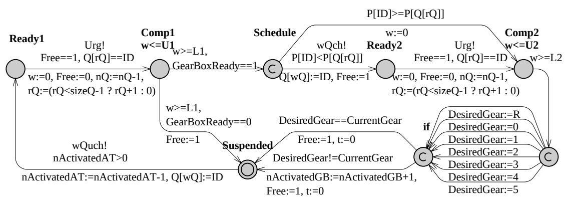
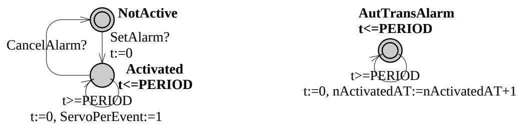
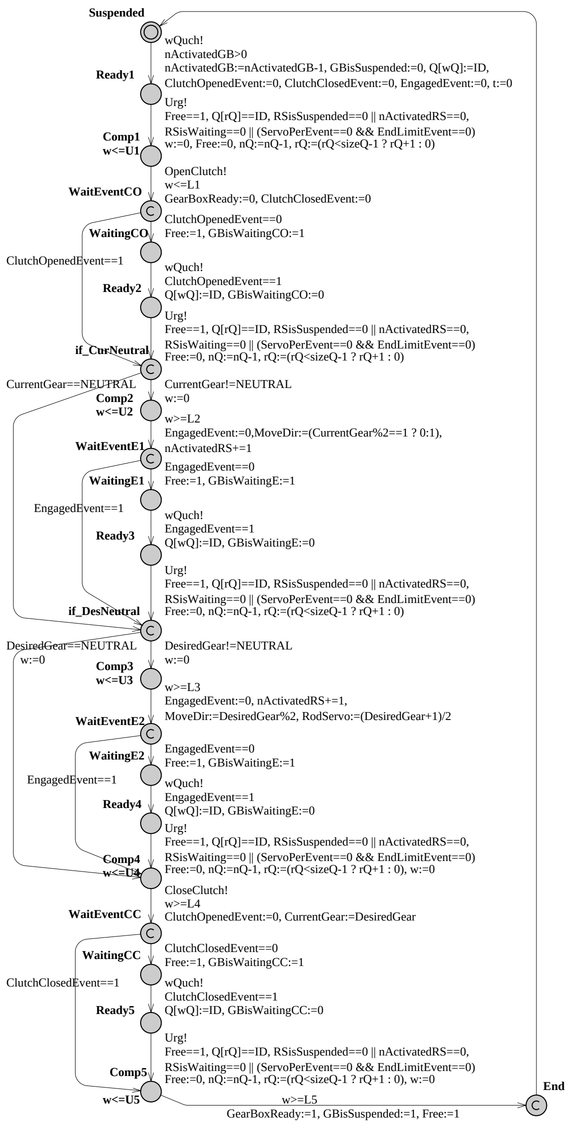


Fig. 3.6. Automated transmission task automaton



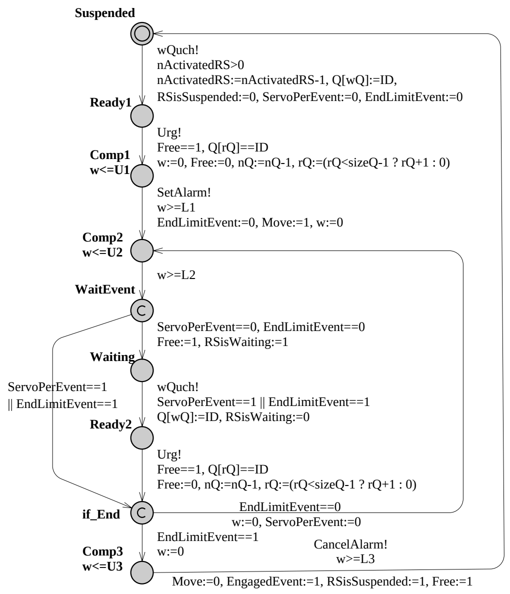
a) *ServoPerAlarm* b) *AutTransAlarm*

Fig. 3.7. Alarms automata



Suspended
wQuch!
nActivatedGB>0
nActivatedGB:=nActivatedGB-1, GBisSuspended:=0, Q[wQ]:=ID,
ClutchOpenedEvent:=0, ClutchClosedEvent:=0, EngagedEvent:=0, t:=0
Ready1
Urg!
Free==1, Q[rQ]==ID, RSisSuspended==0 || nActivatedRS==0,
RSisWaiting==0 || (ServoPerEvent==0 && EndLimitEvent==0)
w:=0, Free:=0, nQ:=nQ-1, rQ:=(rQ<sizeQ-1 ? rQ+1 : 0)
Comp1
w<=U1
OpenClutch!
w<=L1
GearBoxReady:=0, ClutchClosedEvent:=0
WaitEventCO
ClutchOpenedEvent==0
Free:=1, GBisWaitingCO:=1
WaitingCO
ClutchOpenedEvent==1
wQuch!
ClutchOpenedEvent==1
Q[wQ]:=ID, GBisWaitingCO:=0
Ready2
Urg!
Free==1, Q[rQ]==ID, RSisSuspended==0 || nActivatedRS==0,
RSisWaiting==0 || (ServoPerEvent==0 && EndLimitEvent==0)
Free:=0, nQ:=nQ-1, rQ:=(rQ<sizeQ-1 ? rQ+1 : 0)
if_CurNeutral
CurrentGear==NEUTRAL
CurrentGear!=NEUTRAL
w:=0
Comp2
w<=U2
w>=L2
EngagedEvent:=0,MoveDir:=(CurrentGear%2==1 ? 0:1),
nActivatedRS+=1
WaitEventE1
EngagedEvent==0
Free:=1, GBisWaitingE:=1
WaitingE1
EngagedEvent==1
wQuch!
EngagedEvent==1
Q[wQ]:=ID, GBisWaitingE:=0
Ready3
Urg!
Free==1, Q[rQ]==ID, RSisSuspended==0 || nActivatedRS==0,
RSisWaiting==0 || (ServoPerEvent==0 && EndLimitEvent==0)
Free:=0, nQ:=nQ-1, rQ:=(rQ<sizeQ-1 ? rQ+1 : 0)
if_DesNeutral
DesiredGear==NEUTRAL
w:=0
DesiredGear!=NEUTRAL
w:=0
Comp3
w<=U3
w>=L3
EngagedEvent:=0, nActivatedRS+=1,
MoveDir:=DesiredGear%2, RodServo:=(DesiredGear+1)/2
WaitEventE2
EngagedEvent==0
Free:=1, GBisWaitingE:=1
WaitingE2
EngagedEvent==1
wQuch!
EngagedEvent==1
Q[wQ]:=ID, GBisWaitingE:=0
Ready4
Urg!
Free==1, Q[rQ]==ID, RSisSuspended==0 || nActivatedRS==0,
RSisWaiting==0 || (ServoPerEvent==0 && EndLimitEvent==0)
Free:=0, nQ:=nQ-1, rQ:=(rQ<sizeQ-1 ? rQ+1 : 0), w:=0
Comp4
w<=U4
CloseClutch!
w>=L4
ClutchOpenedEvent:=0, CurrentGear:=DesiredGear
WaitEventCC
ClutchClosedEvent==0
Free:=1, GBisWaitingCC:=1
WaitingCC
ClutchClosedEvent==1
wQuch!
ClutchClosedEvent==1
Q[wQ]:=ID, GBisWaitingCC:=0
Ready5
Urg!
Free==1, Q[rQ]==ID, RSisSuspended==0 || nActivatedRS==0,
RSisWaiting==0 || (ServoPerEvent==0 && EndLimitEvent==0)
Free:=0, nQ:=nQ-1, rQ:=(rQ<sizeQ-1 ? rQ+1 : 0), w:=0
Comp5
w<=U5
w>=L5
GearBoxReady:=1, GBisSuspended:=1, Free:=1
End

Fig. 3.8. Gear Box task automaton



Suspended
wQuch!
nActivatedRS>0
nActivatedRS:=nActivatedRS-1, Q[wQ]:=ID,
RSisSuspended:=0, ServoPerEvent:=0, EndLimitEvent:=0
Ready1
Urg!
Free==1, Q[rQ]==ID
w:=0, Free:=0, nQ:=nQ-1, rQ:=(rQ<sizeQ-1 ? rQ+1 : 0)
Comp1
w<=U1
SetAlarm!
w>=L1
EndLimitEvent:=0, Move:=1, w:=0
Comp2
w<=U2
w>=L2
WaitEvent
ServoPerEvent==0, EndLimitEvent==0
Free:=1, RSisWaiting:=1
Waiting
ServoPerEvent==1
|| EndLimitEvent==1
wQuch!
ServoPerEvent==1 || EndLimitEvent==1
Q[wQ]:=ID, RSisWaiting:=0
Ready2
Urg!
Free==1, Q[rQ]==ID
Free:=0, nQ:=nQ-1, rQ:=(rQ<sizeQ-1 ? rQ+1 : 0)
if_End
EndLimitEvent==0
w:=0, ServoPerEvent:=0
EndLimitEvent==1
w:=0
CancelAlarm!
w>=L3
Comp3
w<=U3
Move:=0, EngagedEvent:=1, RSisSuspended:=1, Free:=1

Fig. 3.9. Shifting rod servo task automaton

4 CONCLUSION AND FUTURE WORK

With respect to the processor utilisation and reaction time the non-preemptive scheduling conceived in this article is not the most efficient one, but due to the simplicity reasons many embedded applications, where the available RAM is limited, are often based on similar non-preemptive scheduling. The non-preemptive scheduling approach given in this article allows creating accurate model based on timed automata. The inter-task synchronisation – the most important aspect of real time embedded applications – is taken into consideration in the proposed model.

Existing approaches for design and analysis of real-time applications, like Rate Monotonic Analysis, use very elegant way of deciding whether the application is schedulable or not. But it is needed to mention, that the model checking approach provides a room for verifying more complex properties (e.g. detection of deadlocks in communication, specification of buffer size,…). Model checking provides also room for modelling and verifying of more complex time behaviour of the controlled system, running truly in parallel with the control system (modelled as separate automaton).

As the complexity of the model checking remains very huge in a general case it is motivating to set up the rules applied at a design phase, that would lead into the state spaces of reasonable size. Specification of such rules linked to the identification of the controlled systems represents a possible direction of our future work.

ACKNOWLEDGEMENT

This work was supported by the Ministry of Education of the Czech Republic under Project LN00B096.

REFERENCES

Alur, R. and D.L. Dill (1994). A theory of timed automata. *Theoretical Computer Science* **126**, 183-235.

Amnell, T., E. Fersman, L. Mokrushin, P. Pettersson and W. Yi (2003). TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems. In: *Proceedings of the 1st International Workshop on Formal Modeling and Analysis of Timed Systems FORMATS'03*. Springer-Verlag.

Behrmann, G., A. David, K.G. Larsen, O. Möller, P. Pettersson, and Yi. W. (2001). UPPAAL - Present and Future. In: *Proceedings of the 40th IEEE Conference on Decision and Control (CDC'2001)*. Orlando, Florida, USA.

Bouyer, P., C. Dufourd, E. Fleury and A. Petit (2000). Are Timed Automata Updatable?. In: *Proc. 12th Int. Conf. Computer Aided Verification (CAV'00)*. LNCS **1855**, pp. 464-479, Springer-Verlag.

Buttazzo, G., C. (1997). *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Kluwer Academic Publishers, Boston.

Cassez F. and K. Larsen (2000). The Impressive Power of Stopwatches. In: *Proceedings of CONCUR 2000 - 11th International Conference on Concurrency Theory*. LNCS **1877**, p. 138 ff.

Corbett, J. C. (1996). Timing analysis of Ada tasking programs. In: *IEEE Transactions on Software Engineering*. **22**(7), pp. 461-483.

Fersman, E., P. Pettersson, and W. Yi (2002). Timed Automata with Asynchronous Processes: Schedulability and Decidability. In: *Proceedings of 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2002*. LNCS **2280**, pp.67-82, Springer-Verlag.

Fersman, E., P. Pettersson and W. Yi (2003). Schedulability Analysis using two clocks. In: *Proceedings of TACAS'03*, LNCS **2619**, pp 224-239. Springer-Verlag.

Henzinger, T.A., P.W. Kopke, A. Puri and P. Varaiya (1998). What's decidable about hybrid automata? *Journal of Computer and System Sciences* **57**:94-124

Liu, J.W.S. (2000). *Real-time systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey.

OSEK (2003). *OSEK/VDX Operating System Specification 2.2.1*. http://www.osek-vdx.org/

Sha, L., M. Klein and J. Goodenough (1991). Rate Monotonic Analysis for Real-Time Systems. In: *Foundations of Real-Time Computing: Scheduling and Resource Management*. 129-155. Boston, MA: Kluwer Academic Publishers.

Waszniowski, L. and Z. Hanzálek (2003a). Analysis of Real Time Operating System Based Applications. In: *Proceedings of FORMATS'03*. Springer-Verlag.

Waszniowski, L. and Z. Hanzálek (2003b). Model of Multitasking Applications with Preemption based on Timed Automata. Research report, Department of Control Engineering, Czech Technical University.