Model Checking Real-Time Systems with Schedulers

By Le Vo Hue Quan

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Toshiaki Aoki

September, 2012

Model Checking Real-Time Systems with Schedulers

By Le Vo Hue Quan (1010217)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Toshiaki Aoki

and approved by
Associate Professor Toshiaki Aoki
Professor Kokichi Futatsugi
Associate Professor Kazuhiro Ogata

August, 2012 (Submitted)

Table of content

1 Introduction **6**
- 1.1 Real-time operating systems . . . 6
- 1.2 Verifying real-time systems with model checking tools . . . 7
- 1.3 Problem . . . 8
- 1.4 Our approach . . . 9
- 1.5 Outline of thesis . . . 10

2 Background **12**
- 2.1 Scheduling problem . . . 12
  - 2.1.1 Tasks . . . 12
  - 2.1.2 Scheduler . . . 13
  - 2.1.3 Scheduling problem . . . 14
- 2.2 Scheduling analysis . . . 14
- 2.3 Model checker selection . . . 15
  - 2.3.1 SPIN . . . 15
  - 2.3.2 UPPAAL . . . 16
  - 2.3.3 TIMES . . . 17
  - 2.3.4 Summary . . . 17

3 Abstraction of Scheduler **19**
- 3.1 Overview . . . 19
- 3.2 Model of the abstraction scheduler . . . 19
- 3.3 Converting from model to UPPAAL . . . 23
- 3.4 Generating priority assignment set . . . 24
  - 3.4.1 Implementing priority assignment generation . . . 27

4 Verification Framework **30**
- 4.1 Framework for verification . . . 30
- 4.2 Preemption . . . 34
- 4.3 To invoke scheduler by system call . . . 34
- 4.4 To invoke sporadic task . . . 35
- 4.5 Using verification framework . . . 35

5 Experiment **37**
5.1 Real-time Sorting Machines Problem . . . 37
5.1.1 Design . . . 37
5.1.2 Scheduling analysis . . . 38
5.1.3 Experiment . . . 39
5.1.4 Appropriate scheduling policies . . . 41
5.1.5 Evaluation . . . 49

6 Conclusion and Future work **50**
6.1 Conclusion . . . 50
6.2 Related Works . . . 51
6.2.1 Model-based framework for Schedulability Analysis Using Uppaal 4.1[7] . . . 51
6.2.2 TIMES Tool . . . 52
6.2.3 RTSM group . . . 52
6.3 Future work . . . 53
6.3.1 Provide criterion of choosing appropriate scheduler . . . 53
6.3.2 Prove the correctness of abstracted scheduler model . . . 53
6.3.3 Realistic situation of real-time systems . . . 53

Bibliography **55**

7 Appendix **57**
7.1 UPPAAL . . . 57

List of Figures

1.1 RTOS main Component 6
1.2 Task management structure 7
1.3 The overview of our approach 9

2.1 Overview of Uppaal 16
2.2 Times tool framework 17

3.1 Example abstracted scheduler model 20
3.2 Execution relation of tasks under Round robin algorithm 22
3.3 The relation on task execution under EDF mechanism 23
3.4 Graph representation of this sample ASM model 26
3.5 Implementation design 27

4.1 A system verification framework in which abstracted scheduler work 30
4.2 Basic model of scheduler 31
4.3 Basic model of periodic task 32
4.4 Basic model of sporadic task 33
4.5 Embedded miss deadline error property 34
4.6 Scheduler invocation 35
4.7 Sporadic task invocation 35

5.1 Design for the Real-Time Sorting Machines example 37
5.2 Abstracted Scheduler Model for RTSM problem 39
5.3 RTSM checking result 41
5.4 Modified basic task model 42
5.5 Task transition after a sporadic task invocation 45

7.1 (a) A simple client communicating with (b) a simple buffer. 57
7.2 Three different automata with a local clock 59

List of Tables

3.1 ASM model table 21
3.2 Table representation of this sample ASM model 26

5.1 ASM model for RTSM 40
5.2 Scheduling policies for real-time system 43
5.3 Summary of appropriate scheduling policy for RTSM design 49

Acknowledgements

First and foremost, I would like to express my deepest thanks to my supervisor − Associate Professor Toshiaki Aoki for his guidance through the study program in JAIST, Japan. His valuable advice and support have helped me to overcome many difficulties in the studying and researching during my study in JAIST. He has made me interested in science and doing research. My study could not have been completed without him.

Secondly, I would like to acknowledge Asia Jinzai Program for giving me the chance to study abroad, and the financial support on the study of Master program in JAIST, Japan.

Thirdly, I would like to thank to all teacher, staff at JAIST who have helped me a lot during the study time in JAIST, Japan.

Finally and most importantly, I would like to thank my family for their unconditional love and encouragement through my study program and throughout my life.

Chapter 1

Introduction

1.1 Real-time operating systems

In computer science, real-time computing (RTC), or reactive computing, is the study of hardware and software systems that are subject to a "real-time constraint" e.g. operational deadlines from event to system response. Real-time programs must guarantee response within strict time constraints. Often real-time response times are understood to be in the order of milliseconds and sometimes microsecond. Real-time software may use one or more of the following: synchronous programming languages, real-time operating systems, and real-time networks, each of which provide essential frameworks on which to build a real-time software application.

A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications request. Nowadays real-time systems are used in a wide variety of applications, systems, including space applications, electronic automotive and so on. Due to its application to many areas in society, the correctness of RTOS in time domain is very important. It means that the problem of checking whether the system complete its work and deliver its service on time or not is the most important point in design and developing of RTOS. As the structure of RTOS, there are some key components like following figure:

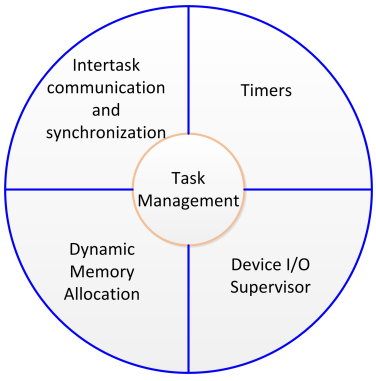


Figure 1.1: RTOS main Component

In the task management component, scheduler with scheduling policies will guarantee the correctness of RTOS in real-time domain. Because a real-time system is composed by many tasks, and scheduler is the component which determine the execution order of task.

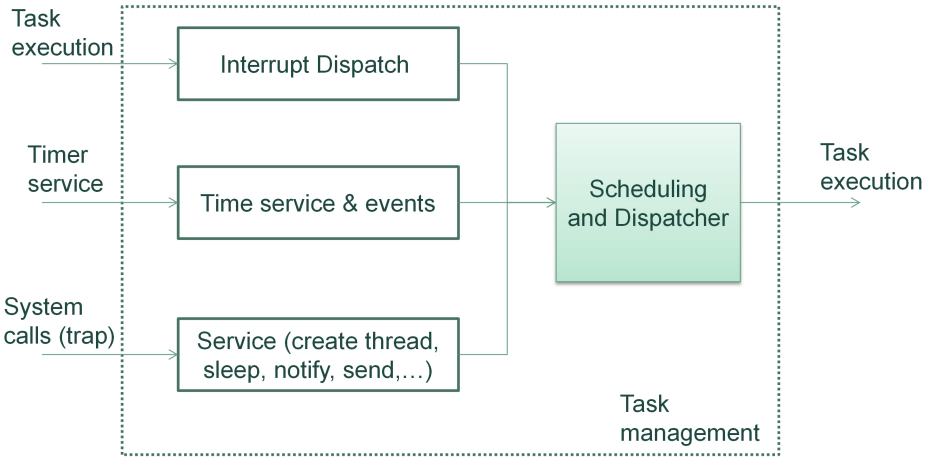


Figure 1.2: Task management structure

The challenge of schedulability analysis is now concerned with guaranteeing that the applied scheduling principles for a set of system properties ensure that the timing deadlines are met.

1.2 Verifying real-time systems with model checking tools

There are two main approaches in verifying real-time systems:

- Simulation: try to make a simulation tool in order to simulate the behaviors of real-time systems.
- Model checking: try to create finite-state models that represent the task's state transition, system environment and scheduler behavior. By using model-checking engine to verify the collaboration between those model.

In the first approach, there are many researches about creating different simulation tools for different systems with the variety of system environments, scheduler policies and task properties. For example

- Realtss: A Real-time Scheduling Simulator - gives a framework to evaluate the real-time scheduling algorithm with given a specific of task set. Using this tool we can evaluate a scheduling algorithm without worry about a system environment implementation.

- RTSSIM: A Simulation Framework for Complex Embedded Systems - provides a framework to analysis the task behavior in time domain and resource usage. Using this tool we can understand how a task will behave in real execution.
- Stress: Simulator for Hard Real-Time Systems - provides a framework to simulate both applications and kernel software on various architectures. It comprises a single generic language for architecture description, kernel software, and application software.
  ...

However, if we need to prove the correctness of real-time system - for example, whether an execution will has a deadlock condition or not - this approach is not appropriate. Because in simulation tools, they just only simulate the execution of the system in some specific circumstances. They can not provide the evidence to prove the correctness of real-time system for all situation of the system.

The second approach to verify real-time system is based on model-checking technique. Recent years, model-checking has turned out to be a useful technique for verifying temporal properties of finite state systems. In the last few year, model-checking has been extended to real-time systems, with time considered to be a dense linear order. We can use this state of the art - timed automaton - in order to deal with scheduling analysis of real-time system. A timed extension of finite automata through addition of a finite set of real valued clock variables has been put forward so called timed automata and the corresponding model-checking problem has been proven decidable for a number of timed logics including timed extensions of CTL (TCTL) and μ-calculus (T_μ) [5].

In this approach, the TIMES tool and a framework created by UPPAAL group [7] are outstanding works. These works provide framework in order to analyze scheduling problems. We need to provide necessary system information into these tool: task properties, scheduling policies, resource constraints, and so on. The tool will base on model checking engine to exhaustively check all system states. By exhaustively check all state of the system, model-based analysis tool can give a firm evidence on our desired properties of real-time systems.

1.3 Problem

Recent years, there are many research about verifying the validity of some specific scheduling mechanism of different system. We also have another direction in scheduling analysis, that is finding the set of condition on a task design so that the scheduling problem is feasible or not. This is classical problem, and it is have been investigated by many researcher. However, in other research they usually deal with a lot worst-case assumption on task and system. These assumption are usually based on a worst case computation time of a task. However typically, systems fail, in a timing sense, in very constrained circumstances, with the average-case, as often seen on the testcase, performing adequately. So the consequent result when we base on worst-case assumption will very pessimistic.

By using second approach - model-based scheduling analysis - we can deal with more exactly timing information of real-time tasks. However, beside the detail task behavior which reflects the detail timing information of task, there are many information related to real-time scheduling analysis. For example, if we want to check if a given task design is suitable with a new environment or not. We have to implement the system environment in model checking tool first, like in [1]. After having an environment, we have to check all combinations of scheduling strategies, task priority, task offset, task activation time. This work in some case is not reasonable. Because, the combination of all above factors is large number and due to the functional limitation of model-checking tool, implement every possible scheduling policy is almost impossible.

> In the case of UPPAAL tool, because we can not access the internal information of a process (template) from another template. So, we can not implement the Slack Stealing scheduling mechanism, because in order to compute slack time of all current task, it requires to have the access to all information of current running task.

1.4 Our approach

To deal with above problem, we proposal a method to abstract the behavior of scheduler. This abstraction makes it possible to check RTS with multiple schedulers at once and analyze broad properties related to the schedulers. Our purpose is by using abstracted scheduler, we can find out in which set of condition on real-time tasks, a given task design will be feasible, like:

- Scheduling policy
- Task priority
- Task cost

Following figure is the overview of our approach:

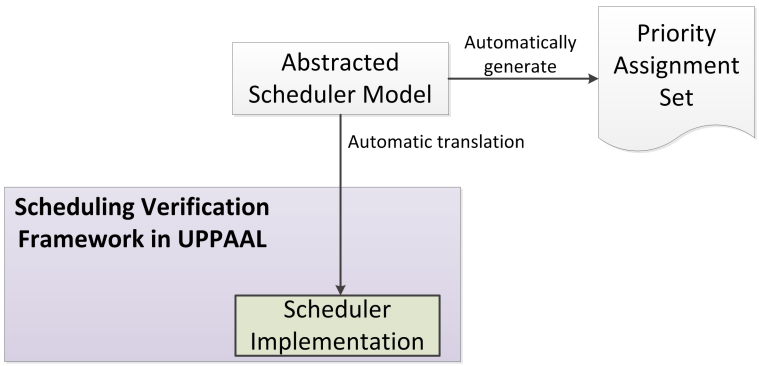


Figure 1.3: The overview of our approach

- **Abstracted Scheduler Model**: we will define what is the abstracted scheduler model in here.
- **Scheduling verification framework in UPPAAL**: we are trying to do scheduling analysis in more effective way than traditional approach. Therefore, we need a framework to verify scheduling problem. The framework is built in UPPAAL model checking tool. The framework will contain task, task behavior and scheduler. The framework is reusable.
- Generate scheduler implementation: in the verification framework, it has scheduler component. This scheduler component is in UPPAAL language, so we need a implementation of our abstracted scheduler in UPPAAL language. We will propose an algorithm in order to automatically translate from abstracted scheduler model into UPPAAL programming language.
- Generating priority assignment: our purpose is through abstracted scheduler to find out set of conditions in which a given task design is feasible. In our work, this set of conditions will contain: task priority assignment and scheduler policy. We will propose an algorithm which automatically generate all possible priority assignment from abstracted scheduler model.

1.5 Outline of thesis

In this thesis we will report our work as following order:

- Chapter background:
  - We will discuss about the common standard of real-time system to do scheduling analysis and our research scope, goals.
  - We will explain why we choose UPPAAL as our model checking tool in order to realize our idea.
- Chapter Abstraction Scheduler
  - We will introduce the formal model of Abstraction Scheduler and its meaning.
  - Proposing an algorithm to transform Abstraction Scheduler model into UPPAAL model.
  - How to generate the priority assignments for a given task design from abstraction scheduler model.
- Chapter Verification Framework:
  - Explaining the system framework in which abstraction scheduler will work.
  - Explain how to implement our idea in UPPAAL model-checking.

- Chapter Experiment and evaluation: we will do an experiment on Real-time sorting-machine system design.
- Chapter Conclusion: we will summarize our achievement in this work and also future improvement needed for our work.

Chapter 2

Background

In this chapter we will mention about what is the important factors need to be considered when we do scheduling analysis. In the second part of this chapter, we will briefly describe about available model checking tools for real-time system, and why we choose UPPAAL as our model-checker tool for this thesis.

2.1 Scheduling problem

In this section, we will give a discussion on what is the scheduling problem of real-time system. At the core of any schedulability problem are the notions of tasks and scheduler. Tasks are logical units of computation. Tasks are executed under the decision of scheduler. The added constraints to this basic setup is what defines a specific schedulability problem. In this section, we define a range of classical schedulability problems.

2.1.1 Tasks

A schedulability problem always consists of a finite set of tasks that we consistently will refer to as $T = t_1, t_2, ..., t_n$.

Depending on the consequences that may occur because of a missed deadline, a real-time task can be distinguished in three categories:[3]

- **Hard**: A real-time task is said to be *hard* if producing the result after its deadline may cause catastrophic consequences on the system under control.
- **Firm**: A real-time task is said to be *firm* if producing the result after its deadline is useless for the system, but not cause any damage.
- **Soft**: A real-time task is said to be *soft* if producing the result after its deadline has still some utility for the system, although causing a performance degradation.

Each task has a number of attributes that we refer to by the following functions:

- INITIAL OFFSET: $T \rightarrow N$

  Time offset for initial release of task.

- BCET: $T \rightarrow N_{\geq 0}$

  Best case execution time of task.

- WCET: $T \rightarrow N_{\geq 0}$

  Worst case execution time of task.

- MIN_PERIOD: $T \rightarrow N$

  Minimum time between task releases.

- MAX_PERIOD: $T \rightarrow N$

  Maximum time between task releases.

- OFFSET: The time offset into every period, before the task is released.

- DEADLINE: $T \rightarrow N_{\geq 0}$

  The number of time units within which a task must finish execution after its release. Often, the deadline coincides with the period.

- PRIORITY: Task priority.

These attributes are subject to the obvious constraints that $BCET(t) \leq WCET(t) \leq DEADLINE(t) \leq MIN\_PERIOD(t) \leq MAX\_PERIOD(t)$. The periods are ignored if the task is non-periodic.

The interpretation of these attributes is that a given task t_i cannot execute for the first $OFFSET(t_i)$ time units and should hereafter execute exactly once in every period of $PERIOD(t_i)$ time units. Each such execution has a duration in the interval $[BCET(t_i), WCET(t_i)]$. The reason why tasks have a duration interval instead of a specific duration is that tasks are often complex operations that need to be executed and the specific computation of a task depends on conditionals, loops, etc. and can vary between invocations. Furthermore, for multi-processor scheduling, considering only worst-case execution times are not is insufficient as deadline violations can result from certain tasks exhibiting best-case behavior.

We say that a task t is *ready* (to execute) at time τ iff:

1. $\tau \geq$ INITIAL_OFFSET(t)

2. t has not executed in the given period dictated by τ.

2.1.2 Scheduler

In order to determine which task to execute and which tasks to hold, scheduling policy is needed. Scheduling policies vary greatly in complexity depending on the constraints of the schedulability problem.

Scheduling strategy for real-time systems can be classified into 2 groups: preemptive and non-preemptive.

- Preemptive scheduling is the act of temporarily interrupting a task being carried out by a computer system, without requiring its cooperation, and with the intention of resuming the task at a later time.
- Non-preemptive scheduling is a style of computer multitasking in which the system will never initiates a context switch from a running task to another.

Preemption characteristic is determined by many factor of task depend on the strategy of scheduler: task deadline, activation time, priority and so on. A good scheduling strategy for real-time system is a policy that guarantee all hard real-time periodic and sporadic task are meet their deadline.

2.1.3 Scheduling problem

Now, we define what it means for a system to be feasible. For a given system's design with scheduling policies is said to be feasible if no execution satisfying the constraints of the system violates a deadline.

2.2 Scheduling analysis

In other researches like [2, 7, 9] scheduling analysis for real-time systems is based on some assumption for the system:

- The system can be in finite number of states.
- There are transitions between states.
- The transitions are sensitive to the time in which they occur.
- There is a global clock in the system that measure its time (global clock tics).
- All states and transitions share the same clock rate (but can have clocks of their own with the same rate)

In our work, we also work with above assumptions, because it will make our work is reasonable when compare with related work. In real-time system or another operating system, the system environment contains a lot of information and components like: scheduler itself, resource (memory, bus, i/o component), communicate control component, tasks (with different types), system kernel. Deal with everything at once will make the problem of scheduling analysis become unnecessary complicated. Therefore, in our work and also in many related works, we want to focus on scheduler itself and system tasks.

For the scheduler and task in real-time system, there are also many factors we need to consider when do scheduling analysis. Among that factors, the important factors need to be considered in many related research are:

- Scheduling strategy

- Number and task type : periodic, sporadic.
- Task properties: activation time, offset, worst and best computation time, deadline, task behavior

These above factors is our aimed system or the scope of our research. About the goal or purpose of performing scheduling analysis is to aim the the following common goals:

- Reachable state: checking if some desired states are reached or not in system executions.
- Deadline miss: check that all hard real-time tasks are done by their deadline or not.
- Deadlock conditions: check that if the deadlock conditions is happen or not in system execution.

2.3 Model checker selection

As the discussion in chapter Introduction, we want to follow model-based scheduling analysis in order to realize our approach. Our intention is choosing a model-checking tool that can be applied for various features of real-time systems like: time related properties, cost.

2.3.1 SPIN

SPIN is a general tool for verifying the correctness of distributed software models in a rigorous and mostly automated fashion. It was written by Gerard J. Holzmann and others in the original Unix group of the Computing Sciences Research Center at Bell Labs, beginning in 1980. The software has been available freely since 1991, and continues to evolve to keep pace with new developments in the field.

Systems to be verified are described in Promela (Process Meta Language), which supports modeling of asynchronous distributed algorithms as non-deterministic automata (SPIN stands for "Simple Promela Interpreter"). Properties to be verified are expressed as Linear Temporal Logic (LTL) formulas, which are negated and then converted into Bchi automata as part of the model-checking algorithm. [18]

In the basic Promela language there is no mechanism for expressing properties of clocks or of time related properties or events. There are good algorithms for integrating real-time constraints into the model checking process, but most attention has so far been given to real-time verification problems in hardware circuit design, rather than the real-time verification of asynchronous software, which is the domain of the Spin model checker. [17] The best known of these algorithms incur significant performance penalties compared with untimed verification. Each clock variable added to a model can increase the time and memory requirements of verification by an order of magnitude. Considering that one needs at least two or three such clock variables to define meaningful constraints, this seems to imply, for the time being, that a real-time capability requires at least three to

four orders of magnitude more time and memory than the verification of the same system without time constraints.

Promela is a language for specifying systems of asynchronous processes. For the definition of such a system we abstract from the behavior of the process scheduler and from any assumption about the relative speed of execution of the various processes. It can be hard to define realistic time bounds for an abstract software system. Typically, little can be firmly known about the real-time performance of an implementation. It is generally unwise to rely on speculative information, when attempting to establish a system's critical correctness properties.

2.3.2 UPPAAL

Uppaal is a tool box for modeling, simulation and verification of real-time systems, based on constraint-solving and on-the-fly techniques, developed jointly by Uppsala University and Aalborg University. It is appropriate for systems that can be modeled as a collection of nondeterministic processes with finite control structure and real-valued clocks, communicating through channels and (or) shared variables.

Uppaal consists of three main parts: a description language, a simulator, and a model-checker. The description language is a non-deterministic guarded command language with data types. It serves as a modeling or design language to describe system behavior as networks of timed automata extended with data variables. The simulator and the model-checker are designed for interactive and automated analysis of system behavior by manipulating and solving constraints that represent the statespace of a system description. They have a common basis, i.e., constraint-solvers.The simulator enables examination of possible dynamic executions of a system during earlymodeling (or design) stages and thus provides an inexpensive mean of fault detection prior to verification by the model-checker which covers the exhaustive dynamic behavior of the system.

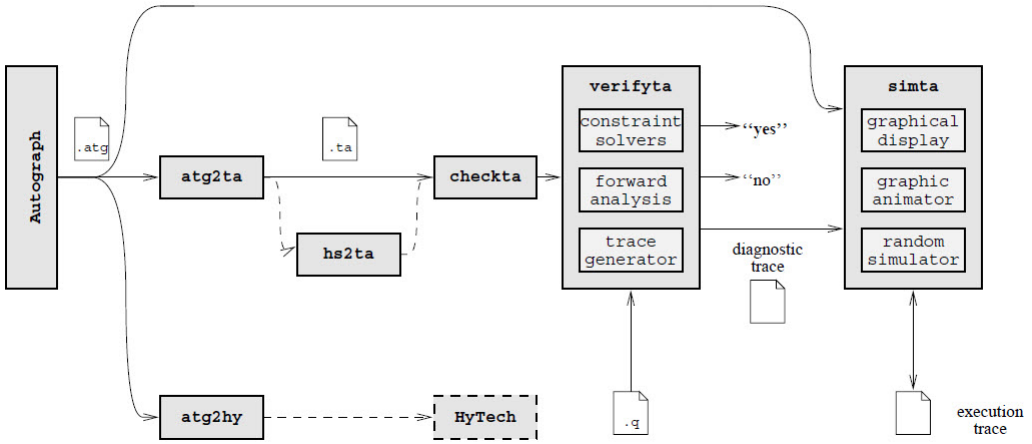


Figure 2.1: Overview of Uppaal

2.3.3 TIMES

Times is a modelling and schedulability analysis tool for embedded real-time systems, developed at Uppsala University. It is appropriate for systems that can be described as a set of preemptive or non-preemptive tasks which are triggered periodically or sporadically by time or external events. It provides a graphical interface for editing and simulation, and an engine for schedulability analysis. [19]

A system specification in Times consists of three parts: the control automata modelled as a network of timed automata extended with tasks, a task table with information about the processes triggered (released) when the control automata changes location, and a scheduling policy.

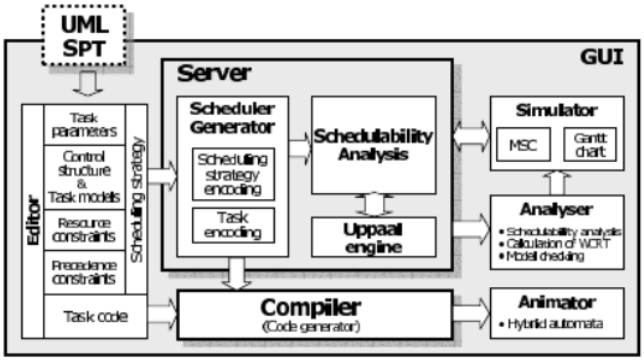


Figure 2.2: Times tool framework

Times tool provides a facility to perform scheduling analysis with expressive task-model. It means that we can set many parameters of tasks of real-time system in Time tool: priority, computation time, deadline, period and behavior. However, in Times tool we can only analyze four kinds of scheduling policy: Rate Monotonic, Deadline Monotonic, Earliest Deadline First, First Come First Served.

2.3.4 Summary

Our goal in this thesis is trying to find out in which set of conditions of tasks in real-time system, scheduling problem is feasible. As the briefly description about Spin model-checker tool above, SPIN can not directly deal with real-time properties. Moreover, in Spin we can not deal with the cost of a task's behavior like: computation time, switching cost, and so on. So the choice of SPIN is not appropriate in our research.

In the case of Times tool, it has the framework for us to perform scheduling analysis with various task properties: periodic, aperiodic, sporadic, task priority, so on. It can give us evidence to prove that with a given task set and a specific scheduling policy, the system is feasible or not based on UPPAAL engine. However, it can not give us in which task properties set, the system is feasible. Moreover, if we can to verify the system in different situations, we have to do this verification one by one. It is not reasonable because the

combination of task properties (task priority, computation time, task behavior, arrival time, deadline, and so on) is large.

Therefore, after considering related model-checking tools used for timed related system, we choose UPPAAL as our model-checking tool in order to realize our idea in this thesis. Because it provides us a open framework to realize our idea in real-time system concept. Moreover, UPPAAL tool also supports stopwatch automata. By using stopwatch automata we can dealwith the fifth assumption 2.2 in scheduling analysis for real-time system. In Appendix part, we will briefly introduce about Uppaal model based on Uppaal tutorial available at `http://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf`

Chapter 3

Abstraction of Scheduler

3.1 Overview

The purpose of scheduler in real-time scheduling is to decide the execution order of system's task such that the real-time properties are hold. Real-time properties are:

- Every hard real-time tasks meet its deadline.
- Every soft real-time tasks finish their task in reasonable time.

In this chapter, we will introduce our idea about creating an abstraction scheduler and explain how to use this model to verify a system.

3.2 Model of the abstraction scheduler

There are two main method in abstraction techniques: under-approximation and over-approximation method. In under-approximation we will create an abstraction model which is a subset of real problem, and try to prove the incorrect facts of real-problem by using under-approximation model. In contrast, over-approximation method, we will create an abstraction model which is a superset of real problem, and try to prove the correct facts of real-problem by using over-approximation model.

A scheduler will create an execution order of task depends on input parameters. Input parameters are scheduling policy, task priority, task deadline, activation time and so on. In our work, we want to propose a method to create an abstraction scheduler. This can abstract the scheduling policy, task priority and task cost. In addition, in our work we can deal with the functional view of real-time task. Therefore we need a mechanism to describe the abstracted scheduler and how to use it.

Firstly, we will introduce the concept in order to define the abstraction scheduler:

Definition A model of abstracted scheduler S is a set of {T,E,f} where T is a set of tasks {t_1, t_2, t_3,...}, and E is a set of edges between the task E $\subseteq \{(t_1, t_2)|t_1, t_2 \in T\}$. Where edge is the relation on tasks represent the execution order of task. That is, each

edge can be followed from one task to another task or itself. We have 2 types of edges: non-preemptive edge and preemptive edge

- Non-preemptive edge: the succeeding task can not preceding task
- Preemptive edge: the succeeding task can preempt preceding task

f: is the assignment function of abstracted scheduler model, it will assign a edge to it's corresponding value: preemptive or non-preemptive.

We can define the abstracted scheduler model formally like following:

Definition Abstraction Scheduler : S = (T,E,f)

- T = $(t_1, t_2, t_3, ...)$
- E $\subseteq$ T x T
- f: E $\rightarrow$ {1, 0.5}
  - "1" is corresponding with non-preemptive edge.
  - "0.5" is corresponding with preemptive edge.

For example, in a system with 5 tasks and we want to have an abstracted scheduler work like following definition:

- Example abstraction scheduler model : S_1 = {T, E, f}
  - T = { t_1, t_2, t_3, t_4, t_5 }
  - E = $\{(t_1, t_2), (t_2, t_3), (t_2, t_4), (t_3, t_4), (t_4, t_5), (t_1, t_5)\}$
  - f: (1,2) $\rightarrow$ 1 ; (2,3) $\rightarrow$ 0.5 ; (2,4) $\rightarrow$ 0.5; (3,4) $\rightarrow$ 1; (4,5) $\rightarrow$ 1; (1,5) $\rightarrow$ 0.5

We can get a abstraction model like in following figure:

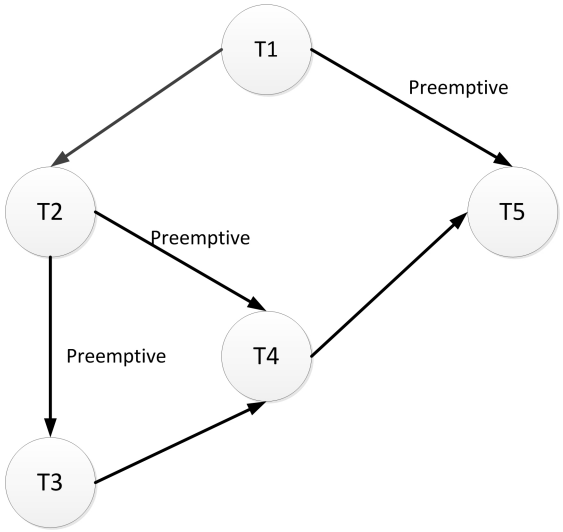


Figure 3.1: Example abstracted scheduler model

Where each node represent a task in the system, each transition represent the edge of abstracted scheduler model (ASM). The transition name represent the name (or value) of each edge in ASM.

- Transition with no name: the succeeding task can not preempt the preceding task
- Name of transition is Preemptive: the succeeding task can preempt the preceding task

After we have a abstracted scheduler model is defined by using our method 3.2, we can represent it by using an 2-dimension array. For example, like in our previous example, the ASM_model can be represented like:

Task	T_1	T_2	T_3	T_4	T_5
T_1	NG	NG	NG	NG	NG
T_2	1	NG	NG	NG	NG
T_3	NG	0.5	NG	NG	NG
T_4	NG	0.5	1	NG	NG
T_5	0.5	NG	NG	1	NG

Table 3.1: ASM model table

In above example of abstracted scheduler model (ASM model), we will have following execution order of this given 5 tasks:

- If there is only one task ready for execution, it will has permission for execution.
- When task 1 is executing, task 2 is ready for execution. Task 2 must wait until task 1 finish it's work.

  However, in this situation, if task 5 is ready for execution, it will preempt the execution of task 1.

  Task 1 will continue it's execution after task 5 finish it's work
- It is the same mechanism in the case that task 2 is executing and task 3 or task 4 is ready for execution.

Through this simple example, we can get an idea of the meaning of abstracted scheduler model. It will make a abstracted schedule among many possible schedules created by different scheduling policies. Each scheduling policy based on it's algorithm and the input information from task in order to make the schedule of task.

For example, in the case of round robin scheduling algorithm, we will have following execution relation between tasks:

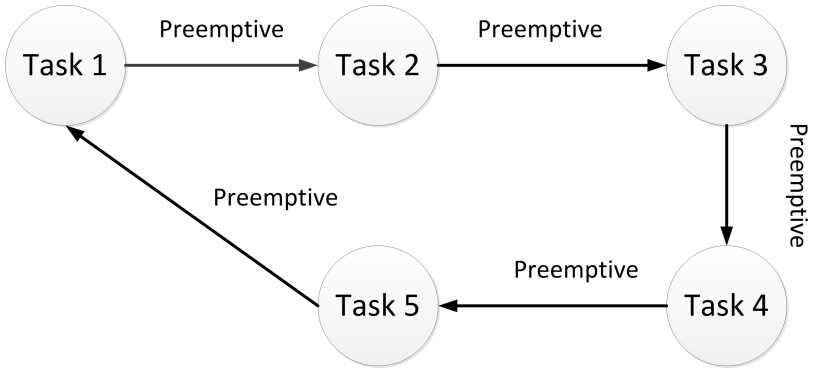


Figure 3.2: Execution relation of tasks under Round robin algorithm

In the case of Earliest Deadline First scheduling algorithm, depend on the task information - task activation time, offset, deadline - it will give a different schedule of task execution. For example, we have following task design information:

- Task activation time
  - Task 1 is activated at 0 time unit.
  - Task 2 is activated at 0 time unit.
  - Task 3 is activated at 4 time unit.
  - Task 4 is activated at 4 time unit.
  - Task 5 is activated at 6 time unit.
- Task deadline
  - Task 1 has to be finished within 7 time unit.
  - Task 2 has to be finished within 4 time unit.
  - Task 3 has to be finished within 5 time unit.
  - Task 4 has to be finished within 5 time unit.
  - Task 5 has to be finished within 10 time unit.

Based on above assumption on task information, we will have following relation between task executions:

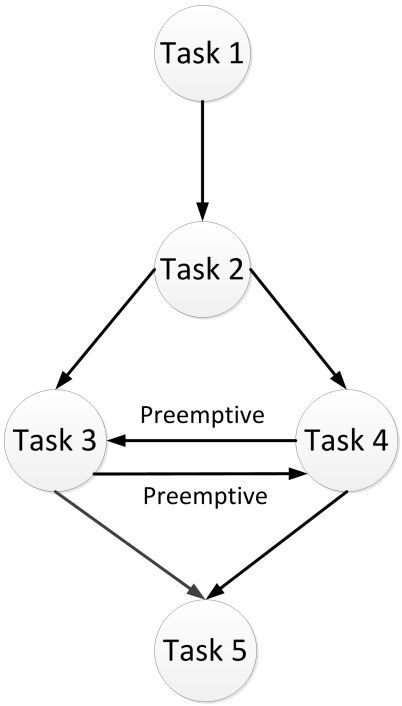


Figure 3.3: The relation on task execution under EDF mechanism

Therefore, we can recognize that the relation between task execution will be various under different scheduling policies and different settings. Our abstracted scheduler model will create an abstracted relation between task execution order. By applying the abstracted model to a concrete example, we can known that which set of conditions it cover.

3.3 Converting from model to UPPAAL

In our work, we will use UPPAAL model checking tool as the verification tool. UPPAAL has its own programming language. So in order to make our idea work, we must have UPPAAL code for abstracted scheduler. This section we will introduce how to convert the abstraction scheduler model (ASM) into UPPAAL code.

By using algorithm 1, we can translate ASM description into UPPAAL code.

input : Abstracted Schedule model table - ASM_table

output: Uppaal scheduling code

```
1  declare iterator i, j;
2  for each task i in system do
3  |  if task i is schedulable then
4  |  |  switch i do
5  |  |  |  case i
6  |  |  |  |  for each task j on i row do
7  |  |  |  |  |  if ASM_table[i,j] = 1 then
8  |  |  |  |  |  |  if task j is schedulable then  selectedTask = j;
9  |  |  |  |  |  end
10 |  |  |  |  |  else if ASM_table[i,j] = 0.5 then  selectedTask = i;
11 |  |  |  |  end
12 |  |  |  endsw
13 |  |  |  otherwise
14 |  |  |  |  selectedTask = i;
15 |  |  |  endsw
16 |  |  endsw
17 |  end
18 end
19 for each task i in system do
20 |  running_array[i] = 0 ;                                  // Stop all task
21 end
22 running[selectedTask] = 1;
```

Algorithm 1: How to convert from ASM model to UPPAAL code

The idea of this algorithm is based on the execution order relation's value of task in table form, to determine which task will be executed first, which task will be next. In the UPPAAL framework, we have an array - **running** - to indicate which task is currently executing. Based on the input table information, we will set the appropriate value to this array.

The UPPAAL programming language is C-like language, however it does not support **switch - case** control, so in the UPPAAL code it will be replaced by **if - then - else** control sequence.

3.4 Generating priority assignment set

In our research, many information of scheduling problem is abstracted. One of important information is task priority. Task priority is needed to schedule real-time tasks in almost scheduling algorithms. Therefore, after having an abstracted scheduler model, we want to extract information that in which priority set the tasks design of system is feasible.

Following algorithm will help us to do that:

input : Abstracted Schedule model table - ASM_table
output: Priority assignment set

```
1  declare iterator i, j and index;
2  List out all possible combination from number of task - n^n combination into
   result array;
3  for each task i in system do
4  |  for each task j in system do
5  |  |  if i == j then
6  |  |  |  next;
7  |  |  end
8  |  |  else
9  |  |  |  index++;
10 |  |  end
11 |  |  if ASM_model[i,j] == 1 then
12 |  |  |  for each k in result array do
13 |  |  |  |  if result[k]i < result[index]j then
14 |  |  |  |  |  remove result[k];
15 |  |  |  |  end
16 |  |  |  end
17 |  |  end
18 |  |  else if ASM_model[i,j] == 0.5 then
19 |  |  |  for each k in result array do
20 |  |  |  |  if result[index]i >= result[index]j then
21 |  |  |  |  |  remove result[k];
22 |  |  |  |  end
23 |  |  |  end
24 |  |  end
25 |  end
26 end
27 Eliminate wrong initial selection;
```

Algorithm 2: How to generate priority assignment from ASM model

The idea this algorithm is we base on the relation of execution order in ASM model, to determine the priority assignment of tasks:

- If a task can preempt another task, this task will have a higher priority than the preempted task.
- If a task can not preempt another task, it will have a lower priority than the preceding task.
- If there is no execution order relation between tasks, these can can have arbitrary priority assignment.

- Because the graph model is transitive, so the relation of task has transitive property.

For example, if we have a ASM model is described as following:

ASM Model S: (T, E, f)

T: t_1, t_2, t_3

E: {(t1,t2), (t1,t3)}

f: (t1,t2) $\rightarrow$ 0.5, (t1,t3)$\rightarrow$ 0.5

Task	$Task_1$	$Task_2$	$Task_3$
$Task_1$	NG	NG	NG
$Task_2$	0.5	NG	NG
$Task_3$	NG	0.5	NG

Table 3.2: Table representation of this sample ASM model

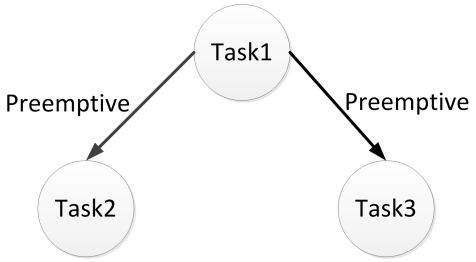


Figure 3.4: Graph representation of this sample ASM model

Based on above algorithm for generating priority assignment, we will get following assignment:

```
(1, 2, 2)
(1, 2, 3)
(1, 3, 2)
(1, 3, 3)
```

Listing 3.1: Priority assignment list for this sample model

3.4.1 Implementing priority assignment generation

Our work is based on UPPAAL model checking tool, however above algorithm is used for evaluating our abstracted scheduler model. Therefore, we will use a structure supported programming language to realize our idea. In this work, we use Java as the language to implement automatic priority generation algorithm. In this sub-section, we will explain how we implemented it in Java.

As the design for this implementation, we will have only one processing component which represent our algorithm, as following figure 3.5

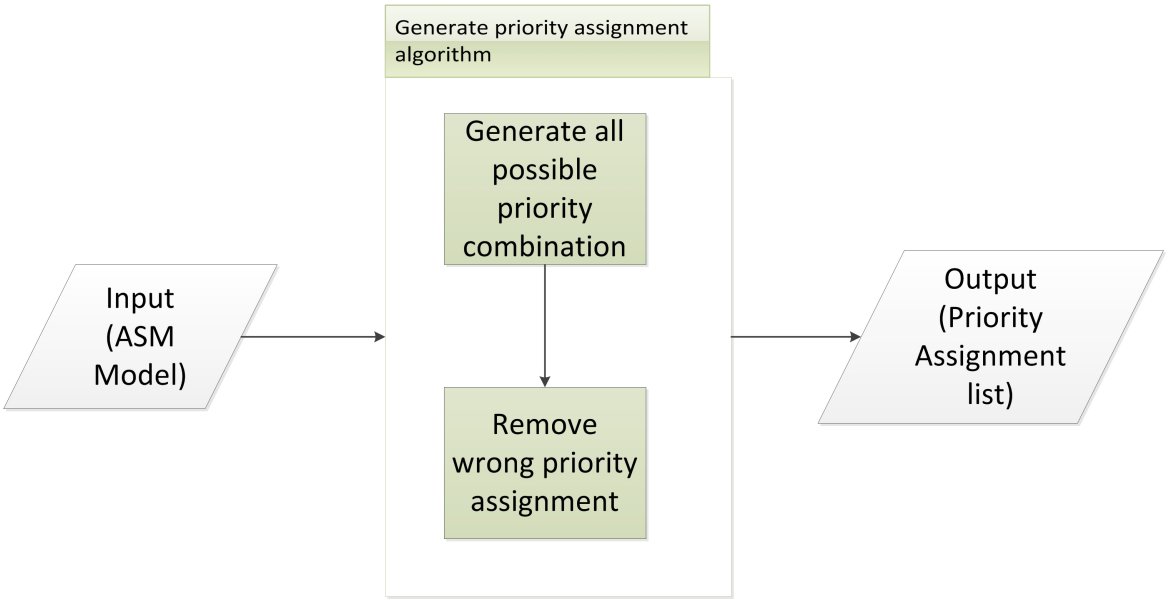


Figure 3.5: Implementation design

As in algorithm description, the input of algorithm is a abstracted scheduler model in table form. We will represent it in Java by using 2-dimensional array.

- The **NG** value will be represented by **0**.
- The **1** value will be represented by **1**.
- The **0.5** value will be represented by **0.5**.

For example, in previous example, input of our algorithm will be:

```
double[][] asm = new double[][] {{0,0,0},{0.5,0,0},{0,0.5,0}}
```

Listing 3.2: Input for our example model

Base on the automatic priority generation algorithm, we have two main steps:

- **Step 1**: List all possible priority combinations and put into result array.

- **Step 2**: Eliminate wrong combination from result array.

In our example, for the first step, by using Java we will get following implementation:

```
int i,j,k;
ArrayList<Object> result = new ArrayList<Object>();
for(i = 1;i<=3;i++)
{
        for(j = 1;j<=3;j++)
        {
                for(k = 1;k<=3;k++)
                {
                        result.add(new int[] {i, j, k});
                }
        }
}
```

Listing 3.3: First step of automatic generating algorithm

The second step, we will base on the abstracted scheduler model in table form to eliminate wrong priority assignment:

```
for(i = 1;i<=3;i++)
{
        for(j = 1;j <= 3;j++)
        {
        if(i != j)
        {
                if(asm[i][j] == 1)
                {
                        for(k = 0;k<result.size();k++)
                        {
        if(((int[])result.get(k))[i] < ((int[])result.get(k))[j])
                result.remove(k);
                        }
                }
                else if(asm[i][j] == 0.5)
                {
                        for(k = 0;k<result.size();k++)
                        {
        if(((int[])result.get(k))[i] >= ((int[])result.get(k))[j])
                result.remove(k);
                        }
                }
        }
        }
```

```
}
```

Listing 3.4: Second step of automatic generating algorithm

The content of **result** array is the priority assignments after running this algorithm, as following listing:

```
(1, 2, 2)
(1, 2, 3)
(1, 3, 2)
(1, 3, 3)
```

Chapter 4

Verification Framework

In this chapter, we will explain how we implement our idea and build a framework for checking real-time system with abstracted scheduler.

4.1 Framework for verification

In order to use the abstraction scheduler model to verify a system, we need a framework in order to describe system component and make everything work together. Following framework allow us to put any new scheduling algorithm into the system. Moreover, in the system model we can describe the detail functional view of tasks, along with some abstracted component like: task cost.

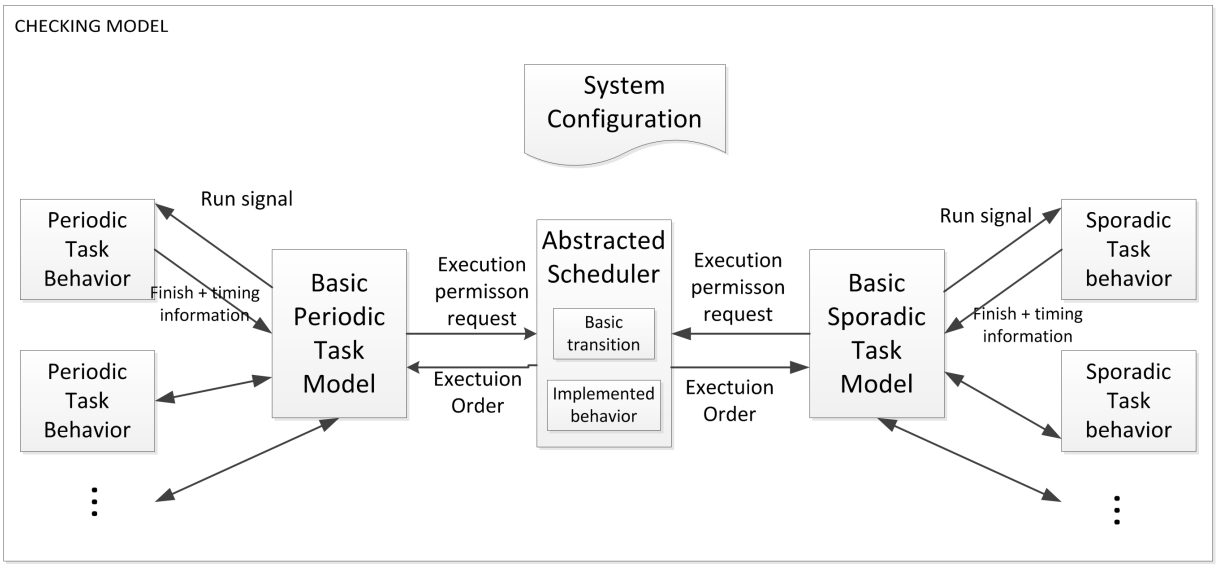


Figure 4.1: A system verification framework in which abstracted scheduler work

Our framework contains 4 main parts:

- System configuration: contains general configuration for the system. We can specify the number of periodic tasks, sporadic tasks, total tasks, and the scheduler ID. In addition, we will specify how many channels are needed in the verification model: channel for communicating between scheduler model and task model, channel for communicating between task behavior model. For system management, we also declare arrays of running thread (task) in the system in this component.
- Scheduler model: it contains 2 part: a basic model and scheduling algorithm implementation part.
  - Scheduler basic model: is depicted as figure 4.2. This model work as the basic state transition of scheduler. Initially the broadcast channel *GO*! is signaled to ensure all threads are in their correct state. The scheduler then executes for *wcet* time. If there are any schedulable threads, the appropriate thread is selected, by setting the corresponding index in the running array to 1. If no threads are schedulable all indices in running are set to 0. This is handled by the two methods *selectThread*() and *idle*() respectively.

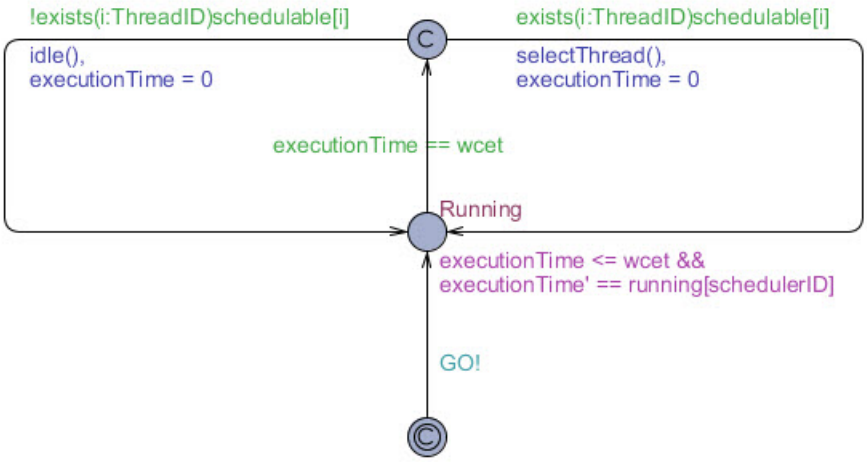


Figure 4.2: Basic model of scheduler

  - Scheduler algorithm implementation: is the implementation for *selectThread*() and *idle*() in scheduler basic model.
- Periodic task model: it also contains 2 component Basic model and Model for describing task behavior.
  - Basic model: it work like a basic state transition of periodic task, depicted as figure 4.3. This basic model must be supplied with parameters to determine its ID, period, deadline, and offset corresponding with task design.

* Initially the task waits if an offset is specified. This waiting process is signaled after receiving "GO" signal from scheduler. Each task in the system will have an clock variable named "releasedTime". This clock variable will measure the time of running task and help us to realize the preemption of scheduling algorithm. After passing specified offset, the periodic task will set its schedulable variable to be **true**, reset it's own clock **releasedTime = 0**, and call function runScheduler() in order to invoke scheduler model.
* Basic model will communicate with task behavior model through **run** channel. The run channel is signaled with the current ID of the thread, **run[pID]!**, it then waits for a response on the same channel, which indicates the task is done with its behavior. It is then ensured that it has completed before its deadline, otherwise it results in a deadlock, and the system is not schedulable. The scheduler model is invoked to determine which task to schedule next. The same procedure continues for each period of the periodic thread.

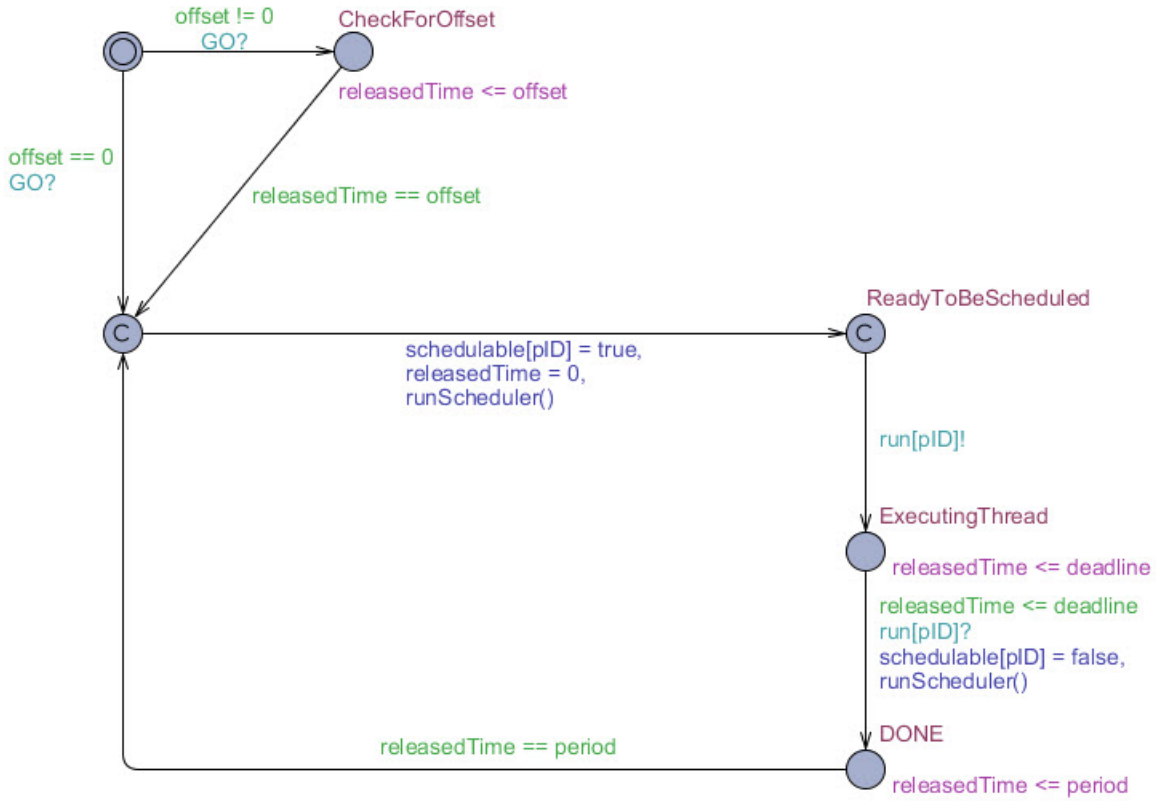


Figure 4.3: Basic model of periodic task

– Task behavior model: it work as the detail description of real-time task. Task cost: switching cost, computation code will be abstracted as a time value.

Each task will has one or a number of task behavior models. But, all behavior models of task must be have a common channel to communicate with basic model. In order to receiving the running signal, and return the computation time to task.

- Sporadic model: it similar to the periodic model. It contains 2 part: a basic model and behavior model. In basic model, it must be invoked by a signaling **fire** with the **sporadicID**. The basic model is depicted in figure 4.4. This model must be supplied with parameters for its ID, minimum inter-arrival time, and deadline. When the minimum inter-arrival time since the last release of this task has passed, the **firable** array is set to true for the specific task, and it is ready to be fired again. The signal **fire** is generated by a invocation from a periodic task's behavior in our framework.

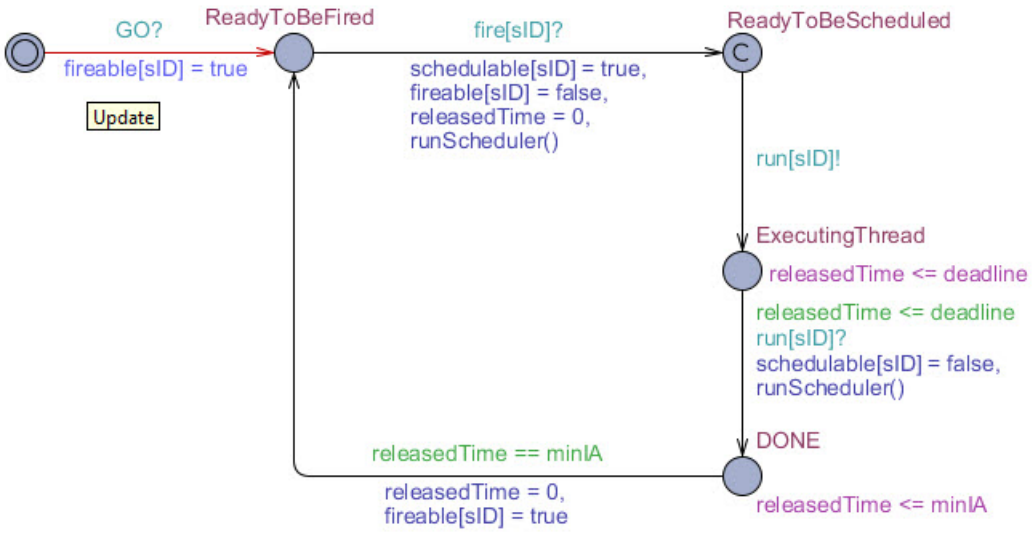
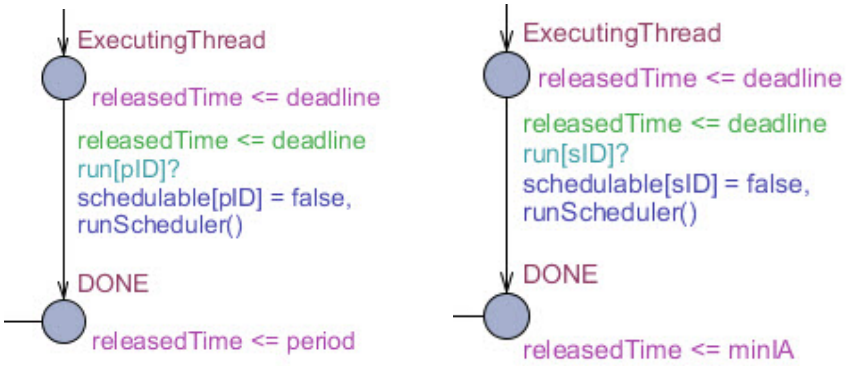


Figure 4.4: Basic model of sporadic task

About checking real-time properties by using our framework. Like we can see in 4.5, when a task model go from "ExecutingThread" state to "DONE" state, it must check guard condition **releasedTime <= deadline**. Therefore, if the execution time of a task behavior exceeds it's deadline, the task model can not perform this transition. Therefore, the system will go to deadlock condition. So, if we want to check deadline miss error, we can do it by checking the deadlock condition in the system. By using following query:

A[] not deadlock

Beside that, if the given task design has deadlock condition by itself, we can also use above query to verify the system.



(a) (b)

Figure 4.5: Embedded miss deadline error property

4.2 Preemption

In our approach scheduler can preempt a running task. To realize this behavior of scheduler, we will base on the stopwatches function of UPPAAL tool. About the definition and power of stopwatches automaton, we can reference to [20]. By using stopwatch, at different location of model, we can set different clock rate

- clock rate = 0: it mean that the current task is not running
- clock rate = 1: it mean that the current task is running

In order to realize the preemption, we put a clock variable into each model (scheduler model, periodic task model, sporadic task model). If we want to preempt current task i, we just need to add this update command into a transition of a model:

executionTime' == running[i]

where running is an array indicate the state of task: running (1) or preempted (0).

4.3 To invoke scheduler by system call

In real-time operating system, scheduler is just only invoked by a system call. In our framework, we only deal with scheduler and system tasks. There is no kernel implementation. In order to realize above fact, the scheduler will be invoked by a signal sent from a task. In the framework initialization period, all task will receive "**GO**" signal from scheduler model. After passing the initialization period, when the task is ready to perform it's work, task model will invoke scheduler by call a function **run_scheduler()** like in task model 4.3, 4.4.

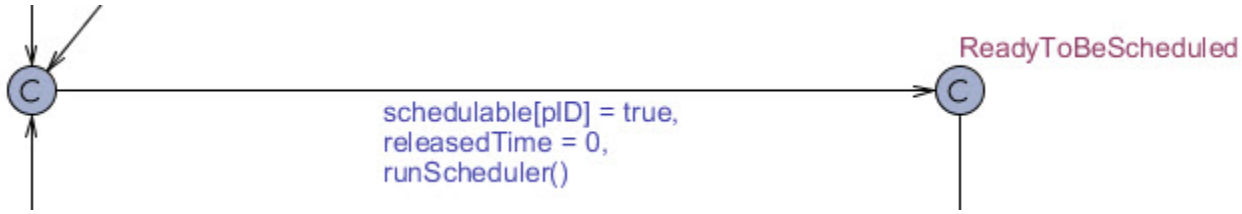


Figure 4.6: Scheduler invocation

4.4 To invoke sporadic task

The sporadic task is a system task respond for a random event. Like in previous section, our framework only deal with scheduler and task. Therefore, there is no input event into our framework. In order to fire a sporadic task, we provide a array which indicate a sporadic task is fireable or not, after the condition on sporadic task design. When we want to fire an sporadic task, we will send a signal through communication channel **fire**. In figure 4.7, when a sporadic task 4 is ready to be fired, we set like:

$$\boxed{\text{fireable}[4] = \text{true}, \text{fire}[4]!}$$

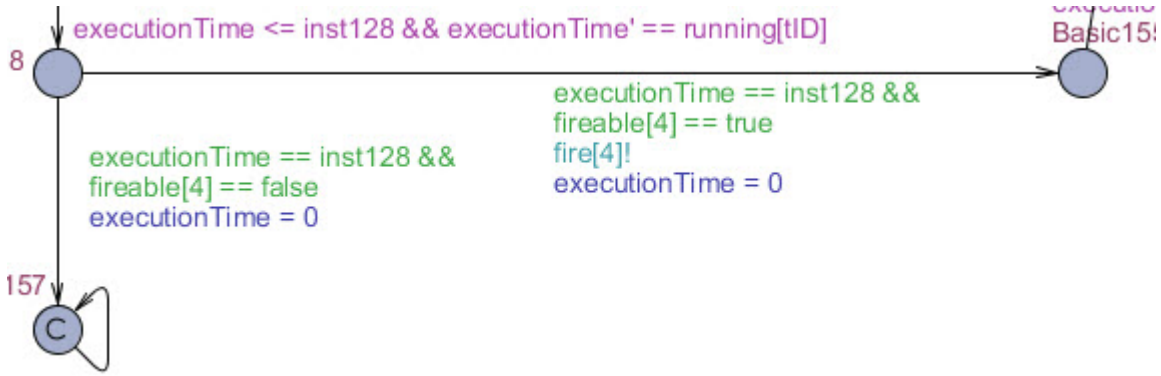


Figure 4.7: Sporadic task invocation

4.5 Using verification framework

Our framework has 4 main components. To use this framework, we can follow following step:

- Create task model:
  - Create the basic task model for both periodic and sporadic task if necessary. 4.3 4.4

  - Declare a communication channel between basic model and task behavior.
  - Create task behavior model for each task, based on task design of the system. In each behavior model, we must declare a clock variable, in order to realize preemption characteristic.

- Create scheduler model:
  - Create the basic scheduler model as depicted in 4.2.
  - Implement selectThread() function as to realize abstracted scheduler.

- In system configuration: declare necessary information for the configuration of system - tasks, scheduler, communication channel.

- In system data: declare necessary global data like: number of task, number of task types.

In next chapter, we will provide an example, in order to give more detail sample of above steps.

Chapter 5

Experiment

In this chapter, we will describe our experiment based on following sample problem:

- Real-time Sorting Machines: this is the sample real-time system designed by UPPAAL research group [15]

5.1 Real-time Sorting Machines Problem

5.1.1 Design

The overall idea of RTSM is to sort candy available in two different colors, white and blue. The candy is from now on referred to as objects. The sorting problems is kept simple, because the focus is to try different real-time aspects. The design of RTSM is based on some requirements.

The overall design idea of RTSM is depicted in Figure 5.1. The objects are placed in the *Feeder* which places one at a time on the conveyor belt with some space between. This ensures a constant flow of objects to be sorted, and prevents two objects from being right beside each other.

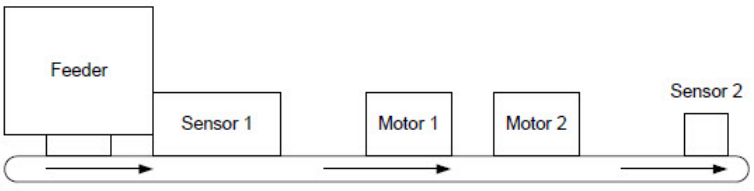


Figure 5.1: Design for the Real-Time Sorting Machines example

The conveyor moves the objects to the right, as the arrows indicate. When the object leaves the *Feeder* it passes by *Sensor 1*, which determines whether it is a white or a blue object. *Sensor 1* on the figure is actually two sensors placed on each side of the conveyor belt, and then encapsulated in a small cube. The cube is built to prevent external light

from interfering with the sensors. The use of two sensors has two purposes. Firstly, the sensors emit light to each other, keeping a constant high amount of light, easing the detection of objects breaking this light, and therefore spot when objects pass by the sensor. Secondly, the object might not always be in the middle of the conveyor belt, making the light it reflects either higher or lower than expected, this is compensated for, by using the average of the two sensor inputs.

Based on what color is detected, either *Motor 1* or *Motor 2* must be activated when the object is in front of the motor. Activation of a motor involves pushing the object into the correct bin, thereby sorting the objects.

Sensor 2 is not used as a part of the actual sorting, but is used during an initial *calibration* phase. This phase is used to determine the values read by *Sensor 1* dependent on the color of the object and the speed of the conveyor. *Sensor 2* is then used to measure the speed of the conveyor by measuring the time difference between when the object passes *Sensor 1* and *Sensor 2*. The speed of the conveyor belt is then used to calculate the following timing constraints:

- An object is added to the conveyor belt each 500ms.
- It takes 180ms to push an object off the conveyor belt.
- It takes 858ms to move from Sensor 1 to Motor 1.
- It takes 1326ms to move from Sensor 1 to Motor 2.

5.1.2 Scheduling analysis

Based on above system design, real-time sorting machine system contains 4 task: 2 periodic tasks and 2 sporadic tasks. Because 2 sporadic tasks are fired by a periodic task, in order to push an object off the conveyor on time. Therefore when these 2 sporadic tasks is ready to be fired, they should be fired as soon as possible. Based on this fact, we can guess that 2 sporadic task must have higher priority than periodic task. Depending on what scheduling policy is applied for this system design, the system is feasible or not.s

This system design is implemented by [15]. Based on the result of experiment we have the worst-case execution time for:

Name	WCET in clock cycles
Periodic Read Sensor	1330
PeriodicMotorSpooler	16022
SporadicPushMotor	9204

The traditional scheduling analysis approach always try to analyze the system in worst-case of interference. So, if we try to follow this approach, we will get following processor utilization for this given task design:

$$\frac{1330}{16022} + \frac{9024}{16022} + \frac{9024}{16022} = 1.209 > 1$$

So, if we follow the traditional approach, this given task design is unschedulable. However, in model-based analysis we can check the correct behavior of task design, and give more accurate result. For example, in this given task design, the periodic task fire 2 sporadic tasks in exclusive region. Therefore, there is no case of 2 sporadic tasks can be fired at the same time. It is one of strong evidence that following model-based analysis is appropriate for real-time systems.

In our approach, we will abstract the scheduling policy by using an abstracted scheduler model. By using the abstracted scheduler model, priority assignment for each task is abstracted. After finding out the correct scheduler abstraction, we will try to extract the set of appropriate priority assignment for this set of task, and also which scheduling policy is suitable for this given task design.

5.1.3 Experiment

In order to use our approach to perform scheduling analysis for this system design, we have to follow steps in 4.5.

First, we will create 2 basic model for periodic task and sporadic task. Based on the implementation of RTSM in [15], we can build task behavior model for 2 sporadic task, and 2 periodic task.

Now we need to define what is our intention abstracted scheduler for this given task design. Our intention will control 4 tasks of this given design are scheduled like in figure 5.2

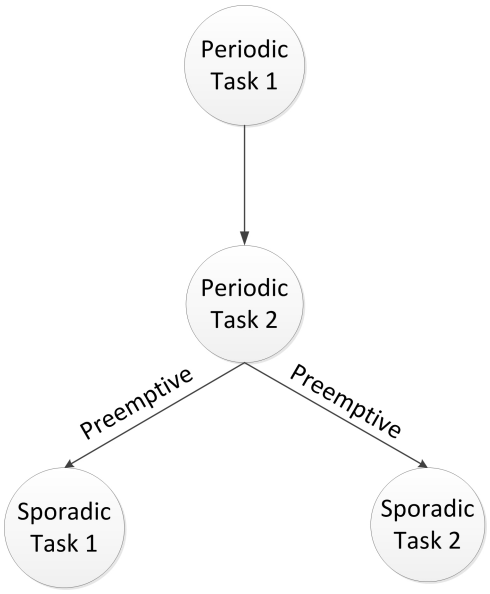


Figure 5.2: Abstracted Scheduler Model for RTSM problem

This abstracted scheduler can be declared by using our formal definition for abstracted scheduler model like following:

- Abstracted Scheduler for RTSM - S = {T, E, f}
  - T = { $t_1, t_2, t_3, ...$ }
  - E = $\{(t_1, t_2), (t_2, t_3), (t_2, t_4)\}$
  - f: (1,2) $\rightarrow$ 1 ; (2,3) $\rightarrow$ 0.5 ; (2,4) $\rightarrow$ 0.5

The representative version in computer understandable mean is:

Task	T_1	T_2	T_3	T_4
T_1	NULL	NULL	NULL	NULL
T_2	1	NULL	0.5	0.5
T_3	NULL	NULL	NULL	NULL
T_4	NULL	NULL	NULL	NULL

Table 5.1: ASM model for RTSM

By using translation algorithm from ASM model to UPPAAL code, we can get following implementation code for this abstracted scheduler in UPPAAL.

```
void selectThread(){
int i;
for(i:ThreadID){
if(schedulable[i]){
    if(i == 1)
        selectedThread = 1;
    else if(i == 2) {
        if(schedulable[1] == false)
            selectedThread = 2;
        else
            selectedThread = 1;
    }
    else if(i == 3)
        selectedThread = 3;
    else if(i == 4)
        selectedThread = 4;
        }   }
for (i = 0; i <= totalThreads; i++){
        running[i] = 0;
}
running[selectedThread] = 1;
}
```

Listing 5.1: Implementation code for ASM model of RTSM

Because UPPAAL does not support switch - case command, so in the implementation code we change it to if - else control command.

After we have a model for scheduler, basic model of task, task behavior model, we need to configure the system component into UPPAAL like:

Listing 5.2: Configuration code for RTSM

```
//declare information for periodic tasks
PeriodicThread1 = PeriodicThread(1, 120000, 120000, 0);
PeriodicThread2 = PeriodicThread(2, 240000, 240000, 0);
//declare information for sporadic      tasks
SporadicThread3 = SporadicThread(3, 240000, 3600);
SporadicThread4 = SporadicThread(4, 240000, 3600);
System
// Sporadic Threads
Template_0003_3, SporadicThread3,
Template_0003_4, SporadicThread4,
// Periodic Threads
Template_0006_1, PeriodicThread1,
Template_0017_2, PeriodicThread2,
// Scheduler
Scheduler;
```

Now we will check our system with following query:

A[] not deadlock

We will get the result of after verifying by using UPPAAL engine - figure 5.3. It means that RTSM system is feasible under our abstracted scheduler abstraction.

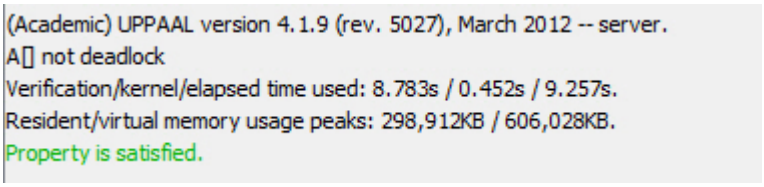


Figure 5.3: RTSM checking result

5.1.4 Appropriate scheduling policies

We have got the RTSM system is feasible under the ASM model5.1. In our approach, priority and scheduling policy is abstracted. Now, we will try to extract the priority assignment set and appropriate scheduling policy for RTSM system through our abstracted scheduler 5.1

Priority assignment

By using the algorithm for generating priority assignment from ASM model, algorithm 2. Will get following priority assignment for this given task design of RTSM system.

```
(2, 1, 2, 2)    (3, 1, 2, 3)    (4, 1, 3, 3)
(2, 1, 3, 4)    (3, 1, 3, 2)    (4, 1, 4, 3)
(2, 1, 4, 3)    (3, 1, 3, 3)    (4, 1, 3, 4)
(2, 1, 4, 4)    (4, 1, 2, 2)    (4, 1, 4, 4)
(2, 1, 3, 3)    (4, 1, 2, 3)    (4, 1, 2, 4)
(3, 1, 2, 2)    (4, 1, 3, 2)    (4, 1, 4, 4)
```

Listing 5.3: Priority assignment for RTSM system

In order to check those priority assignment is valid or not, we have created a slightly different task model and scheduler model in our framework. In the task model, we will add a guard condition when task move from "Start" state to "Ready for executed" state 5.4. The purpose of this guard condition is to check current task has a higher priority assignment than current executing thread or not. Based on that, it will invoke scheduler in order to select the highest priority task to be executed.

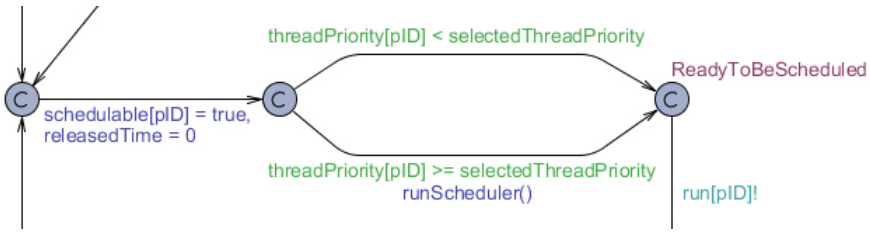


Figure 5.4: Modified basic task model

```
void selectThread(){
int i;
selectedThread = -1;
selectedThreadPriority = -1;
for(i:ThreadID){
if(schedulable[i] && threadPriority[i] >
                        selectedThreadPriority){
        selectedThread = i;
        selectedThreadPriority = threadPriority[i];
        }    }
for (i = 0; i <= totalThreads; i++){
        running[i] = 0;
}
```

```
        running[selectedThread] = 1;
        }
```

Listing 5.4: A scheduler implementation which select the highest priority task

After we assign priority for each system task like in 5.5, we can use UPPAAL engine to verify the validity of our priority assignment

```
int threadPriority[ThreadID] = {3, 1, 3, 3};
```

Listing 5.5: Example of assigning priority to task

Appropriate scheduling policy

In this section, we will discuss how we can examine that a scheduling policy is fix with our abstracted model or not. There are many scheduling policies for real-time system. As we can see in our previous part, we can classify scheduling policy for real-time system into 2 groups:

- Preemptive
- Non-preemptive

Preemptive Scheduling policies	**Non-preemptive scheduling policies**
Earliest Deadline First	Spring
Rate Monotonic	
Deadline monotonic	
Background service	
Earliest Deadline Late Server	
Improved Priority Exchange Server	
Polling server	
Defferable Server	
Priority Exchange	
Sporadic server	
Slack stealer	
Dynamic Priority Exchange Server	
Dynamic Sporadic Server	
Total Bandwidth Server	

Table 5.2: Scheduling policies for real-time system

In this abstracted scheduler model, we have 2 preemptive transitions in the model. Therefore, only preemptive category of scheduling policies is appropriate with our intention. Following we will check which policy is fix with our abstraction model.

- **Earliest Deadline First - EDF**:The principle of this scheduling policy is: the task with the (current) closes deadline is assigned the highest priority in the system. In our sample task design, we have for task with following deadline respectively:
  - Periodic task 1: 120000 μs
  - Periodic task 2: 240000 μs
  - Sporadic task 1: 3600 μs
  - Sporadic task 2: 3600 μs

  Based on above deadline of the tasks, we can have following priority assignment for this scheduling policy:

  $$(2,\ 1,\ 3,\ 4)$$
  $$(2,\ 1,\ 3,\ 3)$$
  $$(2,\ 1,\ 4,\ 3)$$
  $$(2,\ 1,\ 4,\ 4)$$

  Above priority assignment is subset of our feasible priority assignment set. Therefore, this scheduling policy is fix with our abstracted scheduler model.

- **Rate Monotonic - RM** The principle of this scheduling policy is: the static priorities are assigned on the basis of the cycle duration of the task: the shorter the cycle duration is, the higher is the task's priority. In order to apply this scheduling policy, all the task in system must be periodic. However, in our sample design, we have 2 tasks are sporadic with unpreditable inter-interval time. Therefore this scheduling policy is not fix with our abstracted scheduler model.

- **Deadline-monotonic - DM** The principle of this scheduling policy is: each task is assigned a priority inversely proportional to its deadline. Based on the task's deadline, we have following priority assignment for our sample system's tasks:

  $$(2,\ 1,\ 3,\ 4)$$
  $$(2,\ 1,\ 3,\ 3)$$
  $$(2,\ 1,\ 4,\ 3)$$
  $$(2,\ 1,\ 4,\ 4)$$

  Like in EDF policy, above priority assignment is subset of our feasible priority assignment set. Therefore, this scheduling policy is fix with our abstracted scheduler model.

- **Background service** The principle of this policy is: try to schedule sporadic task in background. It means when there is no periodic task instance is ready to be executed, the ready sporadic task can execute. In our tasks design, when periodic

task 2 is executing, it will fire one of 2 sporadic task and wait until sporadic task finish. By examining the execution time of periodic task 2 after firing sporadic task, this periodic task 2 invoke 2 methods, with the executing time is larger than 3600 μs. Moreover, the periodic task 1 always try to detect object on conveyor. So there is no case such that no periodic task is ready for executed.

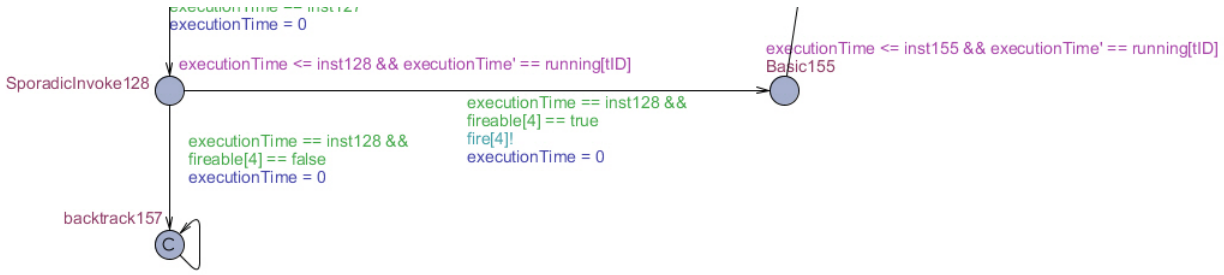


Figure 5.5: Task transition after a sporadic task invocation

Therefore, this scheduling policy is not fix with our abstracted model.

- **Earliest Deadline Late Server - EDLS** The principle of this policy is: to use the idle times of an EDL schedule to execute sporadic requests as soon as possible. When there are no sporadic activities in the system, periodic tasks are scheduled according to the EDF algorithm. Whenever a new sporadic request enters the system (and no previous sporadic is still active) the idle times of an EDL scheduler applied to the current periodic task set are computed and then used to schedule the sporadic request pending.

  In our task design

  - the allocated time for one slot is 24s
  - two periodic tasks take totally 16,1s in WCET
  - one sporadic tasks take 6,4 s in WCET

  So, based on the task information, scheduler can compute that there are 7,9s idle time, it is enough to serve the sporadic task. We can conclude this information because when we do the scheduling analysis with model-checking approach - like using our framework - we can recognize that two sporadic tasks are never activated at the same time.

  However, in our current approach the information guarantees the feasible or not of a scheduling policy is priority assignment.Therefore, under the guidance of our abstracted scheduler model, we can not determine that this scheduling policy is fix with our abstracted model or not.

- **Polling server - PS** The principle of this scheduling policy is: creating a periodic task whose purpose is to serve sporadic task request as soon as possible. Assign the server task with highest priority, capacity equal to one sporadic tasks.

  If we apply this scheduling policy, our system will has another priority task who represent two sporadic tasks. We will have following priority assignment for the system's task:

  $$\{2,\ 1,\ 3,\ 4\}$$
  $$\{2,\ 1,\ 4,\ 3\}$$
  $$\{2,\ 1,\ 3,\ 3\}$$
  $$\{2,\ 1,\ 4,\ 4\}$$

  Above priority assignment is a subset of our feasible priority assignment set. Therefore this scheduling policy is fix with our abstracted scheduler model.

- **Deferrable Server - DS** The principle of this scheduling policy is: creating a periodic task whose purpose is to serve sporadic task request as soon as possible. The capacity of server is remain even if there is no requests are pending upon the invocation of the server. Assign the server task with highest priority, capacity equal to one sporadic tasks.

  In this policy, we also have a periodic task who serve sporadic tasks. However, because of its priority is highest and capacity always remain even if there is no sporadic task. So in our given task design, 2 periodic tasks do not have time for execution. It will lead to the deadline miss error for those tasks.

  Therefore, this scheduling policy is not fix with our abstracted scheduler.

- **Priority Exchange - PE** The principle of this scheduling policy is: like DS, however it differs from DS in the manner in which the capacity is preserved. PE preserves its high-priority capacity by exchanging it for the execution time of a lower-priority periodic task.

  In our experiment, we assume that the task and scheduler switching cost is negligible. We assign the value of these switching cost to zero. So, if we apply this scheduling policy into our system, we will have following priority assignment:

  $$\{2,\ 2,\ 3,\ 4\}$$
  $$\{2,\ 2,\ 4,\ 3\}$$
  $$\{2,\ 1,\ 4,\ 4\}$$
  $$\{2,\ 1,\ 3,\ 3\}$$

  Above priority assignments is not a subset of our feasible priority assignment set. However, there is two priority assignment which belong to our generated assignment set. Therefore this scheduling policy is fix with our abstracted scheduler model.

- **Sporadic Server - SS** The principle of this scheduling policy is: like DS and PE, however SS replenishes its capacity only after it has been consumed by sporadic task execution. The replenishes time is decided as:
  - The replenishment time - RT - is set as soon as SS becomes active and $C_s > 0$. Let T_a be such a time. The value of RT is set equal to T_a plus the server period ($RT = T_a + T_s$)
  - The value of RA (replenishes amount) is set equal to the capacity consumed within the interval $[T_a, T_i]$

  In this scheduling policy, it basically base on DS and PE, so we will have following priority assignment:

  $$\{2,\ 1,\ 3,\ 4\}$$
  $$\{2,\ 1,\ 3,\ 4\}$$
  $$\{2,\ 1,\ 4,\ 4\}$$
  $$\{2,\ 1,\ 3,\ 3\}$$

  However when a sporadic task is activated, server's amount will be replenished. Although the priority assignment is okay. But based on task design, periodic task 1 can not meet its deadline, if sporadic server has highest priority after serving a sporadic task.

  Therefore, this scheduling policy is not fix with our abstracted scheduler model.

- **Improved Priority Exchange Server - IPES** The principle of this scheduling policy is: The idle times of the EDL algorithm can be precomputed off-line and the server can use them to schedule sporadic request, when there are any, or to advance the execution of periodic tasks. In the latter case, the idle time advanced can be saved as sporadic capacity at the priority levels of the periodic tasks executed. As the explanation in Earliest Late Deadline Server, we can not determine this scheduling policy is fix with our abstracted scheduler model or not.

  The only problem of applying this policy is the difficulty of policy implementation in real-time system. The cost of computation idle time of all task in the system, is also the problem need to be considered. However, in our experiment, we assume that such cost is negligible.

- **Slack stealer - SStealer** The principle of this scheduling policy is: when an sporadic request arrives, the Slack stealer steal all the available slack from periodic tasks and uses it to execute sporadic requests as soon as possible. If no sporadic requests are pending, periodic tasks are normally scheduled by RM.

  By applying this scheduling policy, the priority assignment for this system's task will be:

$$
\begin{array}{l}
(2,\ 1,\ 3,\ 4) \\
(2,\ 3,\ 1,\ 4) \\
(2,\ 4,\ 1,\ 3) \\
(2,\ 3,\ 4,\ 1)
\end{array}
$$

Above priority assignment is not a subset of our feasible priority assignment set. However, there is one priority assignment which belong to our generated priority assignment set. So we can determine that this policy is feasible in our system.

- **Dynamic Priority Exchange Server - DPES** The principle of this scheduling policy is: adapted to work with a deadline-based scheduling algorithm. The server will has a lowest priority, and it can be later reclaimed when sporadic requests enter the system.

  This scheduling policy is fix with our abstracted scheduler model. In sense of EDF scheduling algorithm, our sample system just has 4 tasks with 2 periodic tasks and 2 sporadic tasks. Periodic task 2 will fire one of 2 sporadic tasks when it is executing. So the priority assignment will the same as in EDF policy.

- **Dynamic Sporadic Server - DSS** The principle of this scheduling policy is: we can views as the combination of Dynamic Priority Exchange Server with Sporadic Server.

  If we apply this scheduling policy, because it is the combination of DPES and SS, the priority assignment will like in SS. Therefore, this scheduling policy is fix with our abstracted scheduler model.

- **Total Bandwidth Server - TBS** The principle of this scheduling policy is: in order to improve Sporadic Server in the case of the deadline of sporadic task is late, it will assign a possible earlier deadline to each sporadic request.

  Like the explanation in EDLS, any assignment of sporadic task to its earlier deadline will make our sample system feasible. Therefore, this scheduling policy is fix with our abstracted scheduler model.

5.1.5 Evaluation

Scheduling policy	Fix with ASM model or not
Spring	NG
Earliest Deadline First	OK
Rate Monotonic	NG
Deadline monotonic	OK
Background service	NG
Earliest Deadline Late Server	N/A
Improved Priority Exchange Server	N/A
Polling server	OK
Deferrable Server	NG
Priority Exchange	OK
Sporadic server	NG
Slack stealer	OK
Dynamic Priority Exchange Server	OK
Dynamic Sporadic Server	OK
Total Bandwidth Server	OK

Table 5.3: Summary of appropriate scheduling policy for RTSM design

Through this experiment, we can understand clearly that we can find out the appropriate scheduling policy based on the guidance of abstracted scheduler model. Without using the abstracted scheduler model, in order to check above scheduling policies precisely one by one for a given system design, we have to implement all of it in model checking tool. As we explain in 1.3, this work is almost impossible and costly.

In addition, through this experiment with RTSM example, we have demonstrated how we can apply our approach into scheduling analysis with a given task design. Through this experiment we can get an idea of how to use an abstracted scheduler model to do scheduling analysis. We known how to extract priority assignment from abstracted scheduler model, and how to do experiment of getting appropriate scheduling policies.

Chapter 6

Conclusion and Future work

6.1 Conclusion

The correctness of hard real-time systems is crucial, because failures can cause disasters. Many researches have been done to improve different aspects of the development process, ranging from model checking tools, to real-time specific languages and profiles, to WCET analysis and scheduling. In many researches, they are based on WCET analysis. This is important factor of real-time systems, in order to ensure the schedulability of the system. However, if we only based on WCET, we will miss the behavior of the task and lack of information about scheduler.

In this research, we deal with the problem of doing scheduling analysis for real-time system in different settings. We propose the abstracted scheduler model in order to do above problem at once. It means that we can abstract different settings of real-time system into only one setting. By follow this method, we can overcome the limitation of implementation of difficult settings for real-time system in model-checking tools. Beside that, by using our framework, we can deal directly with the functional view of the task. By going to detail of task behavior we can solve the problem of assuming worst-case condition in other scheduling analysis researches. Moreover, our framework is reusable with task model, framework configuration. For a new system, we need to modify the abstracted scheduler implementation and the corresponding task behavior 4.5

In addition, we also provide algorithm in order to extract the correct settings for a given task design for a real-time system. We can generate the appropriate priority assignment for the task after having a feasible abstracted scheduler. For a new task design, we can base on some criterion of choosing appropriate scheduling policy like in our evaluation part.

In conclusion, throughout this research, we have addressed some important aspects in real-time model checking. It is task behavior, scheduler, cost, priority and timing problem. We have proposed a new approach in real-time model checking.

6.2 Related Works

In this section, we will give a small discussion about other related works in scheduling analysis that follow model checking approach. Throughout this discussion we can realize what is our advantage compare with other researches.

There are many researches in scheduling analysis using model-checking technique. However, by using UPPAAL as the model-checking engine we have 3 outstanding researches:

- Model-based framework for Schedulability Analysis Using Uppaal 4.1 [7]
- TIMES tool [19]
- Real-time Sorting Machine design group [15]

6.2.1 Model-based framework for Schedulability Analysis Using Uppaal 4.1[7]

This is a research of UPPAAL research group. This work takes the advantage of model-checking method and overcomes the limitation of simulation approach for multi-core platform. By using UPPAAL engine, they provide a framework to do scheduling analysis for real-time system in various architectures. Their framework contains:

- A rich collection of attributes for tasks, including: off-set, best and worst case execution times, minimum and maximum interarrival time, deadlines, and task priorities.
- Facility to represent task dependencies.
- Assignment of resources, for example processors or busses, to tasks.
- Scheduling policies including First-In First-Out (FIFO), Earliest Dead-line First (EDF), and Fixed Priority Scheduling (FPS).
- Possible preemption of resources.

This work focus on the schedulability problems that: A system of tasks with constraints and resources with scheduling policies is said to be schedulable if no execution satisfying the constraints of the system violates a deadline.

By using this framework we can create a suitable scheduling analysis system so that fix with the desired problem, with:

- Generic task: a task model with basic task transition state. We can specify how many tasks exist in target system with their information.
- Number of resources in target system.
- Scheduling polices that used in target system.

This is a very useful framework to do scheduling analysis for real-time systems. However, in the case of finding out which set of conditions for a given task design such that the system is feasible, by using this framework we must:

- Implement all possible scheduling policy: this work is almost impossible due to the limitation of UPPAAL engine. It does not allow access to another model's state from another model.
- Checking all the combination of the variation of target system design like: priority of task, scheduling policies, task's cost, assignment of task to resource, and so on. By doing this work, it is very tedious. Moreover if the number of system information is large, it will take a lot of time for performing model-checking the system and analyze the result.

6.2.2 TIMES Tool

Like in the discussion of how we choose the appropriate model-checking tool for our research, TIMES is useful tool for performing scheduling analysis and scheduling simulation of systems that can be described as a set of tasks which are triggered periodically or sporadically by time or external events. In order to use TIMES tools to analyze schedulability of a given design, we must:

- Specify how many periodic, sporadic tasks are in the system.
- Specify each task's properties like: activation time, deadline, offset, behavior, priority, best case execution time, worst case execution time.
- Specify what is the scheduler policy in 4 predefined scheduler policies

Like in the previous work, in TIMES tool the number of scheduling policy is limited. Performing scheduling analysis in TIMES does not provide the set of conditions in which a given task design are feasible or not.

6.2.3 RTSM group

The RSTM project not only designed the system but also implemented it in different mean. There are already implemented the system in Java, LRBJOP physical board and develop a version of model-based scheduling analysis.

In model-based version of RTSM project, they have create a way to analyze the schedulability of they system:

- Task: has many properties like: priority, deadline, offset, activation time, best case execution time, worst case execution time.
- Scheduler: they just only implemented the deadline monotonic policies.

In this work, we can only analyze the schedulability of a specific setting of the system. For example the setting of this project is

- Task priority assignment is (2, 1, 3, 4)
- Number of task and task behavior: there are 4 tasks, with 13 task behaviors are modeled in UPPAAL.
- Scheduling policy: deadline monotonic policies.

Therefore, this work can not give us any information about the set of conditions (priority assignment, scheduling policy) in which the RTSM design is feasible.

6.3 Future work

In this section, we will discuss what should be done in order to improve our work.

6.3.1 Provide criterion of choosing appropriate scheduler

Currently, after having a feasible abstracted scheduler, we will base on the experiment of each scheduling policy to choose the appropriate one. In order to improve this process, we need a mechanism that support extracting scheduling policy criterion from abstracted scheduler model. We can base on the transition type in abstracted scheduler model, to decide the criterion for appropriate scheduling policy.

6.3.2 Prove the correctness of abstracted scheduler model

In our work, we have proposed a abstracted scheduler model. As a abstraction, we need to prove the soundness of our model. It means that we need to prove properties that are satisfied in abstracted model are also satisfied in concrete models.

6.3.3 Realistic situation of real-time systems

In order to make our work can handle the realistic situation of real-time systems, we must consider following issues:

Environment interrupt

Currently, in our framework, a sporadic task is only invoked by another task. Therefore, it is more useful if we have an mechanism to enable the interrupt from user or outside event. It can be implemented as a spooling periodic task. It will periodically check interrupt from user, in order to fire corresponding sporadic task. However, it should be possible to model hardware interrupts as sporadic tasks, provided a minimum inter-arrival time is present and obeyed.

Cost of scheduler and task

Now, in our framework, we assume that the task and scheduler switching cost is zero. If we can deal with these switching cost into the framework, the scheduling analysis will be more accurate. As a idea for this improvement, we can add a dummy state into basic scheduler and basic task model. In this dummy state, we will add a execution time represent the switching cost of scheduler. However, we need to consider carefully about the behavior of scheduler after adding this dummy state.

References

[1] Model Checking Multi-task Software on Real-time Operating Systems, *Toshiaki Aoki*, Japan Advanced Institute of Science and Technology, Japan, 2008

[2] Model-Checking for Real-Time Systems, *Kim G. Larsen, Paul Petterson, Wang Yi*, BRICS**, Aalborg University, DENMARK, Uppsala University, SWEDEN

[3] Hard Real-Time Computing Systems - Predictable Scheduling Algorithm and Applications, *Giorgio C. Buttazzo*, Kluwer Academic Publishers, 1997

[4] Realtss: a real-time scheduling simulator, *Arnoldo Diaz, Ruben Batista and Oskardie Castro*, Department of Computer Systems, Instituto Technolgico de Mexiali, Mexico, 2007

[5] T. A. Henzinger, Z. Nicollin, J. Sifakis, and S. Yovine, Symbolic model checking for real-time systems. *In Logic in Computer Science*, 1992

[6] Model-Based Schedulability Analysis of Safety Critical Hard Real-Time Java Programs, *Thomas Bogholm, Henrik Kragh-Hansen, Petur Olsen, Bent Thomsen, Kim G.Larsen*, Aalborg University, 2008

[7] Model-based Framework for Schedulability Analysis Using UPPAAL 4.1, *Alexandre David, Jacob Illum, Kim G. Larsen, Arne Skou*, Aalbord University, 2009

[8] An Analysis of Research on Block Scheduling, *Sally J. Zepeda, R. Stewart Mayers*, University of Geogia, Southeastern Oklahoma State University

[9] An Abstract Model for Scheduling Real-Time Programs, *Alvaro E. Arenas*, Laboratorio de Computo Especializado, Universidad Autonoma de Bucaramanga, Calle 48 No 39 -234, Bucaramanga, Colombia

[10] Counterexample-based abstraction refinement, *Edmund Clarke , Orna Grumberg , Somesh Jha , Yuan Lu , Helmut Veith* , 2000

[11] Schedulability Analysis Using Two Clocks, *ELena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi*, Uppsala University

[12] Decidable and Undecidable Problems in Schedulability Analysis Using Timed Automata, *Pavel Krcl and Wang Yi*, Uppsala University

[13] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. *Times: a tool for schedulability analysis and code generation of real-time systems.* citeseer.ist.psu.edu/amnell03times.html, 2003.

[14] Gerd Behrmann, Alexandre David, and Kim G. Larsen. *A Tutorial on Uppaal.* http://www.it.uu.se/research/group/darts/papers/ texts/new-tutorial.pdf, 2004. Online: 6/12-2007.

[15] Real-time sorting machines project page `http://iprojectideas.blogspot.jp/2011/11/real-time-sorting-machine.html`

[16] Uppsala University and Aalborg University. UPPAAL. `http://www.uppaal.com`

[17] Spin Model Checker `http://spinroot.com/spin/Man/promela.html#section6`

[18] General information about spin and real-time system `http://en.wikipedia.org`

[19] Times tool official website `http://www.timestool.com/`

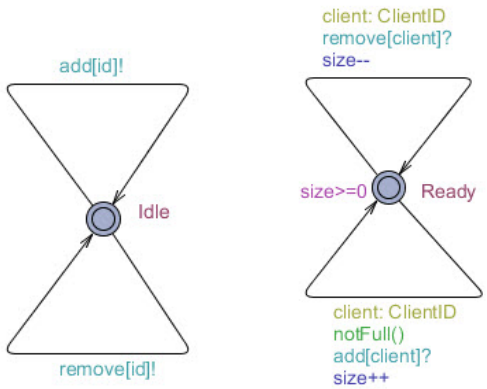
[20] The impressive Power of Stopwatches, *Franck Cassez, Kim Larsen*, IRCCyN/CNRS UMR 6597, France, Dep. of Computer Science, Aalborg University, Denmark

Chapter 7

Appendix

7.1 UPPAAL

Uppaal is a verification tool developed in co-operation between Uppsala University and Aalborg University [16]. This appendix provides an introduction to parts of the syntax used in Uppaal. For a more throughout description see [14]. This is only the most basic of Uppaal in order to understand the models presented in this thesis. A small example is used to describe some of the terminology used in Uppaal, when creating a model. The example illustrates a simple buffer, which allows add and remove instructions, and a couple of clients requesting to add or remove from the buffer. The example is depicted in Figure 7.1, containing both the client and buffer model.



(a) (b)

Figure 7.1: (a) A simple client communicating with (b) a simple buffer.

In general a Uppaal model contains a set of finite state machines (FSMs), called templates. The buffer and the client are templates. These templates represent parts of the system, and consist of two basic elements, states and transitions. A state is represented by a circle, and a transition is a directed edge between two states. Each state can have a unique name attached, e.g. the buffer has a state called *Ready* and the client a state

called *Idle*. The name must only be unique for the given template. Each state can have an *invariant*, a Boolean expression which must always evaluate to true when the FSM is in this state. For example $size >= 0$, while the buffer is in the *Ready* state, this ensures that the buffer can not be dequeued when it is empty. Transitions connect states, and they can loop to the same state. A transition can have four attributes:

- **Select**: Non-deterministically selects a value from a type and assigns it to a variable. The syntax is: $< VariableName >:< Type >$, e.g. the buffer uses *client: ClientID* to represent the client requesting an add or remove. *ClientID* is defined as an integer with a specific range representing the number of clients.

- **Guard**: A Boolean expression which must evaluate to true for the FSM to be able to follow the transition, e.g. *notFull()* prevents the clients from adding more elements if the buffer is full. Here *notFull()* is a custom developed method returning *true* if the buffer is not full, and *false* otherwise. This could also be guaranteed using an invariant stating $size <= MaxSize$. Note even though the guard is fulfilled, the transaction cannot be performed if it violates the invariant of the target node.

- **Sync**: Sends or receives on a channel, which is shared among the machines. An "!"represents send/signal and a "?"represents receive/ listen, e.g. $remove[client]?$ receives a remove request from a specific client.

- **Update**: A comma separated list of variable assignments, using normal assignment operators, e.g. $size++$ increments the size of the buffer.

To ease the readability of the model, the attributes are color-coded by type and attributes should be placed near the corresponding state or transition.

The states in the template can be normal, urgent, or committed. A simple example depicted in Figure 7.2 is used to describe the different states. Three different templates are shown, called P0, P1, and P2 representing three isolated processes, each containing a local clock x. Time is represented by clocks in Uppaal, which are constantly increasing and represented by natural numbers. A template must contain exactly one initial state, which is the starting point, represented by a small circle inside the state. Each of the processes has the initial state set to S0, the clock x is then reset on the transition to state S1, which is normal in P0, urgent in P1, and committed in P2. An urgent state is represented by a small U and committed by a small C inside the state. The following describes how the difference in the S1 state affects the different processes.

- **Normal**: Time can elapse in this state, and the value of the clock x in state S2 cannot be ensured.

- **Urgent**: The time is frozen during an urgent state. The value of the clock x is therefore ensured to also be 0 when entering S2. An urgent state is semantically equivalent to resetting a designated clock, y, on all incoming edges and add the invariant y¡=0 to the state, ensuring it to leave before time elapses.

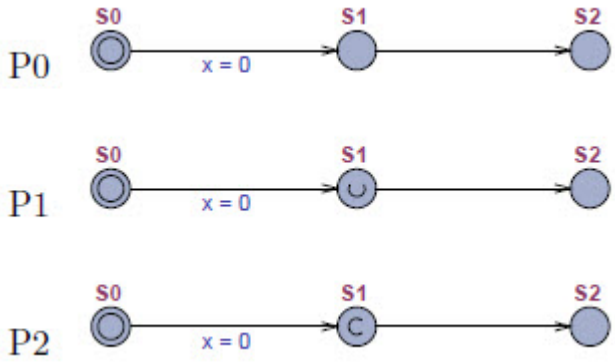


Figure 7.2: Three different automata with a local clock

- **Committed** Similar to urgent states, time is frozen. Furthermore, if a template is in a committed state, the next transaction must leave this state, i.e. when P2 is in S1 the only possible transition is to S2, independent of other states in the system. Note, if P1 is in S1, then P0 is able to take a transition, but if P2 is in S1, then P0 cannot take a transition.

At first glance the difference between urgent and committed might not seem that obvious, but the strength of committed is the ability to create an atomic sequence. When different templates need to exchange values, this must be done through public variables, e.g. the buffer from the first example might be extended to return the dequeued value to the client. If several clients can communicate with several buffers, and the public variable is shared between the buffers, can result in a race condition, the committed state ensure that this is not possible. This concludes the description of Uppaal.