FORMAL VERIFICATION OF ADAPTIVE REAL-TIME SYSTEMS BY EXTENDING TASK AUTOMATA

Leo Hatvani

2014
Abstract

Recently, we have seen an increase in the deployment of safety critical embedded systems in rapidly changing environments, as well as need for on-site customizations and rapid adaptation. To address the extended range of requirements, adaptation mechanisms are added to the systems to handle large number of situations appropriately. Although necessary, adaptations can cause inconsistent and unstable configurations that must be prevented for the embedded system to remain dependable and safe. Therefore, verifying the behavior of adaptive embedded systems during the design phase of the production process is highly desirable.

A hard-real time embedded system and its environment can be modeled using timed automata. Such a model can describe the system at various levels of abstraction. In this thesis, we model the adaptive responses of the system in terms of tasks that are executed to handle changes in the environmental or internal parameters.

Schedulability, a property that all tasks complete execution within their respective deadlines, is a key element in designing hard real-time embedded systems. A system that is unschedulable immediately compromises safety and hard real-time requirements and can cause fatal failure. Given specifications of all tasks in the system, we can model the system, an abstraction of the environment, and adaptive strategies to investigate whether the system retains safety properties, including schedulability, regardless of the changes in the environment and adaptations to those changes.
Abstract

Recently, we have seen an increase in the deployment of safety critical embedded systems in rapidly changing environments, as well as need for on-site customizations and rapid adaptation. To address the extended range of requirements, adaptation mechanism are added to the systems to handle large number of situations appropriately. Although necessary, adaptations can cause inconsistent and unstable configurations that must be prevented for the embedded system to remain dependable and safe. Therefore, verifying the behavior of adaptive embedded systems during the design phase of the production process is highly desirable.

A hard-real time embedded system and its environment can be modeled using timed automata. Such model can describe the system at various levels of abstraction. In this thesis, we model the adaptive responses of a system in terms of tasks that are executed to handle changes in the environmental or internal parameters.

Schedulability, a property that all tasks complete execution within their respective deadlines, is a key element in designing hard real-time embedded systems. A system that is unschedulable immediately compromises safety and hard real-time requirements and can cause fatal failure. Given specifications of all tasks in the system, we can model the system, an abstraction of the environment, and adaptive strategies to investigate whether the system retains safety properties, including schedulability, regardless of the changes in the environment and adaptations to those changes.
To my parents.
To my parents.
Acknowledgments

First of all, I would like to thank my supervisors, Paul Pettersson and Cristina Seceleanu, without whom this thesis would not be possible. Their encouragement through the tough times and continuous guidance were essential to its success.

My parents, Gabor and Ljiljana, have provided me with vital support over the past few years. Although we were separated by more than 1700km, the regular videochat sessions were something that I could always count on. Their words of reassurance and new perspectives were there whenever I needed them.

Finally, I would like to thank all my friends and colleagues, for the infinite supply of new insights, extraordinary conversations, and great times.

Leo Hatvani
Västerås, November, 2014

See Figure 1 for a sampling of the individual names.
Acknowledgments

First of all, I would like to thank my supervisors, Paul Pettersson and Cristina Seceleanu, without whom this thesis would not be possible. Their encouragement through the tough times and continuous guidance were essential to its success.

My parents, Gabor and Ljiljana, have provided me with vital support over the past few years. Although we were separated by more than 1700km, the regular videochat sessions were something that I could always count on. Their words of reassurance and new perspectives were there whenever I needed them.

Finally, I would like to thank all\(^1\) my friends and colleagues, for the infinite supply of new insights, extraordinary conversations, and great times.

Leo Hatvani
Västerås, November, 2014

---

\(^1\)See Figure 1 for a sampling of the individual names.


The included articles have been reformatted to comply with the licentiate layout.


Figure 1
List of Publications

Papers Included in the Licentiate Thesis


---

2The included articles have been reformatted to comply with the licentiate layout.
1 Introduction
1.1 Background
1.1.1 Adaptive Real-Time Embedded Systems
1.1.2 Model-checking Real-time Systems
1.1.3 Schedulability Analysis
1.2 Thesis Outline
2 Research Problems
2.1 Problem Description
2.2 Research Goals
3 Research Results
3.1 Adaptive Task Automata
3.2 Model-checking ATA with Static-Priority Scheduling
3.3 Model-checking ATA with Dynamic-Priority Scheduling
3.4 Contribution of Included Papers
3.4.1 Paper A
3.4.2 Paper B
3.4.3 Paper C
3.4.4 Paper D
4 Research Method
5 Related Work
5.1 Modeling and Verification of High-level Abstractions of Adaptive Embedded Systems
5.2 Analytic Approaches
Contents

I Thesis

1 Introduction
   1.1 Background ........................................ 5
       1.1.1 Adaptive Real-Time Embedded Systems .... 5
       1.1.2 Model-checking Real-time Systems ........ 6
       1.1.3 Schedulability Analysis ................... 7
   1.2 Thesis Outline ................................... 10

2 Research Problems
   2.1 Problem Description .......................... 11
   2.2 Research Goals ................................. 11

3 Research Results
   3.1 Adaptive Task Automata ...................... 13
   3.2 Model-checking ATA with Static-Priority Scheduling .... 14
   3.3 Model-checking ATA with Dynamic-Priority Scheduling ... 15
   3.4 Contribution of Included Papers ............. 16
       3.4.1 Paper A ...................................... 17
       3.4.2 Paper B ...................................... 17
       3.4.3 Paper C ...................................... 18
       3.4.4 Paper D ...................................... 18

4 Research Method .................................. 19

5 Related Work ...................................... 23
   5.1 Modeling and Verification of High-level Abstractions of Adaptive Embedded Systems ........... 23
   5.2 Analytic Approaches ............................ 24
5.3 Related Verification Approaches ........................................... 25

6 Conclusions .......................................................... 27
   6.1 Summary and Conclusions ........................................... 27
   6.2 Future Work ....................................................... 28

Bibliography .................................................................. 29

II Included Papers ...................................................... 35

7 Paper A: 
   7.1 Introduction ............................................................ 39
   7.2 Preliminaries: Task Automata ..................................... 41
   7.3 Adaptive Task Automata ............................................ 43
   7.4 Encoding of the Adaptive Task Automata ....................... 45
      7.4.1 Encoding the Predicate sched() ......................... 47
      7.4.2 Encoding the Fixed Priority Scheduler ................. 48
      7.4.3 Variable Bounding ............................................... 51
   7.5 Examples .................................................................. 52
      7.5.1 Admission Control - A Synthetic Example .......... 52
      7.5.2 Smartphone Task Management Example ............... 53
   7.6 Conclusion ............................................................. 54
Bibliography .................................................................. 57

8 Paper B: 
Modeling and Analysis of Adaptive Embedded Systems using Adaptive Task Automata ................................................. 59
   8.1 Introduction ............................................................ 61
   8.2 Overview of Task Automata ........................................ 61
   8.3 Adaptive Task Automata ............................................ 63
   8.4 Example .................................................................. 65
      8.4.1 Robot Teleoperation ............................................. 65
      8.4.2 Scalability of the Approach ................................. 68
   8.5 Related Work ........................................................... 68
   8.6 Ongoing and Future Work ......................................... 69
Bibliography .................................................................. 71
9 Paper C:  
Adaptive Task Automata with Earliest-Deadline-First Scheduling  
9.1 Introduction ........................................ 77  
9.2 Adaptive Task Automata ............................... 77  
9.2.1 Introductory Example ............................. 78  
9.2.2 Overview of the Existing Framework ............... 79  
9.3 Encoding of ATA_{EDF} ............................... 80  
9.3.1 Timed Automata with Updates ..................... 81  
9.3.2 Earliest-Deadline-First Scheduling Policy ........ 82  
9.3.3 Task Releases .................................... 82  
9.3.4 Schedulability Predicates ......................... 83  
9.3.5 Scheduler and Queue .............................. 84  
9.4 Decidability ........................................... 88  
9.4.1 Decidability of Timed Automata with Updates ... 88  
9.4.2 Model Bisimulation ............................... 89  
9.5 Related Work .......................................... 90  
9.6 Conclusion ............................................ 91  
Bibliography .............................................. 93  

10 Paper D:  
Adaptive Task Automata with Earliest-Deadline-First Scheduling  
(full paper)  
10.1 Introduction .......................................... 99  
10.2 Adaptive Task Automata .............................. 99  
10.2.1 Introductory Example ............................ 100  
10.2.2 Overview of the Existing Framework ............. 101  
10.3 Encoding of ATA_{EDF} .............................. 102  
10.3.1 Timed Automata with Updates ................... 103  
10.3.2 Earliest-Deadline-First Scheduling Policy ..... 104  
10.3.3 Task Releases ................................... 104  
10.3.4 Schedulability Predicates ....................... 105  
10.3.5 Scheduler and Queue ............................ 106  
10.4 Decidability .......................................... 110  
10.4.1 Decidability of Timed Automata with Updates .. 110  
10.4.2 Model Bisimulation .............................. 114  
10.5 Related Work ......................................... 119  
10.6 Conclusion ........................................... 120  
Bibliography .............................................. 123
I

Thesis
Chapter 1

Introduction

Starting a modern car does not only turn on the engine, but an array of computers. They are monitoring the status of the car, as well as taking an active role in the driving experience by increasing safety, regulating fuel consumption, and performing a large number of smaller functions that improve the driving experience.

Like cars, the functioning of many modern devices is regulated by built-in computers, sometimes running entire operating systems. Such computers are built specifically for this purpose, which classifies them as embedded systems. They are also often built based on strict requirements regarding how much time can pass between an input event and the response to that event, thus making them real-time systems.

Due to the progress of technology and demand for increased numbers of functions, we can encounter real-time embedded systems in many areas of our life. In toys, home appliances, medical and industrial equipment, vehicles that travel underwater, on land, in air, in space, and on other planets. Computer systems built into these devices share the same challenges, that they have to be reliable, respond in a timely manner and are not easily updatable nor replaceable.

Since we started building embedded systems, we have been creating methods to ensure their reliability and conformity to the specifications.

Many machines containing real-time embedded systems are deployed in highly dynamic ecosystems where the environment can change suddenly and the system has to handle the change or otherwise risk failure with serious consequences. One can try to make more robust systems, but, due to the technological (e.g. limited battery charge) and financial constraints, this can be
Chapter 1

Introduction

Starting a modern car does not only turn on the engine, but an array of computers. They are monitoring the status of the car, as well as taking an active role in the driving experience by increasing safety, regulating fuel consumption, and performing a large number of smaller functions that improve the driving experience.

Like cars, functioning of many modern devices is regulated by built-in computers, sometimes running entire operating systems. Such computers are built specifically for this purpose, which classifies them as embedded systems. They are also often built based on strict requirements regarding how much time can pass between an input event and the response to that event, thus making them real-time systems.

Due to the progress of technology and demand for increased number of functions, we can encounter real-time embedded systems in many areas of our life. In toys, home appliances, medical and industrial equipment, vehicles that travel underwater, on land, in air, in space, and on other planets. Computer systems built into these devices share the same challenges, that they have to be reliable, respond in a timely manner and are not easily updatable nor replaceable. Since we started building embedded systems, we have been creating methods to ensure their reliability and conformity to the specifications.

Many machines containing real-time embedded systems are deployed in highly dynamic ecosystems where the environment can change suddenly and the system has to handle the change or otherwise risk failure with serious consequences. One can try to make more robust systems, but, due to the technological (e.g. limited battery charge) and financial constraints, this can be
impractical. Because of this, researchers have started considering deployment of adaptive real-time embedded systems. Such systems have different modes of functioning and switch between them based on the changes in their environment or internal parameters.

Ensuring that all the requirements are met for a typical embedded system is already complex, and adaptivity makes it even more so. To handle this added complexity, we can utilize formal analysis techniques. They are a set of mathematically-based techniques applied on abstract models of system behavior, which facilitate answering to various questions with respect to system correctness and reliability. Formal analysis enables designers to completely explore all possible states of the entire system and ensure their correctness. Usually, the system designer creates a model of the system and environment and uses some form of formal analysis to verify that they satisfy all the requirements.

To illustrate this, let us consider a simplified adaptive embedded system. If we were to design a battery powered robotic arm, the following requirements might arise:

1. The actuators should respond in less than $X \text{ms}$ to the control input.
2. Feedback from the sensors should be presented to the user within $Y \text{ms}$. 
3. The arm should enter a power saving mode when the battery is low.
4. The microcontrollers should reduce their frequency when there is a risk of overheating.
5. The movement functionality of the arm is prioritized over the feedback.

If we were to test a system against these individual specifications, we might miss the combination of high temperatures and power saving, for instance. In such situations, formal analysis can be considered as an alternative verification method. A correct model of the environment should allow for the low battery and high temperature to coincide so formal analysis would explore it as well.

In this thesis, we propose a formal framework for modeling, simulation, and verification of adaptive embedded systems and their environments. By verifying the concordance of a model to its specification, we can ensure its correctness to a high degree.

The above example may be too simple to serve as a motivating argument of our work, yet with proper methods we can model complex systems and discover rare, yet possible, situations where the system would fail due to a combination of different factors.
1.1 Background

Our work relies mostly on research results in three specific topics. The first is adaptive embedded systems. This area studies the effects of adaptivity on functional and extra-functional properties of embedded systems. The second is model checking. To model-check a system implies creation of an abstraction (model) of the system and then applying some approach to check if the system behaves as expected. And the third is schedulability analysis. Given a set of tasks and restrictions, schedulability analysis answers whether the tasks can be completed under the given restrictions. In the following three sections, we present these areas in more detail.

1.1.1 Adaptive Real-Time Embedded Systems

Embedded systems are microprocessor based systems that control a specific function or a predefined set of functions [22]. In contrast, general purpose systems are made to enable simple transitions between functions as well as adding new functionality, e.g. by installing additional software that tends to involve heavier resource usage than in most embedded cases.

On the other hand, a real-time system is any system where the correct behavior is defined by the (logical) correctness of the system outputs as well as their timeliness [24]. Such systems can be classified as hard real-time or soft real-time. In hard real-time systems, providing the output outside of the predefined time-window would result in severe consequences, and so it is not acceptable. For soft real-time systems, output outside of the specified time will have lower value, but can be still considered usable.

A classical example of a hard real-time embedded system is the air-bag system commonly found in modern vehicles. The system consists of three components, a sensor that detects the vehicle crash, an actuator that inflates the air-bag and another one that deflates it. The system has to inflate the air-bag in the precisely calculated moment before the driver collides with the dashboard. Moments later, the air-bag needs to be deflated to avoid the possibility of depriving the driver of air. As the precise timing behavior is critical to the life of the driver, this system is considered a safety-critical hard real-time embedded system.

Since the embedded systems are, by design, tailored to perform certain functions with as little maintenance as possible, they have to have a large degree of independence [24]. In many cases this can be achieved by designing the embedded system to be as robust as possible and to foresee all probable
fluctuations in the environment within a single design. However, due to the technological or production cost constraints, it is often not possible to achieve this level of robustness. In that case, designing an *adaptive* real-time embedded system (AES) can be used as a solution.

The AES is designed with a set of features that can be modified to accommodate the possible changes in the environment. The system may be designed with multiple goals in mind, such as optimal performance or low power consumption, between which a trade-off has to be made. In addition, the effect of the changes to the system has to be considered and accounted for.

The characteristics of adaptation goals, mechanisms, and effects are sometimes called modeling dimensions. For a complete review of modeling dimensions, we refer the reader to Cheng et al. [10]. The authors have provided a general framework that describes modeling dimensions for self-adaptive systems that is well applicable to the design of adaptive embedded systems.

### 1.1.2 Model-checking Real-time Systems

Many techniques are used to ensure a system’s correctness (that is, meeting the specified requirements), most widely spread being system testing. However, testing can be done only after a prototype of the system has been already developed and often requires large amount of manpower. The recent increase in the available computing power has opened up opportunities for the application of theorem-proving and exhaustive model-checking.

By applying model-checking techniques to a model of a system, one can explore all possible system states and ensure that in every state that is reachable for a certain environment, the model behaves as required. One of the main reasons why this approach is still used less than testing is that the number of system states grows exponentially as the complexity of the system increases, and modern model-checkers can handle state spaces of about $10^9$ with explicit state-space enumeration [4]. Using symbolic representations of the state space and cleverer algorithms, this number can be raised up to $10^{476}$ for specific problems [4].

Model-checking of a real-time system requires creation of a *model* of the system that can be *verified* against a set of *requirements specifications*. As shown in Figure 1.1, this model describes possible behaviors of the system which are compared to a formalized requirements specifications. For a given model and each of the requirements specifications, the verifier outputs either a confirmation that the requirement is satisfied or a counterexample proving otherwise.
Chapter 1. Introduction

fluctuations in the environment within a single design. However, due to the technological or production cost constraints, it is often not possible to achieve this level of robustness. In that case, designing an adaptive real-time embedded system (AES) can be used as a solution. The AES is designed with a set of features that can be modified to accommodate the possible changes in the environment. The system may be designed with multiple goals in mind, such as optimal performance or low power consumption, between which a trade-off has to be made. In addition, the effect of the changes to the system has to be considered and accounted for.

The characteristics of adaptation goals, mechanisms, and effects are sometimes called modeling dimensions. For a complete review of modeling dimensions, we refer the reader to Cheng et al. [10]. The authors have provided a general framework that describes modeling dimensions for self-adaptive systems that is well applicable to the design of adaptive embedded systems.

1.1.2 Model-checking Real-time Systems

Many techniques are used to ensure a system’s correctness (that is, meeting the specified requirements), most widely spread being system testing. However, testing can be done only after a prototype of the system has been already developed and often requires large amount of manpower. The recent increase in the available computing power has opened up opportunities for the application of theorem-proving and exhaustive model-checking.

By applying model-checking techniques to a model of a system, one can explore all possible system states and ensure that in every state that is reachable for a certain environment, the model behaves as required. One of the main reasons why this approach is still used less than testing is that the number of system states grows exponentially as the complexity of the system increases, and modern model-checkers can handle state spaces of about $10^{9}$ with explicit state-space enumeration [4]. Using symbolic representations of the state space and cleverer algorithms, this number can be raised up to $10^{476}$ for specific problems [4].

Model-checking of a real-time system requires creation of a model of the system that can be verified against a set of requirements specifications. As shown in Figure 1.1, this model describes possible behaviors of the system which are compared to a formalized requirements specifications. For a given model and each of the requirements specifications, the verifier outputs either a confirmation that the requirement is satisfied or a counterexample proving otherwise. The counterexample can be used to refine the model and solve the discovered conflicts. After the model has been refined, the process starts from the beginning. Once the model satisfies all requirement specifications, the model can be used as a basis for development or generation of the real system.

The greatest limit to model-checking tends not to be the verification tool, but that the model representing the system is not an accurate representation of the system. Thus there is a constant search for more user-friendly and semantically understandable modeling frameworks.

1.1.3 Schedulability Analysis

In the context of this thesis, a task is a computation that is executed sequentially on a CPU. Every task is characterized by its worst-case execution time (WCET) denoted by $C$, a relative deadline denoted by $D$, and, if required by the scheduling algorithm, priority denoted by $P$.

Another parameter that commonly characterizes tasks is minimum inter-arrival time, denoted by $T$, which is the shortest amount of time between two consecutive releases of the same task. In the context of this thesis, this is
replaced by task automata models that specify when the tasks can be released.

While alternative forms of computation exist, in this thesis, we are looking at the sequential execution of multiple tasks on a single CPU. To enable this, the operating system uses a concept of a ready queue, a sequence of tasks that will be executed on the CPU one after another [9]. The algorithm that defines how tasks are ordered in the ready queue is the scheduling algorithm that implements a scheduling policy.

Within this thesis, we will observe two specific scheduling policies, the fixed priority first (FPS), and the earliest deadline first (EDF) scheduling policy.

The FPS policy requires that the task priorities are defined before the execution of the system. During the system execution, the priorities remain constant. The ready queue is sorted in such manner that the currently executing task has the highest priority among the tasks in the queue.

For the EDF policy, the priorities are determined at runtime such that the task in the ready queue that is the closest to its deadline is currently executing. The EDF policy is a dynamic priority scheduling policy since we do not know the relative ordering of tasks in the ready queue before the system execution.

Tasks in the task queue can be in one of the following states: ready - a task is waiting for execution on the CPU, running - a task is currently executing on the CPU, or preempted - a task of a higher priority was selected for execution on the CPU so the current task is waiting to resume. After the task has been executing on the CPU for $C$ time units before its deadline expires, it is considered finished, since that amount of the execution accounts for even the worst case scenario. If all tasks in the queue can complete their execution before their deadlines, we say that the queue is schedulable by the assumed scheduling policy. Verifying that all tasks complete by their deadlines is called schedulability analysis.

Generally, real-time tasks are divided into three types, hard, firm, and soft. A hard real-time task has to be completed before the deadline or otherwise cause catastrophic consequences. A firm real-time task is any task that does not damage the system by missing the deadline, but the computed output has no value. A soft real-time task is any task for which the computed output is valuable even if it is computed after the deadline has expired, the output usefulness is decreasing with its tardiness [9]. In this work, all the tasks are considered hard real-time tasks.

Tasks can be also distinguished based on their periodicity. They can be periodic, aperiodic, and sporadic. Periodic tasks are indefinitely released into the queue in regular intervals, as shown in Figure 1.2(a). Aperiodic tasks, shown in Figure 1.2(b), are also indefinitely released into the system, but we do not know their arrival times in advance. A variant on aperiodic tasks are sporadic
Generally, real-time tasks are divided into three types, hard, firm, and soft. A hard real-time task has to be completed before the deadline or otherwise damage the system by missing the deadline, but the computed output has no catastrophic consequences. A soft real-time task is any task for which the computed output is valuable even if it is computed after the deadline has expired, the output usefulness is decreasing with its tardiness \[9\]. In this work, all the tasks are considered hard even if it is computed after the deadline has expired, the output usefulness is decreasing with its tardiness \[9\].

In this thesis, we will observe two specific scheduling policies, the fixed priority first (FPS), and the earliest deadline first (EDF) scheduling policy. The FPS policy requires that the task priorities are defined before the execution of the tasks. The algorithm that defines how the ready queue is ordered in the ready queue is the highest priority among the tasks in the queue.

The EDF policy is a dynamic priority scheduling policy since we do not know their arrival times in advance. A variant on aperiodic tasks are sporadic. In Figure 1.2(b), are also indefinitely released into the system, but we do not know the exact pattern of their release. We only know the minimum interval between two consecutive task releases \[9\].

Tasks can be also distinguished based on their periodicity. They can be periodic, aperiodic, and sporadic. Periodic tasks are indefinitely released into the system. Figure 1.2(a) is a periodic task that all tasks complete by their deadlines is called schedulability analysis. To enable this, the operating system uses a concept of a ready queue. The current task is waiting to resume. After the task has been executing on the CPU, or preempted CPU so the current task is waiting to resume. The ready queue is sorted in such manner that the currently executing task has the highest priority among the tasks in the queue. The ready queue is sorted in such manner that the currently executing task has the highest priority among the tasks in the queue. During the system execution, the priorities remain constant.

Figure 1.2: Task periodicity: (a) periodic, (b) aperiodic, and (c) task pattern.

While alternative forms of computation exist, in this thesis, we are looking at the sequential execution of multiple tasks on a single CPU. To enable this, the tasks are ordered in the ready queue that is the closest to its deadline is currently executing. The relative ordering of tasks in the ready queue before the system execution.

The ready queue is sorted in such manner that the currently executing task has the highest priority among the tasks in the queue. The ready queue is sorted in such manner that the currently executing task has the highest priority among the tasks in the queue. Within this thesis, we will observe two specific scheduling policies, the fixed priority first (FPS), and the earliest deadline first (EDF) scheduling policy. The FPS policy requires that the task priorities are defined before the execution of the tasks. The algorithm that defines how the ready queue is ordered in the ready queue is the highest priority among the tasks in the queue.

Schedulability Analysis with Timed Automata. Timed automata \[3\] are automata with continuous real clocks. Properties of timed automata, such as location reachability and liveness, can be verified using UPPAAL\(^2\) model-checking tool. The current location in a timed automaton as well as values of all variables make up the state of the timed automaton. The state is said to be reachable if there exists a path from the initial state to the state in question. Liveness properties are relevant to the infinite sequences of transitions and

\[1\]In order to model this specific task release pattern as a verifiable task automaton, the sequence of prime numbers would need to repeat itself at an arbitrary point.

\[2\]UPPAAL model-checking tool can be found at http://www.uppaal.org/
specify properties such as “if action A occurs infinitely often, so does the action B” [2].

Task automata [14] are an extension of timed automata designed for modeling and verification of requirements related to task execution within a real-time embedded system. They have well-defined formal semantics, so that scheduling by model-checking can be applied. The semantics of task automata are given in terms of timed transition systems.

In Figure 1.3 we can see an example of a task automaton. It consists of one initial location and six locations in which the task \( t_1 \) is released. The variable \( x \) is a continuous real clock that can be reset to an integer value and increases linearly. The locations are connected by edges.

Each of the locations is annotated by an invariant that determines how much time the automaton can spend in that location. The edges are annotated by guards that determine when the edge can be traversed. Just before reentering the initial location, the clock \( x \) is reset.

Properties regarding schedulability of task automata can be verified using Times Tool\(^3\).

1.2 Thesis Outline

The remainder of the thesis is organized into two parts:

**Part I** includes the first six chapters. Chapter 2 describes the research problem addressed in this thesis as well as introduces the individual research goals. Chapter 3 provides a brief overview of the research results and their correlation to the research goals. Chapter 4 presents a brief overview of the research method applied in this thesis. Chapter 5 positions the work presented in this thesis with respect to other relevant work in the field. Chapter 6 presents our general conclusions and possible future extensions.

**Part II** presents the technical contributions of the thesis in the form of four papers that are organized in Chapters 7 to 10.

\(^3\)Times Tool can be found at [http://www.timestool.com/](http://www.timestool.com/)
Chapter 2

Research Problems

2.1 Problem Description

Although there are significant advancements in the domain of schedulability analysis for periodic and sporadic tasks in the adaptive contexts [23, 26, 27], few works address the issues of analyzing tasks that are neither truly periodic nor sporadic, but have a known release pattern that can be influenced by either internal or external triggers that react to changes in the environment. In a system that changes task release patterns in response to environmental fluctuations, adaptivity can be of great benefit. The goal is to create a system that can gracefully handle exceptional changes in the environmental factors while ensuring the highest quality of service in other situations.

2.2 Research Goals

In order to meet the above desideratum, we have defined the following research goal that our research has tried to address:

**Goal.** To provide a framework for the design of adaptive hard real-time embedded systems with non-uniformly recurring computation tasks.

As we have previously described, the need for adaptive hard real-time embedded systems has motivated us to proceed with the development of a framework for designing adaptive, formally verifiable, embedded systems. Since
the goal above is still fairly abstract, we have further split it into three more concrete subgoals.

In order to be able to provide a framework for designing adaptive ES, we need an expressive model with well-defined semantics that would support describing adaptive behavior at an abstract level. This justifies the first subgoal given below:

**Subgoal 1.** Develop a *formal model* for adaptive hard real-time embedded systems in which the system adapts based on its state and the state of the environment, plus the schedulability of potentially released tasks.

The first subgoal establishes the basis for the next two subgoals, in that it results in a model that can be verified and formally examined. The next step is then concerned with proposing means of verifying the assumed model scheduled by a given fixed-priority scheduling policy (FPS), which gives rise to the second subgoal as follows:

**Subgoal 2.** Describe a way to formally verify the proposed model assuming static (task) priorities (e.g. FPS) w.r.t. schedulability, reachability and liveness properties.

In the second subgoal, we analyze the decidability (possibility to compute the truth value) of the verification of our newly created model. This subgoal is specific in that it analyzes only systems in which the relative task priorities are static and predicted before the verification of the system.

To address dynamic scheduling policies also, which are deemed optimal (meaning that all tasks that pass the specific schedulability test can be scheduled by the policy), we have formulated the third subgoal as follows:

**Subgoal 3.** Describe a way to formally verify the proposed model assuming dynamic (task) priorities (e.g. EDF) w.r.t. schedulability, reachability and liveness properties.

The last subgoal partially relaxes the restrictions on the previous subgoal in that we are verifying the systems that have dynamic task priorities during the verification, but they are consistent after the tasks are released into the system.
Chapter 3

Research Results

In this section we will give an overview of our research results that address the subgoals presented in Section 2.2. The chapter is divided into four parts. First we present our modeling framework proposed in this thesis called adaptive task automata (Section 3.1). Next, we present the model-checking specifics and results of verifying the two variants of ATA – with fixed-priority scheduling (Section 3.2) and dynamic-priority scheduling (Section 3.3).

3.1 Adaptive Task Automata

In this thesis we introduce adaptive task automata (ATA) for modeling adaptive hard real-time systems. ATA consists of task automata [14] extended by the schedulability predicates.

Adding schedulability predicates to ATA makes it possible to model tasks that have release patterns dependent on events of other tasks in the system, or alter task release patterns based on the influence of potential task releases on already released tasks. Thus providing an effective approach to modeling adaptive hard real time systems.

Here, we provide a brief overview of the ATA framework; for the full description of ATA, we refer the reader to Chapters 7 to 10.

In the adaptive task automata framework, we model task release patterns using timed automata, an extension of automata with continuous time variables called clocks [2, 3, 6].

Let us introduce ATA by Figure 3.1, in which an example of an adaptive
Figure 3.1: An example of adaptive task automaton.

The automaton consists of three locations, one initial (1), and two ordinary locations (5). The initial location is associated with a task $t_1$ and has an invariant (2). The model contains one clock variable $x$ that is initialized to 0 when the system starts and then progresses until it is reset to zero (4) on the edges between locations with the task $t_1$ and tasks $t_2, t'_2$, respectively. The tasks $t'_2$ can be considered an alternative to the task $t_2$ that achieves the same purpose at a lower quality and thus has reduced computation time. The invariant (2) is a clock constraint that limits the amount of time the system can spend in a location, whereas guards (3) are also Boolean conditions that need to be satisfied in order for an edge to be traversed.

In order to create models of systems that choose tasks to be released depending on their schedulability, we have introduced schedulability predicates as part of the ATA framework. These predicates determine the schedulability of a task within the context of the current ready queue. In Figure 3.1, the predicate $sced(t_1, t_2)$ is part of the guard. This predicate is true when the task $t_1$ will complete in a timely manner even if the task $t_2$ is released into the system.

### 3.2 Model-checking ATA with Static-Priority Scheduling

Our first result [20] is proving the decidability of ATA with fixed-priority scheduling, with respect to schedulability properties. In order to achieve this, we transform our model into timed automata that can be verified by UPPAAL and show that the new timed automata model is indeed decidable.

The procedure of encoding can be summarized in the following steps. First, we define an automaton that represents the scheduler and the queue. Second, for each location that is annotated with tasks, e.g. shown in Figure 3.2(a), we
3.3 Model-checking ATA with Dynamic-Priority Scheduling

Our next goal has been to try to address the verification of schedulability with ATA, in the context of dynamic priority scheduling, by assuming a variant of
Table 3.1: Contribution of the individual papers to the research subgoals

<table>
<thead>
<tr>
<th>Paper</th>
<th>Subgoal 1</th>
<th>Subgoal 2</th>
<th>Subgoal 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper A</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Paper B</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Paper C</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Paper D</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

3.4 Contribution of Included Papers

This thesis includes four research papers. In the following we summarize the contributions of the thesis per paper, as well as my specific contributions to each paper. The relationship of each paper to the subgoals is presented in Table 3.1.
3.4 Contribution of Included Papers

3.4.1 Paper A


Summary. In this paper, we introduce the adaptive task automata framework and by that address the Subgoal 1 and Subgoal 2. At this point, we have proposed a framework for verifying the schedulability of adaptive systems that assume a fixed-priority task scheduling policy. The framework is based on timed automata with tasks and extended with primitives that support testing of schedulability of a given task. By addressing the fixed priority scheduling first, we have covered a large number of possible uses of our system, while at the same time covering an entire class of fixed priority schedulers.

Contribution. The initial idea of maintaining the response times for all the tasks in the queue was presented by Fersman et al. [15], and brought to my attention by my supervisors. I have done most of the work on the development of the idea into a functioning framework and proof of concept implementation in the UPPAAL tool.

3.4.2 Paper B


Summary. In this paper, we present a summary of the first paper as well as some additional, more general examples. The examples outline the potential for our framework in practical applications for designing adaptive hard real-time embedded systems. As a natural extension of Paper A, this paper continues to address Subgoals 1 and 2.

Contribution. I have done most of the work on conceiving and implementing the examples, as well as writing the corpus of the paper.
3.4.3 Paper C


**Summary.** In this paper, we start from the framework introduced in Paper A and improve it with the earliest deadline first scheduling policy. Our extension to dynamic scheduling policies has required the exploration of decidability of a subclass of timed automata with clock-to-clock assignments. The third paper addresses Subgoals 1 and 3.

**Contribution.** I have defined the specific theorems and proof sketches, based on the proofs developed in previous work [3, 14].

3.4.4 Paper D


**Summary.** We conclude this collection of papers with a technical report that provides a detailed proof of decidability and bisimilarity between ATA and its encoding in the framework of timed automata (TA) sketched in Paper C. Since the framework itself is not changed in this paper, it contributes only to Subgoal 3.

**Contribution.** Similar to Paper C. The structure of the proofs is inspired by previous work and I have developed the proofs needed in our case.
Chapter 4

Research Method

The research method used for this thesis is derived from the scientific method for computer science [13], and software architecture [33].

Figure 4.1: A simplified illustration of the scientific method

The scientific method for computer science [13] proposes the following approach to the process of research, as illustrated in Figure 4.1: (i) first, a question is proposed in the context of the existing knowledge, (ii) second, a hypothesis is formed as a tentative answer to the question, (iii) third, predictions are made about the hypothesis, (iv) fourth, the hypothesis is tested carefully and checked if it fits within the current knowledge or if adjustments to the already existing knowledge need to be made, and (v) fifth, when consistency is reached, the hypothesis becomes a theory and provides a coherent set of propositions that define a new theoretical concept.

Our research method was mostly based on the scientific method for computer science with a strong accent on the influence of the related work on the
research. As illustrated in Figure 4.2, the research starts with a formulation of a general research problem based on the currently available scientific knowledge. This research problem is then ported into the research setting and is further refined into more tangible research problems. After the research problem has been understood, it is conceptualized. A conceptualized research problem corresponds to the question of the scientific method. From the conceptualization, a research goal is derived, which corresponds to the hypothesis.

Once a research goal is defined, using original ideas and the ideas from related work, a solution is designed, together with tests for testing the validity of this solution within the context of the research goal. Using information gained from testing the solution, we can further refine both the research goal, and the solution design until consistency is achieved. Further, when all the conceptualizations have been assembled in a study, the study is validated by comparing it to the related work.

The general research problem that initiated our research was on how to
address the challenges of creating safe adaptive embedded systems. Within this topic we have then located specific goals of verifying schedulability in an adaptive hard real-time embedded system, as presented in Goal 2.2 and Subgoals 1-3.

From this point, we have constructed the concept of the adaptive task automata framework. In the first iteration, the framework was refined for static priority scheduling, and adjusted for dynamic priority scheduling. Each of the refinements of the theoretical model was supported by mathematical proofs of the claims. Although we have mathematically showed correctness of our claims, we have have not done validation on real-world case study or in an industrial environment.
Chapter 5

Related Work

This thesis focuses on providing a framework for schedulability analysis via formal verification of real-time task sets, in the context of adaptive embedded systems. Although not many research results have been published on the exact topic, there are several related works that are relevant to our research.

The related research can be classified into three clusters. In the first, we encompass the research that is considering design and verification of higher abstractions of the adaptive embedded systems. The second analyzes adaptive embedded systems and their task sets using analytic approaches. The third considers verification of schedulability by automated model checking, yet without any focus on adaptive characteristics.

5.1 Modeling and Verification of High-level Abstractions of Adaptive Embedded Systems

Most of the research on the adaptive embedded systems considers higher level descriptions of the adaptation behavior. The following are some examples that are most closely related to our work. All of the following examples support modeling of adaptive behaviors and some form of verification while, in contrast to our research, verification of schedulability is not directly supported.

Schaefer [30] has provided several approaches on verifying adaptive embedded systems specified as Synchronous Adaptive Systems - high level representations of modeling concepts used in the MARS modeling approach [35]. The solution integrates model slicing of various granularities to reduce
Chapter 5
Related Work

This thesis focuses on providing a framework for schedulability analysis via formal verification of real-time task sets, in the context of adaptive embedded systems. Although not many research results have been published on the exact topic, there are several related works that are relevant to our research.

The related research can be classified into three clusters. In the first, we encompass the research that is considering design and verification of higher abstractions of the adaptive embedded systems. The second analyzes adaptive embedded systems and their task sets using analytic approaches. The third considers verification of schedulability by automated model checking, yet without any focus on adaptive characteristics.

5.1 Modeling and Verification of High-level Abstractions of Adaptive Embedded Systems

Most of the research on the adaptive embedded systems considers higher level descriptions of the adaptation behavior. The following are some examples that are most closely related to our work. All of the following examples support modeling of adaptive behaviors and some form of verification while, in contrast to our research, verification of schedulability is not directly supported.

Schaefer [30] has provided several approaches on verifying adaptive embedded systems specified as Synchronous Adaptive Systems - high level representations of modeling concepts used in the MARS modeling approach [35]. The solution integrates model slicing of various granularities to reduce the
complexity and enable automated model checking of the models by means of theorem proving. The technique is tested on adaptive vehicle stability control system.

Schneider et al. [31] have proposed a method to describe and analyze adaptation behavior in embedded systems in which the data flow is augmented with quality descriptions used by configuration rules to determine potential adaptations. Then, the augmented system can be transformed into a transition system and serve as input for a model checker.

Adler et al. [1] use Kripke structures as the underlying system model and specify the system’s properties using LTL. This representation of the system is then verified using the Averest\(^1\) framework.

Goldsby et al. [16] provide the AMOEBA-RT model focused on run-time verification and monitoring.

Tan [34] has introduced a design workflow that takes a system specified as hybrid automata, augments it with reconfiguration information and builds a self-adaptive model. Self-adaptive models are achieved by transformation and addition of monitoring automata. Further, such models can be transformed into program code. While self-adaptivity is an interesting venue, design time verification of such system has not been investigated yet.

5.2 Analytic Approaches

In the area of adaptive scheduling, most of the work [23, 26, 27] aimed at achieving a lower energy consumption by exploiting dynamic voltage scaling features of modern CPUs. While such approaches can be used to analyze schedulability in some adaptive contexts, our approach is more general in some aspects. Besides periodic, aperiodic, and sporadic tasks, we can analyze tasks that are released into the system based on complex interactions with internal and environmental events.

The work by Shakhlevich et al. [32] proposes an adaptive approach for fine tuning of a realistic heuristic scheduling algorithm. While the paper is focused more on job shop scheduling rather than tasks, the approach could eventually provide a heuristic that can be verified as correct by our framework.

Lawrence et al. [25] have proposed a control-theoretic approach to distribution of computing resources. However, with the growing complexity of the scheduling algorithm, verification complexity of schedulability in a hard real-time system increases.

\(^1\)The Averest framework is available at http://www.averest.org.
Another relevant line of research, but based on different set of assumptions is fault-tolerant scheduling [29, 17]. While our work assumes that any errors will lead to failure, fault-tolerant scheduling tries to recover from errors while minimizing loss.

Beccari et al. [5] analyze adaptive soft real-time tasks and alter the rate of task releases depending on the available resources. While this work proposes analytical solutions over our verification solution, the main other difference is in the definition of tasks. This work defines ranges of acceptable admission rates for tasks, while our framework can define arbitrary admission criteria.

5.3 Related Verification Approaches

The application of schedulability verification has already targeted multiprocessor systems in the work by Yu et al. [38], or satellite systems in the work by Mikučionis et al. [28], and results on generalized frameworks for schedulability analysis have also been provided by David et al. [11]. However, in these studies the non-schedulability of the system cannot be predicted soon enough such that the system does not reach an error state, but only after a task misses its deadline.

Wang et al. [37] have proposed usage of verification techniques to find the optimal schedule for energy constrained systems. The authors have developed a cost-reward variant of timed automata that makes it possible to directly model energy expenditure of different tasks.
Chapter 6

Conclusions

6.1 Summary and Conclusions

The main goal of this thesis is to improve the state-of-the-art regarding modeling and verification of adaptive embedded systems. To achieve this, we have focused our efforts on adaptive hard real-time embedded systems and their task-level abstractions.

A key component to our work is that we can model networks of adaptive task automata. We have achieved this by grounding our work in task automata that already had this feature. Networks of task automata provide support for compartmentalization of adaptive task automata models so that different functions can be modeled separately. While this does not influence their decidability or expressiveness, for us, it means that a designer can clearly distinguish the model of the environment from the model describing internal responses to the changes in the environment. The internal responses are then reflected in the change of task release patterns, which can also be described in our framework.

Tasks in our framework can be released periodically, sporadically, or aperiodically, as long as their behavior can be modeled using task automata. The resulting pattern of task releases allowed by the task automata model constitutes our assumed task release pattern.

A change in the task release pattern to adjust to the altered environmental conditions can cause already existing tasks in the system to miss their deadlines and result in system failure. Thus, as the main contribution of this thesis, we have defined and shown decidability for a set of predicates that can inspect the state of the scheduler and queue. By combining these predicates with the model...
Chapter 6

Conclusions

6.1 Summary and Conclusions

The main goal of this thesis is to improve the state-of-the-art regarding modeling and verification of adaptive embedded systems. To achieve this, we have focused our efforts on adaptive hard real-time embedded systems and their task-level abstractions.

A key component to our work is that we can model networks of adaptive task automata. We have achieved this by grounding our work in task automata that already had this feature. Networks of task automata provide support for compartmentalization of adaptive task automata models so that different functions can be modeled separately. While this does not influence their decidability or expressiveness, for us, it means that a designer can clearly distinguish the model of the environment from the model describing internal responses to the changes in the environment. The internal responses are then reflected in the change of task release patterns, which can also be described in our framework.

Tasks in our framework can be released periodically, sporadically, or aperiodically, as long as their behavior can be modeled using task automata. The resulting pattern of task releases allowed by the task automata model constitutes our assumed task release pattern.

A change in the task release pattern to adjust to the altered environmental conditions can cause already existing tasks in the system to miss their deadlines and result in system failure. Thus, as the main contribution of this thesis, we have defined and shown decidability for a set of predicates that can inspect the state of the scheduler and queue. By combining these predicates with the model
of the task release pattern, we can prevent deadline misses in the tasks already present in the system and thus mitigate the effects of the adaptive processes on the performance of the system.

In conclusion, we have created a framework that is suitable for modeling task-level abstractions of adaptive hard real-time embedded systems. We have also proved the model’s decidability, and implemented one of its variants in UPPAAL\(^1\).

### 6.2 Future Work

During working on the thesis’ topic and writing the dissertation, we have encountered many possibilities to further explore and extend our work.

A significant value could be gained by creating a dedicated tool that supports schedulability verification using the ATA framework. This can be done by extending the already existing tool TIMES\(^2\) or building a new tool tied to the UPPAAL\(^1\) verification engine. To improve usability, additional research can be carried out to determine the optimal way to represent the adaptivity features.

Another possible extension of our work could be towards adopting statistical model checking (SMC) [12] to replace the full state space exploration that we currently use. This would introduce additional challenges, such as ensuring that the state space exploration is not biased, and determining a safe bound for the probabilistic verification. On the positive side, the model checking is done at an exponentially higher speed, and there is a possibility of using distributed model checking [8].

In our framework, the tasks are described using only their fixed worst-case execution time as the main definition of the length of task’s execution time. An interesting venue to explore would be to allow the execution time of tasks to be described as intervals, or in the case of statistical model checking as probability distributions.

Once the tasks are released into the queue, with the current framework, they cannot be significantly modified due to the constraints of the model. It could be interesting to create an encoding where tasks can be arbitrarily removed from the queue, which would free up the time that the task is supposed to spend if it needs to be replaced by another task. This extension would further improve the adaptation capabilities of the framework.

\(^{1}\)The Uppaal tool is available at [http://www.uppaal.org/](http://www.uppaal.org/).

\(^{2}\)The TIMES Tool is available at [http://www.timestool.com/](http://www.timestool.com/).
Chapter 6. Conclusions

of the task release pattern, we can prevent deadline misses in the tasks already present in the system and thus mitigate the effects of the adaptive processes on the performance of the system.

In conclusion, we have created a framework that is suitable for modeling task-level abstractions of adaptive hard real-time embedded systems. We have also proved the model’s decidability, and implemented one of its variants in UPPAAL 1.

6.2 Future Work

During working on the thesis’ topic and writing the dissertation, we have encountered many possibilities to further explore and extend our work. A significant value could be gained by creating a dedicated tool that supports schedulability verification using the ATA framework. This can be done by extending the already existing tool TIMES 2 or building a new tool tied to the UPPAAL 1 verification engine. To improve usability, additional research can be carried out to determine the optimal way to represent the adaptivity features.

Another possible extension of our work could be towards adopting statistical model checking (SMC) [12] to replace the full state space exploration that we currently use. This would introduce additional challenges, such as ensuring that the state space exploration is not biased, and determining a safe bound for the probabilistic verification. On the positive side, the model checking is done at an exponentially higher speed, and there is a possibility of using distributed model checking [8].

In our framework, the tasks are described using only their fixed worst-case execution time as the main definition of the length of task’s execution time. An interesting venue to explore would be to allow the execution time of tasks to be described as intervals, or in the case of statistical model checking as probability distributions.

Once the tasks are released into the queue, with the current framework, they cannot be significantly modified due to the constraints of the model. It could be interesting to create an encoding where tasks can be arbitrarily removed from the queue, which would free up the time that the task is supposed to spend if it needs to be replaced by another task. This extension would further improve the adaptation capabilities of the framework.

1The Uppaal tool is available at http://www.uppaal.org/.

2The TIMES Tool is available at http://www.timestool.com/.

Bibliography


