Analyse of real-time systems from scheduling perspective

Mounir Chadli

▶ **To cite this version:**

Mounir Chadli. Analyse of real-time systems from scheduling perspective. Performance [cs.PF]. Université de Rennes, 2018. English. NNT : 2018REN1S062 . tel-02066632

HAL Id: tel-02066632
https://theses.hal.science/tel-02066632

Submitted on 13 Mar 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THESE DE DOCTORAT DE

L'UNIVERSITE DE RENNES 1
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *(Informatique)*

Par

« Mounir CHADLI »

« Analyse des Systèmes Temps-Réel de point de vue Ordonnancement »

Thèse présentée et soutenue à « Rennes », le « 21/11/2018 »
Unité de recherche : Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA)

Rapporteurs avant soutenance :

Tiziana Margaria	Professeur à l'Université de Limerick
Cristina Seceleanu	Professeur Associé à l'Université de Mälardalen

Composition du Jury :

Président :	Olivier Barais	Professeur à l'Université de Rennes1
Examinateurs :	Kim Guldstrand Larsen	Professeur à l'Université d'Aalborg
	Saddek Bensalem	Professeur à l'Université Grenoble Alpes (UGA)
Dir. de thèse :	Axel Legay	CR Inria Rennes Bretagne Atlantique

Remerciements

First of all, I have to thanks God who give me courage and faith that help me to finish this modest work.

I would like to thank my supervisor, Professor Axel Legay, for the patient guidance, encouragement and advice he has provided throughout my time as his student. I have been extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so promptly.

I would also like to thank all the members of TAMIS team who helped me in my work. In particular I would like to thank Louis-Marie Traounouez, Thomas-Given Wilson and Fabrizio Biondi for their trust and invaluable support.

I am very honored to thank the presence on my thesis jury and I would like to thank:

Mr. Olivier Barais, Professor at the University of Rennes, for the honor he gave me by accepting to be the president of my thesis jury. I wish to assure him of my deep appreciation for his interest in this work.

Mrs Cristina Secelenau, Associate Professor at Malardalen University in Sweden, for the honor she gave me for her participation in my thesis jury as rapporteur of my work, for the time spent reading this thesis, and for the suggestions and the judicious remarks that she indicated to me.

Mrs Tiziana Margaria, Professor at the University of Limerick, for the honor she gave me for her participation in my thesis jury as rapporteur for my work and for all the interesting remarks she made me made.

Mr Kim Guldstrand Larsen, Professor at Aalborg University, for having accepted to be part of the jury of this thesis. I thank him for the scientific advice he has given, and for his immense help in carrying out this work.

Mr. Saddek Bensalem, Professor at Grenoble Alpes University, for his interest in participating as a guest member of this jury.

On a more personal note, I warmly thank my wife Naima, for her continued support and encouragement. I also want to thank the patience of my mother, my father and my

brother who have experienced all the ups and downs of my research.

Completing this work would have been all the more difficult were it not for the support and friendship provided by the other members of TAMIS team, and the IRISA laboratory members in Rennes. I am indebted to them for their help. Those people, such as the postgraduate students at MathSTIC doctoral school, who provided a much needed form of escape from my studies, also deserve thanks for helping me keep things in perspective.

Résumé

De nos jours, les machines sont devenues une partie intégrante de notre vie quotidienne, elles sont utilisées dans différents domaines comme les transports, l'industrie ou la médecine. Ces machines sont contrôlées par des logiciels et des programmes très complexes qui assurent la bonne exécution des taches affectées à ces machines. La plupart de ces machines doit communiquer avec le monde extérieur pour accomplir leurs missions, cette interaction exige que les systèmes qui contrôlent ces machines devraient avoir la possibilités de recevoir des données extérieurs, les traiter et donner les réponses adéquates dans le temps opportun. Ce type de systèmes est appelé *Systèmes Temps Réel*.

Les systèmes temps réel sont des systèmes de traitement d'information qui doivent répondre aux entrées reçues de l'extérieur, en vérifiant des contraintes temporelles strictes sur leur temps de réponse. Ces systèmes sont fréquemment utilisés dans des systèmes critiques tels que les systèmes embarqués, ce qui exige un très haut niveau de sécurité. Une des principales problématiques pour développer ces systèmes est de vérifier que ces contraintes temporelles seront toujours vérifiées, quelques soient les entrées reçues. En plus des contraintes de temps, d'autres contraintes peuvent être prises en considération, comme la consommation d'énergie ou la sécurité des données.

Plusieurs méthodes de vérification ont été utilisées ces dernières années, comme la révision attentive du code des programmes qui s'applique essentiellement sur les logiciels, ou le test qui consiste à réaliser un prototype et puis appliquer différents tests pour vérifier son exactitude. Cependant, avec la croissance de la complexité des logiciels embarqués, ces méthodes ont atteint leur limitation. C'est pourquoi les recherches se concentrent actuellement à développer de nouvelles méthodes et formalismes pour pouvoir vérifier l'exactitude des systèmes les plus complexes.

Le principal travail de ce manuscrit porte sur le développement de techniques avancées d'ordonnancement basées sur des modèles formels. Le but est d'analyser et de valider la satisfaction d'un certain nombre de propriétés sur les systèmes temps réel, dont des propriétés non temporelles telles que la consommation d'énergie ou la fuite d'informations.

L'ordonnancement est un processus de prise de décision utilisé dans de nombreux domaines tels que la fabrication, l'industrie, la médecine et ainsi de suite. Il traite de l'allocation des ressources aux tâches sur une période donnée, et son but est d'optimiser un ou plusieurs objectifs.

Les ressources et les tâches peuvent prendre plusieurs formes différentes. Les ressources peuvent être des machines dans un atelier, des pistes dans un aéroport, des unités de traitement dans un environnement de calcul, et ainsi de suite. Les tâches peuvent être des

opérations dans un processus de production, l'atterrissage dans un aéroport, l'exécutions de programmes informatiques, etc. Chaque tâche peut avoir un certain niveau de priorité, un instant de début et une date d'échéance. Les objectifs peuvent également prendre plusieurs formes différentes. Un objectif peut être la minimisation du temps d'achèvement de la dernière tâche. Un autre peut être la minimisation du nombre de tâches accomplies après leurs échéances respectives.

L'ordonnancement, en tant que processus décisionnel, joue un rôle important dans la plupart des processus de fabrication et des systèmes de production ainsi que dans la plupart des environnements de traitement de l'information. Dans les systèmes temps réel l'ordonnancement consiste à planifier de l'utilisation des ressources de calcul de façon à satisfaire toutes les contraintes temporelles.

Les algorithmes classiques d'ordonnancement des systèmes temps réel se basent sur la priorité des taches et leurs échéances respectives. La priorité de la tâche peut être une valeur fixe donnée initialement, ce type d'algorithmes est dit *Statique*, ou peut être une valeur variable calculer lors de l'exécution de l'ordonnancement, ce type d'algorithmes est dit *Dynamique*.

Pour décrire des systèmes plus complexes les techniques basées sur des modèles utilisent des modèles formels tels que les *Systèmes de Transition* et les *Automates Temporisés* pour spécifier le comportement logique du système et les contraintes temporelles. Des extensions aux automates temporisés permettent également de spécifier des mécanismes d'ordonnancement ou des problèmes d'énergie.

Une première contribution de la thèse est de proposer un nouveau modèle stochastique pour décrire des tâches dont le temps d'exécution n'est pas fixe, mais décrit par une distribution de probabilités. Ce modèle peut être utilisé pour représenter des tâches apériodiques dont leur déclenchement est dû à des événements extérieurs au système, et leur temps d'exécution n'est pas fixe et dépend de l'ensemble des tâches qui constituent le système.

Pour analyser ces modèles formels complexes on utilise des techniques de vérification automatisée tel que le *model-checking*. Cette technique permet de vérifier la satisfaction de propriétés décrites en utilisant une logique formelle. Elle consiste à explorer toutes les exécutions possibles du modèle et vérifier la satisfaction de la propriété étudiée à chaque exécution.

Une alternative est la technique de *model-checking statistique (SMC)* qui traite un échantillon de l'ensemble de scénarios des exécutions possibles pour quantifier la probabilité de satisfaction d'une propriété donnée. Cette méthode est utilisée pour gérer le problème d'explosion d'état, c'est-à-dire le nombre d'états nécessaires pour modéliser

le système avec précision, qui peut facilement dépasser l'espace mémoire disponible sur l'ordinateur.

Enfin cette thèse propose d'utiliser un langage de haut niveau pour avoir une description graphique simplifiée des modèles formels. Cette représentation graphique est transformée automatiquement en un ensemble de modèles formels, puis les différentes propriétés peuvent être vérifiées sur ces modèles en utilisant l'outil de vérification Uppaal. Les résultats de la vérification formelle sont analysés et les informations les plus pertinentes sont affichées graphiquement sur le modèle graphique.

Cette thèse étudie trois catégories de problèmes d'ordonnancement dans les systèmes temps réel. Le premier modèle analyse des *systèmes d'ordonnancements hiérarchiques (HSS)*. Les HSS sont des systèmes d'ordonnancement complexes avec plusieurs algorithmes d'ordonnancement. Nous construisons tout d'abord une banque de modèles formels générique à l'aide d'automates temporisés. Ces modèles sont utilisés pour représenter le comportement des différents composants du système étudié. Nous utilisons en particulier le modèle de tâches temps réel stochastiques implémenté à l'aide d'automates temporisés probabilistes (PTA). Cette banque de modèles est utilisée pour construire des HSS complexes.

Pour modéliser les HSS on utilise un langage de haut niveau spécifique implémenté avec l'outil Cinco. Cinco est un générateur d'outils de modélisation. Il permet de spécifier les fonctionnalités d'une interface graphique dans un langage de méta-modèle compact, et il génère automatiquement à partir de ce méta-modèle un outil d'analyse spécifique au domaine avec une interface graphique.

A l'intérieur de cet outil d'analyse, nous pouvons définir les spécifications d'un HSS et les propriétés qu'il doit satisfaire. Nous pouvons ensuite lancer l'analyse des propriétés. Cela génère automatiquement les modèles d'automates temporisés en utilisant les composants de notre banque de modèles. Les outils Uppaal et Uppaal SMC sont ensuite utilisés pour effectuer l'analyse. Cette approche permet de masquer complètement les modèles formels utilisés par le concepteur du système qui peut se concentrer sur la structure et les paramètres du HSS.

Pour illustrer les travaux réalisés pour résoudre ce premier problème, nous présenterons les résultats de nos expériences réalisées sur un cas d'étude qui consiste en un système de contrôle d'un aéronef. Ces expériences qui consistent à vérifier la possibilité dâĂŹordonnancement des tâches du système étudié de façon hiérarchique, du plus bas niveau vers le haut. Le but final est de calculer le budget minimal nécessaire pour l'ordonnancement du système global.

Le deuxième problème d'ordonnancement étudié analyse la consommation d'énergie sur une plate-forme multiprocesseur. En effet beaucoup de CPS évoluent dans des environnements restreints avec une quantité limitée d'énergie. Le défi principal de ce type de systèmes est de s'assurer qu'ils puissent accomplir leurs missions uniquement avec la quantité d'énergie initialement allouée.

Pour formaliser les systèmes d'ordonnancement multiprocesseurs avec ressources énergétiques, nous étendons les modèles formels précédents avec des informations sur la consommation d'énergie. Après cela, en utilisant Cinco, nous donnons une représentation graphique au modèle formel. Ce modèle est constitué de deux couches. La première couche modélise la plate-forme matérielle avec un système d'ordonnancement composé de tâches temps réel et de CPUs. La deuxième couche modélise l'application qui est composée d'un ensemble d'actions.

Une des contributions de la thèse est de proposer une nouvelle technique d'optimisation pour les ordonnanceurs multiprocesseurs. Cette technique détermine les mappages optimaux des tâches aux processeurs afin de minimiser la consommation d'énergie du système et/ou le temps de réponse en utilisant des tests statistiques (ANOVA et Tukey HSD). Tout d'abord, nous déterminons tous les mappages possibles entre tâches et processeurs. Ensuite en utilisant Uppaal SMC nous évaluons la consommation d'énergie moyenne et le temps de réponse moyen de chaque mappage. Enfin en utilisant les tests ANOVA et Tukey HSD nous comparons les moyennes et classons les mappages selon leurs consommation d'énergie et/ou temps de réponse.

Nous présentons aussi un algorithme statistique de détection de changement appelé CUSUM. Le principe est de suivre l'évolution d'une mesure de probabilité à des intervalles de temps successifs au cours d'une seule exécution du système. L'algorithme détecte ensuite la position où la probabilité de satisfaire la propriété change de façon significative.

Pour mieux expliquer notre approche pour le problème de consommation d'énergie, nous proposons comme cas d'étude un système multiprocesseur se composant de deux niveaux. Le premier niveau consiste à un ensemble de composants qui comporte une série d'action à accomplir. Le deuxième niveau comporte un nombre de processeurs sur lesquels les tâches vont être exécutées. Chaque tâche est destinée à exécuter un ou plusieurs composants du système étudié. Les résultats des expériences réalisés sur ce modèle et leurs synthèses sont présentés en fin du Chapitre 5.

Le dernier problème étudie les fuites d'informations lorsque des processus avec différents niveaux de sécurité se partagent l'espace mémoire lors de leurs exécutions. Typiquement cela inclut le chargement d'informations confidentielles, telles que les clés de chiffrement, les données médicales et les coordonnées bancaires, pour les utiliser dans des

processus de haute sécurité. Ces informations confidentielles doivent être étroitement contrôlée et ne pas être divulgué à des processus de faible niveau sécurité.

Dans cette thèse nous proposons de traiter la confidentialité, mesurée par le résultat de fuite d'informations sécurisées, en tant que ressource quantitative que l'ordonnanceur peut exploiter. Ce qui permet une meilleure quantification de la fuite qui en résulte dans différents scénarios, ainsi que d'avoir une mesure claire du coût des différents choix d'ordonnancement. En outre, cela permet la création de planificateurs qui peuvent faire de meilleurs choix de programmation et aussi respecter les contraintes de fuite d'informations confidentielles.

Nous présentons pour cela un nouveau modèle qui considère que les tâches doivent être composées d'étapes, chacune d'entre elles à un temps d'exécution, une valeur de fuite d'information et un niveau de sécurité. Chacune de ces étapes est implicitement une séquence atomique d'actions qui peuvent être prises dans une tâche sans interruption par le planificateur. Ainsi, une tâche consiste en une séquence ordonnée d'étapes à effectuer, qui donne le comportement total de la tâche.

Nous présentons aussi dans ce manuscrit une nouvelle approche pour résoudre le problème de fuite de l'information. Cette approche consiste à appliquer une procédure combinent des *Pré-processus* qui intervient sur un ensemble initial de tâches pour produire un nouveau ensemble de tâche. Les *Post-processus* interviennent sur la trace résultante de l'application d'un algorithme d'ordonnancement classique sur l'ensemble de tâche à planifier. Pour cela, on a proposé un ensemble d'algorithmes pour les deux phases de notre procédure afin de trouver la bonne combinaison pour réduire la fuite d'information. Pour mieux illustrer notre approche, on a présenté les résultats des expériences qu'on a réalisées en fin du Chapitre 6.

Abstract

Softwares become an important part of our daily life as they are now used in many heterogeneous devices in our daily life. These devices are dotted with a number of embedded systems that run in real-time , which means that they must react to external events. These systems are used in most domains of life, even the critical ones. That is why the safety of these systems is very important, and can be primordial.

The correctness of *Real-Time Systems* does not depend only on the correctness of their treatment results, but it also depends on the timings at which these results are given. There exist many methods that can be used to analyze the correctness of these kind of systems, but with the increasing of their complexity, the necessity to find new methods become an urgent requirement.

Today, a well-used class of verification methods are model-based techniques. These techniques describe the behavior of the system under consideration using mathematical formalisms, then using appropriate methods they give the possibility to evaluate the correctness of the system with respect to a set of properties.

In this manuscript we focus on using model-based techniques to develop new advanced scheduling techniques in order to analyze and validate the satisfiability of a number of properties on real-time systems. The main idea is to exploit scheduling theory to propose these new techniques. To do that, we propose a number of new models in order to verify the satisfiability of a number of properties as schedulability, energy consumption or information leakage.

Our first model treats the *Hierarchical Scheduling* problem, it consists in analyzing a number of properties as scheduling and response time of complex real-time systems in a tree manner. To do that, we propose a new model based on *Stochastic Times Automata* in order to represent complex behaviors of the real-time systems under consideration.

In a second time, we extend our first model in order to treat the energy consumption problem. To do that, we implement two algorithms that exploit ANOVA method in order to optimize the energy consumption of a multi-processor platform. Adding to that, we propose an adaptation of the CUSUM algorithm to handle any relevant variation in the probability of satisfying energy consumption properties.

Our last model uses the property studied, which is the information leakage, as a scheduling resource. For that, we propose a new model that quantifies the amount of information leaked by the execution of secured process on the shared memory. Using this information we propose a number of heuristic algorithms that provide new solution with less amount of resulting leakage.

Finally, we illustrate our frameworks for each model presented previously by a case-study and the results of our experiences.

Contents

Introduction

Softwares became an important part of our daily life as they are now used in many heterogeneous devices, such as our phones, our cars, our home appliances, etc. Modern cars for example are dotted with a number of embedded softwares, each handling a specific task, like braking, airbags or fuel injection. These embedded softwares are designed to run inside larger systems with various and heterogeneous hardware and limited resources.

Communication systems, mobile phones, medical systems, transport are using a vast amount of embedded software. The use of embedded softwares is motivated by the flexibility and the simplicity that these softwares can guarantee, and to minimize the cost. These embedded softwares are designed to run in specific environments with limited resources. This forces the developers to optimize the size, the cost, the power consumption, the reliability and the performance.

Cyber-Physical System (CPS) are softwares used to control physical systems. CPS are often embedded and run in real-time, which means that they must react to external events. A complex CPS can contain many real-time systems. Then a major challenge is to find an optimal policy to share system resources that guarantee the accomplishment of the missions of the CPS. The fact that these systems can be used in critical domains like medicine or transport requires a high level of safety for these systems.

Real-Time Systems by definition are processing information systems that have to respond to externally generated inputs, and they are called real-time because their response must respect strict timing constraints. Therefore, the correctness of these systems does not depend only on the correctness of their treatment results, but it also depends on the timings at which these results are given.

The main problem with using real-time systems is the difficulty to verify their timing constraints. A way to verify timing constraints can be to use *Scheduling* theory which is a strategy used in order to share the system resources between its different components. In addition to the timing constraints, other constraints should be taken into consideration, like energy consumption or security.

Several verification methods have been used in the last years, but with the increasing complexity of the embedded softwares these methods reach their limitation. That is why researchers are now focusing their works on finding new methods and formalisms capable of verifying the correctness of the most complex systems.

Today, a well-used class of verification methods are model-based techniques. These techniques describe the behavior of the system under consideration using mathematical formalisms, then using appropriate methods they give the possibility to evaluate the correctness of the system with respect to a set of properties.

The main work in this manuscript is about using model-based techniques for developing new advanced scheduling techniques in order to analyze and validate the satisfiability of a number of properties on real-time systems. The main idea is to exploit scheduling theory to propose new techniques in order to analyze different properties like energy consumption or information leakage.

Thesis Structure The remainder of the manuscript is composed as follows:

Chapter 1 presents an introduction about scheduling real-time systems. Scheduling is a decision-making process that is used in many domains such as manufacturing, industry, medicine and so on. It deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives.

The resources and the tasks can take many different forms. The resources may be machines in a workshop, runways at an airport, processing units in a computing environment, and so on. The tasks may be operations in a production process, take-offs and landings at an airport, executions of computer programs, and so on. Each task may have a certain priority level, an earliest possible starting time and a due date.

The objectives can also take many different forms. One objective may be the minimization of the completion time of the last task and another may be the minimization of the number of tasks completed after their respective due dates.

Scheduling, as a decision-making process, plays an important role in most manufacturing and production systems as well as in most information processing environments. It is also important in transportation and distribution settings and in other types of service industries.

We begin by presenting a number of techniques used for analyzing real-time systems. *Peer reviewing*, *testing*, *emulation* or *simulation* are used as methods to verify the correctness of real-time systems. Even if these methods proved their efficiency before, on-going technological advances make real-time systems become more and more complex and the

methods presented before show their limitation to handle these kinds of systems.

Another kind of methods can be used to verify the correctness of complex real-time systems, basing on the behavior of these systems a model is created and using appropriate algorithms and techniques the correctness of the system can be checked. In this chapter, we present general models that can be used in order to represent the behavior of the system under consideration.

Finally, we present in this chapter different types of real-time scheduling algorithms. Scheduling algorithms are categorized according to the policy used to assign resources to the different processes that need to be executed.

Chapter 2 presents the models used to describe the behavior of real-time systems. First, we define *transition systems* which are oriented graphs that can be used to describe the behavior of the system under consideration using *locations* to define the different states of the system and *edges* to describe the transition from a state to another.

Second, we introduce *timed automata* for specifying timing constraints. Timed automata are an extension of transition systems with timing aspects called *clocks*. Clocks are used to describe the timing constraints during the execution of real-time systems. In particular clocks control the amount of time that the system can stay in each state, and the timing constraints for a transition from a state to another. We also introduce extensions of timed automata for specifying scheduling mechanisms or energy concerns.

In this chapter, we present our first contribution that consists of a new stochastic model for stochastic tasks and a dispatcher in order to model the variation of execution time with respect to the computation logic. This representation is very important in order to model complex real-time systems behavior.

Finally, we present the model-based approach used in our work to specify scheduling problems with the formal models previously introduced. We first give a view about precedent works done in this field. Then we present the formal models of the components of the scheduling systems analyzed in our work.

Chapter 3 presents the languages used to express correctness properties of RTS and the techniques used to verify these properties. *Temporal logics* are formalisms used to express the timing constraints of the properties needed to be verified with a mathematically precise notation. Temporal logics express constraints over sequences of events.

After that, we present an automated verification method called *model checking (MC)* that we use to verify the satisfiability of properties described using the logic above on a given model. This method proceeds by exploring all possible executions of the model

in a brute-force manner and checks the satisfiability of the property under consideration on each execution. The result will be positive, i.e. the model satisfies the property, or negative, i.e. the model does not satisfy the property. If the result is negative, a counter example will be given and analyzed to explain the violation of the property, and to update the model or the system design to correct these problems. .

We also present an alternative to model checking called *statistical model checking (SMC)* that reasons about average scenarios to quantify the probability of satisfying a property. This method is used to handle the state explosion problem, i.e. the number of states needed to model the system accurately may easily exceeds the amount of available computer memory.

The last part of this chapter presents a high level language called Cinco that we used in our work to have a graphical description of the models under consideration. Cinco can be used to make a graphical representation of the models that describe the different components of the system. This graphical representation will be translated to formal models and then the different properties can be checked on these models using the model-checking tool Uppaal. The results of the formal verification can be parsed and the most relevant information can be graphically displayed.

Chapter 4 presents our first model for analyzing *Hierarchical Scheduling Systems (HSS)*. HSS are complex scheduling systems with multiple scheduling algorithms. We first introduce a new model-based compositional framework with stochastic real-time tasks in a HSS. This framework is designed with timed automata and probabilistic timed automata that constitute a model bank to describe HSS. In particular we introduce new probabilistic timed automata (PTA) models to instantiate stochastic tasks where task real-time attributes, such as deadline, execution time or period, are characterized by probabilities. This allows to design generic models that cover more cases of CPS.

Then we encapsulate this formal framework into Cinco, a generator for domain-specific modeling tools. Cinco allows to specify the features of a graphical interface in a compact meta-model language, and it generates automatically from this meta-model specification a domain specific analysis tool with a graphical interface.

Inside this analysis tool we can design the specifications of a hierarchical scheduling system and the properties it must satisfy. We can launch analysis of the properties, which generates automatically the timed automata models using the components of our model-bank, and it calls the tools Uppaal and Uppaal SMC to perform the analysis. This approach allows to completely hide the formal models being used from the system designer that can concentrate on the structure and the parameters of the hierarchical

scheduling system. Finally, we illustrate our framework with a case-study and the results of our experiences.

Chapter 5 presents our study of *Energy Consumption of Multi-Processor Scheduling Systems*. Many CPS are mission critical systems, it means that these systems have a specific mission to achieve with a limited amount of energy. Number of researches focused on analyzing these systems by verifying that the system can accomplish its mission only using its initial budget of energy.

To formalize multi-processor scheduling systems with energy resources, we extend formal models presented in the precedent chapter with information about energy consumption. After that, using Cinco we give a graphical representation of the formal model. The graphical representation consists of two layers. The first layer, *platform layer*, models the hardware platform with a scheduling system composed of real-time tasks and CPUs. The second layer, *application layer*, models the application that is composed of a set of actions.

In this chapter we present a new optimization technique for multi-processor scheduling system. It determines optimal mappings from tasks to processors in order to minimize the energy consumption of the system and/or response time using statistical tests (ANOVA and Tukey HSD). First, we determine all the possible schedulable mapping from tasks to processors. Using Uppaal SMC we evaluate the energy consumption and the response time of each schedulable mapping. After that, using statistical test ANOVA we determine if the means of the treatments are significantly different. If it is the case we use Tukey HSD to compare the means of every treatment to the means of every other treatment. Based on this comparison, we classify the precedent mapping according to their energy consumption and/or response time. Finally, we choose the appropriate mapping according to the property needed to be verified.

In this chapter also, we present a statistical algorithm for change detection called CUSUM. The principle is to monitor the evolution of a probability measure at successive positions during a single execution of the system. The algorithm then detects the position where the probability to satisfy the property changes significantly.

Chapter 6 presents a new model that handles the information leakage when processes with different security levels shared memory during their execution. Information leakage here means, the different pieces of information that can be left by a high level security process at different moments of its execution. Typically, this includes loading confidential information, such as encryption keys, medical data, and bank details, into memory for

use within high-security processes. These confidential pieces of information may be vital to the operation of the high-security processes, but must also be tightly controlled and not be leaked to low-security processes.

In this chapter we propose to treat confidentiality, measured by the resulting leakage of secure information, as a quantitative resource that the scheduler can exploit. This allows for a better quantification of the resulting leakage in different scenarios, as well as having a clear measure of the cost of different scheduling choices. Further, this allows for the creation of schedulers that can make better scheduling choices and also respect confidential information leakage constraints.

In this chapter we present also a new model that considers tasks to be composed of steps, each of which has an execution time, leakage value, and security level. Each one of these steps is implicitly an atomic sequence of actions that can be taken within a task without preemption by the scheduler. Thus a task consists of an ordered sequence of steps to be performed, that yields the total behavior of the task.

Chapter 1

Real-Time Scheduling

In this chapter, we give a brief introduction about real-time systems scheduling theory. Scheduling this kind of systems must take on account the timing constraints of these systems. Generally speaking, scheduling these systems consists in finding a schedule for the processes of the system such that all the processes respect their timing constraints. Solution will be considered true if all the processes respect their timing constraints. First, we introduce the main techniques for analyzing real-time system. Second, we present the main concepts of tasks and jobs used to design real-time scheduling systems. Finally, we present some examples of scheduling algorithms and we list the properties that must be satisfied.

1.1 Introduction

Cyber Physical System (CPS) are software-implemented control systems that control physical objects of the real world. The physical system observes the environment by means of its sensors. The software receives the information from the sensors and sends signals to actuators. The information sent by sensors can be periodic or irregular depending on the events happening in the environment. In all cases the system must react to these demands, and there will be a time bound for the response that must be respected. The software must be able to deal with the case where there is more than one treatment to do and where each one of these treatments have a time constraints.

CPS have to schedule the computation between all the received requests in order to satisfy each treatment to get response within its required time bound. The no respect of the treatment time bound by the software can have different consequences, in some cases it can have no negative consequence, in other cases it can have a few negative consequences that can be solved, but in other case it can have disastrous consequences.

The correctness of these systems is not only based on the correctness of the results given by the system, but on the time when those results are given too. That is why the timing constraints are as important as the correctness of the results given.

In this chapter we present basics about the model formalizing of the CPS systems used in our work. First we give a brief definition of the different methods used to analyze real-time systems, then we present the formalizing model used in our work. After that we present the different algorithms used to analyze scheduling problems. Finally, we present the different kinds of properties that can be analyzed on this model.

1.2 Analysis Techniques for Real-Time Systems

The criticality of the real-time systems requires efficient analysis methodology to verify the correctness of these systems. A number of existing techniques can be used to analyze these systems. *Peer reviewing* and *testing* are the most used verification techniques for the software systems in practice. For hardware analysis there exists other techniques like *emulation* or *simulation*.

Peer reviewing consists of the inspection of the software statically in order to find any problem, this inspection is done before the compilation of the software by a neutral engineer group, preferentially that has not been involved in the software development. The testing technique gives test values to the compiled software and observes the treatment results, basing on this observation weather the correctness of the system is satisfied or not.

The emulation technique is a kind of testing used to verify hardware systems, it consists of configuring the emulator to behave like the hardware under consideration and in the same way as the testing technique, it gives test values to the emulator and compares the generated output with the expected output to decide the correctness of the system under consideration. Simulation consists in constructing a model for the hardware that simulates this hardware. Simulation is like testing, but it is applied on a model. The main limitation of this method is that simulation gives a possible scenarios to the system execution, but the number of scenarios to be checked to get high confidence about the correctness of the system can be very high and cannot be accomplished by a simulator in a reasonable way.

Formal techniques are very used to analyze complex real-time systems, the general idea is to apply a number of mathematical techniques on a *Model* to verify the satisfiability of a number of *Properties*. The model is an abstraction describing the real-time system, it represents the behavior of the system respecting the time constraints. The properties

can be results that the system must produce or specific behaviors of the system.

In the next section, we will present a general model to describe different components of real-time systems.

1.3 Tasks and Jobs

In this section we present a formalization of a real-time system. As we mentioned before, real-time systems are such that the correctness do not depend only on the correctness of the results, but also on the time when these results are given. A real-time system is normally composed of a number of *tasks*, also called threads. Each task is designed to accomplish a specific work. Each time the task recurs it is called a *job*. The job is instantiated at a regular time or not depending on the task nature, the jobs are periodically produced if the task is periodic. According to the system architecture the Jobs are executed on one or different processors, but at given instant a processor is able to execute only one Job, processor here refers to a single core architecture.

We can divide the nature of the tasks on two categories, soft and hard tasks.

Definition 1.1 (Soft tasks)**.** *We designate soft tasks those for which meeting their time constraints is not necessary, this means it is acceptable that this category of tasks do not finish its work before its deadline.*

Definition 1.2 (Hard tasks)**.** *We designate hard tasks those for which meeting their time constraints is mandatory, this means it is not acceptable that this category of tasks do not finish its work before its deadline. The no respect of their time constraints can have disastrous consequences on the system.*

The events happening in the environment in which the real-time system performs are picked up by the system sensors. The system receives the information given by sensors and affects each treatment to a specific task. The tasks can be periodic or aperiodic depending on the nature of the treatment.

Definition 1.3 (Periodic tasks)**.** *We call periodic tasks those where the task is repeated after a fixed amount of time called Period P, they are called Time-triggered.*

A simple example of real-time systems with periodic tasks, is the traffic light system. Each light is turned on for a fixed amount of time, that can be modeled as the *execution time* E before it is turned off. The amount of time between every two activations of the light can be modeled as the period.

Definition 1.4 (Aperiodic tasks)**.** *We call aperiodic tasks those which have an irregular arrival time, they are called Event-triggered.*

A simple example of an aperiodic task is the air-bag activation system, if there is a collision the sensor sends the signal to the system, the system takes the power of the collision and evaluates the necessity to activate the air-bag or not, if yes the system sends a signal to the actuator to activate the air-bag, all these treatments must be done with a time constraint to guarantee the safety of the conductor.

It is a great difficulty to represent such aperiodic tasks. A solution is to model these kinds of tasks using *sporadic tasks.*

Definition 1.5 (Sporadic tasks)**.** *Sporadic tasks are such that their period can be modeled using different probability distributions in order to have a bound on the task arrival time.*

Each task or job is characterized by the following parameters, the release time, execution time and the deadline.

Period P is the minimal amount of time between two consecutive jobs. For periodic task, period P is a fixed value. For aperiodic tasks, period P cannot be defined because their arrival time are irregular. For sporadic tasks, period P is a bound on the task arrival time that can be modeled using different probability distributions.

Release Time is the date at which a job is instantiated. If the job is instantiated from periodic tasks, the release time R of the job is computed by the following formula: $R = (n-1) * P$, where n is the number of the job and P is the period of the task. If the job is instantiated from aperiodic task, the release time R of the job cannot be predicted because the release time of the task is irregular.

Execution time E is the time needed by each task or job to finish its execution. We can define several types of execution time: best case execution time BE, is the minimum time needed by the task or job to finish its execution; worst case execution time WE, is the maximum time needed by the task to finish its execution.

Deadline D is the time at which the task or job must finish its execution relative to the release time, we call it *relative deadline.*

1.4 Scheduling Algorithms

For a given set of jobs, the scheduling problem is to find an order for which all the jobs are executed satisfying all their time constraints. Usually, each job is parametrized by its release time R, its absolute deadline A, its execution time E and resource requirements. Each job execution may or may not be interrupted (preemptivity). The order given by the scheduler must be respected, it means that each job cannot be executed before the complete execution of its predecessors.

We can distinguish two types of scheduling techniques, static techniques called off-line techniques and dynamic techniques called on-line techniques.

Off-line Scheduling Algorithms We designed an off-line scheduling algorithms systems that have an entire knowledge about the set of jobs scheduled, the scheduler have knowledge about each task, its release time, execution time and deadline before beginning the execution. This type of algorithms is mostly used when the system contains periodic tasks only.

On-line Scheduling Algorithms Contrary to the off-line algorithms, the on-line algorithms have a partial knowledge about the set of jobs scheduled, the scheduler receives requests at any time and must answer these requests. This kind of algorithms is suited to handle the scheduling of aperiodic tasks, because the aperiodic tasks produce jobs at an irregular time. That is why the scheduler cannot have an entire knowledge about the arrival time of the aperiodic tasks.

Scheduling algorithms can also be classified according to the policy used for the classification of the ready jobs for execution. The jobs are classified according to their criticality, the most critical job must be executed in first and so on. This criticality can be expressed using a *priority* value where the job with the highest priority value will be executed first. The scheduling algorithms can be classified into two classes *fixed priority* and *dynamic priority* scheduling algorithms.

Fixed Priority Algorithms Fixed priority algorithms are those where the priorities of the tasks do not change during the execution time. The priority of each task is fixed at the design time according to the criticality of the task, and this priority will not be affected during the execution.

A good example of the fixed priority algorithms is the *Rate Monotonic Algorithm (RM)*, this algorithm assigns the highest priority to the task with the smallest period P.

Dynamic Priority Algorithms Dynamic priority algorithms are those where the priorities of the tasks are calculated on the fly during the execution time of the system. The scheduler calculates the priorities depending on the tasks parameters.

An example of the dynamic priority algorithm is the *Earliest Deadline First (EDF)*. EDF determines the priority of jobs according to their absolute deadline, at any given point of time, out of the currently available jobs, the job with the earliest absolute deadline is scheduled first.

In some cases, the real-time system requires that some specific tasks must be executed in priority over the other tasks. For that, the scheduler can stop the execution of a lower priority task in order to promote another high priority task. The interruption action called *preemption* can be used by *preemptive scheduling algorithms*. Other algorithms are *non-preemptive scheduling algorithms*.

Preemptive Scheduling Algorithms We call *Preemptive Scheduling Algorithms* those accepting that a job in execution can be preempted (stopped) by another job with a higher priority. The precedent job is returned to the ready queue in order to finish its execution.

Non-Preemptive Scheduling Algorithms Contrary to the precedent kind of algorithms, the non-preemptive ones do not accept that a running job can be preempted. The running job keeps all the necessary resources until finishing its execution even if there is another job with a higher priority that needs to be executed.

The preemptivity of the scheduling algorithms depends on the nature of the job and the priority of its execution, some jobs must be executed rapidly because of their criticality. That is why these jobs must have a higher priority value. The priority of each job can be fixed by the system constructor in advance, or can be calculated according to the tasks attributes. Mixed systems are also possible, the system can accept preemptivity but in the same time can guarantee for some jobs to be executed without be preempted.

Example Let consider the two periodic tasks $\mathcal{T}_1(3,3,2)$ and $\mathcal{T}_2(6,6,2)$, where $\mathcal{T}_1$ has a period of 3 units time and must finish its computation before 3 units time with an execution time of 2 units time. $\mathcal{T}_1$ has a period of 6 units time and must finish its computation before 6 units time with an execution time of 2 units time. Let consider that $\mathcal{T}_1$ has a higher priority than $\mathcal{T}_2$.

Figure 1.1 represents a possible execution of $\mathcal{T}_1$, $\mathcal{T}_2$ on a single processor platform using a non-preemptive scheduling algorithm. Let consider that the release time of both tasks is zero, the task $\mathcal{T}_1$ is executed first because its priority is higher. Task $\mathcal{T}_1$ finishes

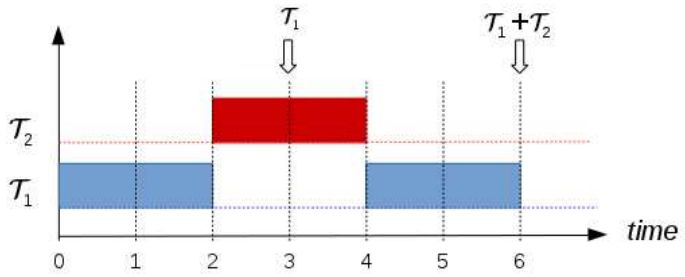


Figure 1.1 – Example of scheduling two tasks using a non-preemptive Scheduling algorithm

its execution after 2 time units. Then the task $\mathcal{T}_2$ begins its execution. A new job of task $\mathcal{T}_1$ is instantiated at time 3, but at this moment the processor is not available because the task $\mathcal{T}_2$ is running. Task $\mathcal{T}_1$ must wait until task $\mathcal{T}_2$ finishes its execution. Even if we know that $\mathcal{T}_1$ has a higher priority than $\mathcal{T}_2$, the non-preemptive algorithm protects $\mathcal{T}_2$ from any interruption before finishing its execution.

Figure 1.2 represents another possible execution of the precedent model using a preemptive scheduling algorithm. Task $\mathcal{T}_1$ is executed first because its priority is higher. Task $\mathcal{T}_1$ finishes its execution after 2 time units. Then task $\mathcal{T}_2$ begins its execution. A new job of task $\mathcal{T}_1$ is instantiated at time 3, this time the algorithm interrupts the execution of the running job since a higher priority task needs to be executed. The job instantiated by task $\mathcal{T}_1$ at time 3 will interrupt the execution of $\mathcal{T}_2$ in order to be executed. Task $\mathcal{T}_2$ will resume its execution when $\mathcal{T}_1$ finishes its execution.

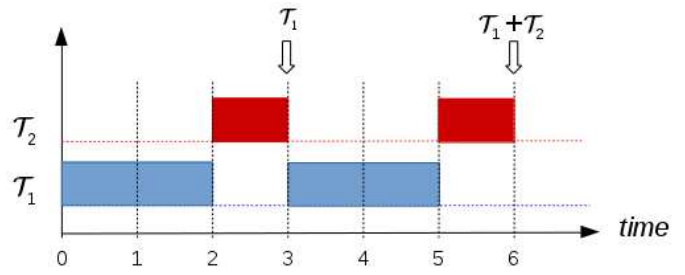


Figure 1.2 – Preemptive Scheduling algorithm

1.5 Scheduling Properties

The principal objective of analyzing real-time systems is to verify that these systems accomplish their designated work correctly while respecting time constraints. The correctness of the results, and the time constraints are modeled as properties. These properties can be classified generally on two main classes, safety and liveness properties.

Safety Properties *Safety Properties* are those that declare that "something bad never happens". Generally, safety requirements means that the system does not have any deadlock or any similar state that can crash the system.

Deadlock is considered as a safety property. We call deadlock the states where the system cannot progress, like interminable loops or terminal states. In such systems terminal states are undesirable and mostly represent a design error. A simple example of deadlock scenario occurs when several components wait for the progress of the other component.

A way to verify safety properties is to search using specific algorithms in the set of states of the system about any undesirable state like state where the system do not finish its work but in the same time the system cannot progress any more. This search should outputs a trace with a counter example leading to the deadlock problem.

Another important safety property is the *schedulability property*, this property consists of verifying that each component of the system finishes its treatment respecting its time constraints.

Example An example of safety property, the mutual exclusion problem. A simple logical representation of this problem is $\neg(\mathcal{T}_0 \wedge \mathcal{T}_1)$, it means that the Task $\mathcal{T}_0$ and Task $\mathcal{T}_1$ cannot be in the critical section at the same time.

Liveness Properties Contrary to safety properties, liveness properties attest that "something good will eventually happen", safety properties are violated within a finite time, while liveness properties are violated in an infinite time. For example the liveness property in the mutual exclusion problem can be presented as follow:

- Each task must enter its critical section.
- Each task must enter its critical section each time it is needed.
- Each waiting task must enter its critical section in some time in the future.

From what we see above, the fact that real-time systems are today used in many critical domains of our live, their correctness becomes a primordial necessity. Over the years, many methods have been used to analyze the correctness of these kinds of systems. In this thesis, we focus our work on model-based methods. These methods design a model to describe the behavior of the real-time system under consideration, and apply different techniques to analyze the correctness of this system. The next chapter describes a well-used model to describe this kind of systems.

The next chapter presents formal models used in our work to model, verifies and validates real-time systems.

Chapter 2

Formal Models for Scheduling Real-Time Systems

This chapter introduces formal methods in order to model, verify and validate real-time systems using discrete event models extended with time. This formal model gives us the possibility to represent the behavior of complex systems with their timing constraints. This representation can be exploited by formal methods in order to verify the satisfaction of specific properties on the system under consideration. In this chapter we present the transition system theory to represent the discrete behavior of the system. Then we add timing constraints to represent real-time systems, and stochastic constraints to represent systems with complex behaviors or real-time systems. To do that, we use the time automata. Finally we present the model-based method that we use in our work, and we present some models of scheduling components.

Key Contributions In this chapter we present new stochastic models for stochastic tasks and dispatcher in order to model the variation of execution time with respect to the computation logics. This representation is very important in order to model complex real-time systems behavior.

2.1 Introduction

Today, formal methods became one of the most used techniques for analyzing real-time systems. The possibility of analyzing complex systems make the use of this methods very desirable in the design process of the real-time systems. During the last two decades, research in formal methods has led to the development of some very promising verification

techniques that facilitate the early detection of defects. These techniques are accompanied by powerful software tools that can be used to automate various verification steps.

Model-based verification techniques are based on models describing the possible behavior of the system under consideration in a mathematically precise and unambiguous manner. Such problems are usually only discovered at a much late stage of the design. The system models are accompanied by algorithms that systematically explore all states of the system model in order to detect any undesirable behavior.

This chapter first introduces *transition systems*, a standard class of models to represent systems under consideration. Different aspects for modeling concurrent systems are treated, ranging from the simple case, in which processes run completely autonomously to more realistic settings, where processes communicate in some ways.

Then to model real-time systems we need a model that takes into account timing aspects. For that we introduce *timed automata*. That is an extension of transition systems with *clocks* to handle timing aspects. Finally, we present a model-based framework based on timed automata to describe scheduling systems.

2.2 Transition Systems

2.2.1 Definitions

Transition systems can be used to model the behavior of real-time systems. Transition systems are oriented graphs containing *nodes* and *edges*. The nodes represent the state (or location) of the system in a specific moment of its behavior. The edges represent the transitions from a state to another state. As an example the state of the traffic light is the current color of the light and the edge is the switch from one color to another.

There exist different types of transition systems. In our work we decide to use transition systems with *action names* on the edges and *atomic propositions* on the locations. The action names on the edges describe communication mechanisms of the system processes controlling the transition from one location to another. The atomic propositions express different characteristics of the system at this moment.

Definition 2.1 (Transition System)**.** *A transition system is a tuple* (Loc, Act, E, I, AP, L) *where*

- Loc *is a set of locations;*
- Act *is a set of action;*
- $E \in (S \times AP \times S)$ *is a transition relation;*

- $I, I \subseteq S$ *is a set of initial states;*
- AP *is a set of atomic proposition;*
- $L : S \to 2^{AP}$ *is a labelling function.*

The transition system starts at some initial location l_0, where $l_0 \in I$, and progresses according to its set of transitions E. The transition system moves from location l to another l' and executes an action $\alpha \in Act$, which we write $l \xrightarrow{\alpha} l'$. In case the location l has more than one outgoing edge, it is important to precise that the next edge is chosen in a non-deterministic manner. In the following we denote actions using Greek alphabet (such as α or β).

Each location in the transition system contains one or more atomic proposition $a \in AP$. These propositions express some knowledge about the system at this state. The atomic propositions will be denoted using letters from the beginning of the alphabet (such as a, b, c ..). Examples of atomic propositions are "light is green" or "x equals 2". The labeling function L relates a set of atomic propositions $L(l)$ to a location l. The labeling function stands for the atomic propositions $a \in AP$ satisfied by the location l.

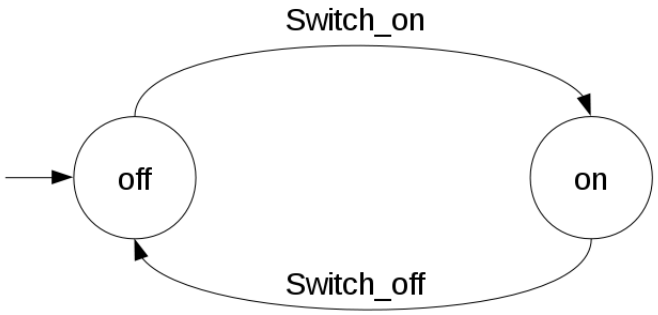


Figure 2.1 – Light Switch Transition System *TA*

Example Let consider the example presented in Figure 2.1. It models a simple design of a light switch with a transition system. The locations of the model are designed by circles and the transitions are designed by labeled edges. The name of each location is written inside the circle, the initial locations are those having an incoming edge without a source.

The set of locations is $Loc = \{on, off\}$. The set of initial locations consist of a single location $I = \{OFF\}$. The set of actions is $Act = \{Switch\_on, Switch\_off\}$ The action $Switch\_on$ moves the system from the initial location off to the location on,

and the action $Switch\_off$ moves the system from the location on to the location off. Examples of transitions are:

$$off \xrightarrow{Switch\_on} on \qquad \text{and} \qquad on \xrightarrow{Switch\_off} off$$

Definition 2.2 (α_*successors* and α_*predecessors*)**.** *Let consider $TS = \{Loc, Act, E, I, AP\}$ a transition system. For $l \in Loc$ and $\alpha \in Act$, the set of α_ successors of l is defined as follow:*

$Post(l, \alpha) = \{l' \in Loc \mid l \xrightarrow{\alpha} l'\}$. *To generalize we write* $Post(l) = \bigcup_{\alpha \in Act} Post(l, \alpha)$

The set of α_predecessors is defined as follow:

$Pre(l, \alpha) = \{l' \in Loc \mid l' \xrightarrow{\alpha} l\}$. *To generalize we write* $Pre(l) = \bigcup_{\alpha \in Act} Pre(l, \alpha)$

Each location l' in the α_ successors (resp. α_predecessors) set is a direct successor (resp. predecessor) of the location l.

Definition 2.3 (Terminal Location)**.** *Terminal locations, also called blocking locations, are locations without any outgoing transition. When a transition system reaches a terminal location the system execution terminates.*

It is important to precise that terminal locations are used to represent the ending of the system running. In some type of systems, this kind of locations is undesirable.

2.2.2 Paths and Traces

Let consider $TS = (Loc, Act, E, I, AP)$ a transition system, the executions (or paths) of the transition system define its possible behaviors.

Definition 2.4 (Path Fragment)**.** *A sequence of locations, $\pi = l_0 l_1 l_2 ... l_n$ (when the sequence is finite) or $\pi = l_0 l_1 l_2 ...$ (when the sequence is infinite) is called:*

- A *path fragment* if $\forall i$, $l_{i+1} \in Post(l_i)$, it means for each location l_i in the path the next location l_{i+1} is a direct successor of the state l_i.
- *Initial path fragment* if the sequence is a path fragment and the first location of the sequence l_0 is an initial location, i.e $l_0 \in I$.
- *Maximal path fragment* if the sequence is a path fragment and the last location of the sequence do not have a direct successor, i.e $Post(l_n) = \emptyset$ or the sequence π is infinite.

Definition 2.5 (Path). *A sequence of locations is called a path if the sequence is an initial and maximal path fragment.*

As an example π = off on off on... is a path of the transition system describing the behavior of the light switch example.

Let consider a transition system *TS*. Let *Paths(TS)* denote the set of all paths in *TS*.

Definition 2.6 (Trace). *Let TS be a transition system with no terminal locations, it means that all its paths are infinite. If $\pi = l_0 l_1 l_2 ...$ is an infinite path, its trace is defined as follow:*

$$Trace(\pi) = L(l_0)L(l_1)L(l_2)...$$

We denote by *Traces(TS)* the set of all traces of the transition system (TS), it is defined by

$$Traces(TS) = \{Trace(\pi), \pi \in Paths(TS)\}$$

2.3 Timed Automata

In the precedent section, we have presented transition systems as a way to model the behavior of real-time systems. But until now, we did not present how to model the timing aspects of these systems, that is, information about residence time in a state or the possibility of taking a transition within a timing interval. These information give us the possibility to verify the satisfiability of the timing constraints of the real-time system under consideration.

As a modeling formalism for real-time systems, the notion of *timed automata* has been developed, an extension of transition systems with clock variables that measure the elapse of time. This model includes means to impose constraints on the residence times of states, and on the timing of actions.

2.3.1 Clocks and Clock Constraint

Timed automata are used to model the behavior of time-critical systems, time is a continuous entity. That is why to express the timing information it uses real-valued variables called *clocks*. All clocks in a system progress at the same rate. The only operations possible on a clock are reading the value of the clock and resetting the clock to zero. Intuitively, a clock represents the amount of time elapsed since the last reset of the clock.

In timed automata we reason about timing aspects in an abstract way as a sequencing of events.

To express the variations of the clocks values, we use a valuation function $v : C \to \mathbb{R}^+$ that assigns to each clock $c \in C$ a non-negative value $v(c)$. For an element t of $\mathbb{R}^+$ and a subset $x \subseteq C$, the valuations $v + t$ and $v[x \leftarrow 0]$ are defined as follow:

$$(v + t)(c) = v(c) + t \text{ for each clock } c \in C$$

$$v[x \leftarrow 0](c) = \begin{cases} 0 \text{ if } c \in x \\ v(c) \text{ otherwise} \end{cases}$$

To express conditions over clocks, we use clock constraints. Clock constraints can be used on location and transition. In the first case, it is called a location *Invariant*. The location invariant represents the time allowed to the system to stay in this location. When the invariant does not hold, the location must be left. In the second case, it is called a *Guard*. A transition is available as long as the guard holds. When the guard evaluates to false, the transition cannot be taken.

Definition 2.7 (Clock Constraint)**.** *A clock constraint over a set of clocks C can be written according to the following grammar:*

$$g ::= c < k \mid c \leqslant k \mid c > k \mid c \geqslant k \mid g \wedge g$$

where $k \in \mathbb{N}$, and $c \in C$. We denote $CC(c)$ the set of clock constraints over the set C. We write $v \vDash \varphi$ when valuation v satisfies the clock constraint φ.

Note that:

- A clock constraint can be written in an abbreviated mode, i.e. $(c \geqslant k_1) \wedge (c \leqslant k_2)$ can be written as $c \in [k_1, k_2]$, where $k_1, k_2 \in \mathbb{N}$;
- Clock difference constraints as $c_1 - c_2 \geqslant k$ can be added using a more complex theory, in this work we focus on atomic clock constraints, without any difference.

2.3.2 Syntax and Semantics of Timed Automata

A timed automaton is a transition system extended with a finite set of real-valued clock variables and clock constraints.

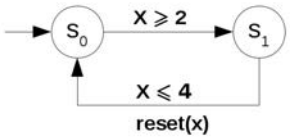
Definition 2.8 (Timed Automaton). *A timed automaton is a tuple* $TA = (Loc, Act, C, E, I, Inv, AP, L)$ *where:*

- Loc *is a set of locations;*
- $Act = Act_i \uplus Act_o \uplus \{\varrho\}$ *is the set of actions, where* Act_i *is a set of input actions,* Act_o *is a set of output actions and* ϱ *is an internal action;*
- C *is a set of clocks;*
- $E \subseteq S \times CC(C) \times Act \times 2^C \times S$ *is a set of edges.*
- $I \subseteq S$ *is a set of initial locations;*
- $Inv : S \to CC(C)$ *is a function assigning invariants to locations;*
- AP *is a set of atomic propositions;*
- L *is a set of labelling.*

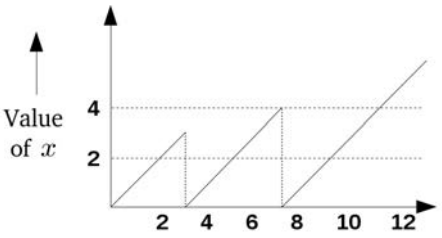
Edges are labeled with the tuple (g, α, D), where $g \in CC(C)$ represents the guard that must hold to enable the transition, $\alpha \in Act$ is an action and $D \in 2^C$ is the set of clocks that must be reset to zero when the transition is taken. Intuitively, $l \xrightarrow{g:\alpha,D} l'$ means that the timed automaton can move from location l to location l' when the guard g holds. When moving from location l to location l', each clock in the set D will be reset to zero and the action α is carried out. The function Inv assigns to each location a location invariant that indicates the amount of time that the timed automaton can stay in the designated location.

To represent timed automaton we adopt the drawing conventions for transition systems. Invariants are indicated inside locations and are omitted when equal true. Edges are labeled with the guards, the action, and the set of clocks to be reset. Empty sets of clocks are often omitted. The same applies to clock constraints that are constantly true. The reset of set D of clocks is sometimes indicated by $reset(D)$. If the actions are irrelevant, they are omitted.

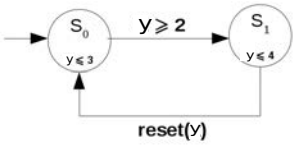
Example Figure 2.2 describes a simple timed automaton with a single clock x and some possible evolution of this clock through time. In Figure 2.2a, we can distinguish two guards, the first guard $x \geqslant 2$ means that the system cannot move from location l_0 to location l_1 before (2) time units, the second guard $x \leqslant 4$ means that the system can move from location l_1 to location l_0 if the clock x is less or equal to (4) time units. When the system takes the transition from location l_1 to location l_0 the clock x is reset to zero.



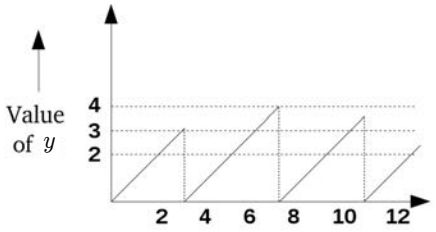
(a)



(b)



(c)



(d)

Figure 2.2 – Examples of timed automata with a single clock and one example of the evolution of their clock over time.

Figure 2.2b describes the evolution of the clock x through time. The system do not have any restriction about the time passed in each location, that is why if the system stays in one of the two locations more than (4) time units, then the system cannot take anymore the transition from location l_1 to the location l_0 due to the restriction on the clock x.

In Figure 2.2c, we add two invariants, the first invariant $y \leqslant 3$ in location l_0 to ensure that the system cannot stay more than (3) time units in location l_0, the second invariant $y \leqslant 4$ means that the system must move from location l_1 to location l_0 before (4) time units. These two invariants avoid that the system is blocked in any location.

Operational Semantics The semantics of a timed automaton are defined as a transition system where a state or configuration consists of the current location and the current values of clocks. There are two types of transitions between states. The automaton may either delay for some time (a delay transition), or follow an enabled edge (an action transition).

Definition 2.9 (Operational Semantics). *The semantics of a timed automaton is a transition system (also known as a timed transition system) where states are pairs (l, u), and transitions are defined by the rules:*

- $\langle l, u\rangle \xrightarrow{d} \langle l, u+d\rangle$ *if* $u \in I(s)$ *and* $(u+d) \in I(l)$ *for a non-negative real* $d \in \mathbb{R}^+$;
- $\langle l, u\rangle \xrightarrow{a} \langle l', u'\rangle$ *if* $l \xrightarrow{g:a,r} l', u \in g, u' = [r \mapsto 0]u$ *and* $u' \in I(l')$.

2.3.3 Parallel Composition of Timed Automata

A practical manner to model complex systems consists of using parallel composition of timed automata. This makes it possible to model time-critical systems in a compositional manner. We consider a parallel composition operator, denoted $\|$.

Let consider the two following timed automata $TA^1 = (Loc^1, Act^1, C^1, E^1, I^1, Inv^1, AP^1, L^1)$ and $TA^2 = (Loc^2, Act^2, C^2, E^2, I^2, Inv^2, AP^2, L^2)$, with $Act_0^1 \cap Act_0^2 = \emptyset$. The timed automaton $TA = TA^1 \parallel TA^2$ is a tuple $TA = (Loc, Act, C, E, I, Inv, AP, L)$ defined as follow:

- $Loc = Loc^1 \times Loc^2$ is the set of locations;
- $Act = Act_i \uplus Act_o$, where $Act_i = Act_i^1 \backslash Act_o^2 \cup Act_i^2 \backslash Act_o^1$ and $Act_o = Act_0^1 \cup Act_o^2$ is the set of actions;
- $C = C^1 \uplus C^2$ is the set of clocks;
- $I = I_1 \cup I_2$ is the set of initial locations;
- $Inv(s^1, s^2) = Inv_1(s^1) \wedge Inv_2(s^2)$ is the set of invariants for the new location (s^1, s^2);
- $L(\langle l_1, l_2\rangle) = L(l1) \cup L(l_2)$ is the set of labels

The set of the edges is defined as follow:

- If $(l^1, a, \alpha^1, c^1, l'^1) \in E^1$ with $a \in Act^1 \backslash Act^2$ then for each location $l^2 \in Loc^2((l^1, l^2), a, \alpha^1, c^1, (l'^1, l^2)) \in E$;
- If $(l^2, a, \alpha^2, c^2, l'^2) \in E^2$ with $a \in Act^2 \backslash Act^1$ then for each location $l^1 \in Loc^1((l^1, l^2), a, \alpha^2, c^2, (l^1, l'^2)) \in E$;
- If $(l^1, a, \alpha^1, c^1, l'^1) \in E^1$ and $(l^2, a, \alpha^2, c^2, l'^2) \in E^2$ with $a \in Act^1 \cap Act^2$ then $((l^1, l^2), a, \alpha^1 \wedge \alpha^2, c^1 \cup c2, (l'^1, l'^2)) \in E$;

The location invariant of a composite location is simply the conjunction of the location invariants of its components. For $\alpha \in Act$, the guard of the synchronized transition is defined by the conjunction of the guards of the transitions in the initial timed automata.

That implies that each action in Act can only be taken if it is true in both timed automata. The clocks that are reset in the initial automata are all reset. The operator $\|$ is associative for a fixed set Act.

Example Parallel composition can be done by synchronizing inputs and outputs in a broadcast manner. This means that when an TA executes one output, all those TA that can receive it must be synchronized. We denote input actions with a channel name followed by **?** and output actions with the channel name followed by **!**.

2.4 Hybrid Time Automata

Hybrid automata [Hen00] are an extension of timed automata that extends the dynamic of clocks with ordinary differential equations. Let X be a set of continuous variables. As for clocks, a variable valuation is a function $\nu : X \to \mathbb{R}$. We write $\mathbb{R}^X$ for the set of valuations over X. Valuations over X evolve according to delay functions $F : \mathbb{R}_{\geq 0} \times \mathbb{R}^X \to \mathbb{R}^X$, where for a delay d and a valuation ν, $F(d, \nu)$ is a the new valuation. Delay functions are assumed to be time additive ($F(d_1, F(d_2, \nu)) = F(d_1 + d_2, \nu)$).

Definition 2.10. *A hybrid automaton (HA) is a tuple* $\mathcal{H} = (Loc, I, C, Act, E, F, Inv)$.

- *Loc is a finite set of locations;*
- $I \in L$ *is a set of initial locations;*
- C *is a finite set of continuous variables.*
- $Act = Act_i \uplus Act_o \uplus \{\varrho\}$ *is a finite set of actions partitioned into inputs* (Act_i)*, outputs* (Act_o) *or internal (labelled with* ϱ*).*
- E *is a finite set of edges of the form* (l, g, a, ϕ, l')*, where* l *and* l' *are locations (resp. the* source *and the* destination*),* g *is a predicate on* $\mathbb{R}^X$ *(called the* guard*),* $a \in Act$ *is an action label,* ϕ *is a binary relation on* $\mathbb{R}^X$ *that defines the* clock updates*.*
- *For each location* l*,* $F(l)$ *is a delay function;*
- $Inv(l)$ *is an* invariant *predicate.*

The semantics of $\mathcal{H}$ are a transition system, whose states are pairs $(l, \nu) \in L \times \mathbb{R}^X$ with $\nu \models I(l)$, and whose transitions are either, delay transitions $(l, \nu) \xrightarrow{d} (l, \nu')$ with $d \in \mathbb{R}_{\geq 0}$ and $\nu' = F(l)(d, \nu)$, or, discrete transitions $(l, \nu) \xrightarrow{a} (l', \nu')$ if there is an edge $(l, g, a, \phi, l') \in E$, such that $\nu \models g$ and $\phi(\nu, \nu')$. An execution of $\mathcal{H}$ is an alternating

sequence of delay and discrete transitions. As timed automata, HA can be combined in networks of HA via parallel composition.

The above definition deliberately left open the syntax for the delay functions F, the guards g, the update predicates ϕ and the invariants I. Their concrete definition depends on the class of hybrid automata that is considered.

Timed automata (*TA*) [AD94] is the most restrictive class of HA we use as presented in Section. 2.3. This means that for any clock $x \in X$, the delay functions $F(l)$ defines an implicit rate $x' = 1$.

Stopwatch automata (*SWA*) [CL00] extend TA by allowing to stop and resume clocks. The rates of the variables are therefore either $x' = 1$ (for running clocks) or $x' = 0$ (for stopped clocks).

Priced timed automata (*PTA*) [BFH$^+$01, ALTP04] allow the continuous variables to be either clocks as in TA, or cost-variables with a rate $x' = e$, where e is an expression that only depends on the discrete part of the current state. These cost-variables cannot be used in guards, updates and invariants of the *PTA*, which implies that they cannot affect the behavior of the model.

Hybrid automata (*HA*) is the most general case. It allows to use ordinary differential equations to define delay functions F and invariants I.

2.4.1 Modeling Hybrid Automata in Uppaal

Uppaal is one the most famous tools for modeling and analyzing timed automata and their hybrid extensions. The tool has been developed for more than 20 years by a collaboration between Uppsala University in Sweden and Aalborg University in Denmark. It allows to design models that belong to one of the four classes of hybrid automata presented previously. It additionally provides many syntactic constructions that help the design of complex models. In the following of the thesis we will heavily use these constructions for designing models of scheduling systems. We will succinctly explain the syntax and semantics of our models, but we cannot present here the full syntax of Uppaal models, and therefore we redirect the reader to the documentation of the tool (at `http://www.uppaal.org/`) for a more precise description. Some of the main capabilities offered by the tool are:

- **Data variables.** In addition to clocks, the tool allows to use data variables (integer, float, arrays, and structures). They can be updated during transitions, and tested in guards or invariants. Synchronization channels can also be defined in arrays.
- **Functions.** The tool allows to write functions using a syntax similar to the C language. They can be used in guards, invariants, and updates of variables. When synchronizing transitions on a channel, the update functions of all the transitions involved in the synchronization are performed.
- **Templates automata.** Hybrid automata can be defined as templates with input parameters. This allows to instantiate several automata in a model using the same template (for instance several tasks with different parameters).

In this thesis we will show several examples of hybrid automata by using screen captures from automata designed in Uppaal. In these figures the transitions have guards in green, synchronization actions in light blue (τ actions are omitted), updates in blue. Locations have a name and an invariant (possibly with clock rates) in purple.

Example We present in Figure 2.3 four examples of the different types of models. All these models implement a simple real-time task with various functionalities, depending on the type of model being used.

The model in Figure 2.3a implements a task with no preemption using a timed automata. It has a clock x to measure the length of the period and a clock y to measure the execution time. It starts its execution when receiving the event schedule?. It sends an event done! as soon as the clock y has reached the best case execution time (bcet) and before reaching the worst case execution time (wcet). Otherwise it goes to the location MissingDeadline with an internal transition when the clock exceeds the deadline. Finally it returns to location JobDone to wait for the next execution round and it sends the signal ready! to the scheduler.

The model in Figure 2.3b implements a preemptive task using a stopwatch automaton. It refines the previous model with a stopwatch on clock y: the clock is stopped in location Ready (denoted y'=0), otherwise it is assumed that its execution rate is 1. The task can be preempted by the scheduler when it receives the signal not_schedule?, in which case it returns to location Ready.

The model in Figure 2.3c additionally computes the energy consumed by the running task using a priced timed automaton with a variable e to measure the energy. The energy can only increase in location Executing at a rate given by the constant POWER.

Finally, the model in Figure 2.3d is additionally aware of the frequency FREQ at which the processor is running. This frequency defines the rate at which the task executes by setting y'==FREQ in the invariant of location Executing.

2.4.2 Stochastic Hybrid Automata

Hybrid automata (and their sub-classes) may be used with a stochastic semantics [DLL+11, DDL+12] that refines all non-deterministic choices with probability distributions. This impacts the choice of delay, output and next state. For each state $s = (l, \nu)$ of an HA $\mathcal{H}$ we assume there exists the following probability distributions:

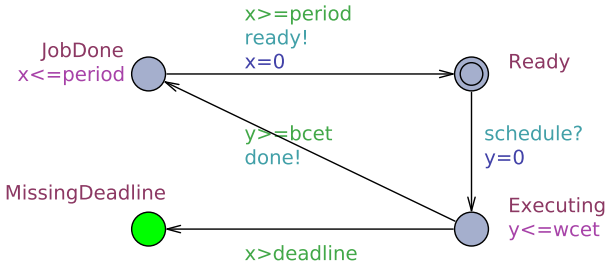
- the delay density function μ_s over delays in $\mathbb{R}_{\geq 0}$, that defines when the component will perform an output,
- the output probability function λ_s, that assigns probabilities to each available outputs $o \in \Sigma_o$,
- the next-state density function η_s^a, that provides stochastic information on the next state $s' = (l', \nu') \in \mathbb{R}^X$ given an action a.

Adding stochastic information Stochastic hybrid automata are analyzed with Uppaal SMC. Without additional information the tool is also able to run classical TA, SWA, *PTA* or HA with a stochastic semantics, that apply uniform distributions to delays in states with bounded delay, to outputs and to next states. Additionally the user can provide the rate of an exponential distribution for each location with unbounded delay, and discrete probability distributions between different outputs and the next states.

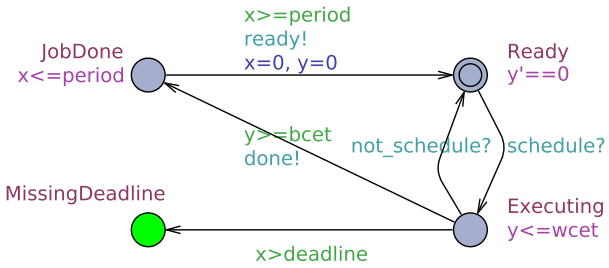
These distributions can be sampled from executions or simulations of the system, or set as requirements from the specifications. For instance in avionics, display components have a lower criticality. They can include sporadic tasks generated by user requests. In that case, average user demand will be efficiently modeled with a probability distribution. Similarly, timing executions may vary due to the content being displayed and can be measured from the system.

If analyzed with Uppaal model-checker, stochastic information from a stochastic hybrid automaton is discarded to consider only the underlying non-deterministic model.

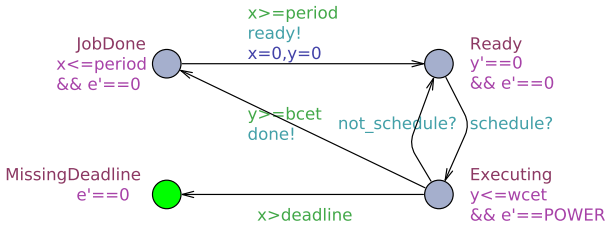
Example Stochastic hybrid automata with discrete probability distributions are useful to initialize the parameters of a model with random values, e.g., to specify that the period or the deadline of a task depends on some random information. They can be designed



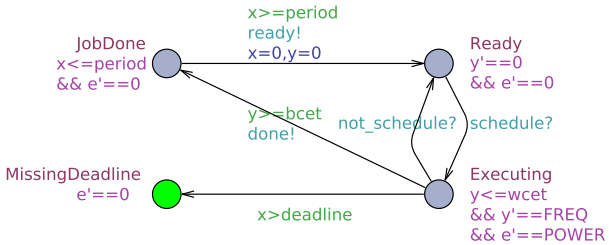
(a) Timed automata



(b) Stopwatch automata



(c) Priced timed automata



(d) Hybrid automata

Figure 2.3 – Implementations of a simple real-time task with timed, stopwatch, priced and hybrid automata

in Uppaal using a special node with one incoming transition (possibly with guard), and several outgoing transitions (displayed with dashed lines) that perform different updates and reach different locations, each associated to a probability weight (the probability of the transition is then the ratio of the probability weight over the sum of all the weights).

For instance, the simple automaton in Figure 2.4 allows to select two values for the period of the task: 10 with probability 2/3 or 15 with probability 1/3. In what follows, we will call this automaton a dispatcher.

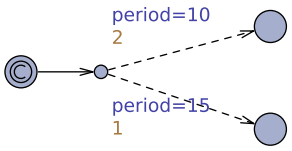


Figure 2.4 – Stochastic dispatcher implemented with a stochastic *TA*

In a network of stochastic HA the components repeatedly race against each other, i.e. they independently and stochastically decide on their own how much to delay before outputting, the "winner" being the component that chooses the minimum delay.

2.5 Model-based Scheduling Approach

Analytical scheduling methods determine if a set of tasks are schedulable by a given scheduling algorithm, using a scheduling test that is a function on the parameters of the tasks. Though effective, these techniques are limited to specific classes of scheduling policies and systems. An alternative is to use model-based approaches, with formal models of the components of a scheduling system (tasks, scheduler), and formal techniques such as model-checking and statistical model-checking. A recent series of papers [DLLM12, BDK+13, BDK+15a, KLT+16a] show that model-based approaches, implemented with timed automata and their extensions, are flexible enough to embed various types of scheduling policies, that go beyond those in the scope of analytical tools.

Model-based approaches also enable to use stochastic tasks whose real-time attributes, such as deadline, execution time or period, are characterized by probability distributions. This is particularly useful to describe mixed-critical systems and to make assumptions on the hardware domains. These systems combine hard real-time periodic tasks, with soft real-time sporadic tasks. Analytical scheduling techniques can only reason about worst-case analysis of these systems, and therefore always return pessimistic results. Using stochastic verification techniques like SMC we can instead analyze the system in

an average scenario and provide more accurate measures.

Analytical Methods for Analysis of Sporadic Tasks Sporadic tasks were first introduced in [BMR90, Mok83] as an extension of the Liu and Layland [LL73] task model.

The authors in [BMR90] proposed an exact schedulability analysis by providing some necessary and sufficient conditions for a sporadic task system.

In [ZKG+08], the authors propose a framework for the schedulability analysis of real-time systems, where they define a generalized model for sporadic tasks to more precisely characterizes the task arrival times. Each task is characterized by two constraints: higher instantaneous arrival rate, which bounds the maximum number of task arrivals during some small time interval; lower average arrival rate, which is used to specify the maximum number of arrivals over some longer time interval. The work of [MCG13] considers systems with probabilistic execution times and probabilistic inter-arrival times. However it does not handle dynamic scheduling policies. Moreover, the method is a numerical analysis technique whose complexity is exponential in proportion to the number of samples and tasks. In [TDP12], the authors propose a method to control the preemptive behavior of real-time sporadic task systems by the use of CPU frequency scaling. They introduced a new sporadic task model in which the task arrival may deviate, according to a *discrete* time probability distribution, from the minimum inter-arrival time. Based on the probability of arrivals, the authors propose an on-line algorithm computing CPU frequencies that guarantee non-preemptiveness of task behavior while preserving system schedulability.

Model-based Analysis of Stochastic Sporadic Tasks In the context of model-based analysis, the authors in [CBF+11] present a symmetric multi-core framework where a flat scheduling system can be described in the Prelude language. The schedulability can be checked using generated Uppaal models.

The authors in [MEP07] formally characterize stochastic tasks for various platforms and presents a model-based analysis technique to check the schedulability of the tasks. The main idea is to compute the probability distribution of a task termination time by a convolution of the probability density functions of the task starting time and execution time. However, it is restricted to non-preemptive stochastic tasks, and the analysis complexity is also exponential.

Using the statistical model checking technique in Uppaal, the work in [BDK+14] proposes a way of estimating the "degree of schedulability" of sporadic tasks and also presents the Uppaal models used to implement the concepts as well as an avionics case-

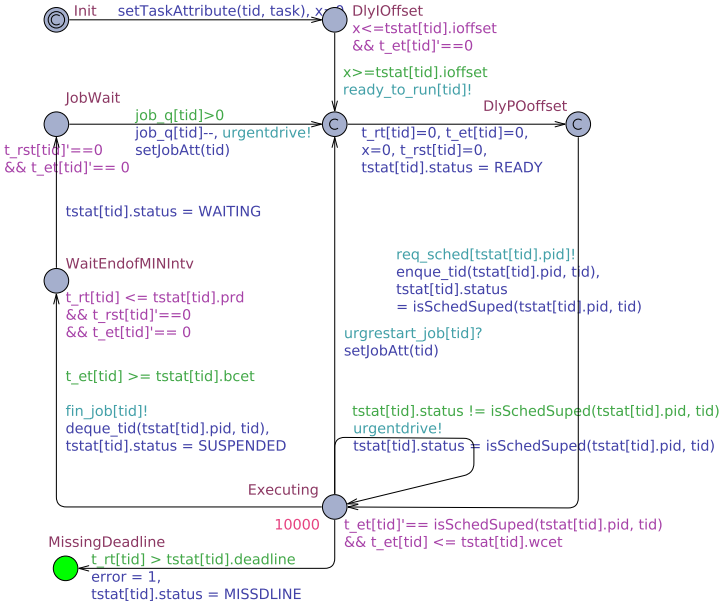


Figure 2.5 – *SWA* model of a stochastic task

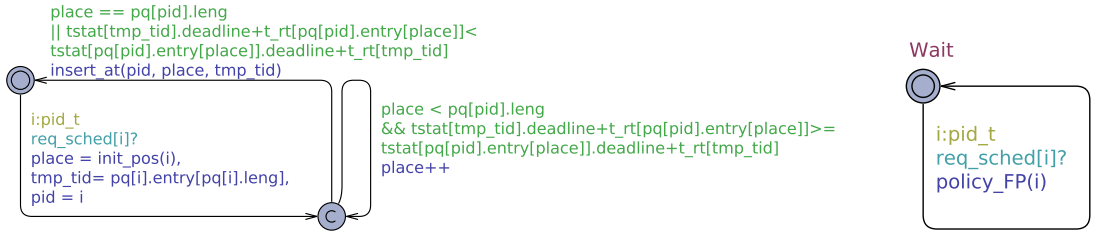
study.

The analysis technique of our work is based on extending the models of Timed Automata (TA) and Stopwatch Automata (SWA) in [BDK+13, BDK+15a], to present a model of hierarchical scheduling systems, based on the stochastic sporadic tasks of [BDK+15b] but with dynamic stochastic updates of real-time attributes. In this thesis, we reuse these models and extend them to define stochastic tasks.

2.5.1 Formal Models of Scheduling Components

The formal models of our scheduling systems are inspired by [BDK+13, BDK+15a].

Tasks Tasks are implemented with a SWA shown in Figure. 4.4. From the Init location, a first job is initialized with real-time attributes obtained from the function setTaskAttribute(...). This job is queued for execution at location DlyPOoffset. There it requests the scheduler to assign a CPU, which is granted by a synchronisation on the channel req_sched[tstat[tid].pid], and reaches location Executing. Its execution can be stopped and resumed according to the availability of the CPU resource. This is implemented by a stopwatch clock t_et[tid]. The clock progresses only when the CPU is available, that is when the function isSchedSuped(...) returns 1. Finally, the job exits from location Executing when it has completed its execution time. This releases the CPU resource



(a) EDF (b) FP

Figure 2.6 – *SWA* models of schedulers

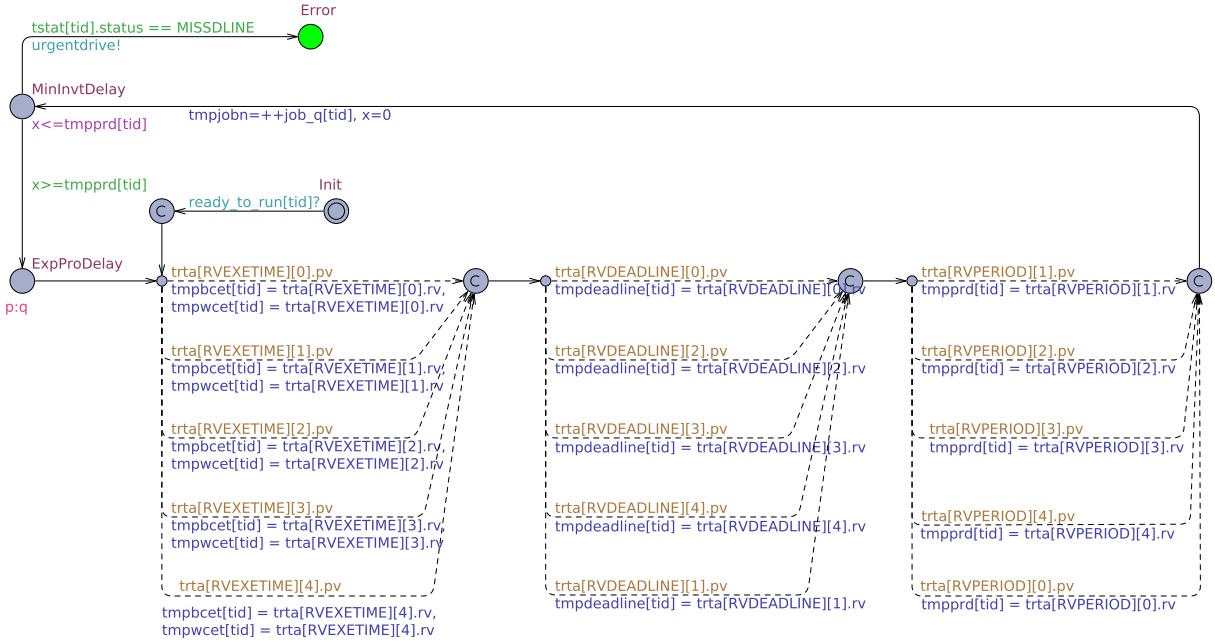


Figure 2.7 – *PTA* of the stochastic dispatcher

using function deque_tid(...). The SWA waits the end of the minimal inter-arrival time (WaitEndofMINIntv) and then instantiates a new job.

Scheduler The scheduler SWA implements the scheduling policy. We use two types of scheduling policy: earliest deadline first (EDF), implemented with the SWA in Figure. 2.6a, and fixed priority (FP), implemented with the SWA in Figure. 2.6b. These schedulers synchronize with the task model on the channel req_sched.

2.5.2 Stochastic Scheduling Systems

In [CKL+16] we have extended these models to use stochastic tasks whose real-time attributes (period, delay, execution time) depend on probability distributions, and are dynamically chosen by a stochastic dispatcher. This stochastic feature is of interest to model the variation of execution time with respect to the computation logic and the capability of the execution environments (CPU, memory, I/O and caches, etc). Such real values can be obtained by sampling the execution times from the real world system. Observe that other tasks parameters such as the deadline and the period are determined according to the timing requirements of the functionality implemented by a set of tasks. For instance, some video decoder and encoder would update the deadline and period of tasks according to the frequency of input streams. For those reasons, they can also be represented by probability distributions.

In a stochastic task the stochastic attributes are determined by a stochastic dispatcher at each new instantiation of a job (when calling the functions setTaskAttribute and setJobAtt). The stochastic dispatcher is implemented with a stochastic timed automata using discrete probabilistic choices. Figure. 2.7 presents an example of a dispatcher that configures the three attributes with probabilistic choices between five values.

Chapter 3

Model Checking, Statistical Model Checking and High Level Language

In this chapter, we present model checking MC and statistical model checking SMC as formal methods that we used in our work to analyze real-time systems. We first introduce the formal language used for the properties specification. Then we present the principals of MC and SMC. Finally we present high level languages that allow to embed these formal specifications into a user friendly graphical interface.

3.1 Introduction

The formal models defined in the previous chapter can be used to perform automatic verifications. For that, They must be accompanied by a specification of the properties of interest that need to be verified. This chapter introduces some important, though relatively simple, classes of properties. These properties are based on temporal logic to express requests about sequences of events. We present the syntax and semantics of this logic. Next, we present formal methods *model checking (MC)* and *statistical model checking (SMC)* to verify properties on a given model. Finally, we present a high level language called Cinco used to produce a graphical representation of the model and the properties under consideration.

3.2 Temporal logics

To express the timing constraints an appropriate formalism must be used. Here the timing constraints are defined as a sequence of events and not as a quantitative values.

Temporal logic is a formalism used to express the timing aspects of the property that needs to be verified with a mathematically precise notation. Temporal logic considers time in one of the two following manners. The first one is in linear way, which means that at each state of execution the system has a unique possible future state. The second manner is to consider time in a branching way, which means that at each state of the execution of the model the system can have more than one possible future. The second one considers the structure of time as a tree.

3.2.1 Linear Temporal Logic

Linear Temporal Logic (LTL) reasons in linear way about time. At each moment in time LTL considers that there is only one possible future of the system. Formulas of LTL are constructed from the set of atomic proposition AP of the transition system using the usual boolean connectors $(\neg, \wedge)$ and the temporal operators $(\bigcirc, \bigcup)$, where $\bigcirc$ designs the temporal operator *Next* and $\bigcup$ designs the temporal operator *Until*.

3.2.1.1 Syntax

The LTL formulas over set of atomic proposition are constructed according the the following grammar:

$$\varphi = true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \bigcup \varphi_2$$

where $a \in AP$.

The $\bigcirc$ operator is a unary prefix operator. $\bigcirc\varphi$ holds if it holds in the next step. The operator $\bigcup$ is a binary one. Formula $\varphi_1 \bigcup \varphi_2$ holds at the current state l_i if there is some state l_j in the future for which the formula φ_2 holds and the formula φ_1 holds at all the states until l_j.

The precedence order of the operators is given as follow. the unary operators are stronger than the binary ones. $\neg$ and $\bigcirc$ are equal strong, the temporal operator $\bigcup$ takes precedence over $\wedge, \vee$, and $\rightarrow$. Operator $\bigcup$ is right associative, $\varphi_1 \bigcup \varphi_2 \bigcup \varphi_3$ means $\varphi_1 \bigcup (\varphi_2 \bigcup \varphi_3)$.

From the negation and the conjunction we can define the usual boolean connectors as disjunction and implication as follow:

- $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ (disjunction)
- $\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$ (implication).

Using the until operator, we can express the temporal modalities $\Diamond$ (eventually in the future) and $\Box$ (now and forever in the future) as following:

$$\Diamond\varphi = true \bigcup \varphi \qquad\qquad \Box\varphi = \neg\Diamond\neg\varphi$$

Intuitively, $\Diamond\varphi$ means that eventually in the future the property φ will be true. $\Box\varphi$ holds if and only if $\neg\varphi$ do not holds eventually in the future.

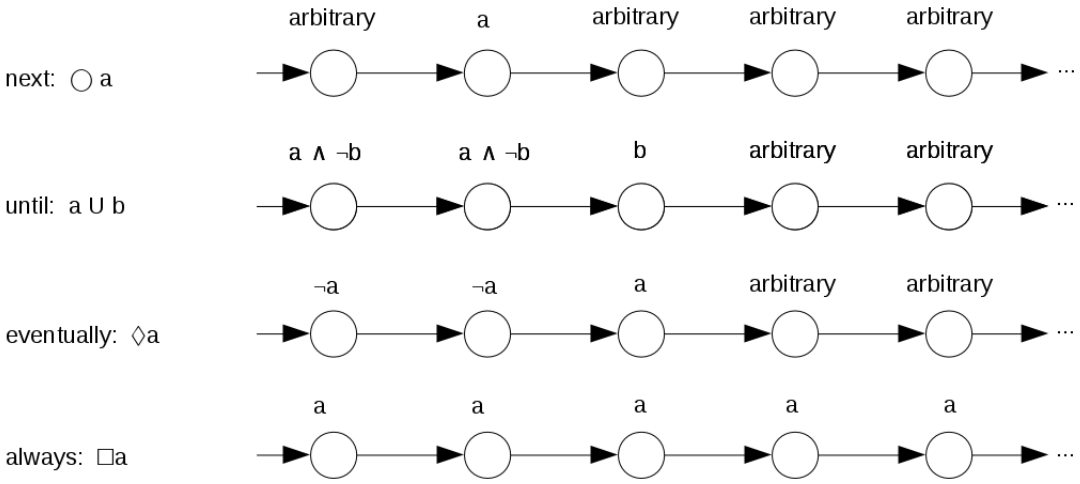


Figure 3.1 – Examples of the satisfaction of simple temporal modalities over an execution trace

New temporal modalities can be obtained by combining the two temporal modalities $\Diamond$ and $\Box$. For example, $\Box\Diamond a$ (always eventually a) means that at any moment i there is a moment $j, j \geqslant i$ at which the proposition a is true, it means that the proposition a will be true infinitely often. Another example is $\Diamond\Box a$ (eventually forever) which means that after some moment j only the proposition a will be true.

Figure 3.1 represents a graphical representation of given temporal modalities for a simple case where the arguments of the modalities are atomic propositions $\{a, b\}$. In the left side of the figure, some LTL formulae are given, while in the right side their corresponding graphical representation as an example of the satisfaction relation over an execution trace.

Example Let consider two processes P_1, P_2. For each process we can define three states. A non critical section state non_crit_i, a wait state $wait_i$, when the process is ready to enter its critical section, and the critical section $crit_i$. The LTL formula to express the requirement that the two processes can not enter their critical section simultaneously is:

$$\Box(\neg crit_1 \vee \neg crit_2)$$

This formula expresses that always ($\Box$) at least one of the two processes is not in its critical section.

3.2.1.2 Semantics

LTL formula expresses properties on paths (or in fact their trace). This means that a path can either satisfy an LTL formula or not. To precisely formulate when a path satisfies an LTL formula, we proceed as follows. First, the semantics of LTL formula φ is defined as a language $Words(\varphi)$ that contains all infinite words over the alphabet 2^{AP} that satisfy φ.

Definition 3.1 (Semantics of *LTL*). *Let consider an LTL formula φ over atomic propositions AP. The language accepted by φ is:*

$$Words(\varphi) = \{\sigma \in (2^{AP})^w | \sigma \vDash \varphi\}$$

with $\sigma = A_0A_1A_2... \in (2^{AP})^w$, $\sigma[j...] = A_jA_{j+1}A_{j+2}...$ is the suffix of σ starting in the $(j+1)$st symbol A_j, and the satisfaction relation $\vDash \subseteq (2^{AP})^w \times LTL$ is the smallest relation with the properties:

- $\sigma \vDash true$
- $\sigma \vDash a$ *iff* $a \in A_0$ *(i.e.* $A_0 \vDash a$*)*
- $\sigma \vDash \varphi_1 \wedge \varphi_2$ *iff* $\sigma \vDash \varphi_1$ *and* $\sigma \vDash \varphi_2$
- $\sigma \vDash \neg\varphi$ *iff* $\sigma \nvDash \varphi$
- $\sigma \vDash \bigcirc\varphi$ *iff* $\sigma[1...] = A_1A_2A_3... \vDash \varphi$
- $\sigma \vDash \varphi_1 \bigcup \varphi_2$ *iff* $\exists j \geqslant 0, \sigma[j...] \vDash \varphi_2$, *for all* $0 \leqslant i < j$, $\sigma[i...] \vDash \varphi_1$

For the derived operators $\Diamond$ and $\Box$ the satisfaction relation will be:

$$\begin{aligned} \sigma \vDash \Diamond\varphi \quad & \text{iff} \quad \exists j \geqslant 0, \sigma[j...] \vDash \varphi \\ \sigma \vDash \Box\varphi \quad & \text{iff} \quad \forall j \geqslant 0, \sigma[j...] \vDash \varphi \end{aligned}$$

As a subsequent step, we determine the semantics of LTL formula with respect to a transition system. The LTL formula φ holds in location l if all paths starting in l satisfy φ. The transition system TS satisfies φ if TS satisfies $Words(\varphi)$, i.e., if all initial paths of $Paths(TS)$ starting in an initial state $l_0 \in I$ satisfies φ.

Definition 3.2 (Semantics of *LTL* over Paths and States)**.** *Let consider φ as an LTL formula over atomic propositions AP, and let $TS = (Loc, Act, E, I, AP, L)$ be a transition system without any terminal location.*

- *For infinite path fragment π of TS, the satisfaction relation is defined by:*

$$\pi \vDash \varphi \text{ iff } Trace(\pi) \vDash \varphi$$

- *For location $l \in Loc$, the satisfaction relation is defined by:*

$$l \vDash \varphi \text{ iff } \forall \pi \in Paths(l), \pi \vDash \varphi$$

- *TS satisfies φ, denoted by $TS \vDash \varphi$, if $Traces(TS) \subseteq Words(\varphi)$*

3.2.2 Computation Tree Logic

In a transition system a location can have more than one direct successor, it means that we can have different paths starting from the same location. The satisfaction of an LTL formula φ in a state requires that the LTL formula holds in the location l if all possible computations starting in l satisfies φ.

In LTL, it is not simple to verify the existence of *some* paths starting at location l that satisfies the property φ. For example, to verify if there exists some paths starting at location l and that satisfies property φ, we can exploit the duality between universal and existential qualifications. We may check whether $l \vDash \neg\varphi$; if this formula is not satisfied, then there must be some computations satisfying φ.

For more complicated properties, like "for every computation it is always possible to return to the initial location", this is, however, not possible. A naive attempt would be to require $(\Box\Diamond l_0)$ to hold for every computation, where the location l_0 uniquely identifies the initial state. This is, however, too strong as it requires a computation to always return to the initial state, not just possibly. Other attempts to specify the intended property also fail, and it turns out that the property cannot be specified in LTL.

To overcome these problems, another kind of temporal logic can be used. Contrary to LTL, *Computation Tree Logic* (CTL) reasons about time in a branching manner, i.e at each state there can be several different futures. Due to this branching notion of time, this class of temporal logic is known as a *branching* temporal logic. The semantics of a branching temporal logic is defined in terms of an infinite, directed *tree* of states rather than an infinite sequence. Each traversal of the tree starting in its root represents a

single path. The tree itself thus represents all possible paths, and is directly obtained from a transition system by "unfolding" at the state of interest.

The temporal operators in a branching temporal logic allow the expression of properties *some* or *all* paths that starts at location l. To that end, it supports an existential path quantifier (denoted $\exists$) and a universal path quantifier (denoted $\forall$). For instance, the property $\exists\Diamond\varphi$ denotes that there exists a path along which $\Diamond\varphi$ holds. It means that there is at least one possible path in which a state that satisfies φ is eventually reached. The property $\forall\Diamond\varphi$, in contrast, means that all paths satisfy the property $\Diamond\varphi$.

3.2.2.1 Syntax

CTL distinguishes between state formula and path formula. State formula expresses a property of a state, while path formula expresses a property of a path, i.e. an infinite succession of states. The temporal operators $\bigcirc$ and $\bigcup$ have the same meaning as in LTL. The CTL formula over a set of atomic proposition is defined as follow:

$$\varphi ::= \ true \ | \ a \ | \ \varphi_1 \wedge \varphi_2 \ | \ \neg\varphi \ | \ \exists\Phi \ | \ \forall\Phi$$

where $a \in AP$ and Φ is a path formula. CTL path formulae are formed according to the following grammar:

$$\Phi ::= \ \bigcirc\varphi \ | \ \varphi_1 \bigcup \varphi_2$$

where φ, φ_1, and φ_2 are state formulae.

Path formula can be turned into state formula by adding path quantifier $\exists$ (pronounced it exists some paths) or the quantifier $\forall$ (pronounced for all paths). Temporal modalities "eventually", and "always" can be derived as follow:

$$\begin{aligned} \text{eventually: } \exists\Diamond\varphi &= \exists(true \bigcup \varphi) \\ \forall\Diamond\varphi &= \forall(true \bigcup \varphi) \\ \text{always: } \exists\Box\varphi &= \neg\forall\Diamond\neg\varphi \\ \forall\Box\varphi &= \neg\exists\Diamond\neg\varphi \end{aligned}$$

Example Let consider the precedent example presented in section 3.2.1, the mutual exclusion property can be written in CTL as follow:

$$\forall\Box(\neg crit_1 \vee \neg crit_2)$$

It means that for all paths we have always one process at least out of its critical section.

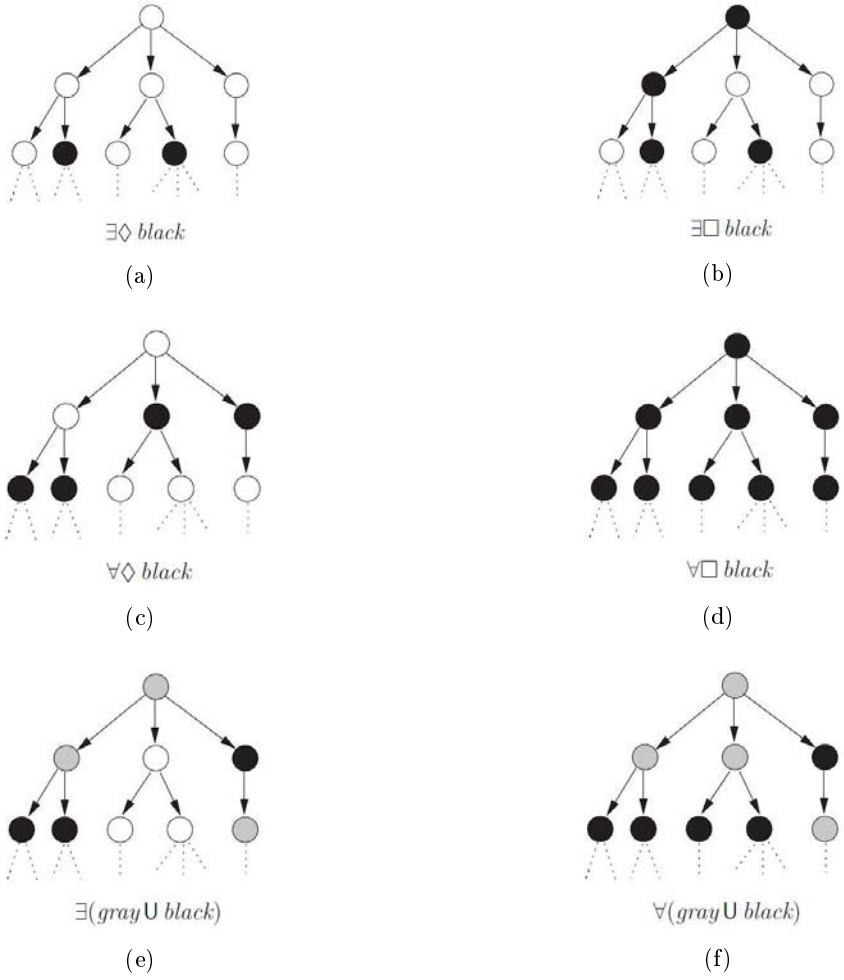


Figure 3.2 – Examples of satisfaction of some *CTL* formula

Figure 3.2 describes the graphical representation of the semantics of some CTL formulae. The first graphic Figure 3.2a describes the property "black holds potentially". The second graphic Figure 3.2b describes the property "black holds potentially always". The two graphics Figure 3.2c and Figure 3.2d respectively describe the properties "black is inevitable" and "invariably black". The last two graphics Figure 3.2e and Figure 3.2f express the properties "there exists paths where gray holds until black holds" and "for all paths gray holds until black holds", respectively.

3.2.2.2 Semantics

CTL formulae are interpreted over the states and paths of transition system. Formally,

given a transition system, the CTL formulae are defined by two satisfaction relations: one for the state formula and one for the path formula. For the state formula, $\vDash$ is a relation between the states in the transition system and the state formula Φ, we write $s \vDash \Phi$, to mean that state s satisfies the state formula Φ. For the path formula, $\vDash$ is a relation between maximal path fragment π and path formula φ, we write $\pi \vDash \varphi$, to mean that π satisfies the path formula φ.

Definition 3.3 (Satisfaction relation for *CTL*). *Let $TS = (Loc, Act, E, AP, L)$ be a transition system without terminal state, $a \in AP$ an atomic proposition, Φ, Ψ a CTL state formula, φ a CTL path formula. We define the satisfaction relation $\vDash$ as follow:*

- $l \vDash a$ *iff* $a \in L(l)$
- $l \vDash \neg\Phi$ *iff* *not* $l \vDash \Phi$
- $l \vDash \Phi \wedge \Psi$ *iff* $(l \vDash \Phi)$ *and* $(l \vDash \Psi)$
- $l \vDash \exists\varphi$ *iff* $\pi \vDash \varphi$ *for some* $\pi \in Paths(l)$
- $l \vDash \forall\varphi$ *iff* $\pi \vDash \varphi$ *for all* $\pi \in Paths(l)$

For paths π, the satisfaction relation $\vDash$ is defined as follow:

- $\pi \vDash \bigcirc\Phi$ *iff* $\pi[1] \vDash \Phi$
- $\pi \vDash \Phi \bigcup \Psi$ *iff* $\exists j \geqslant 0, (\pi[j] \vDash \Psi \wedge (\forall 0 \leqslant k < j, \pi[k] \vDash \Phi))$

where for path $\pi = l_0 l_1 l_2 ...$ and for $i \geqslant 0$, $\pi[i]$ denotes the $(i+1)$th location of π.

In CTL atomic propositions, negation, and conjunction are interpreted over states, whereas in LTL they are interpreted over paths. State formula $\exists\varphi$ holds in l if and only if there exists some paths starting in l that satisfies φ. The state formula $\forall\varphi$ holds in l if and only if all paths starting in l satisfies φ. The semantics of path formula is identical to that for LTL. For instance, $\exists \bigcirc \Phi$ holds in location l if and only if there exists paths starting in location l where the next location satisfies Φ, this is equivalent to the existence of a direct successor l' of the location l where $l' \vDash \Phi$. $\forall(\Phi \bigcup \Psi)$ holds is location l if and only if for all paths starting in location l there exists an initial finite prefix such that Ψ holds in the last location of this prefix and Φ holds for all the locations along this prefix.

3.3 Model Checking (*mc*)

Model checking is an automated verification technique that explores all the possible executions of a model in a brute-force manner to verify if it satisfies a property written in a formal logic. Model-checking can thus be used to assess the schedulability of a system, for any of its executions. This corresponds to the so-called worst-case analysis.

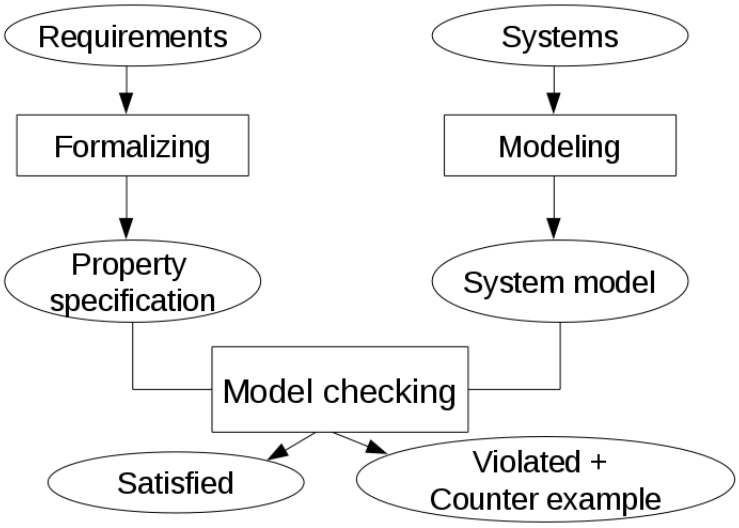


Figure 3.3 – Graphical representation of the model checking approach

Figure 3.3 represents the model checking approach. The system model is usually automatically generated from a model description that is expressed in some appropriate language. Note that the property specification describes *what* the system should do, or what should not do, while the model specification describes *how* the system behaves. The model checker analyses all the relevant system states and verifies if they satisfy the desired property. If the model checker detects a system state that does not satisfy the desired property, it provides a counterexample that indicates how the model can reach the undesired state. The counterexample represents an execution path starting from the initial state until the state that violates the desired property. The user can analyze the counterexample to adapt the model accordingly.

When using model checking to analyze a given system, we can distinguish the following phases:

Modeling The modeling phase consists of:

- Model the system under consideration using the model description language of the model checker.

- Make some initial simulations to detect and fix any simpler errors before using any form of checking.
- Formalize the properties to be checked using appropriate temporal logic.

Running The running phase executes the model checker to verify the satisfiability of the properties under consideration.

Analysis The analysis phase interprets the results of the model checker:

- If the property is satisfied then check the next property if it exists. If all properties are checked and are satisfied, then we conclude that the model satisfies all the desired properties;
- If the property is not satisfied then:
  1. analyze the generated counterexample;
  2. refine the model, the system design, or the property;
  3. repeat the entire procedure.
- If the model is too large to be handled (state space of real-life system is often too large to be stored in the available memory) then reduce the model and try again.

Typically, properties of qualitative nature can be checked using model checking. For example, "is the generated result OK?" or "can the system reach a deadlock situation?". Additionally, timing properties can be checked, like "can the system reach a deadlock situation within one hour after a system reset?", or, "is the response always received after sending a question?". In that case, model checking requires a precise and unambiguous statement of the properties to be examined.

CTL checking problems are decidable for TA. Even basic model checking problems (reachability) are undecidable for SWA and HA. For these models it is only possible to perform exhaustive analysis with an over-approximation of the reachable states. The alternative is to exploit the stochastic semantics of HA and to resort to simulations and statistical model-checking (SMC).

The model checker Uppaal uses a restricted version of temporal logic CTL to express the properties under consideration. It is defined as follow:

$$\varphi ::= \mathsf{A}[]P \mid \mathsf{A}<>P \mid \mathsf{E}[]P \mid \mathsf{E}<>P$$

The operator A represents the operator for all paths" $\forall$ defined above in section. 3.2.2. In the same way E represents the operator "there exists a path" $\exists$. [] and <> are state

operators, meaning respectively ‘"all states of the path" and "there exists a state in the path", the corresponding state operators defined in section. 3.2.2 are respectively $\Box$ and $\Diamond$. P is an atomic proposition that is valid in some state. For example the formula "A[] not error" specifies that in all the paths and all the states on these paths we will never reach a state labeled as an error. For instance for schedulability analysis, an error state is one where a task has missed a deadline.

Strengths and Weaknesses The main important qualities that constitute the strength of model checking are:

- It is a general verification approach that can be applied on different ranges of systems.
- The properties can be checked separately, i.e. each property can be checked individually.
- It gives diagnostic information if the property is not validate, these information are very important for debugging purposes.
- It can be easily integrated in existing development cycles.

The main important weaknesses of model checking are:

- It verifies a system model not the actual system itself. Any obtained result is thus as good as the system model.
- It checks only stated requirements, i.e., there is no guarantee of completeness. The validity of properties that are not checked cannot be judged.
- It suffers from the state-space explosion problem, i.e., the number of states needed to model the system accurately may easily exceeds the amount of available computer memory.
- Its usage requires some expertise in finding appropriate abstractions to obtain smaller system models and to state properties in the logical formalism used.

Despite the above limitations we conclude that *model checking is an effective technique to expose potential design errors.* Thus, model checking can provide a significant increase in the level of confidence of a system design.

3.4 Statistical Model Checking (*SMC*)

To overcome the above difficulties we also propose to work with *Statistical Model Checking SMC* [KZH+11, You05, You06, SVA04, SVA05a, SVA05b], an approach that has recently been developed as an alternative to avoid an exhaustive exploration of the state-space of the model.

SMC allows to reason on the average scenario, and to quantify the results with a probability measure. The principle is to combine formal verification and techniques from the statistic area in order to compute the probability that a system achieves a given objective.

There exists several SMC algorithms, see [LDB10] for details. In this thesis, we focus on the Monte-Carlo algorithm. This algorithm performs N executions ρ and then estimates the probability γ that the system satisfies a logical formula φ using the following equation:

$$\tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\rho \models \varphi)$$

where $\mathbf{1}$ is an indicator function that returns 1 if φ is satisfied and 0 otherwise. The number of simulations N defines the precision of the results. It guarantees that the estimate $\tilde{\gamma}$ is close enough to the true probability γ, such that if $N = \lceil (\ln 2 - \ln \delta)/(2\varepsilon^2) \rceil$ the probability of error is $Pr(\mid \tilde{\gamma} - \gamma \mid \geq \varepsilon) \leq \delta$, where ϵ and δ define the confidence interval and the confidence level, respectively.

Bounded Linear Temporal Logic (BLTL) is a restricted version of LTL that expresses bounds on step or time units in order to reduce the paths or time on which the desired property will be verified. These bounds give the length of the run on which the property under consideration will be verified. Any decidable property on states or paths can be used in the formulae including BLTL operators. Thus, the semantics of BLTL logic is the semantics of LTL logic restricted to a time interval. The BLTL temporal operators are defined as follow:

- "eventually within time t" : $\Diamond^t \varphi = true \bigcup^t \varphi$ where $t \in \mathbb{R}^+$
- "always up to time t" : $\Box^t \varphi = \neg \Diamond^t \neg \varphi$ where $t \in \mathbb{R}^+$

The statistical model checker Uppaal SMC uses BLTL formulas to ask for the probability that a given property holds within a fixed bound of time.

Uppaal SMC queries express properties over a single trace using BLTL. These queries associate an LTL formula and probability operator Pr and a time bound. The following query for instance "Pr[<=maxTime](<> error)" asks to compute the probability of reaching an error state before maxTime. Additionally, Uppaal SMC allows writing simulate queries that only examine traces without computing a probability.

3.5 High Level Language: Cinco

Currently, many models and tools are successfully used to analyze properties of CPS. But they are domain-specific, which means they cannot easily be applied to other systems. Moreover, these models and tools require high technical knowledge about the theoretical formalisms used to design models and write properties, which most system engineers do not master. In this section we demonstrate a flexible and formal analysis engineering approach for analyzing scheduling properties of CPS.

We encapsulate the formal models for scheduling systems presented in Chapter 2.5.1 into Cinco [NLKS17, NTI+14] a generator for domain-specific modeling tool, these models constitute a model bank that will be used to represent the different systems under consideration. This bank will be enriched in the next chapters.

Cinco allows to specify the features of a graphical interface in a compact meta-model language, and it generates automatically from this meta-model specification a domain-specific analysis tool with a graphical interface. Inside this analysis tool we can specify scheduling systems and the properties they must satisfy. Cinco allows also to add implementations in order to achieve additive works, in our work we implement number of algorithms that are designed to:

- Transform the graphical representation of the components and the properties specifications of the system under consideration to timed automata;
- Allow to launch analyses in the graphical interface;
- Call Uppaal [BDL+06] and Uppaal SMC [DLL+15] to perform the analysis;
- Parse the results and displays them by modifying the graphical elements.

These transformations from the graphical representation to the formal models consist in translating the graphical components and the properties specifications of the system under consideration using the appropriate timed automata from the model bank. Consequently it constructs a full formal model that can be analyzed by Uppaal in order to

verify the satisfiability of the properties studied. This approach allows to completely hide the formal models being used from the system designer, who can concentrate on the structure and the parameters of the scheduling system.

The last challenge is to give significant feedbacks to the user in the most friendly manner. Indeed, results of formal verification from academic tools like Uppaal can be difficult to interpret, all the more when the models used by these tools have been automatically generated. Cinco provides an API for model transformations that allows to program actions that can update the model. We have used this functionality to parse the results of the analyses output by Uppaal and to show graphically the most relevant information.

3.5.1 Domain-Specific Code Generator: CINCO

Cinco is a generative framework for the development of domain-specific graphical modeling tools. It is based on the Eclipse Modeling Project [Gro08], but with a strong emphasis on simplicity [MS10], so that the user (i.e. the developer of a tool generated with Cinco) does not need to struggle with the underlying powerful but complicated EMF metamodeling technologies [SBPM08] directly. This is achieved by focusing on graph model structures (i.e. models consisting of various types of nodes and edges) and automatically generating the required Ecore metamodel as well as the complete corresponding graphical editor from an abstract specification in terms of structural constraints. In a sense, this approach turns constraint-based variability management [JLM+12, LNS13] into a tool generation discipline, where a product line is just characterized by the tools' modeling capacities.

3.5.1.1 Meta-Modeling

Central to every Cinco product is the definition of a file in the Meta Graph Language (MGL). It defines what kind of modeling components the model consists of and what attributes they have. Every modeling component is either a node type, a container type (i.e. a special node that can hold other nodes) or an edge type. It is also possible to define which kind of nodes can be connected to which kind of edges and express cardinality constraints on those connections.

Example For instance, Listing 3.1 presents of a portion of an MGL file with the definition of a container node to represent a simple task. The definition precises some attributes (period, wcet, deadline ...). It needs exactly one input transition and one or more output transitions. Furthermore, it can contain other nodes of type Query.

```
@style(task,"${tid}","${period}","${wcet}","${deadline}","${priority}")
container Supplier {
  attr EInt as tid
  attr EInt as period
  attr EInt as wcet
  attr EInt as priority
  attr EInt as deadline
  incomingEdges (Transition[1,1])
  outgoingEdges (Transition[1,*])
  containableElements (Query)
}
```

Listing 3.1 – Part of the MGL file that specifies a task.

The second important file is a specification in the Meta Style Language (MSL), which is used for defining shapes (rectangle, ellipse, polygon,image, text, etc.) and appearances (colors, line style, line width, etc.) for nodes and edges. To change the look of the model depending on runtime information (e.g. the value of a node's attribute) one can either use the attribute directly within a text shape or implement an appearance provider that is invoked by the framework and may contain Java code that decides on the appearance by arbitrary external or internal factors.

Example Listing 3.2 contains the style definition for the previous task node. It is displayed with a red rounded rectangle and some texts in the top precessing the identifier of the task, as shown in Figure 3.4.

```
nodeStyle task(1) {
  roundedRectangle r {
    appearance extends default {
      foreground (220,15,15)
  background (255,255,255)
    }
    size(100,80)
    corner(20,20)
    text {
  position relativeTo r (CENTER,TOP)
  value "Task %s"
    }
  }
}
```

Listing 3.2 – Part of the STYLE file that configures the display of the task node.

Those specifications are already enough for Cinco to generate the complete graphical modeling tool. But Cinco also provides mechanisms to integrate the code that interprets

Figure 3.4 – Example of task display generated by the style configuration.

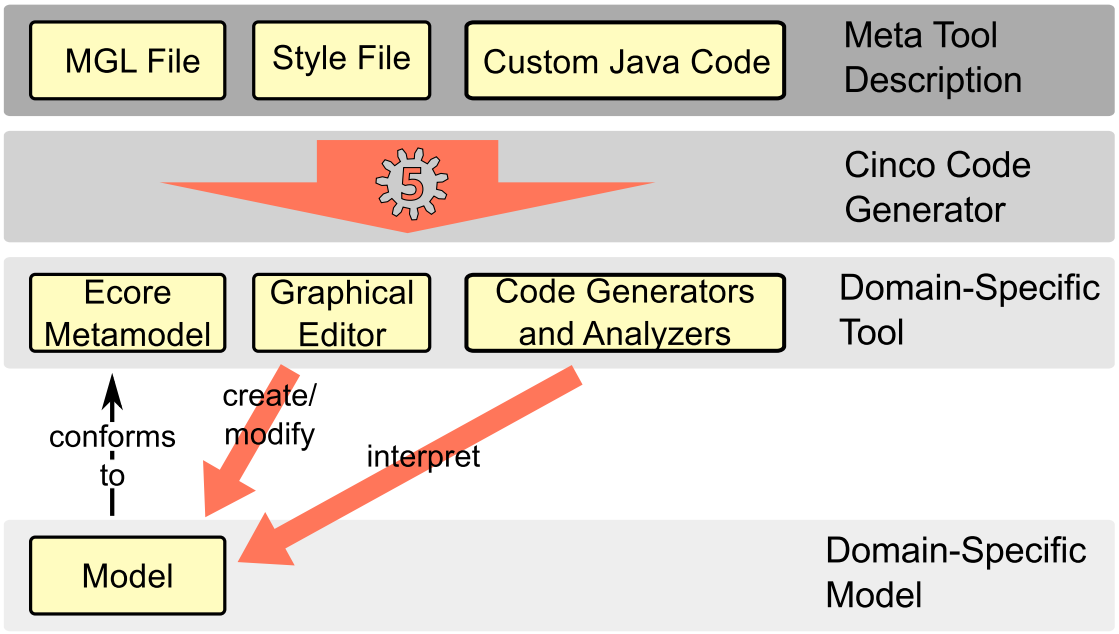


Figure 3.5 – Main principles of domain-specific tools generation with Cinco.

or transforms the models. It automatically generates APIs specific to the model type and seamlessly integrates code implemented against it into a ready-to-run modeling tool, which is a realization of the one-thing-approach [MS09].

The main principles for the generation of a domain-specific tool with Cinco are depicted in Figure 3.5. From the MGL and Style definitions, Cinco generates an Ecore metamodel as well as a corresponding graphical editor for the domain-specific tool. The user can then create a model in the tool that conforms to the given specification. This model can be analyzed by custom Java code, embedded in the tool during the automatic generation by Cinco.

3.5.1.2 Domain Specific Tool

Besides the tool meta-modeling, the second important feature of Cinco needed to develop a domain specific tool is the possibility to enhance the graphical editor by adding custom code. This code can call an API generated by Cinco to interact with the meta-model. Please refer to [NLKS17, NTI+14] or the website[1] for more detailed introductions.

The easiest way to enhance the graphical editor is by adding a custom action to a node type, which is then available via the nodes' context menu or on double-click. For a custom action two methods need to be implemented: `canExecute` and `execute`. Both receive the node on which the action should be performed as a parameter. While the first decides whether the action is available (i.e. not disabled/greyed out in the context menu), the second one actually performs it. The generated enhanced API for the metamodel simplifies the implementation of those methods, as one can easily access related modeling elements in a semantic and type-safe way, e.g. by accessing all successors (i.e. target nodes of outgoing edges) of a certain type.

Furthermore, Cinco makes it especially easy to perform changes to the edited model. Usually, with the common Eclipse approaches, the visual representation as well as the underlying model structure need to be changed separately. The transformation API that Cinco generates for every model type handles the synchronous and consistent modification of both parts automatically, so that it becomes very straightforward to program transformations for the model, as the generated API provides the same actions the tool user can perform within the editor, e.g. change attributes, add new elements, connect them with edges, or move/resize/delete them.

3.5.2 Implementation of the Framework and Tool Chain

We have implemented the domain-specific analysis framework. The tool chain involved in the generation of these frameworks and then in their usage is described in Figure 3.6.

The framework is developed in Java and with Cinco. The graphical interface of the framework is specified with the meta-modeling languages of Cinco, presented in Section 3.5.1.1. Then we have developed Java programs for generating complete formal models from the high-level specifications. These generators use existing formal models from a model bank (the models presented in Section 2.5.1). Finally we have developed Java custom analysis programs. These programs are linked to the code generated by Cinco such that they can be started directly from the graphical interface, either by a right-click menu or double-click actions. These programs solve the problems listed in

[1] `http://cinco.scce.info`

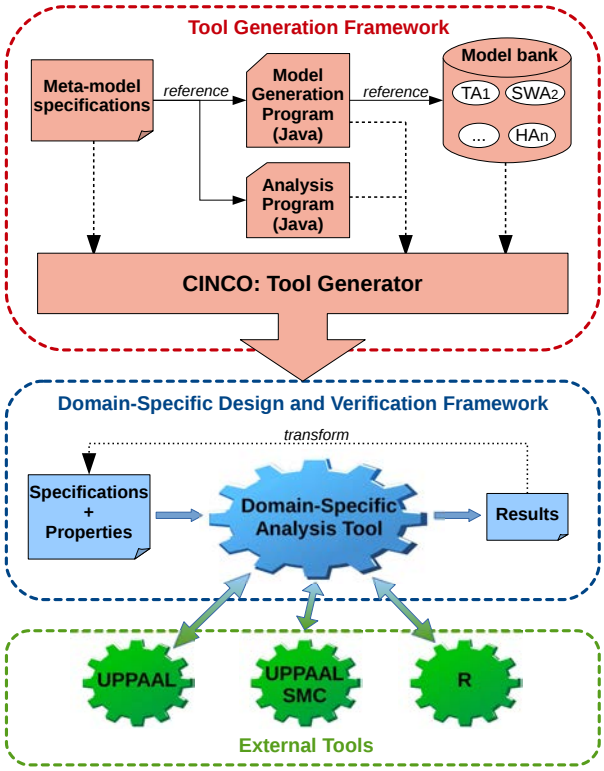


Figure 3.6 – Tool chain for generating and using domain-specific analysis frameworks

Section 4.2 using the techniques presented in Section 4.4.1. In the background, they launch Uppaal and Uppaal SMC via the command line interface to perform formal verifications, and they use the tool R for statistical analysis. The textual results of these verifications are then analyzed by our programs to determine the relevant results of the analysis. Then, the transformation API of Cinco is used to visualize the results on the model's high-level abstract view, either by creating pop-up windows or by making modifications on the model designed in the interface.

Using the meta-model specification, our custom Java code and the model bank, Cinco automatically generates a domain-specific framework, that includes the Java code and the Ecore specifications of an Eclipse graphical interface. This domain-specific framework allows to design scheduling problems, using the high-level graphical languages presented in Section 3.5.1.2. It then launches analysis by calling external tools (Uppaal, Uppaal SMC and R). It produces results and consequently can transform the original high-level specification.

In the next two chapters we will demonstrate the use of Cinco in combination with MC and SMC.

Chapter 4

Hierarchical Scheduling Systems

In this chapter, we extend the model-based approach of Section 2.5 in order to model hierarchical scheduling systems HSS, that are complex scheduling systems with multiple heterogeneous scheduling levels. The next step consists of expressing a number of problems that must be verified on this model. After that, we detail the methods used to solve each problem. Then, we present a new high-level framework for specifying and verifying the models and the problems. Finally, a number of experiences using the high-level framework are presented with a discussion on their results.

Key Contributions. The key contributions in this chapter are as follows:

- Formal models for HSS using stochastic tasks of Chapter 3 .
- A high level framework for specifying and verifying HSS models.
- A set of experiments executed on a case study which is an avionic system.

4.1 Hierarchical Scheduling Systems

One of the trends in developing CPS is to execute many heterogeneous real-time components into a single high-performance platform. This does not only reduce the costs, but also improves the performances and maximizes the utilization of hardware resources. However, these heterogeneous components must be partitioned, such that errors caused by one component are alienated from the other components. For instance, heterogeneous operating platforms in avionics and automotive systems manage various and different integrity-level applications. They are integrated using a high-performance hardware

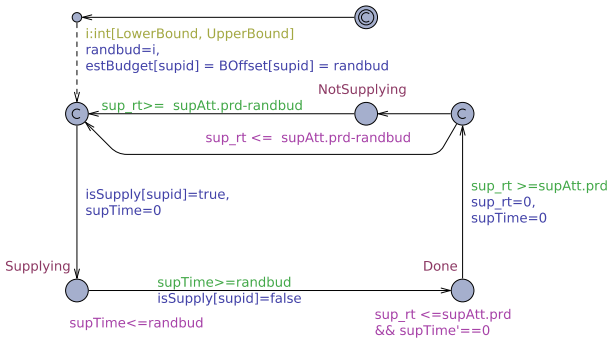


Figure 4.1 – Periodic Resource Model supplier with stochastic budget

platform, supported by multi-core processors, advanced memories, and multi-level cache architectures.

This has motivated research on hierarchical scheduling systems (HSS), where a global scheduler is used to distribute a shared resource among several local schedulers. This mechanism can be duplicated in a multi levels system, effectively building a hierarchy of schedulers organized in a tree structure. On one hand, analytical methods have been proposed for HSS [SEL08a, PLE+11]. Though they are easy to apply once proven correct, proving their correctness is a difficult research topic, and they only provide a coarse abstraction that grossly overestimates the amount of resources needed.

On the other hand, there exists model-based techniques [DLLM12, BDK+13, BDK+15a]. Since the complexity of the entire HSS is too large to be analyzed with formal methods, we rely on compositional approaches that allow to analyze each local scheduler independently [SL03].

In our formal framework, a HSS is a set of scheduling units organized in a tree structure. Each scheduling unit is composed of a set of real-time tasks, a scheduler, that implements a scheduling algorithm, and a queue, that manages jobs instantiated by tasks. To perform a compositional analysis of the system, we provide each scheduling unit with a resource supplier that abstracts the behavior of the parent scheduling unit. The models for tasks and schedulers are defined in Chapter 2.5.1.

Resource Supplier The resource supplier is responsible for supplying a scheduling unit with the resource allocated from a parent scheduling unit. We adopt the periodic resource model (PRM) [SL03]. It supplies the resource for a duration of Θ time units every period Π. To speed up the schedulability analysis using model checking techniques, it only generates the extreme cases of resource assignment: either the resource is provided at the beginning of the period (from 0 to Θ) or at the very end (from $\Pi - \Theta$ to Π). The

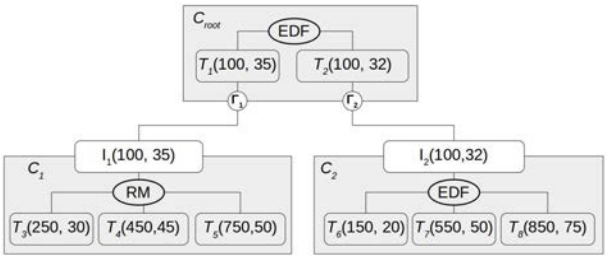


Figure 4.2 – Example of Hierarchical Scheduling System

choice between the two assignments is non-deterministic.

The PRM automaton communicates with the task model through a shared variable isSupply that is set to true during the supply period.

We also use the probabilistic supplier model presented in [BDK+15a]. This probabilistic supplier is implemented with the SWA of Figure 4.1. Instead of using a fixed budget Θ, it uses a range of values specified between an interval [LowerBound,UpperBound]. This will allow to perform a parameter sweep with SMC by selecting uniformly a value of the budget, and it will help determining the optimal budget.

Example We present in Figure 4.2 a small example of HSS with three schedulers: a top scheduler C_{root}, with an EDF policy, and two bottom schedulers C_1 and C_2, with Rate Monotonic (RM) and EDF policies, respectively. The top scheduler schedules two tasks $\mathcal{T}_1$ and $\mathcal{T}_2$ that distribute the resource to the interfaces I_1 and I_2 of the lower schedulers. These interfaces use the PRM, each with a period of 100, and a budget of 35 for I_1 and 25 for I_2. The lower schedulers schedule three real-time tasks each using the resource they receive from the interfaces I_1 and I_2.

4.2 Scheduling Problems

In this section we present the different problems that we want to solve in scheduling systems.

Problem 1: Correctness and performance We want to evaluate several properties of the scheduling system to assess its correctness and measure its performances:

1. **Absence of deadlock:** We check that the formal models have been correctly designed, because they cannot reach a deadlock state in which time is blocked and no action is available.

2. **Schedulability:** We determine whether the tasks are schedulable, i.e, none of them misses a deadline. In case of HSS, we check that all the scheduling units are schedulable.

3. **Maximum response time:** We measure the maximum response time of tasks, i.e., the maximum time between a job instantiation and its completion.

Problem 2: Optimal configuration of the system Depending on their nature, our scheduling systems may admit different configurations. Then, we may evaluate each configuration according to one or several measures presented in Problem 1 in order to select an optimal configuration.

In a HSS, each scheduling unit is analyzed independently using the budget provided by the PRM. To configure the system we determine which budget values make the system schedulable. Our goal is to find minimum budgets, such that all the scheduling units are schedulable.

4.3 Formal Model-based Compositional Framework for HSSs

Our model-based compositional analysis tool implements a model-based analysis framework of HSSs that is flexible enough to represent any scheduling systems.

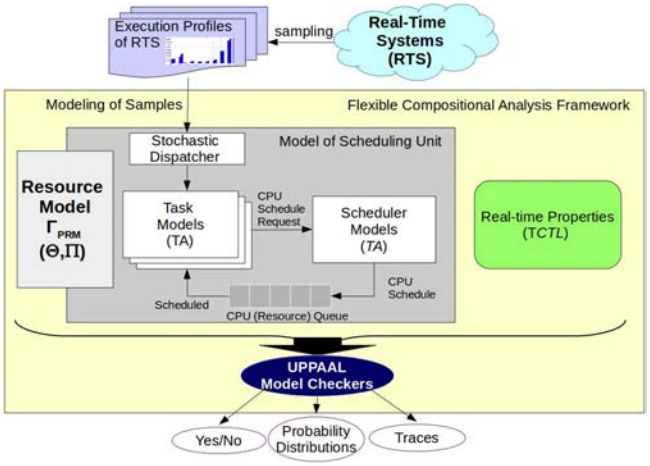


Figure 4.3 – Flexible Compositional Analysis Framework

As shown in Figure 4.3, the framework is composed of a set of component models (tasks, a scheduler and a stochastic dispatcher), that are used to configure the scheduling

units of a HSS, and a set of real-time properties that must be analyzed.

The configuration of the scheduling units of a HSS is determined by the user, who defines the structure of the HSS and specifies the real-time attributes of individual tasks. Once the configuration of the HSS has been made, our tool enables the designer to check the configuration of the HSS against real-time properties. In our setting, three important real-time properties are checked: the deadlock freedom of a HSS, and the schedulability and the worst-case response time of individual tasks.

4.3.1 Stochastic Task

In this section, we use the model of stochastic tasks presented in Section 2.5 whose real-time attributes depend on probability distributions. An execution of a task is characterized by 3 real-time attributes: an execution time, a period, and a deadline. The difference between these stochastic tasks and the previous work [BKD+, BDK+15a, MCG13] is that the three real-time attributes are dynamically configured according to the condition in which the system is running. This dynamic configuration is modeled by a stochastic dispatcher with an extension of timed automata with configuration actions that depends on the probability distributions.

A task represents the time spent for executing some computation. Its execution time may vary due to the length of executions of the computation logic and the capability of the execution environments, such as CPU, memory, I/O and caches, etc. Real values can be obtained by sampling the execution times from the real world system. The sampled execution times can then be captured by a probability distribution.

Meanwhile, the deadline and the period are determined according to the timing requirements of the functionality implemented by a set of tasks. For instance, some video decoder and encoder would update the deadline and period of tasks according to the frequency of input streams. In a similar way, they can also be represented by probability distributions.

In our stochastic task model we consider discrete probability distributions, defined with a random variable X given by:

$$X = \begin{pmatrix} x_1, ..., x_n \\ p_1, ..., p_n \end{pmatrix} \tag{4.1}$$

where $\{x_1, .., x_n\}$ are samples, $P(x_i) = p_i$ is the probability of each sample x_i and $\sum_{i=1}^{n} p_i = 1$. The probability of any variable x is given by $P(x)$ if $x \in \{x_1, .., x_n\}$, otherwise $P(x) = 0$.

Figure 4.4 shows the TA for our stochastic task model. The only difference of this TA model with the periodic TA task model of [BDK+15a] is that it begins the execution if its job's queue job_q[tid] is not empty.

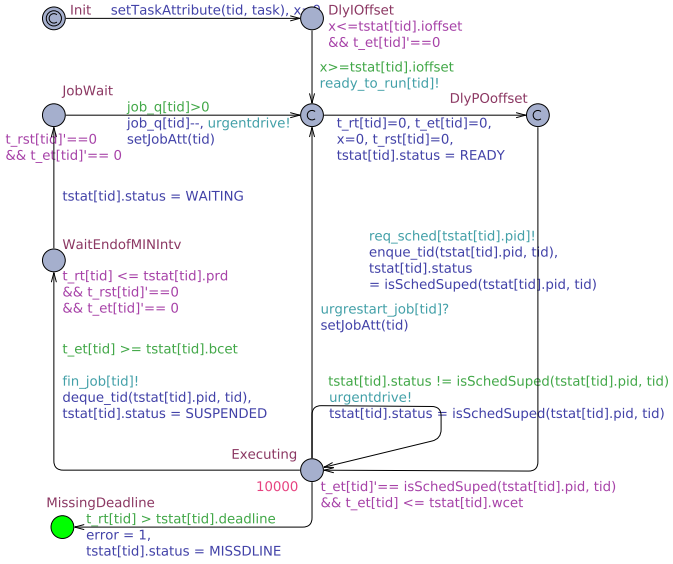


Figure 4.4 – TA template of a stochastic task (T_i)

If a process of the TA stochastic task at the Init location is instantiated, it reads the default attributes by function setTaskAttribute() and initializes a job. Then, the job requests the scheduler to assign a CPU by synchronizing the channel req_sched(pid) and queues at a resource (ready) queue by inserting its id (tid) to the queue.

A job process may stay at location Executing as long as job's execution time is not fulfilled and it does not miss the deadline. The process stops and resumes its execution on that location according to the availability of CPU resource, i.e. the job process can make progress when a CPU is available, otherwise, it must stop its execution.

In our model, there is no preemption location to denote that a task is waiting for CPU after it has been preempted but preemption is implemented by a stopwatch clock t_et[tid]. This clock measures the CPU-consuming time of a task since a job of the task has been instantiated; the clock can stop and resume when a CPU is available to the task. At location Executing, the invariant expression t_et[tid]'=isSchedSuped(tstat[tid].pid),tid) is associated to the stopwatch clock. This condition is such that the clock progresses if the function isSchedSuped() returns 1, otherwise, it does not progress.

The process of a task exits from location Executing when it has fulfilled its execution time and it releases the CPU resource using function deque_tid(tstat[tid].pid, tid). Then,

it joins the location WaitEndofMINIntv and waits the end of the minimal inter-arrival time. Finally, the process of a task joins the location JobWait to be instantiated by a job dispatcher.

4.3.2 Stochastic Dispatcher

These stochastic tasks are combined with a stochastic dispatcher that configures the real-time attributes of the tasks at each individual execution round. In other words, the stochastic dispatcher determines the configuration of tasks real-time attributes at the beginning of each execution round when the task is waiting at the location JobWait.

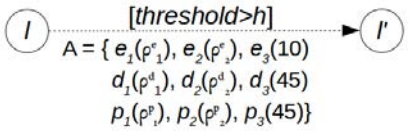


Figure 4.5 – An action to configure stochastic real-time attributes

To represent this dynamic configuration of real-time attributes, we extend the timed automata (TA) with a configuration action. An example of the configuration of a set of 3 tasks is given in Figure 4.5: The transition l to l' is enabled if the condition $threshold > h$ holds. When the transition is taken, the set A of actions are carried out, meaning that the execution e_1 of task T_1 is chosen randomly according to the probability distribution p_1^e. In the similar way, the deadline and period of each individual stochastic tasks are taken from the corresponding probability distributions. Note that e_3, d_3 and p_3 are assigned to constants values, 10, 45 and 45, respectively.

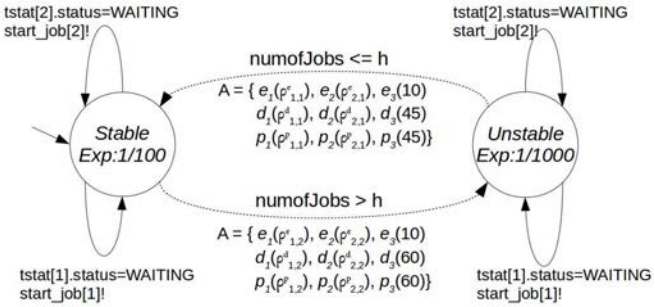


Figure 4.6 – An action to configure stochastic real-time attributes

Figure 4.6 shows an example of a job dispatcher that uses the configuration actions. On the initial location Stable, two recursive transitions trigger the events start_job[1] and start_job[2] to instantiate the corresponding jobs if the conditions tstat[1].status=WAITING

and tstat[2].status=WAITING hold. Even if the transitions are enabled, they are actually taken by the exponential distribution with rate $\lambda = 1/100$. If the condition $numofJobs > h$ holds, the transition heading for location Unstable can be taken. Then, a new configuration on the real-time attributes of T_1, T_2 and T_3 are made and, in particular, the real-time attributes of T_1 and T_2 are taken from the associated probability distributions, such as $\rho^e_{1,2}$, $\rho^p_{1,2}$, $\rho^d_{1,2}$, etc.

4.3.3 Formal Analysis Model of Scheduling Unit

The approach we pursue is compositional: each scheduling unit is individually analyzed with respect to an interface that abstracts the behavior of the other components. For the analysis of HSSs, the interface we are using is the PRM [SEL08b] that assigns the amount Θ of resources every period Π.

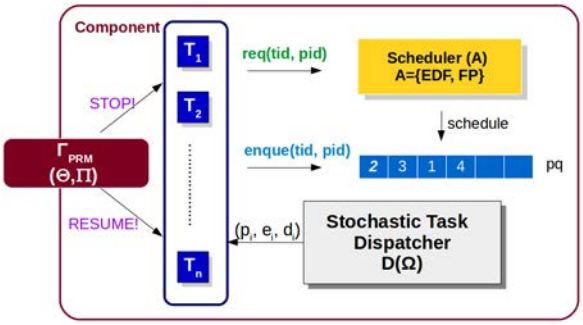


Figure 4.7 – Conceptual model of a scheduling unit of a HSS

Figure 4.7 depicts the conceptual model of a scheduling unit of a HSS: The scheduling unit is composed of a set of tasks (T_i), a scheduler (A), a queue (pq) and a stochastic dispatcher D. The unit is given a PRM ($\Gamma_{PRM}(\Pi, \Theta)$) that is used to analyze the component in a compositional manner. We will call this resource model the supplier.

Our framework supports two types of tasks: periodic task and stochastic task. A periodic task instantiates at the same period. Meanwhile, a stochastic task instantiates with a minimum inter-arrival time by an event. The real-time attributes of stochastic tasks are determined by the stochastic dispatcher D using a set Ω of probability distributions, as shown in Figure 4.7.

Once a job is instantiated by a task, it asks the scheduler for CPU computation time by firing the event req(tid,pid), which inserts the task's Id into the ready queue pq. Then, the scheduler sorts task's identities according to a scheduling policy and chooses the id of the task having the highest priority. This task can carry out its jobs until it finishes the jobs or it is preempted.

The model of the scheduling unit is extended with a resource model $\Gamma_{PRM}(\Pi, \Theta)$ in Figure 4.7 in order to analyze the HSS in a compositional manner. The resource model Γ_{PRM} in Fig. 4.7 can stop and resume the execution of a running task. It determines when to stop and resume according to the timing requirement (Π, Θ). In our work, the TA model of resource model Γ_{PRM} is created such that it supplies the Θ amount of resources at every period Π in a non-deterministic way, i.e. a task that is scheduled to use CPU is allowed to execute only for Θ time units at any time within its period.

Such a non-deterministic behavior simulates every resource supplying patterns of a parent task, including the extreme cases when the longest starvation of the resource assignment occurs, as mentioned in Section 4.1.

4.3.4 Resource Model

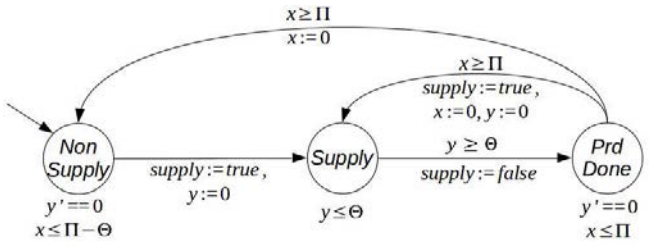


Figure 4.8 – Abstract PRM model in TA

To speed up the schedulability analysis using model checking techniques, we adapt the PRM to generate the extreme cases more often, as depicted in Figure 4.8: The TA model of the PRM uses two clocks, x and y. The clock x is reset every new period and used to measure the current time since a new period has begun. The clock y denotes the time of supplying the resources, so it may progress only when the process resides on the Supply location. A new supply period starts when the clock x reaches Π at location PrdDone. Then, one of the two transitions existing from location PrdDone is taken non-deterministically. One of the transitions leads to location Supply where immediately starts the resource supply. Otherwise, the other transition leads to the location NonSupply that postpones the resource supply up to the time $\Pi - \Theta$ that is the laxity time of the resource supply.

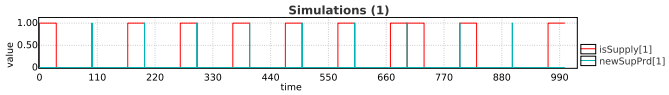


Figure 4.9 – A simulation of PRM behavior model

Figure 4.9 shows a simulation of the PRM behavior that provides a resource for 33 time units every 100 time units. The spike in blue denotes a period of the supply and the graph in red denotes a resource supply. Note that the resource supply begins and terminates in synchronization with the beginning and end of a period, which implies that the longest starvation of the supplying resources can occur extremely often.

4.4 Resolution of the Problems

In this section we detail the techniques we use to solve the scheduling problems presented in Section 4.2.

4.4.1 Checking Correctness and Evaluating Performances with MC and *SMC*

The properties associated to Problem 1 are translated into formal queries in the format of the tool Uppaal MC and Uppaal SMC.

Absence of deadlock We use the CTL formula A[] not deadlock that is checked with model-checking by the tool Uppaal.

Schedulability In our formal models we check schedulability by searching for error states in tasks, that correspond to the tasks missing their deadline. All these error states are identified by a single Boolean variable error, set to true when a task misses a deadlines.

Then, schedulability is analyzed by Uppaal SMC using the following probabilistic query:

simulate nbSim [<=runTime] {error} : 1 : {error}

It asks to perform nbSim simulations of length runTime t.u., until one reaches a state labelled with error. If such a state is found, then the system is not schedulable.

Uppaal SMC performs a quick evaluation of the schedulability. If the system is not schedulable it may find quickly a counterexample execution. However, for an exhaustive result, we rely on model-checking with Uppaal using the CTL formula A[] not error. If the system contains stopwatches the analysis is performed with an over-approximation: if the result is true then the system is surely schedulable; if the result is false it may not be schedulable.

Maximum response time We measure this property using Uppaal SMC with the following query:

E[<= runTime;nbSim](max:t_resp[2])

It runs nbSim simulations of runTime t.u. and it computes the average value over these simulations of the maximum response time of the task with ID 2 (the response time of task 2 is measured in the model with a variable t_resp[2]).

4.4.2 Optimization of a Hierarchical Scheduling System

To optimize a HSS we must determine the minimum budgets for the resource suppliers such that all the scheduling units are schedulable. For this purpose we use the stochastic model of the resource supplier presented in Figure 4.1 that specifies a range of possible budgets Θ. Then we use Uppaal SMC to randomly select a value within this range and check whether the scheduling unit is schedulable with this value.

We use the following probabilistic BLTL formula:

Pr[estBudget[1]<=runTime](<>globalTime>=runTime and error)

It computes the probability distribution of all the possible budget values that are not schedulable. With Uppaal SMC we can plot the probability density distribution in a graph, as shown in Figure 4.10. By looking at the support of this distribution we can determine the minimum budget whose probability is zero, that is the minimum budget necessary to schedule all the tasks of the scheduling unit.

Example We consider the HSS example presented in Figure 4.2. We analyze scheduling unit C_1 to compute the possible budgets for the resource supplier of this scheduling unit (such that the unit is schedulable). In Figure 4.2, this budget was arbitrarily set at 35 over a period of 100. We would determine if this value is sufficient and if it can be lowered.

We set the range of budgets between 0 and 100. Using Uppaal SMC we analyze the probabilistic BLTL formula presented above and we compute the probability density distribution shown in Figure 4.10. It tells us that all the budgets lower than 34 have a non zero probability of being not schedulable. Therefore the minimum budget needed for the scheduling unit is 35 over a period of 100.

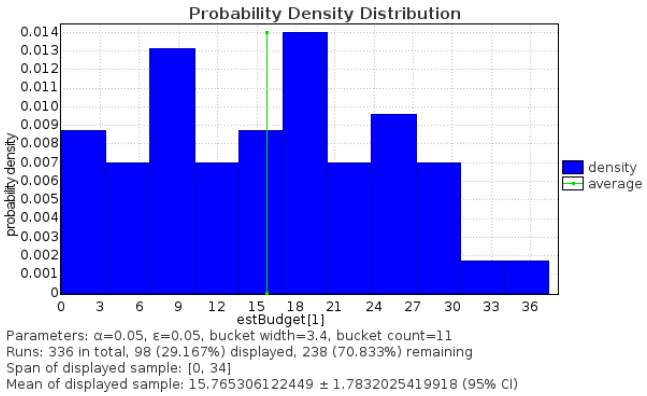


Figure 4.10 – Probability density distribution for the budgets for the scheduling unit C1.

4.5 High Level Framework

This section presents how to use the high-level domain-specific language presented in Section 3.5 in order to design our high-level framework dedicated to the design and analysis of hierarchical scheduling systems.

4.5.1 High-Level Framework for Hierarchical Scheduling Systems

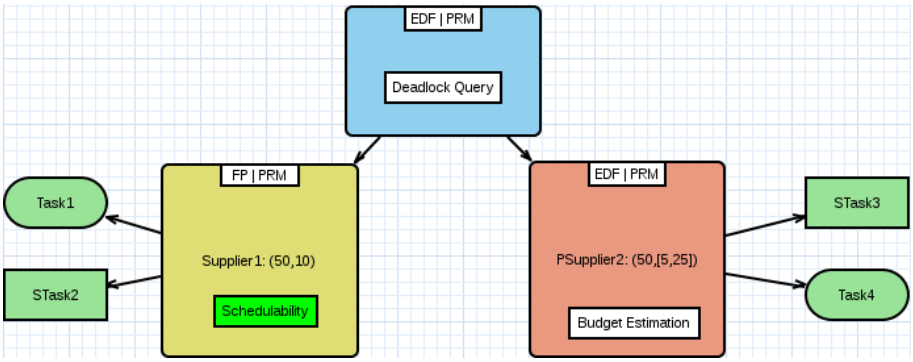


Figure 4.11 – HSS with 3 scheduling units

As presented in Section 4.1, HSS are best represented by a tree structure. This is the format we adopt for our graphical specification of HSS. Figure 4.11 presents an example of an HSS designed in our framework. The nodes of the tree correspond to the components of the scheduling units (tasks, suppliers). In the rest of the section we detail the available components of our high-level language and their configuration parameters.

Resource suppliers TopSupplier(policy), in blue, is the root of the HSS tree. It supplies

the resource to all the scheduling units. Its only parameter is the scheduling policy. Supplier(policy,period,budget), in yellow, are intermediate suppliers (e.g., Supplier1 in Figure 4.11) that receive the resource from an upper level and supply real-time tasks or lower level suppliers. Their parameters are a scheduling policy, a period and a budget within this period. To estimate the necessary budget of a scheduling unit we use a probabilistic supplier, in red, (e.g., PSupplier2 in Figure 4.11) whose budget is chosen randomly between values given in an interval. It is denoted ProbSupplier(policy,period,budget), where budget is an interval of the form [LowerBound,UpperBound].

Tasks Tasks are the leaf of the HSS tree. They represent the time spent for executing some computations. A task is denoted Task(period,deadline,bcet,wcet,priority) and represented in the model with a green rounded box. As presented in Section 2.5.1, we propose a new model of stochastic task whose attributes may be probability distributions. This type of task is denoted STask(period,deadline,execution,priority) and represented by a green rectangle. Here period, deadline and execution are discrete probability distributions. Instead of having a worst case and a best case execution time, we input a probability distribution of execution times.

Queries Queries are associated to the suppliers. The following queries, that correspond to the formal properties presented in Section 4.4.1, are available: deadlock query, schedulability, maximum response time, and budget estimation. In Figure 4.11 for instance, PSupplier2 is assigned a budget estimation query and Supplier1 a schedulability query. Queries that have been verified are colored automatically by the tool, in green if they are satisfied, or in red if they are not satisfied.

We detail below the basic steps performed by our analysis programs to solve the different scheduling problems.

Correctness and performance The program that solved these problems first generates the Uppaal model from the model designed in the graphical interface of the framework. It also generates a text file with the Uppaal query needed for the analysis. It then launches Uppaal or Uppaal SMC and analyses the results. The following results are displayed in the interface:

- The absence of the deadlock is shown in a pop-up window. The color of the query is turned to green or red according to the result.
- The schedulability analysis produces a pop-up window with the result. The color

of the query is turned to green or red. Additionally, if the result is false the color of the task that has missed a deadline is turned to red.

- The measures of maximum response times is displayed in pop-up windows and in the queries.

Optimisation of Hierarchical Scheduling Systems The program that solves this problem generates the Uppaal model of the HSS with a probabilistic supplier, as the one presented in Figure 4.1. It analyses the schedulability query with Uppaal SMC. This generates a probability distribution, as the one presented in Figure 4.10. The program analyses this distribution to determine the minimum budget. It displays the result in a pop-up window.

4.6 Experiments

We apply our framework for HSS to model and verify an avionic scheduling system. We consider the specification of avionic tasks presented in [LLG90]. This is a mixed-critical system with multiple tasks of various criticality running together. We arrange these tasks in a hierarchical scheduling system by grouping tasks from similar functions and criticality (Navigation, Targeting, Weapon control and Controls and displays). Each function is associated to a scheduling unit. The three scheduling units of the most critical functions (Navigation, Targeting and Weapon control) are further grouped under a "Hard-Subsystem" scheduling unit. These results in the hierarchical scheduling systems are presented in Figure 4.12.

The goal of our study is to determine if the complete system is schedulable and to find appropriate parameters for each scheduling unit, such that they are all schedulable.

High-level model We design the HSS in our domain-specific tool generated by Cinco, using the high-level language presented in Section 4.5.1. Sporadic tasks are modelled with stochastic task nodes and are associated to probability distributions. To estimate their necessary budget, each scheduling unit is modelled using a probabilistic supplier.

Verification procedure We analyze each scheduling unit, starting from the bottom, with the budget estimation query. We configure the scheduling unit, by selecting several values for the period of the probabilistic supplier. The period must be lower than the minimum period of the tasks being supplied. Then, we configure the minimum and the maximum budget for the estimation between $[1, period]$. The tool computes the minimum

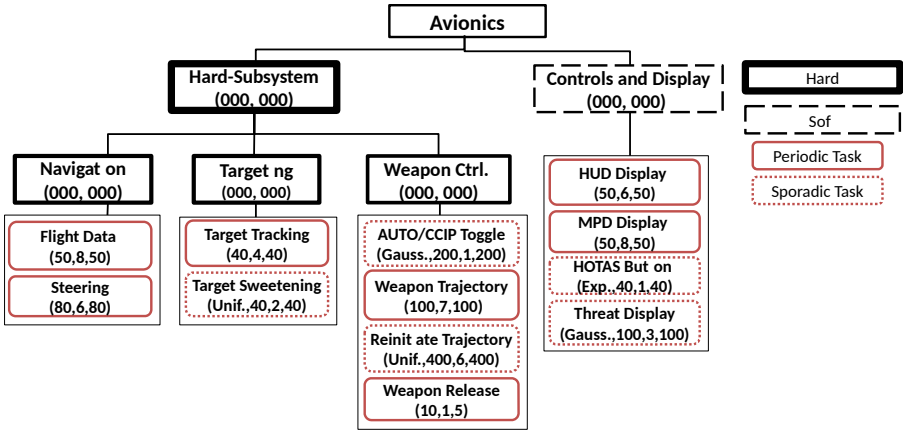


Figure 4.12 – Hierarchical scheduling of avionic tasks

budget such that the tasks are schedulable. The ratio *budget/period* gives us the load factor of the scheduling unit. Our goal is to find the lowest load factor among the choices of possible values for the period.

When all the bottom units have been analyzed we can replace them with normal suppliers using the minimum budget that has been computed. We then repeat the procedure to compute the minimum budget for the upper scheduling units.

Results We present in Figure 4.13 the results obtained from the analysis of the 3 bottom scheduling units (Navigation, Targeting, Weapon control). The graph plots the load factor of the scheduling unit using the minimum budget computed with SMC for several values of the periods. From these results we select the points with the lowest load factor and the highest period. The values that we choose are listed in Table 4.1.

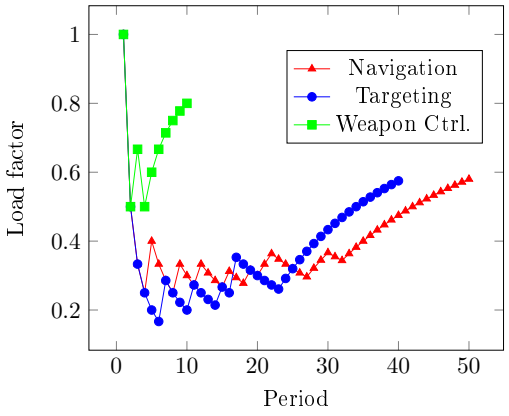


Figure 4.13 – Budget estimation for Navigation, Targeting and Weapon control

Unit	Period	Budget	Load factor
Navigation	8	2	0.25
Targeting	6	1	0.17
Weapon Ctrl.	4	2	0.5
Hard-Subsystem	4	4	1
Controls and Display	3	1	0.33

Table 4.1 – Minimum budget for the scheduling units

We can now replace these probabilistic Suppliers with normal suppliers and confirm the schedulability of the units using the schedulability query, that is checked either with MC or SMC.

We then determine the period and the budget for the Hard-Subsystem unit. Its period must be lower than 4, the chosen period of the Weapon control unit. Since the combined load factor of the 3 lower scheduling units is 0.92, only a budget of 4 over 4 can schedule the Hard Subsystem unit, which we verify with the schedulability query.

We also determine the necessary budget for the Controls and display scheduling units. We found the best budget to be 1 over a period of 3.

From our results we conclude that the two upper scheduling units (Hard Subsystem and Controls and Display) are each schedulable. However since the load factor of the Hard Subsystem is already 1, it cannot be scheduled with the second unit using the same resources.

Chapter 5

Energy Consumption For Multi-Processor Scheduling Systems

Let consider the energy consumption problem on a multiprocessor platform. In this chapter we present two new techniques for scheduling analysis. The first performs runtime monitoring using the CUSUM algorithm to detect alarming change in the system. The second performs optimization using efficient statistical techniques. In this chapter we present our framework on which we implement these two techniques, then we illustrate our framework on two case studies.

Key Contributions. The key contributions in this chapter are as follows:

- New formal models for specifying complex scheduling system with models for multi-processor scheduling systems and energy measure.
- A high-level framework for specifying and verifying scheduling problems. It is automatically generated using a meta-modeling approach.
- Two new techniques for solving scheduling problems. The first one optimizes multi-processor scheduling systems. The second one performs runtime monitoring to detect expected events.
- A case-study that demonstrates the high-level framework and the verification techniques.

5.1 Introduction

Many Cyber Physical Systems (CPS) are mission critical systems. It means that these systems must complete their missions within a period of time. In some cases the system must also complete its mission with limited resources: in particular a limited amount of energy. Number of researches focused on analyzing these systems by verifying that the system can accomplish its mission only using its initial budget of energy.

Besides schedulability, various objectives can be asked upon the scheduler. One of these can be to measure and minimize the energy consumption. This is a great concern in energy limited systems, like cell phones or satellites, and more generally the power consumption of computing devices is an emerging topic.

In this chapter we present a new optimization technique for multi-processor scheduling systems. It determines optimal mappings from tasks to processors in order to minimize the energy consumption of the system and/or response time. We propose algorithms that use statistical tests (ANOVA and TukeyHSD) to determine the optimal mappings.

Related Work Scheduling problems with energy costs are studied in [ORC15]. This work studies an energy-flexible flow shop scheduling problem, that is a multi-objective optimization problem whose goal is to minimize both overall completion time and global energy consumption. It employs stochastic local search techniques. We address a similar problem in our framework for multi-processor scheduling systems. Instead of execution modes and machines switch-off, the configuration options that we study are assignments of tasks to processors and we use statistical model-checking combined with the ANOVA technique to estimate energy cost and response time.

There also exists approaches that perform scheduling via timeline-based planning. The work of [CMR17] proposes such an approach and uses timed game automata (timed automata with controllable and uncontrollable actions) to find strategies for the timeline-based planning problem. Timed games and satisfiability modulo theory are also used in [CMR16] to solve control problems with temporal constraints. Our model-based approach is also based on extensions of timed automata but we mostly rely on statistical model-checking for finding solutions to the scheduling problems. This allows us to consider more complex scheduling systems, with sporadic tasks, hierarchical scheduling or energy constraints, that would not be solvable using exhaustive techniques such as model-checking or timed games.

Another work presented in [CFO+11] uses a logic-based approaches. The planning problem is encoded in a high level action notation modeling language [SFC08] and then

translated into linear temporal logic modulo rational arithmetic formula. In our work, we introduce new high level graphical notations for complex scheduling problems. These notations are specific to the scheduling framework being studied (either hierarchical or multi-processor scheduling systems). These domain-specific notations allow to have a simpler and more accurate description of a scheduling system than using existing formalisms. Moreover we can rely on the tool generator Cinco for easily generating a domain-specific tool that implements a graphical editor for these notations.

5.2 Formalisation

To formalize multi-processor scheduling systems with energy resources, we use formal models presented in Section. 2.5.1, and we extend formal models to take into account the energy consumption.

Adding energy to our timed automata models requires to extend the models with continuous variables and costs, using priced timed automata and hybrid automata. For measuring energy consumption we consider a multi-processor scheduling system with processors of different capabilities (frequencies). Based on CMOS technology, the power consumption is dominated by dynamic power dissipation P_d when the processor is used by some task, given by the following formula:

$$P_d = C * V^2 * f$$

where C is the capacitance, V the voltage and f the frequency. The processor speed is almost linear to the voltage:

$$f = k \cdot \frac{V - V_t}{V}$$

where k is a constant and V_t is the threshold voltage. We therefore get an approximated power consumption:

$$P_d = k' * f^3$$

k' being a constant ($C * k$). In our study we want to compare different configurations of the system according to the trade-off between speed (higher frequency) and energy consumption (lower frequency). We will therefore consider that the different configurations have the same constant characteristics by setting $k' = 1$ and only compare the energy consumption using the formula:

$$P_d = f^3$$

Then, our formal models for multi-processor scheduling systems define a set of pro-

cessors, each having a frequency and its own scheduler. Processors can use different scheduling algorithms. Tasks are statically assigned to one processor.

To measure the energy consumption of the system, we add to our formal models a simple *PTA*. It defines a cost variable energy whose rate is energy'=totalPow, where totalPow is the power of all the running processors. If we increase the speed of a processor (the frequency f) we increase the energy consumption, but in return the task using the processor can run faster. We take this into account in our task model. The stopwatch clock t_et[tid] becomes a continuous variable that progresses at a rate t_et[tid]'=f, where f is the frequency of the processor that executes the task.

5.3 Scheduling Problems

In this section we present the different problems that we want to solve on scheduling systems.

Problem 1: Correctness and performance To analyze the correctness of multi-processor scheduling systems, we evaluate the properties presented in Section 4.2, i.e. (absence of deadlock, schedulability, and maximum response time), and additionally we evaluate energy consumption property:

Energy consumption: We measure the average and maximum energy consumed by the system over a period of time.

Problem 2: Optimization, Configuration We consider a multi-processor system, with CPUs having different frequencies, and a set of real-time tasks. Our goal is to assign each task to a CPU. Then we evaluate the configurations of the scheduling system in terms of schedulability, response time and energy consumption.

Problem 3: Change detection We now want to monitor our scheduling system in order to detect emerging behaviors or an expecting event. We consider a property of the system, based on the measures presented in Problem 1, e.g., the energy is always lower than a given value. We consider our system as a stochastic process and we evaluate the property at regular steps during an execution. This allows us to compute at runtime the probability to satisfy the property. Then, our goal is to detect an abrupt variation of this probability, which will be the sign that some event happened.

Formally, let S be a set of states and $T \subseteq \mathbb{R}$ be a timed domain. A stochastic process (S, T) is a family of random variables $\mathcal{X} = \{X_t \mid t \in T\}$, each X_t having range S. An

execution of the stochastic process is any sequence of observations $\{x_t \in S \mid t \in T\}$ of the random variables $X_t \in \mathcal{X}$. It can be represented as a sequence $\pi = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n)$, such that $s_i \in S$ and $t_i \in T$, with time stamps monotonically increasing, e.g. $t_i < t_{i+1}$. Let $0 \leq i \leq n$, we denote $\pi^i = (s_i, t_i), \ldots, (s_n, t_n)$ the suffix of π starting at position i.

Let φ be a property that can be evaluated to true or false on an execution. We consider a sequence of Bernoulli variables Y_i such that $Y_i = 1$ iff $\pi^i \models \varphi$. We define that the execution π satisfies a change $\tau = Pr[\pi \models \varphi] \geq p_{change}$, where $p_{change} \in]0, 1[$, iff $Pr[Y_i = 1] < p_{change}$ for $t_i < t$ and $Pr[Y_i = 1] \geq p_{change}$ for $t_i \geq t$. The first time t_i when this is detected it is the time of change.

Consider for instance a stochastic scheduling system as presented previously. We can evaluate at regular time intervals the probability that the energy consumption during the time interval exceeds a given value. This probability may change at runtime if the load of the scheduling system changes, because for instance some new tasks have been added. With change detection we would like to raise an alarm when the change occurs.

5.4 Methods

In this section we detail the techniques we used to solve the problems presented in Section. 5.3.

5.4.1 Checking Correctness and Evaluating Performances with MC and SMC

To solve the first three problems,i.e (Absence of deadlock, Schedulability, and Maximum Response Time), we use the same techniques presented in Section. 4.4.1.

Energy consumption We first measure the average energy consumed over a period of time. We use the following query:

$$\mathsf{E}[<= \mathsf{runTime};\mathsf{nbSim}](\max: \mathsf{PlatformEnergy.energy})$$

PlatformEnergy is the PTAthat measures the energy using a cost variable energy. Uppaal SMC runs nbSim simulations of runTime t.u. and it computes the average value of the energy at the end of these simulations.

We can also check if the energy is always lower than a maximum value. We use the

following probabilistic BLTL formula:

$$\mathsf{Pr}[<= \mathsf{runTime}]([]\ \mathsf{PlatformEnergy.energy} <= \mathsf{maxEnergy})$$

where runTime is the time length for the simulations and maxEnergy is an energy bound. With Uppaal SMC we compute the probability that the property is satisfied.

5.4.2 Optimization of a Multi-processor Scheduling System with ANOVA

We consider a set of CPUs, $\mathcal{C} = (CPU_1, CPU_2, \ldots, CPU_k)$ and a set of real-time tasks $\mathcal{T} = (T_1, T_2, \ldots, T_l)$. A multi-processor scheduling system is configured by specifying a mapping $\gamma : \mathcal{T} \mapsto \mathcal{C}$.

For each possible mapping, we would like to evaluate first, if the system is schedulable, and second, the average energy consumption and/or the maximum response time of a task $T_i \in \mathcal{T}$. The Uppaal query that we use to evaluate the energy consumption is:

$$\varphi_e = \mathsf{simulate\ nbSim}[<= \mathsf{runTime}]\{\mathsf{PlatformEnergy.energy}\} : 1 : \mathsf{false}$$

and to evaluate the maximum response time of a task with id i:

$$\varphi_t = \mathsf{simulate\ nbSim}[<= \mathsf{runTime}]\{\mathsf{max\_resp[i]}\ \} : 1 : \mathsf{false}$$

Finally we would like to compare the different configurations in order to select a schedulable configuration that has a minimum energy consumption and/or a minimum response time. If we want to achieve both objectives we are faced with a multi-objective optimisation problem. A simple solution would be to analyze each configuration with SMC experiments in order to compute values for the energy consumption and the response time. However this requires a lot of simulations per configuration to be able to compare them, as the confidence intervals should not overlap. Fortunately, there exists a more efficient statistical technique to solve this problem that is called analysis of variance (ANOVA). The test has already been used to perform optimisation with SMC [DDGL+13]. We propose in this chapter new algorithms based on this test.

ANOVA is a statistical test used to compare several probability distributions. We use it in a single factor configuration with a fixed effects model, as presented in [Mon06]. We have k treatments of a single factor (the system configuration defined by a mapping γ) that we wish to compare. For each $1 \leq i \leq k$, the observed response for treatment i is a random variable X_i (the energy or response time) for which we draw n random values

$x_{i,1}, \ldots, x_{i,n}$ (computed by running n simulations of the system using the mapping γ_i and a property φ_e or φ_t). We denote $\overline{X_i}$ the mean of the random variable X_i and $\overline{X}$ the total mean all the values. ANOVA tests the null hypothesis that all the means of the treatments are equal, against the alternative hypothesis that at least two treatments have different means.

ANOVA is based on a comparison between the variability observed between the treatments and the variability observed within the treatments using the following F-value:

$$F = \frac{1/(k-1)\sum_{i=1}^{k}(\overline{X_i} - \overline{X})^2}{1/(n-k)\sum_{i=1}^{k}\sum_{j=1}^{n}(X_{i,j} - \overline{X_i})^2}$$

If the null hypothesis is true this F-value should follow a F-distribution defined by the degrees of freedom of the experiment, that are $k-1$ and $n-k$. To determine if the null hypothesis holds a classical hypothesis testing solution is to compute the P-value of the test. The P-value is the probability of observing a more extreme F-value than the actual result. It corresponds to the area under the probability density function of the distribution greater than the F-value, as shown in Figure. 5.1. Therefore, the lower the P-value, the lower the probability that the F-value computed actually follows the F-distribution, and consequently the more likely the null hypothesis should be rejected. To make a decision we compare this P-value to a confidence level α, for instance $\alpha = 0.05$ for a 95% confidence. If P-value $\leq \alpha$ then the null hypothesis is rejected, i.e. some treatments have different means, with a 5% chance of making a Type I error.

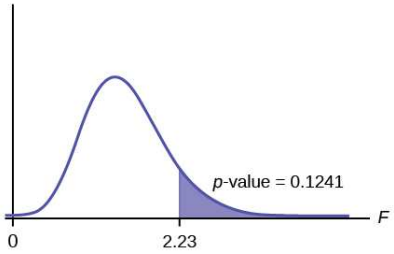


Figure 5.1 – F-distribution example with the p-value computed for F=2.23.

Tukey HSD If ANOVA shows that the means of the treatments are significantly different, then we would like to determine which treatments differ in order to compare them. In [DDGL+13] the test was used with treatments that are continuous variables (temperature thresholds). In their context, using ANOVA alone, the authors were able to valid a linear regression over the continuous variables in order to optimize the system

In our context, the treatments (the different mappings) cannot be compared directly with ANOVA. The result of the test is only that at least two treatments differ, but we do not know which ones. Therefore we need an additional test to compare the treatments. This cannot simply be done by a series of pairwise T-test, as it would greatly increase the likelihood of false positive.

There exists however a multiple comparison test called Tukey HSD (Tukey's Honest Significant Difference test) that compares the means of every treatments to the means of every other treatment. It computes the pairwise differences $\overline{X_i} - \overline{X_j}$ with a confidence interval. If the endpoints of the confidence interval have the same sign (both positive or both are negative), then 0 is not in the interval and we conclude that the means are different. If the endpoints of the confidence interval have opposite signs, then 0 is in the interval and we cannot determine whether the means are equal or different. Tukey HSD is based on a studentized range distribution. As for the ANOVA test, each comparison of the Tukey test can be associated to a P-value to measure the level of significance.

Note that if the number of mappings is reduced to two, then Tukey HSD should be replaced by a T-test.

Algorithms Using the two statistical tests previously presented, we propose two new algorithms to optimize multi-processor scheduling systems. The algorithms determine dynamically the number of simulations needed to compare the means of energy consumption and/or response time with a sufficient confidence. Algorithm 1 has a single objective (minimizing the energy consumption or the response time), while Algorithm 2 considers both objectives simultaneously.

In these algorithms `Simulate` is a function that performs n simulation of a mapping γ and computes the values specified in the property φ (e.g. energy consumption or the response time). `RunANOVA` runs the ANOVA test on the simulations to determine if the mappings values are significantly different. It returns the P-value of the test. `RunTukeyHSDSingle` runs the Tukey HSD test on the simulations and determines the best mappings, which can be a single mapping, or a set of mappings that cannot be distinguished because there is not enough significance, or because they have the same probability distributions. `RunTukeyHSDMulti` runs the Tukey HSD test and returns True if all the differences have either a significant difference (P-value $\leq \alpha$) or are equal (P-value $\geq 1 - \alpha$). `ComputeMeans` computes the means of the values for each mapping given in parameter over all the simulations of the mapping.

Finally we are able to select the "best" configurations. Let $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ be the set of schedulable configurations. We denote $\mathsf{energy}(\gamma_i)$ the average energy consumed

Algorithm 1: Single objective multiprocessor optimization

Input:

Γ: list of mappings
n: initial number of simulations
α: confidence level
$\varphi \in \{\varphi_e, \varphi_t\}$: simulation property.

Output:

bestMappings: set of mappings with minimum energy consumption or response time.
min: minimum mean of energy consumption or response time.

Let $conf$ be a Boolean, initialised $conf \leftarrow False$
$bestMappings \leftarrow \Gamma$
foreach $\gamma \in \Gamma$ **do**
  Let $simulations[\gamma]$ be the set of simulations of the mapping γ, initially empty.
while *not* $conf$ **do**
  foreach $\gamma \in bestMappings$ **do**
    $simulations[\gamma] \leftarrow simulations[\gamma] \cup \texttt{Simulate}(\gamma, n, \varphi)$
  P-value $\leftarrow \texttt{RunANOVA}(simulations)$
  if *P-value* $\geq 1 - \alpha$ **then**
    $conf \leftarrow True$
  if *P-value* $\leq \alpha$ **then**
    $bestMappings \leftarrow \texttt{RunTukeyHSDSingle}(simulations, \alpha)$
    if $|bestMappings| = 1$ **then**
      $conf \leftarrow True$
    foreach $\gamma \in \Gamma \setminus bestMappings$ **do**
      Remove $simulations[\gamma]$ from $simulations$
$min \leftarrow \mathsf{Min}(\texttt{ComputeMeans}(simulations, bestMappings))$

Algorithm 2: Multi-objectives multiprocessor optimization

Input:

Γ: list of mappings
n: initial number of simulations
α: confidence level
φ_e, φ_t: simulation properties

Output:

$means\_e, means\_t$: means of energy consumption and response time for each mapping

Let $conf\_e$ and $conf\_t$ be Booleans, initialised $conf\_e \leftarrow False$ and $conf\_t \leftarrow False$
foreach $\gamma \in \Gamma$ **do**
  Let $simulations\_e[\gamma]$ be the measures of energy consumption of the mapping γ, initially empty.
  Let $simulations\_t[\gamma]$ be the measures of response time of the mapping γ, initially empty.
while *not* $conf\_e$ *or not* $conf\_t$ **do**
  if *not* $conf\_e$ **then**
    foreach $\gamma \in \Gamma$ **do**
      $simulations\_e[\gamma] \leftarrow simulations\_e[\gamma] \cup \texttt{Simulate}(\gamma, n, \varphi_e)$
    P-value $\leftarrow \texttt{RunANOVA}(simulations\_e)$
    if *P-value* $\geq 1 - \alpha$ **then**
      $conf\_e \leftarrow True$
    if *P-value* $\leq \alpha$ **then**
      $conf\_e \leftarrow \texttt{RunTukeyHSDMulti}(simulations\_e, \alpha)$
  if *not* $conf\_t$ **then**
    foreach $\gamma \in \Gamma$ **do**
      $simulations\_t[\gamma] \leftarrow simulations\_t[\gamma] \cup \texttt{Simulate}(\gamma, n, \varphi_t)$
    P-value $\leftarrow \texttt{RunANOVA}(simulations\_t)$
    if *P-value* $\geq 1 - \alpha$ **then**
      $conf\_t \leftarrow True$
    if *P-value* $\leq \alpha$ **then**
      $conf\_t \leftarrow \texttt{RunTukeyHSDMulti}(simulations\_t, \alpha)$
$means\_e \leftarrow \texttt{ComputeMeans}(simulations\_e, \Gamma)$
$means\_t \leftarrow \texttt{ComputeMeans}(simulations\_t, \Gamma)$

over a fixed period of time and $\mathsf{resp}(\gamma_i)$ the maximum response time of one task. If we consider only one objective we select the configuration with the minimum estimated value for $\mathsf{energy}(\gamma_i)$ or $\mathsf{resp}(\gamma_i)$. If we consider both objectives simultaneously we should select a configuration that is Pareto-efficient. Formally, a configuration γ_i is Pareto-efficient if there exists no other configuration γ_j such that $\mathsf{energy}(\gamma_j) \leq \mathsf{energy}(\gamma_i)$ and $\mathsf{resp}(\gamma_j) \leq \mathsf{resp}(\gamma_i)$. We can plot the results on a graph and draw a Pareto-efficiency curve that links the Pareto-efficient configurations.

We consider that Algorithm 2 produces the results given in the graph shown in Figure. 5.2, with values energy and resp for a set of configurations from A to F. Configurations A to D are Pareto-efficient. Configuration E is not Pareto-efficient because $\mathsf{energy}(C) < \mathsf{energy}(E)$ and $\mathsf{resp}(C) < \mathsf{resp}(E)$. Similarly, configuration F is no Pareto-efficient because $\mathsf{resp}(B) < \mathsf{resp}(F)$ and $\mathsf{energy}(B) = \mathsf{energy}(F)$.

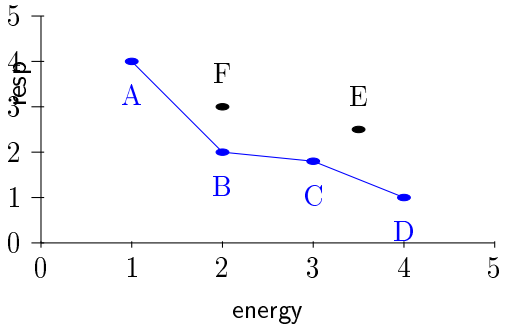


Figure 5.2 – Pareto-efficiency curve

5.4.3 Change Detection with CUSUM

CUSUM [Pag54, BN93] is a statistical algorithm used for monitoring change detection. The principle is to monitor the evolution of a probability measure at successive positions during a single execution of the system. The algorithm then detects the position where it drastically changes from original expectation. We have previously adopted the CUSUM algorithm to monitor a BLTL property over an execution trace of a stochastic process and to detect the position in the trace when the probability to satisfy the property changes significantly [LT16].

Let $\pi = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n)$ be an execution of the stochastic process and φ a BLTL property to monitor during this execution. As defined in Problem 3, Y_i are Bernoulli variables such that $Y_i = 1$ iff $\pi^i \models \varphi$. We note $p_k = Pr[Y_i = 1 | i <= k]$ the probability of satisfying φ from (s_0, t_0) to the state (s_k, t_k). CUSUM will decide between

the two following hypothesis:

- $H_0 : \forall\ k,\ 0 \leq k \leq n, p_k < p_{change}$, *i.e.*, no change occurs.
- $H_1 : \exists\ l,\ 0 \leq l \leq n$ such that the change occurs at time t_l: $\forall k,\ 0 \leq k \leq n$, we have $t_k < t_l \implies p_k < p_{change}$ and $t_k \geq t_l \implies p_k \geq p_{change}$.

We assume that we know the initial probability $p_{init} < p_{change}$ of $Pr[\pi \models \varphi]$ before the change occurs. One solution is to estimate this probability with the Monte Carlo algorithm using an ideal version of the system in which no change occurs.

Like the Sequential Probability Ratio Test (SPRT) [Wal45], the CUSUM comparison is based on a likelihood-ratio test: it consists in computing the cumulative sum S_k of the logarithm of the likelihood-ratios s_i over the sequence of samples $Y_1, \ldots Y_k$. The change is detected as soon as S_k satisfies a stopping rule.

$$S_k = \sum_{i=1}^{k} s_i \quad s_i = \begin{cases} \ln \frac{p_{change}}{p_{init}}, & \text{if } X_i = 1 \\ \ln \frac{1-p_{change}}{1-p_{init}}, & \text{otherwise} \end{cases}$$

The typical behavior of the cumulative sum S_k is a global decreasing before the change, and a sharp increase after the change. Then the stopping rules purpose is to detect when the positive drift is sufficiently relevant to detect the change. It consists in saving $m_k = \min_{1 \leq i \leq k}\ S_i$, the minimal value of CUSUM, and comparing it with the current value. If the distance is sufficiently great, the stopping decision is taken, *i.e.*, an alarm is raised at a time $t_a = \min\{t_k : S_k - m_k \geq \lambda\}$, where λ is a sensitivity threshold.

CUSUM Calibration

The efficiency of the CUSUM algorithm depends on several parameters. First, it is important to note that the likelihood-ratio test assumes that the considered samples are independent. This assumption may be difficult to ensure over a single execution of a system, but heuristic solutions exist to guarantee independence. One of them is to introduce delays between the samples. In that case Monte Carlo SMC analysis can evaluate the correlation between the samples, and help to select appropriate delays.

Second, the CUSUM sensitivity depends on the choice of the threshold λ. A smaller value increases the sensitivity, *i.e.*, the false alarms rate. A false alarm is a change detection at a time when no relevant event actually occurs in the system. Conversely, big values may delay the detection of the change. The false alarms rate of CUSUM is

defined as $E[t_a]$, the expected time of an alarm raised by CUSUM while the system is still running before the change occurs. Ideally, this value must be the biggest as possible ($E[t_a] \to +\infty$). The detection delay is defined as the expected time between the actual change at time t and the alarm time t_a raised by CUSUM ($E[t_a - t \mid t < t_a]$). Ideally, this value has to be as small as possible.

To calibrate the sensitivity a solution is to use two versions of the model: one in which the change never occurs and one in which it always occurs. Running the CUSUM on the first model allows to determine the minimum sensitivity such that no detection occurs. Then, the CUSUM is run on the second model to determine the detection delay.

5.5 High Level Framework

This section presents how to use the high-level domain-specific language presented in Section. 3.5 in order to represent our high-level framework dedicated to the design and analysis of the multi-processor scheduling systems with energy constraints.

5.5.1 High-Level Framework for Multi-Processor Scheduling Systems

For the design of CPS with multi-processor we consider a two-layer approach as proposed in [KLL+15a]. The first layer models the hardware platform, with a scheduling system composed of real-time tasks and CPUs. The second layer models the application that is composed of a set of actions. The link between the two layers is implemented by a mapping from actions to tasks, that specifies for each action of the application on which task it is intended to run. In our current framework this mapping is static and determined before the execution.

This design allows a separation of concerns that facilitates the verification of formal properties:

- Scheduling properties are verified on the platform layer only.
- Logical properties of the application are verified on the application layer only.
- Energy consumption or execution time properties need to consider both layers simultaneously.

We have implemented with Cinco a high-level framework that allows to design a two-layer multi-processor scheduling system.

Platform Layer The platform layer is composed of a set of processors and a set of real-time tasks. Each processor has its own scheduling mechanism and is parametrized by its own frequency. The frequency defines the speed of the processor and its energy consumption when running. Real-time tasks can be either hard real-time, with a deadline, a period and execution times, or soft real-time, with only period and execution times. Tasks are statically assigned to a processor. A model of a platform layer designed in our framework is presented in Figure. 5.3. This model is translated into a set of timed automata, using the models presented in Section **??**.

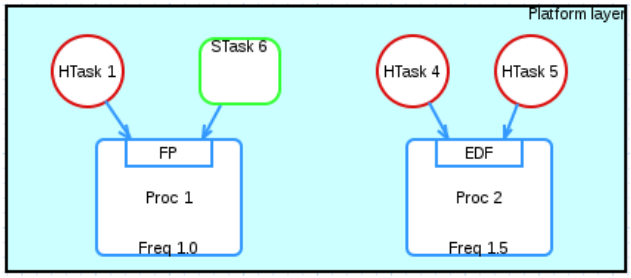


Figure 5.3 – Platform layer with 2 processors, 3 hard real-time tasks and 1 soft real-time task

Application Layer Applications running in CPS are unlimited, with no fixed design. To demonstrate the use of our framework we consider a simple design methodology for writing applications with stochastic behavior. Our application is composed of a set of components. Each component consists in a sequence of actions. Each action has a delay mechanism, implemented with either a uniform or an exponential probability distribution, and minimum and maximum execution time. Actions are also parametrized with an energy consumption parameter (between $[0, 1]$) that defines how much power the action will take from the CPU. The semantics of this language is to execute each component in parallel by running their actions iteratively. A component that has completed its last action will continue in a loop with the execution of the first action. An example of application is presented in Figure. 5.4. These models are translated in a set of stochastic timed automata.

Mapping between application and platform The mapping between the two layers is done by linking each action of the application to a real-time task, as presented in Figure. 5.5.

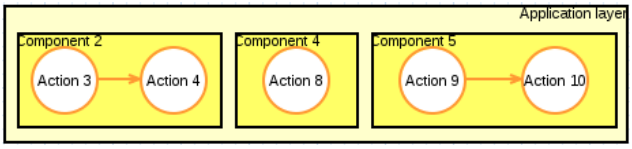


Figure 5.4 – Application layer with 3 components and 5 actions

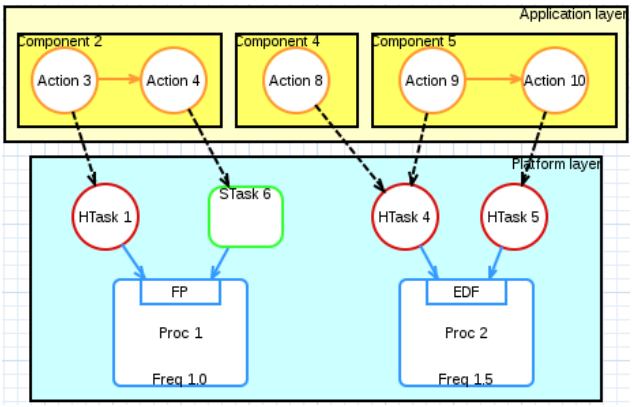


Figure 5.5 – Mapping between application layer and platform layer

Queries: In this framework we consider different type of queries, some of them associated to the platform and some of them associated to the application. On the platform layer we verify schedulability queries and we determine optimal mapping between tasks and processors with ANOVA, as presented in Section 5.4.2. On the application layer we measure average energy consumption and we use CUSUM to detect changes in the application behavior.

5.5.2 Implementation of the Framework and Tool Chain

To implement our domain-specific framework dedicated to the design and analysis of multi-processor scheduling systems, we use the tool chain described in Section. 3.5.2. We detail below the basic steps performed by our analysis programs to solve the different scheduling problems.

Optimisation of Multi-Processor Scheduling Systems We have implemented Algorithms 1 and 2. Our program generates a set of Uppaal models, each corresponding to a configuration of the system with a mapping from tasks to processors. It then runs the optimisation algorithm. This algorithm launches some simulations with Uppaal SMC and extracts the numerical results. The results are written in some temporary files. that

are analysed with the statistical tool R to perform the RunANOVA, RunTukeyHSDSingle, RunTukeyHSDMulti and ComputeMeans procedures. According to the results the program determines if more simulations are needed, or outputs the result.

For the single objective problem the program directly shows the optimal mapping by drawing it on the interface using the transformation API of Cinco.

For the multi-objectives problem the program opens a pop-up window and draws into the Pareto diagram. This window allows to select one of the Pareto-efficient mapping that is then drawn on the interface.

Change detection The program that performs change detection implements the CUSUM algorithm. It first generates the Uppaal model and it will run the CUSUM algorithm on this model several times. For each execution, it generates with Uppaal SMC a simulation trace that corresponds to the total length of the experiment. It then splits this execution into a set of samples and it analyses each sample to evaluate the query and update the CUSUM ratio. If the value of ratio exceeds the sensitivity threshold it outputs a detection with the detection time in a pop-up window.

5.6 Experiments

This section presents an example of a multi-processor scheduling system designed and analyzed in our framework. We first describe the model and then we present the experiments performed in our framework to solve the problems presented in Section 5.4.

5.6.1 Example

The proposed example is composed of two layers, following the modeling framework presented in Section 5.5, a Platform layer and an Application layer.

The Platform Layer is composed of 3 periodic hard real-time tasks and 2 processors. The tasks parameters are configured according to the following order: Task(period,deadline, bcet,wcet,priority), and are respectively $\mathcal{T}_1(10, 10, 3, 4, 9)$, $\mathcal{T}_2(20, 20, 5, 6, 8)$ and $\mathcal{T}_3(30, 30, 6, 8, 7)$. The 2 processors are P_1, with a 1.5 MHz frequency and a FP scheduling policy,and P_2, with a 1.0 MHz frequency and an EDF scheduling policy. We initially distribute $\mathcal{T}_1$ and $\mathcal{T}_2$ on processor P_1, while $\mathcal{T}_3$ is running alone on processor P_2.

The Application Layer consists of 3 components, each composed of a succession of actions as presented in Figure. 5.6. Component C_1 is composed of actions A_1, A_2

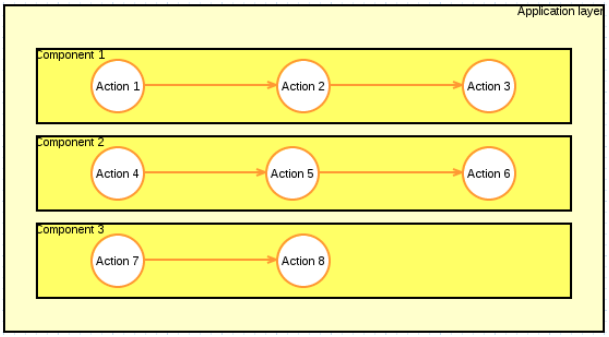


Figure 5.6 – Application layer of our case-study model

and A_3, whose execution time is respectively 4, 3, 5. These actions are executed on task T_1. Component C_2 is composed of actions A_4, A_5 and A_6, whose execution time is respectively 4, 5, 5. These actions are executed on task T_2. Component C_3 is composed of actions A_7 and A_8, whose execution time is respectively 5 and 6. These actions are executed on task T_3.

Each action has an energy parameter that defines how much energy it takes when running on a processor, with a maximum value of 1 meaning that it takes the full power of the processor.

Finally, random delays with uniform distributions are set between the execution of each actions. As explained in Section 5.5 the execution of each component is cyclic: it runs sequentially each action, and then starts again at the first action. Action A_8 is additionally delayed, such that is starts only after 50 or 100 executions of action A_7. Using the change detection problem and CUSUM we will try in our experiments to detect the beginning of execution of this action.

5.6.2 Checking Correctness and Evaluating Performances

Experiments Using SMC we perform the following experiments on the initial model:

1. Schedulability analysis.
2. Measure of energy consumption, considering the platform only and both the platform and the application.
3. Measure of the maximum response time for each task.

We use 100 simulations and a runtime of 60 t.u. This runtime allows to execute the model over the smallest common multiple of the periods of our tasks (the hyper-period).

Results The results of these experiments are presented in Table 5.1. We give for each result the time taken by the analysis. If the result is a measure we give its estimated value and the confidence interval that corresponds to the SMC analysis.

Analysis	Result	Time (s)
Schedulability	True	4.47
Energy consumption (platform)	69.408 ± 0.46	1
Energy consumption (application)	37.69 ± 0.72	4.2
Maximum response time of T_1	3.86 ± 0.025	3.8
Maximum response time of T_2	9.36 ± 0.055	3.78
Maximum response time of T_3	4.89 ± 0.061	3.78

Table 5.1 – Correctness and performances analyzed with Uppaal SMC

5.6.3 Optimization with ANOVA

Experiment This second experiment consists in finding optimal mappings between tasks and processors, such that the system is schedulable and has optimal performances. Therefore we start by removing in our model the mapping used in the previous section. Then we use the ANOVA method with the multi-objectives Algorithm 2 proposed in Section 5.4.2. Our two objectives are to minimize the energy consumption of the scheduling system and the maximum response time of one of the tasks. The result is a Pareto efficiency diagram.

For this experiment we will use SMC with 100 simulations to determine schedulability, and Algorithm 2 with ANOVA and Tukey HSD techniques with a 95% confidence.

Results Table 5.2 presents the results of executing Algorithm 2. We perform 3 executions of the algorithm (Exec. 1, Exec. 2 and Exec. 3) that are differentiated according to the task for which we want to minimize the maximum response time. One execution takes approximately 40 seconds. We determine that there are 8 mappings schedulable, simply named mapping-i with i from 1 to 8. Then for each execution we give in column E the energy consumption of the processors, and in column $t(\mathcal{T}_i)$ the maximum response time of a task $\mathcal{T}_i$.

Let us now consider that task $\mathcal{T}_2$ is our critical task for which we want to minimize the maximum response time. From the results given in columns 4 and 5 we can plot in our framework a Pareto diagram in a pop-up window, as shown in Fig. 5.7. From this window we can select one of the Pareto-efficient mapping that will then be automatically

Mapping	Exec. 1		Exec. 2		Exec. 3	
	E	$t(\mathcal{T}_1)$	E	$t(\mathcal{T}_2)$	E	$t(\mathcal{T}_3)$
mapping-1	98.1	2.57	98.1	6.21	97.8	7.15
mapping-2	115	2.58	115	6.24	115	11.7
mapping-3	77.2	2.57	77.3	5.74	77.5	12.2
mapping-4	95.3	2.57	95.1	5.76	94.9	7.18
mapping-5	70.1	3.81	70.2	3.58	70.1	9.99
mapping-6	88.5	3.80	88.5	3.74	88.8	8.29
mapping-7	50.9	3.87	50.8	9.34	50.7	19.3
mapping-8	69	3.86	68.7	9.33	69.2	4.91

Table 5.2 – Optimization of the mapping between tasks and processors according to energy consumption and maximum response time of tasks $\mathcal{T}_1$, $\mathcal{T}_2$ or $\mathcal{T}_3$

applied to the model.

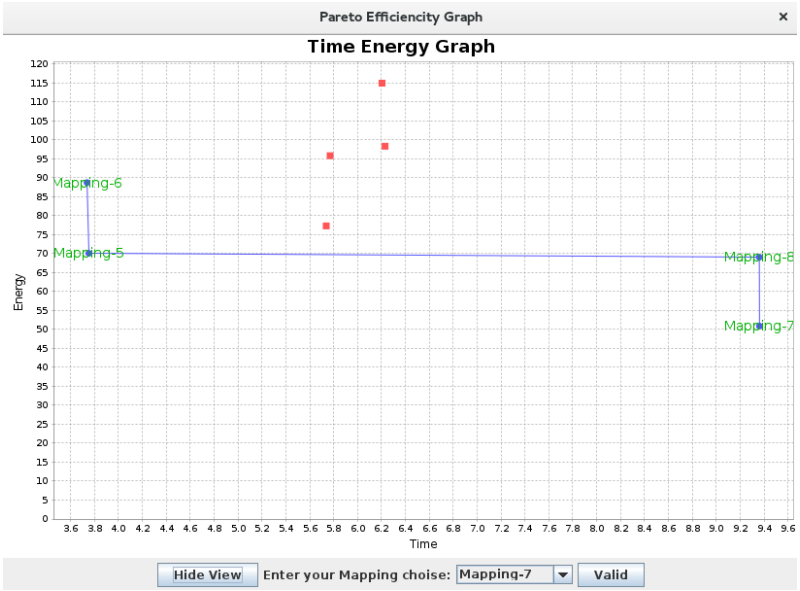


Figure 5.7 – Pareto Efficiency diagram for optimizing energy consumption and maximum response time of task $\mathcal{T}_2$

5.6.4 Change Detection with CUSUM

Experiment In our third experiment we analyze each of the 4 optimal mappings found in the previous experiment and shown in Fig. 5.7. We use the CUSUM algorithm presented in Section 5.4.3 to detect the beginning of execution of action A_8. While action A_7 that precedes A_8 consumes only 80% of the CPU power, A_8 when it starts consuming the full power. This difference should increase the probability that the maximum energy

consumption during a sample exceeds a given level. We consider a sample time of 60 time units, that corresponds to the hyper-period of the executing platform. We will observe at each sample of an execution the probability to exceed the maximum energy consumption. This probability should raise when action A_8 starts executing. We will monitor the variation of this probability during an execution of 300 samples, i.e. 18'000 time units. The CUSUM algorithm detect a change of probability and measure the detection time. We repeat the CUSUM 100 times and we compute the detection time as the average detection time over all the execution of the CUSUM.

To configure the CUSUM algorithm we first need to determine the initial probability. In this example we choose to estimate this probability by executing the optimal model, that is the model without action A_8 that provokes the change. The second parameter that we need to configure is the deviation from the initial probability when the change occurred. This parameter is estimated by computed the energy consumption on a model in which that action is already running at the beginning of the execution.

After fixing these two parameters, we proceed to the calibration of the CUSUM algorithm. This step consists in computing the sensitivity threshold λ. It is done by executing CUSUM on the optimal model, without the action responsible for the change, and using the initial probability and the deviation computed before. The threshold λ will be the minimal value such that no detection is observed for all simulations.

Results We run CUSUM on the set of Pareto-efficient mappings of Figure. 5.2. The analysis of one model takes approximately 20 minutes. The results for each mapping are presented in the following tables. In these tables, the first column (Energy) is the energy level used for the detection, the second column (Init. prob.) is the initial probability, the third column (Deviat.) is the probability deviation, the fourth column (λ) is the sensitivity threshold λ, the fifth and sixth columns (T.Detect) are the detection times, in the cases when action A_8 starts after 50 or 100 executions of A_7.

Table 5.3 presents the results obtained for mapping-6, that executes $\mathcal{T}_1$ on processor P_1 and $\mathcal{T}_2, \mathcal{T}_3$ on processor P_2. With this mapping we can measure experimentally with Uppaal that action A_8 starts after approximately 1470 t.u. when its start parameter is 50, and 2950 t.u. when its start parameter is 100.

Table 5.4 presents the results obtained for mapping-5, such that $\mathcal{T}_1$ and $\mathcal{T}_3$ are executed on processor P_1, and $\mathcal{T}_2$ is executed on processor P_2. In this mapping action A_8 begins after approximately 1480 t.u. for a start of 50, and 2970 t.u. for a start of 100.

The third mapping is mapping-8 such that $\mathcal{T}_1$ and $\mathcal{T}_2$ executes on P_1 and $\mathcal{T}_3$ executes on P_2. The results are presented in Table 5.5. In this mapping action A_8 begins after

Energy	Init. prob.	Deviat.	λ	T. Detect (50)	T. Detect (100)
48	0.665	0.27	7.2	2745	4282
50	0.227	0.432	7.4	2278	3814
52	0.042	0.215	8.4	3109	4396

Table 5.3 – Change detection results for mapping-6

Energy	Init. prob.	Deviat.	λ	T. Detect (50)	T. Detect (100)
44	0.867	0.059	5.3	7277	8646
46	0.492	0.255	5.4	9381	9546
48	0.135	0.124	5.5	4961	6425
50	0.026	0.03	5.7	10725	11761

Table 5.4 – Change detection results for mapping-5

approximately 1510 t.u. for start of 50, and after approximately 3010 t.u. for a start of 100.

Energy	Init. prob.	Deviat.	λ	T. Detect (50)	T. Detect (100)
39	0.48	0.475	9.0	2362	3851
41	0.289	0.518	4.7	1932	3416
43	0.118	0.489	7.0	2339	4004
45	0.033	0.271	8.0	2557	4070

Table 5.5 – Change detection results for mapping-8

The last Pareto-efficient mapping is mapping-7 that executes all tasks on P_1. Results for this mapping are presented in Table. 5.6. In this mapping the action A_8 begins after approximately 1480 t.u.for start of 50, and after approximately 2980 t.u. for a start of 100.

Energy	Init. prob.	Deviat.	λ	T. Detect (50)	T. Detect (100)
23	0.976	0.013	4.0	3687	3714
25	0.832	0.067	20.8	16211	16730
27	0.612	0.086	19.8	13067	13683
29	0.388	0.119	13.5	7379	8587
31	0.188	0.106	3.8	6836	8170
33	0.038	0.103	8.3	7348	8092

Table 5.6 – Change detection results for mapping-7

Discussion In these experiments, we are mainly interested in the detection delay, that is the delay between the true occurrence of the event and its detection by our CUSUM algorithm. Since our models are stochastic and our experiments are based on statistics there is inevitably some variance in the results. First we have configured our algorithm in order to limit to the minimum the occurrences of false alarms. As we can see in the results there is no detection before the true occurrence of the event. There is however some detection delay. Since our algorithm is based on the measure of energy consumption, the event that we monitor (the start of action A_8) needs some time to produce effects on the energy consumption. Indeed the change produced by this event is quite subtle (a change from 80% CPU power to 100% CPU power, when the action is running). Nevertheless the algorithm always manages to raise a detection.

Looking more closely at the results from the different mappings, we can observe that the best results are obtained from mapping-8, a model in which action A_8 (that runs on task $\mathcal{T}_3$) is executed alone on processor P_2. This result can be explained by the fact that A_8 running alone on P_2 is not perturbed by the preemption from other tasks, and therefore tends to produce more deterministic effects on the energy consumption. In Table. 5.2 we can see that mapping-8 also provides the best maximum response time for task T_3.

Chapter 6

Information Leakage

High-security processes have to load confidential information into shared resources as part of their operation. This confidential information may be leaked (directly or indirectly) to low-security processes via the shared resource. In this chapter, we consider leakage from high-security to low-security processes from the perspective of scheduling. The workflow model is here extended to support preemption, security levels, and leakage. Formalization of leakage properties is then built upon this extended model, allowing formal reasoning about the security of schedulers. Several heuristics are presented in the form of compositional preprocessors and postprocessors as part of a more general scheduling approach. The effectiveness of these heuristics is evaluated experimentally, it shows significantly better schedulability than the state of the art. Modeling of leakage from cache attacks is presented as a case study.

Key Contributions. The key contributions in this chapter are as follows:

- A model to reason quantitatively on the amount of information leaked by scheduling tasks with different security levels on a shared resource system.
- A scheduling approach with compositional and specialized pre- and postprocessors that schedule tasks while reducing the amount of confidential information leaked.
- Several heuristic pre- and postprocessing algorithms that can reduce leakage.
- Experimental evaluation of the combinations of the pre- and postprocessors, showing that the approach provides significantly better schedulability and lower information leakage than the state of the art.

- A case study showing how to adapt the model to other scenarios and kinds of leakage, demonstrated with cache attacks.

6.1 Introduction

This chapter considers a shared resource system where processes are classified as either high-security or low-security. High-security processes work with confidential information that should not be leaked to low-security processes. Typically, this includes loading confidential information into memory for use within high-security processes. Examples of such confidential information include encryption keys, medical data, and bank details. This confidential information may be vital to the operation of the high-security processes, but must also be tightly controlled and not be leaked to low-security processes. For instance, in an embedded sensor, high-security encryption processes handle encryption keys that must not be leaked to low-security data compression processes.

Example Consider the small example in Figure 6.1, written in Intel x86-64 assembly code for Linux compiled to ELF format[1]. There are two processes: `Process 1` doing some (trivial) encryption operations, and `Process 2` attempting to access the encryption key. `Process 1` takes a key `$KEY` and a message `$MSG` then encrypts the message with the key using an exclusive or `XOR` operation. The result is then output to the disk (represented by `$DISK1`). `Process 2` writes to a different disk location (represented by `$DISK2`) the content of register `r13`. It is clear that if `Process 2` is executed after the first operation and before the fourth operation of `Process 1`, then the value of the key is directly leaked.

However, high-security processes may not properly flush confidential information from the shared resource, or context switching may interrupt their execution before such flushing can be applied. Consequently, confidential information remaining in the shared resource becomes (directly or indirectly) available to low-security processes.

If a scheduler is aware of a process' access level, then the scheduler can take actions to prevent confidential information being leaked to low-security processes. Recent work [MYPB14, PPY+15] has explored these kinds of problems in a real-time setting by scheduling a complete resource (memory) flush after any high-security process that is followed by a low-security process. However, this provides only limited options to the scheduler since such a complete resource flush is expensive and may prevent real-time

[1] Technical details for X86-64 (https://software.intel.com/sites/default/files/article/402129/mpx-linux64-abi.pdf) and ELF initialization (http://lxr.linux.no/linux+v3.2.4/arch/ia64/include/asm/elf.h).

```
; Process 1:
 mov    r13,$KEY   ; load key to register r13
 mov    r14,$MSG   ; load message to register r14
 xor    r14,r13    ; encrypt message with key
                   ; using XOR, store result in r14
 xor    r13,r13    ; wipe value of key
 out    $DISK1,r14 ; output the ciphertext (r14)

; Process 2:
 out    $DISK2,r13 ; output r13 (may store the key)
```

Figure 6.1 – Example Processes with schedule-dependent confidential information leakage.

tasks from meeting their deadlines. Furthermore, when flushing is not possible, current approaches do not quantify the information leakage, simply considering any leakage unacceptable.

In this chapter we propose treating confidentiality, measured by the resulting leakage of secure information, as a quantitative resource that the scheduler can exploit. This allows for a better quantification of the resulting leakage in different scenarios, as well as having a clear measure of the cost of different scheduling choices. Further, this allows for the creation of schedulers that can make better scheduling choices and also respect confidential information leakage constraints.

We built our work upon the workflow model commonly used to represent real-time systems [BcRS13, Gra66, YMCS16]. In the workflow model a set of tasks periodically produces jobs that have to be scheduled to complete before deadlines.

The workflow model is here extended by considering tasks to be composed of steps, each of which has an execution time, leakage value, and security level. Each one of these steps is implicitly an atomic sequence of actions that can be taken within a task without preemption by the scheduler. Thus a task consists of an ordered sequence of steps to be performed, that yields to the total behavior of the task.

6.2 Related Work

Real-time systems need to communicate with the outside world, such as receiving data from sensors or communicating with other systems, sometimes over unsecured networks. This communication has allowed attacks against even air-gapped industrial control systems [FMC11].

The real-time scheduling requirement itself can be exploited to generate additional

vulnerabilities. For instance, a process can modulate its use of a resource to affect the scheduling of another process, and use this to covertly transmit information [SA06, SMD00].

Further vulnerabilities can occur in any system with shared resource. When processes with different security levels share the same memory resources, it is possible for low-security processes to access confidential information that should only be accessed by high-security processes [MYPB14]. Using separated memories for processes with different security levels is expensive, particularly if the system has more than two security levels. Mohan et al. [MYPB14] consider a shared memory scenario where low-security processes executing after high-security processes could access the high-security processes' memory space resulting in information leakage. To prevent this, they propose completely flushing the memory after the execution of high-security processes when followed by a low-security process. In [PPY+15], Pellizzoni et al. generalize this work by introducing a binary relation NO-LEAK on tasks, where no-leak$(\mathcal{T}_x, \mathcal{T}_y)$ holds if no leakage can occur from $\mathcal{T}_x$ to $\mathcal{T}_y$. The authors also determine the number of memory flushes needed to enforce the no-leak relation, and consequently construct a preemptivity-assignment scheduling algorithm. This work proposes a more fine-grained approach to confidentiality in similar scenarios.

Another less formal approach is that used in [VRS14] where they limit the time between preemptions between virtual machines in an online scheduling scenario to prevent cache attacks. This could be analysed using the formal approach and methodology here, albeit as a specific case study.

Formal analysis of scheduling system under resource constraints has been performed by Kim et al. [KLL+15b, KLT+16b]. The proposed approach can be extended to confidentiality as a resource using the model proposed in this chapter.

6.3 Model

The model here is based upon the workflow model recalled in Section 1.3. The extension here is to represent more precise information about the internal operations and preemptivity of tasks by dividing them into steps. These steps include their own execution time (like a task or job from the workflow model), and are extended to include leakage value and security level. Special tasks are also added to model other operations of the system.The rest of this section details this extended model and presents illustrative examples that motivate the choices in this chapter.

6.3.1 Concept

This section considers concepts and motivations behind our information leakage model. In particular, the division of tasks into steps, how to account for leakage in practice, and justification for special tasks.

Steps. This model considers the possibility to divide tasks into fine-grained steps. A step represents an atomic sequence of operations that cannot be interrupted by preemption. The practical implementation of steps depends on the architecture and granularity of the scheduling system.

The model is agnostic to step implementation details as long as an execution time, leakage value, and security level can be defined for each step. The most fine-grained approach would be to consider each CPU operation as a step. For instance, `Process 1` in Figure 6.1 would be represented as a task divided into five steps. Thus, a task could be preempted after each CPU operation. Although very simple, in practice this approach is too fine-grained. In lightweight and embedded systems it is common to delegate part of the handling of preemption and atomicity to the programmer, so it is reasonable to consider that the programmer itself could define the steps.

Special Tasks. We consider two special tasks representing special system operations: flush and wait. The flush task flushes all confidential information from the shared resource, for instance by overwriting all shared memory with zeroes. This preserves compatibility with the state of the art [MYPB14, PPY+15] where flushing is used as the main tool to preserve confidentiality. The wait task represents idle processor time. Apart from the obvious use, scheduling of idle time can impact confidentiality of the system.

Leakage Values. The leakage value of a step represents the amount of confidential information that would be leaked to an attacker able to read the shared resource just after the execution step. The model does not constrain the way the leakage value is obtained: leakage can be added by the programmer as an annotation, computed by an automatic tool [BLTW13, CMS14, CKN14, VEB+16], or possibly both.

For instance, the programmer could specify critical zones in which the program must not be interrupted, and the leakage values would be computed automatically by a tool (for both critical and non-critical zones).

An alternative, variable-based approach would be to have the programmer annotate some variables as containing confidential information at a certain point (and as cleared

of confidential information at a later point). Taint analysis can be used to identify which variables are tainted at each point. Information leakage quantification can be used to quantify leakage from the tainted variables.

6.3.2 Formal Model

6.3.2.1 Steps, Tasks and Jobs

Definition 6.1 (Step)**.** *Formally, each* step *is a tuple* $\mathcal{S}(E, L, X)$ *where* E *denotes the (worst case)* execution time *that the step takes to be completed,* L *denotes its (potential)* leakage value*, and* X *denotes its security level (either* high $\top$ *or* low $\bot$*).*

The (potential) leakage value L of a step $\mathcal{S}$ is a measure of the amount of confidential information left in a shared resource at the completion of $\mathcal{S}$. Again, the exact meaning of the leakage value is unimportant here. Here $\top$ indicates that the step contains confidential information and therefore is high-security. Similarly, $\bot$ indicates that the step should not have access to confidential information and therefore is low-security. Since $\top$ and $\bot$ are used to indicate whether the step has access to confidential information, $\bot$ steps typically have leakage zero. This is not a strict requirement, see Section **??**. The choice of having two security levels here is to clearly illustrate the model, however the extension to any number of security levels is straightforward.

Example For instance, consider `Process 1` in Figure. 6.1. Each assembly instruction can be represented by a single step with an execution time of one time unit and a security level of $\top$. The first three instructions have a leakage value of one, representing the fact that one word of confidential information (the key) is in the shared resource (in register `r13`). However, the remaining instructions have a leakage value of zero since the fourth instruction wipes `r13`.

The system operates with a set of tasks $\Gamma = \{\mathcal{T}_\alpha, \mathcal{T}_\beta, \ldots\}$.

Definition 6.2 (Task)**.** *Each* task $\mathcal{T}_x \in \Gamma$ *is a tuple* $\mathcal{T}_x(P_x, D_x, \widehat{\mathcal{S}_x})$ *where* P_x *is the* period *of the task,* D_x *is its* relative deadline*, and* $\widehat{\mathcal{S}_x}$ *is a sequence of* steps $\mathcal{S}_{xa}, \mathcal{S}_{xb}, \ldots$ *making up the ordered actions of the task.*

Tasks are named with Greek letters, e.g. $\mathcal{T}_\beta$. Steps are named with the corresponding tasks Greek letter and a Latin letter in alphabetical order, e.g. step $\mathcal{S}_{\beta c}$ represents the third step of task $\mathcal{T}_\beta$.

Example Observe that `Process 1` in Figure 6.1 can be modeled by the following task:

$$\begin{aligned}\mathcal{T}_\alpha \;=\; \mathcal{T}(&P_\alpha, D_\alpha, \mathcal{S}_{\alpha a}(1,1,\top)\\ &\mathcal{S}_{\alpha b}(1,1,\top)\\ &\mathcal{S}_{\alpha c}(1,1,\top)\\ &\mathcal{S}_{\alpha d}(1,0,\top)\\ &\mathcal{S}_{\alpha e}(1,0,\top))\ .\end{aligned}$$

Similarly, `Process 2` in Figure. 6.1 can be modeled by the following task:

$$\mathcal{T}_\beta \;=\; \mathcal{T}(P_\beta, D_\beta, \mathcal{S}_{\beta a}(1,0,\bot))\ .$$

Definition 6.3 (Job). *Each* job $\tau_{x,k}$ *is created by the activation of the task* $\mathcal{T}_x$ *at* release time $R_{x,k} = (k-1)P_x$ *for* $k \in \mathbb{N}_0$*, and is a tuple* $\tau_{x,k}(R_{x,k}, A_{x,k}, \widehat{\mathcal{S}_{x,k}})$ *where* $A_{x,k} = R_{x,k} + D_x$ *is the job's absolute deadline, and* $\widehat{\mathcal{S}_{x,k}}$ *is the sequence of steps inherited from task* $\mathcal{T}_x$*.*

Jobs are named with the corresponding task's Greek letter and the number k, so job $\tau_{\beta 4}$ is the fourth job generated by task $\mathcal{T}_\beta$ and step $\mathcal{S}_{\beta 4c}$ is the third step of job $\tau_{\beta 4}$.

For simplicity, a task (resp. job) will be referred to as $\top$ or $\bot$ when all steps within that task (resp. job) are either $\top$ or $\bot$, respectively.

Flush and Wait. The model uses a task to represent complete flushing of the shared resource. The flush task is defined by $\mathcal{T}_\mathcal{F}(-,-,\mathcal{S}_\mathcal{F}(E_\mathcal{F},0,\top))$ where $E_\mathcal{F}$ is the execution time to completely flush the shared resource. Observe that *after flushing the shared resource the leakage is reduced to zero.* This is achieved by the single step $\mathcal{S}_\mathcal{F}(E_\mathcal{F},0,\top)$ that takes all the execution time of the flush task and has a zero leakage value. Since the flush task is always available to be scheduled, it has no defined period or deadline (denoted here as -), being able to schedule (or not) at whim. The security level of flush is $\top$ since it is acceptable for flush to have access to confidential information, and for use in calculating the resulting leakage (see below). For simplicity and when no ambiguity may occur, $\mathcal{F}$ is used for the flush task or step.

To represent idle processor time, define the wait task as $\mathcal{T}_\mathcal{W}(-,-,\mathcal{S}_\mathcal{W}(1,*,*))$. Similar to flush, wait is always available to be scheduled and has no period or deadline (again denoted as -). Wait also has a single step that has the minimal runtime of one time unit. However, the leakage value of wait is here denoted by $*$ since waiting does not change

the shared resource, instead the $*$ denotes that *the leakage value of a wait step is the same as the previous step.* Similarly, the security level is also represented by $*$ because it is the same as the previous step. Again for simplicity and where no ambiguity may occur, $\mathcal{W}$ may be used in place of the wait task or step.

6.3.2.2 Traces, Solutions, and Resulting Leakage

Definition 6.4 (Trace). *A* trace $\widetilde{\mathcal{S}} = (\mathcal{S}_1(E_1, L_1, X_1), \mathcal{S}_2(E_2, L_2, X_2), \dots)$ *is a (possibly infinite) sequence of* $n \in \mathbb{N} \cup \{\infty\}$ *steps that may come from any number of jobs.*

In a trace, Step $\mathcal{S}_1$ starts execution at time $t_1 = 0$, and each step $\mathcal{S}_i$ for $i > 1$ starts execution at time $t_i = \sum_{j=1}^{i-1} E_j$ and terminates execution at time $t_i + E_i$. The notation $\widetilde{\mathcal{S}_1}{+}{+}\widetilde{\mathcal{S}_2}$ is used to indicate concatenation of traces $\widetilde{\mathcal{S}_1}$ and $\widetilde{\mathcal{S}_2}$, and $\widetilde{\mathcal{S}} \setminus \mathcal{S}_1$ the removal of the step $\mathcal{S}_1$ from the trace $\widetilde{\mathcal{S}}$. The focus of this chapter is upon solutions.

Definition 6.5 (Solution). *A trace* $\widetilde{\mathcal{S}}$ *is a* solution $\overline{\mathcal{S}}$ *if:*

1. *for each job* $\tau(R, A, \widehat{\mathcal{S}})$*:*
   (a) *each step in* $\widehat{\mathcal{S}}$ *appears in the trace* $\widetilde{\mathcal{S}}$ *in the order that it appears in* $\widehat{\mathcal{S}}$*;*
   (b) *the first step of* $\widehat{\mathcal{S}}$ *does not start execution before* R*;*
   (c) *the last step of* $\widehat{\mathcal{S}}$ *does not terminate execution after* A*;*
2. *each step that is not wait* $\mathcal{W}$ *or flush* $\mathcal{F}$ *appears exactly once in the trace* $\widetilde{\mathcal{S}}$*.*

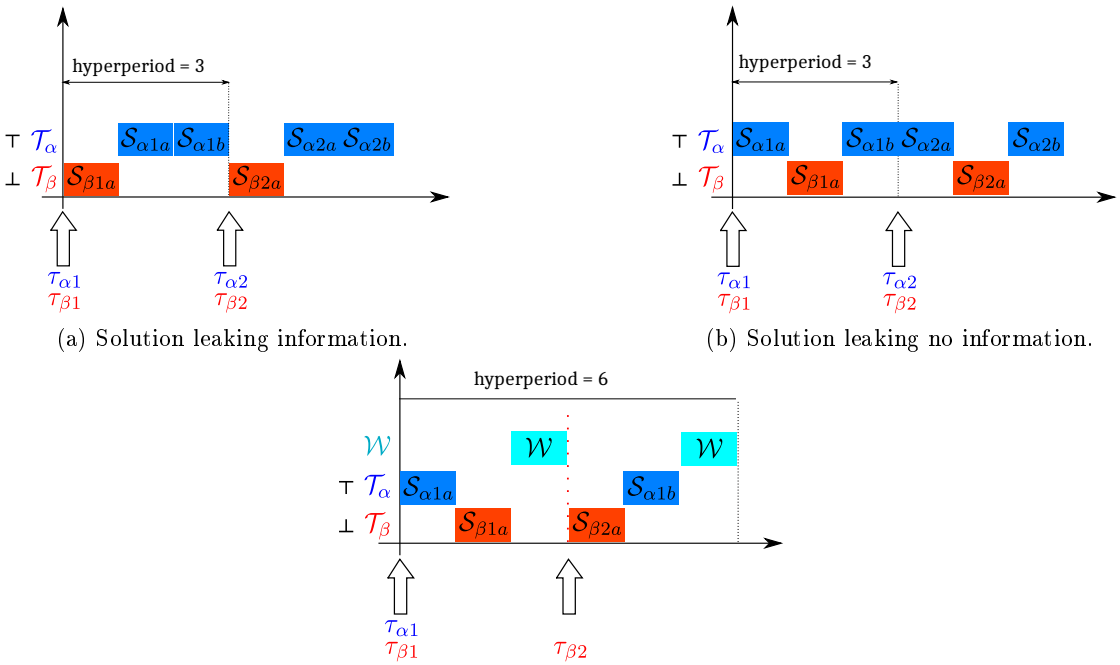
Given a set of tasks Γ, a solution $\overline{\mathcal{S}}$ is a solution for Γ, written $\overline{\mathcal{S}}_\Gamma$, iff $\forall \mathcal{T}_x \in \Gamma, \forall k \in \mathbb{N}_0$ then for each job $\tau_{x,k}(R_{x,k}, A_{x,k}, \widehat{\mathcal{S}_{x,k}})$ it holds that every step in $\widehat{\mathcal{S}_{x,k}}$ is in $\overline{\mathcal{S}}$.

A solution $\overline{\mathcal{S}}$ is periodic if it periodically repeats the same sequence of steps up to job indexing. For simplicity, a periodic solution may be represented by the periodically repeated sequence alone.

Given a trace $\widetilde{\mathcal{S}}$ the resulting leakage $\mathscr{L}(\widetilde{\mathcal{S}})$ of trace $\widetilde{\mathcal{S}}$ represents the total amount of information leaked during the execution of the jobs scheduled according to $\widetilde{\mathcal{S}}$.

Definition 6.6 (Resulting leakage). *Given a trace* $\widetilde{\mathcal{S}}$ *composed of* n *steps with* $n \in \mathbb{N} \cup \{\infty\}$*, the* resulting leakage $\mathscr{L}(\widetilde{\mathcal{S}})$ *of the trace* $\widetilde{\mathcal{S}}$ *is defined inductively as follows:*

- *if* $n \leq 1$*, then* $\mathscr{L}(\widetilde{\mathcal{S}}) = 0$*;*
- *if* $n > 1$ *and the second step* $\mathcal{S}_2$ *of trace* $\widetilde{\mathcal{S}}$ *is* $\top$*, then the resulting leakage is the leakage of the trace without the first step* $\mathcal{S}_1$*:* $\mathscr{L}(\widetilde{\mathcal{S}}) = \mathscr{L}(\widetilde{\mathcal{S}} \setminus \mathcal{S}_1)$;



(a) Solution leaking information.

(b) Solution leaking no information.

(c) Solution with waiting.

Figure 6.2 – Periodic Solutions for Leakage between hyperperiods.

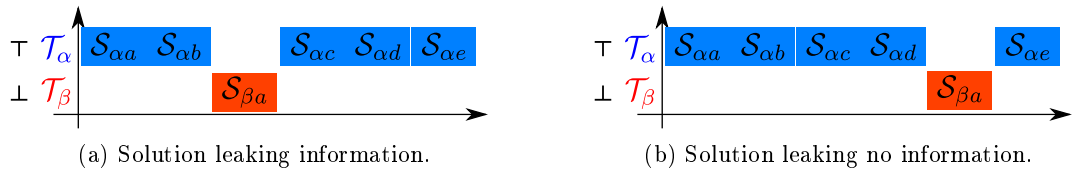
- *if $n > 1$ and the second step $\mathcal{S}_2$ of trace $\widetilde{\mathcal{S}}$ is $\perp$, then the resulting leakage is the leakage of the trace without the first step $\mathcal{S}_1 = \mathcal{S}(E_1, L_1, X_1)$ plus the leakage value L_1 of the first step $\mathcal{S}_1$: $\mathscr{L}(\widetilde{\mathcal{S}}) = \mathscr{L}(\widetilde{\mathcal{S}} \setminus \mathcal{S}_1) + L_1$.*

Since every solution $\overline{\mathcal{S}}$ is a trace $\widetilde{\mathcal{S}}$, a solution resulting leakage $\mathscr{L}(\overline{\mathcal{S}})$ is defined in the same manner.

Example Recall the example from Figure 6.1. The solution in Figure 6.3a has resulting leakage one, since `Process 2` is executed when the key is in the shared resource and so the step $\mathcal{S}_{\beta a}$ is able to access the key.

However, the solution in Figure 6.3b has resulting leakage is zero, since `Process 2` is executed after the key has been wiped from the shared resource.

If a solution is periodic, the periodic leakage can be calculated as follows. Given one instance of the periodically repeated sequence of steps $\widetilde{\mathcal{S}} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_i)$, the periodic leakage is the resulting leakage of the sequence $\widetilde{\mathcal{S}} {+\!+} \mathcal{S}_1$.



(a) Solution leaking information.

(b) Solution leaking no information.

Figure 6.3 – Schedulings for the processes in Figure. 6.1.

6.3.3 Illustrating Examples

This section presents three examples illustrating the utility of the model. Each presents a different aspect of using the model to find solutions with good resulting leakage.

Periodic Leakage This example illustrates leakage due to the periodic nature of tasks and how to account for this when scheduling. Consider two tasks: a $\top$ task $\mathcal{T}_\alpha(3, 3, \mathcal{S}_{\alpha a}(1, 0, \top), \mathcal{S}_{\alpha b}(1, 4, \top))$ and a $\bot$ task $\mathcal{T}_\beta(3, 3, \mathcal{S}_{\beta a}(1, 0, \bot))$. The goal is to find a solution with minimal (here zero) resulting leakage.

Two periodic solutions to these tasks are depicted in Figure 6.2a & 6.2b. Note that the $\top$ step $\mathcal{S}_{\alpha 1a}$ has leakage value zero, so even if $\mathcal{S}_{\alpha 1a}$ is followed by the $\bot$ step $\mathcal{S}_{\beta 1a}$ this does not increase the resulting leakage. Thus both periodic solutions have a resulting leakage of zero within their periodically repeated sequence (here corresponding to their hyperperiod). However, when the periodically repeated sequence is repeated, the periodic solution in Figure 6.2a has non-zero periodic leakage, since at time four the $\top$ step with leakage value four $\mathcal{S}_{\alpha 1b}$ is followed by the $\bot$ step $\mathcal{S}_{\beta 2a}$ on periodic scheduling. Hence, only the periodic solution in Figure. 6.2b has periodic leakage zero.

Waiting can Reduce Leakage This example illustrates the utility of making $\mathcal{W}$ available to the scheduler. In some cases exploiting $\mathcal{W}$ reduces the resulting leakage of a solution. Consider two tasks: a $\top$ task $\mathcal{T}_\alpha(6, 5, \mathcal{S}_{\alpha a}(1, 0, \top), \mathcal{S}_{\alpha b}(1, 4, \top))$ and a $\bot$ task $\mathcal{T}_\beta(3, 2, \mathcal{S}_{\beta a}(1, 0, \bot))$. Again the goal is to find a solution with the minimal (zero) resulting leakage.

One periodic solution to this example with zero resulting leakage is presented in Figure 6.2c. The periodic solution exploits $\mathcal{W}$ to have the two $\bot$ steps executed together (with only $\mathcal{W}$ in between). This allows the $\top$ step $\mathcal{S}_{\alpha 1b}$ (with positive leakage value)

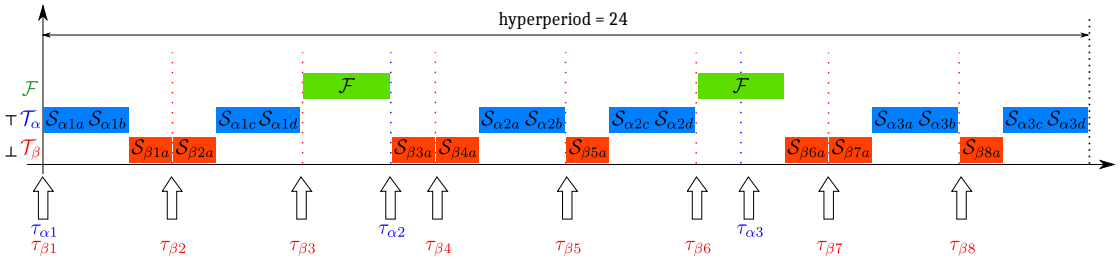


Figure 6.4 – Hyperperiodic flush.

to be scheduled after the last $\bot$ step in the periodically repeated sequence (again corresponding here to the hyperperiod) and still meet its deadline. Observe that since $\mathcal{S}_{\alpha 2a}$ has zero leakage value, the periodic solution has zero periodic leakage. This periodic solution with zero periodic leakage would not be possible if $\mathcal{W}$ was not available to be scheduled at any time (in particular as an alternative to scheduling the step $\mathcal{S}_{\alpha 1b}$), since without $\mathcal{W}$ then $\mathcal{S}_{\alpha 1b}$ would be scheduled before $\mathcal{S}_{\beta 2a}$ and thus increase periodic leakage.

Periodic Flush Since finding a solution for a set of tasks is generally best solved in a periodic manner, it is possible to exploit this periodic nature when constructing the solution. For example, the total amount of time units not used by jobs can be calculated, and then these time units can be used to consider adding $\mathcal{F}$. Typically, such free time units would be fragmented inside the solution. However, with this information, the scheduler can use sufficiently long empty spaces (or create them) to schedule $\mathcal{F}$. Hence, even if it may not be possible to flush the memory after each $\top$ step followed by a $\bot$ step, some additional $\mathcal{F}$ can be scheduled to reduce the solution's resulting leakage while maintaining schedulability.

For example, consider two tasks: a $\top$ task $\mathcal{T}_\alpha(8, 8, \mathcal{S}_{\alpha a}(1, 5, \top), \mathcal{S}_{\alpha b}(1, 1, \top), \mathcal{S}_{\alpha c}(1, 6, \top), \mathcal{S}_{\alpha d}(1, 4, \top))$ and a $\bot$ task $\mathcal{T}_\beta(3, 3, \mathcal{S}_{\beta a}(1, 0, \bot))$. Here let the execution time of $\mathcal{F}$ be two, i.e. $E_\mathcal{F} = 2$. Consider the scheduling of the two tasks over their hyperperiod of twenty-four time units (when developing periodic solutions, the hyperperiod is a convenient choice since periodicity is guaranteed). A periodic solution can be seen in Figure 6.4.

No solution with resulting leakage zero exists. Further, it is not possible to schedule the jobs by inserting an $\mathcal{F}$ after every $\top$ step followed by a $\bot$ step since this would not be schedulable (this is the state of the art as in [MYPB14]). However, the periodic solution in Figure 6.4 achieves a low periodic leakage of three per hyperperiod while maintaining schedulability by adding two $\mathcal{F}$ steps to minimize the periodic leakage.

6.4 Problems

Information leakage (or just leakage) quantifies the amount of privileged information leaked (lost to an attacker) by a system, and is widely used to obtain a measure of the (in) security of the system [KLT+16b, BLMW15, ACPS12, BKR09]. In our work, leakage is used to measure the amount of privileged information that a high-security job leaves in the shared memory at different moments of its execution. We will consider the unit of measure of leakage to be bits, following the standard for information-theoretical leakage measures [BCGL17]. However, the same leakage model could be used to appropriately measure the loss of any measurable security property, where zero represents no loss. Similarly, we do not constraint the way the leakage value is obtained: it could be added by the programmer as an annotation in the source code, or automatically computed by one of the many tools available [BLTW13, CKN14, VEB+16].

6.5 Methods

The overarching goal of the approach proposed in this chapter is to produce a solution with a low resulting leakage for a given set of tasks. To achieve this, standard offline scheduling algorithms are extended with a preprocessing and a postprocessing phase. The preprocessing phase transforms a set of tasks Γ into a set of preprocessed tasks Γ'. Then scheduling is applied to Γ' obtaining a solution $\overline{\mathcal{S}'}_{\Gamma'}$ for Γ'. Finally, the postprocessing phase transforms the solution $\overline{\mathcal{S}'}_{\Gamma'}$ into a postprocessed solution $\overline{\mathcal{S}''}_{\Gamma'}$. Both the pre- and postprocessing phases can affect the desired solution $\overline{\mathcal{S}''}_{\Gamma'}$, here with the goal of reducing the resulting leakage. The rest of this section presents various heuristic algorithms used for the experiments (see Section 6.6). The scheduling algorithms considered are EDF and LSF. Note that EDF and LSF do no consider the security-level or leakage of the steps (for discussion of this see Section 6.7). The rest of this section focuses upon the pre- and postprocessors. The division in phases creates a modular and compositional approach, allowing for a better comparison of different pre- and postprocessors.

6.5.1 Preprocessing

Preprocessors are algorithms that take a set of tasks Γ and produce a set of tasks Γ' to be scheduled.In our work, we consider preprocessors that attempt to "merge" adjacent steps with the same security level within each task in Γ. The merged step has the sum of the execution times of the merged steps, the leakage value of the last merged step, and the same security level as the merged steps. For instance, the steps $\mathcal{S}_{\alpha a}(1, 0, \top)$ and

Algorithm 3: Total Merge Preprocessor

Data: task $\mathcal{T}(P, D, \widehat{\mathcal{S}})$
Result: processed task $\mathcal{T}'$
$E_T = 0$; $L_T = 0$; $X_T = \bot$
for $i = 1$ *to* $|\widehat{\mathcal{S}}|$ **do**
  let $\mathcal{S}(E_i, L_i, X_i) = \mathcal{S}_i$
  $E_T = E_T + E_i$
  $L_T = L_i$
  $X_T = X_i$
end
Return $\mathcal{T}' = \mathcal{T}(P, D, \mathcal{S}_T(E_T, L_T, X_T))$

$\mathcal{S}_{\alpha b}(1, 4, \top)$ could be merged producing the step $\mathcal{S}_{\alpha a'}(2, 4, \top)$. The rest of this section presents three preprocessing algorithms that exploit merging.

Total Merge. The Total Merge algorithm merges all the steps in a task into a single step (as detailed in Algorithm 3). The merging is achieved by starting with a step that has execution time and leakage value zero. The execution time for each other step in the task is then added, and the leakage value from the last step being merged is preserved. The security level is set to that of the last step. Finally, the processed task uses this single merged step as its only step.

One-Step Merge. The One-Step Merge algorithm attempts to merge pairs of adjacent steps. Adjacent pairs are merged if the leakage of the former step is higher than the latter. This is achieved by iterating through the steps $\mathcal{S}_i$ of the task. If $L_i > L_{i+1}$, then the steps $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$ are merged. Otherwise, $\mathcal{S}_i$ is maintained unchanged. This algorithm generates a new sequence of steps $\widehat{\mathcal{S}'}$, that are then used in the processed task Details can be seen in Algorithm 4.

n-Step Merge. A straightforward extension to the One-Step Merge algorithm is to allow the merging of any number of steps. This appears in the results as n-Step Merge.

6.5.2 Postprocessing

Postprocessing algorithms take one solution and produce another solution. This can be done by any possible manipulation of the steps within the original solution $\overline{\mathcal{S}'}_\Gamma$ to produce the new solution $\overline{\mathcal{S}''}_\Gamma$ that does not break the property of being a solution for Γ. The rest of this section presents four such postprocessors.

Algorithm 4: One-Step Merge Preprocessor

Data: task $\mathcal{T}(P, D, \widehat{\mathcal{S}})$
Result: processed task $\mathcal{T}'$
$\widehat{\mathcal{S}}' = \emptyset;\ \ i = 1$
while $i < |\widehat{\mathcal{S}}| + 1$ **do**
  let $\mathcal{S}(E_i, L_i, X_i) = \mathcal{S}_i$
  let $\mathcal{S}(E_{i+1}, L_{i+1}, X_{i+1}) = \mathcal{S}_{i+1}$
  if $L_{i+1} < L_i$ **then**
    $\widehat{\mathcal{S}}' = \widehat{\mathcal{S}}' {+\!+} \mathcal{S}(E_i + E_{i+1}, L_{i+1}, X_{i+1});\ \ i = i + 2$
  else
    $\widehat{\mathcal{S}}' = \widehat{\mathcal{S}}' {+\!+} \mathcal{S}_i;\ \ i = i + 1$
  end
end
Return $\mathcal{T}' = \mathcal{T}(P, D, \widehat{\mathcal{S}}')$

Add Flush. The Add Flush algorithm replaces sequences of $\mathcal{W}$ with $\mathcal{F}$ where possible (detailed in Algorithm 5). Add Flush operates by finding sequences of $\mathcal{W}$ whose length is greater than or equal the execution time of $\mathcal{F}$. If such a sequence is found, a $\mathcal{F}$ is added to the produced solution instead of the initial sequence of $\mathcal{W}$ with execution time equal to the $\mathcal{F}$. Any remaining $\mathcal{W}$ in the solution are maintained.

Algorithm 5: Add Flush Postprocessor

Data: solution $\overline{\mathcal{S}}$, and wait $\mathcal{W}$
Result: solution $\overline{\mathcal{S}'}$
$\overline{\mathcal{S}'} = \emptyset;\ \ i = 1$
while $i < |\overline{\mathcal{S}}| + 1$ **do**
  if $\mathcal{S}_i == \mathcal{W}$ **then**
    $j = \mathsf{CountWaitsFrom}(\overline{\mathcal{S}}, i)$
    if $j \geq E_{\mathcal{F}}$ **then**
      $\overline{\mathcal{S}'} = \overline{\mathcal{S}'} {+\!+} \mathcal{F}$
      $j = j - E_{\mathcal{F}}$
    end
    $\overline{\mathcal{S}'} = \overline{\mathcal{S}'} {+\!+} \mathsf{Repeat}(\mathcal{W}, j)$
  else
    $\overline{\mathcal{S}'} = \overline{\mathcal{S}'} {+\!+} \mathcal{S}_i$
    $i = i + 1$
  end
end
Return $\overline{\mathcal{S}'}$

Swap. The Swap algorithm attempts to reduce the resulting leakage by swapping steps within the solution (as in Algorithm 6). Swap works by considering each step $\mathcal{S}_i$. Then each possible swap $[\mathcal{S}_i \leftrightarrow \mathcal{S}_j]$ between the step $\mathcal{S}_i$ and a following step $\mathcal{S}_j$ is considered. If the trace with this swap applied has less resulting leakage and is still a solution, then this solution $[\mathcal{S}_i \leftrightarrow \mathcal{S}_j]\overline{\mathcal{S}}$ is kept as the best possible solution so far. Finally, once all possible swaps have been considered, the best swap to the solution is applied and i is incremented.

Algorithm 6: Swap Postprocessor

```
Data: solution S̄
Result: solution S̄'
for i = 1 to |S̄| do
    S̄' = S̄
    for j = i to |S̄| do
        S̄'' = [S_i ↔ S_j]S̄
        if L(S̄'') < L(S̄') && isSolution(S̄'') then
            S̄' = S̄''
        end
    end
    S̄ = S̄'
end
Return S̄
```

Move. The Move algorithm moves one step to a new position in the solution. Move works in the same manner as the Swap postprocessor, except instead of swapping $[\mathcal{S}_i \leftrightarrow \mathcal{S}_j]\overline{\mathcal{S}}$ the steps $\mathcal{S}_i$ and $\mathcal{S}_j$, the move $[\mathcal{S}_i \longrightarrow \mathcal{S}_j]\overline{\mathcal{S}}$ moves the step $\mathcal{S}_i$ to be after $\mathcal{S}_j$. For example: $[\mathcal{S}_1 \longrightarrow \mathcal{S}_3]\mathcal{S}_a, \mathcal{S}_b, \mathcal{S}_c = \mathcal{S}_b, \mathcal{S}_c, \mathcal{S}_a$

where the first step $\mathcal{S}_a$ is moved to be after the third step $\mathcal{S}_c$. The rest of the algorithm is the same as Swap, finding the best possible move and ensuring the trace after the move is a solution. The algorithm is identical to the Swap algorithm substituting $[\mathcal{S}_i \leftrightarrow \mathcal{S}_j]\overline{\mathcal{S}}$ with $[\mathcal{S}_i \longrightarrow \mathcal{S}_j]\overline{\mathcal{S}}$ in Line 4.

1-Swap. Observe that if only swapping or moving with the following step is considered, that is $[\mathcal{S}_i \leftrightarrow \mathcal{S}_{i+1}]$ or $[\mathcal{S}_i \longrightarrow \mathcal{S}_{i+1}]$, then the swap and move postprocessors coincide. This postprocessor is denoted as 1-Swap in the results.

6.6 Experiments

This section discusses the results obtained by running experiments with the preprocessing, scheduling, and postprocessing algorithms in this chapter.

The experiments were conducted by using approximately 30,000 randomly generated sets of tasks[2], and then testing each possible combination of one preprocessing, one scheduling, and one postprocessing algorithm. Each set of tasks consists of 2 to 6 tasks with at least one $\top$ task and one $\bot$ task, with each task having 1 to 8 steps, and each step execution time from 1 to 5.

Preprocessor	**Postprocessor**				
Merge	**None**	**Add Flush**	**Swap**	**Move**	**1-Swap**
None	2	116	1919	1903	190
One-Step	1	93	1567	1489	149
n-Step	1	88	1486	1404	141

Table 6.1 – Average execution time (in ms) for each combination of pre- and postprocessor (except Total Merge) using the EDF scheduling algorithm.

Sets of tasks with a hyperperiod over 5000 have been discarded to reduce testing time. The code[3] to perform the tests and implement the preprocessing, scheduling, and postprocessing is written in Java 1.8, and all experiments conducted on a Linux 3.13 64-bit kernel on an Intel Core i7-3720QM 2.60GHz CPU with 8GB of RAM.

A demo[4] is available that shows examples, and allows users to conduct their own GUI-based experiments.

The rest of this section discusses experimental outcomes.

The first point of interest is the schedulability of the set of tasks used in each experiment. Merging task steps in a preprocessor can make a set of tasks unschedulable, and the EDF and LSF scheduling algorithms are not equally able to find solutions. The failure percentage for each combination of preprocessing and scheduling algorithm is shown in Figure 6.5.

Figure 6.5 clearly shows that a greater merging of steps leads to more schedulability failures. In particular, indicating that Total Merge is not an effective algorithm to use in practice despite being considered as the current state of the art [MYPB14, PPY+15]. This is a strong motivation for the approach presented in this work to consider fine-grained preprocessing and preemption of tasks. Due to its high failure rate, Total Merge

[2] 30,000 sets of tasks were generated, 22 were discarded as unschedulable.
[3] Available via git from: https://scm.gforge.inria.fr/anonscm/git/secleakpublic/secleakpublic.git
[4] Demo available via website at: http://secleakpublic.gforge.inria.fr/

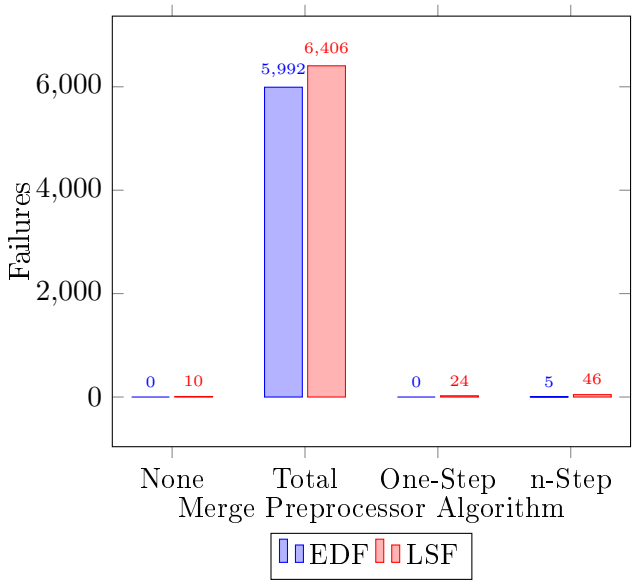


Figure 6.5 – Number of failures for each combination of preprocessor and scheduling algorithm, out of ∼30,000 experiments. Note that Total Merge, corresponding to the state of the art [MYPB14, PPY+15], fails ∼20% of the time.

will not be considered further in this chapter.

Figure 6.5 also shows that, for all preprocessing algorithms, EDF performs better for schedulability than LSF. (This is expected since EDF is guaranteed to find a solution if the tasks are schedulable, while LSF is not.) The two scheduling algorithms produce almost the same results for every other measure tested, so the rest of this chapter shall present only experimental results using the EDF scheduling algorithm.

Comparing the experimental results from postprocessing algorithms, the average resulting leakages for each combination of pre- and postprocessor is shown in Figure 6.6, while the average running times to generate a solution are shown in Table 6.1.

As expected, solutions without any postprocessing produce the highest resulting leakage. The best resulting leakage is obtained by the Add Flush algorithm. (This would correspond to the approach in [MYPB14, PPY+15] when combined with the Total Merge, however as noted above this is often not schedulable.) Note that merging preprocessors reduce total time, since they reduce the number of steps that the scheduler has to schedule.

1-Swap slightly reduces the resulting leakage, however Table 6.1 shows that it is significantly more expensive than the scheduling operation, so 1-Swap could be applied after Add Flush only if the cost is acceptable. Swap and Move do not reduce the resulting leakage significantly more than 1-Swap and are significantly more expensive to compute.

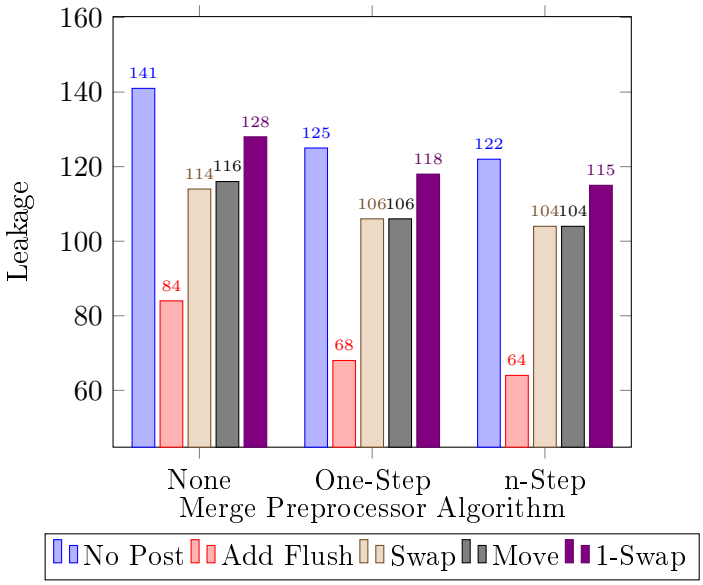


Figure 6.6 – Information leakage of the solutions for each combination of pre- and post-processor (except Total Merge) using the EDF scheduling algorithm.

This indicates that there is a balance to be found depending on the scenario. Taking significant time to pre-compute an optimal scheduling strategy for a sensor or other real-time system prior to shipping could be worth the time cost. However, for online scheduling with limited (or no) ability to look ahead and consider such options, the cost of anything more complex than Add Flush or 1-Swap may be too much.

6.7 Discussion

On the Division of Scheduling into Three Phases. The division into three phases is to separate out distinct parts of an overall scheduling from tasks to a solution. This approach allows for separation conceptually of different phases, and also for composition of simple algorithms in the pre- and postprocessing phases. For example, a postprocessor could move steps in a solution around to maximize contiguous $\mathcal{W}$s and then be composed with the Add Flush postprocessor to improve the resulting leakage further. This also allows different strategies to be employed in different phases, including strategies with different goals. For example, processors for resulting leakage minimization and energy consumption could be combined during pre- or postprocessing (or both).

Online Scheduling. In our work, we consider offline scheduling, i.e. when the tasks to be scheduled are known beforehand. In most real cases the tasks appear at runtime, requiring online heuristics to decide the scheduling. The division in steps and the leakage model presented in this chapter extend immediately to the online scenario. While the preprocessors and postprocessors do not, they provide insight that can be used to build online heuristics that reduce leakage. We consider this as a future work.

Execution Time. This chapter consideres the execution time to be essentially fixed for each step. Although formally the execution time is worst case, the scheduling here does not exploit when steps may terminate prior to their (worst case) execution time. This could naturally be incorporated into online scheduling (above), but even in a purely offline scheduling system this could be exploited. For example, consider the cache attack scenario, where flushing not only affects the leakage, but by flushing the cache the execution time will go up due to cache misses.

Conclusion

In this manuscript we focused our work on finding new advanced techniques for exploiting the scheduling theory in order to analyze the correctness of CPS under various types of properties.

First, we have presented a software engineering approach that generates model-based analysis tools for the schedulability analysis of CPS. This approach is based on one side on a set of formal models for describing complex scheduling problems, and on the other on meta-models of high-level specification languages to easily specify these scheduling problems. Our approach generates automatically domain-specific analysis tools based on the Cinco framework. These tools allow to specify scheduling problems using graphical components, and they can launch formal analyses by calling model-checking tools such as Uppaal and Uppaal SMC.

On Hierarchical Scheduling Our first domain-specific tool is designed to analyze hierarchical scheduling systems, using a stochastic model to represent stochastic behavior of the system, this model gives us the possibility to represent more complex systems and analyze their schedulability.

As an example for our first framework we propose to model an avionic system, on which we execute a number of experiments in order to verify its schedulability, and also to compute the minimum budget for which the whole system is schedulable. We present in this manuscript the results of these experiments.

On Energy Consumption We have also presented new statistical model-checking algorithms that perform optimization or runtime monitoring. These algorithms are based on statistical tests like ANOVA and CUSUM. They are implemented and embedded into our analysis tools. Our second domain-specific tool is designed to analyze scheduling of multi-processor systems with the energy constraint.

For this second framework, we propose an example to illustrate how to use the algorithms proposed for the optimization of the energy consumption and the detection of significant probability variation. The results of these experiments and their analyses are presented to consolidate our work.

Second, we proposed a new model that consider information leakage as quantitative resource that the scheduler can exploit In a system with shared resources, the security of confidential information is a major concern.

On Information Leakage This manuscript allows reasoning about leakage of confidential information by extending the workflow model to support fine-grained preemption and confidentiality. This allows confidentiality to be addressed by quantifying the amount of information leaked by the system, including different leakage models.

Scheduling in this new model is then considered using pre-and post-processors. These can be computationally combined for scheduling that exploits different techniques and approaches, including focusing on different aspects of the overall problem. Several pre- and post-processing heuristic algorithms are presented that can operate on the model. These are focused on improving resulting leakage, but the principles can be adapted to other problems as well.

In order to evaluate these heuristic algorithms, experiments were conducted by using approximately 30,000 randomly generated sets of tasks, then testing each possible combination of one pre-processing, one scheduling, and one post-processing algorithm. The results of these experiments demonstrate that the model and the heuristics improve over the state of the art and show that even simple heuristics can be effective.

Future work the next step in our work would be to improve our models by extending the model banks in order to analyze more complexe systems. Adding to that, we can also focus on improving treatment time by using parallelism theory advantages, these improvements allow as to treat more information in a small time which would improve the confidence of our results.

Another axis that we can follow could be to generalize our models to deal with multi-resource approaches, where scheduling algorithms considers a number of different properties at the same time. For example consider confidentiality, energy consumption, schedulability, etc.

In fact, multi-resource approaches could be a very good solution to analyze complex systems that have energy constraints, and must interact with other systems to accomplish its mission at the same time. These interactions could expose the confidential information of the system under consideration.

Concerning the model that consider confidentiality problem, one direction that can be followed would be to generalize this model in order to treat on-line scheduling algorithms. On-line scheduling algorithms are designed to deal with real-time systems that interact with external environment through different sensors. The on-line scheduling algorithm must schedule the entering task and take into consideration the confidentiality problem.

Another direction would be to consider theoretical complexity, that can help to improve the efficiency of the heuristic algorithms. The heuristic algorithms could be proposed to aim different objectives, as example one objective could be to find a solution that gives optimal confidentiality by reducing the resulting leakage to zero. Another objective could be to find a solution with a fixed value of the resulting leakage.

Finally, in our work we proposed a composed strategy that combines pre and post-processors in order to solve the confidentiality problem, a good direction that could be followed is to improve this strategy or to propose other strategies that could give better solutions for this problem.

Listes de travaux de Mounir Chadli

1. **Titre:** High-level frameworks for the specification and verification of scheduling problems.
   Authors: Mounir Chadli, Jin Hyun Kim, Kim Guldstrand Larsen, Axel Legay, Stefan Naujokat, Bernhard Steffen, Louis-Marie Traonouez.
   Actes: International Journal on Software Tools for Technology Transfer (STTT), pages 1-26, 2017.

2. **Titre:** Information Leakage as a Scheduling Resource.
   Authors: Fabrizio Biondi, Mounir Chadli, Thomas Given-Wilson, Axel Legay.
   Actes: International Workshop on Formal Methods for Industrial Critical Systems and Automated Verification of Critical Systems (FMICS-AVoCS), LNCS, volume 10471, pages 83-99, 2017

3. **Titre:** A Model-Based Framework for the Specification and Analysis of Hierarchical Scheduling Systems.
   Authors: Mounir Chadli, Jin Hyun Kim, Axel Legay, Louis-Marie Traonouez, Stefan Naujokat, Bernhard Steffen, Kim Guldstrand Larsen.
   Actes: International Workshop on Formal Methods for Industrial Critical Systems and Automated Verification of Critical Systems (FMICS-AVoCS), LNCS, volume 9933, pages 133-141, 2016

Bibliography

[ACPS12] Mário S. Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In Stephen Chong, editor, CSF. IEEE, 2012.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. Theor. Comput. Sci., 126(2):183–235, April 1994.

[ALTP04] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. Theor. Comput. Sci., 318(3):297–322, June 2004.

[BCGL17] Fabrizio Biondi, Mounir Chadli, Thomas Given-Wilson, and Axel Legay. Information leakage as a scheduling resource. In FMICS-AVoCS, volume 10471 of Lecture Notes in Computer Science, pages 83–99. Springer, 2017.

[BcRS13] Anne Benoit, Ümit V. Çatalyürek, Yves Robert, and Erik Saule. A survey of pipelined workflow scheduling: Models and algorithms. ACM Comput. Surv., 45(4):50:1–50:36, August 2013.

[BDK+13] A. Boudjadar, A. David, J.H. Kim, K.G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Hierarchical scheduling framework based on compositional analysis using uppaal. In Proceedings of the 10th International Symposium on Formal Aspects of Component Software (FACS), Revised Selected Papers, volume 8348 of LNCS, pages 61–78. Springer, 2013.

[BDK+14] A.Jalil Boudjadar, Alexandre David, Jin Hyun Kim, Kim G. Larsen, Marius Mikučionis, Ulrik Nyman, and Arne Skou. Degree of schedulability of mixed-criticality real-time systems with probabilistic sporadic tasks. In Theoretical Aspects of Software Engineering Conference (TASE), 2014, pages 126–130, Sept 2014.

[BDK+15a] Abdeldjalil Boudjadar, Alexandre David, JinHyun Kim, KimGuldstrand Larsen, Marius Mikučionis, Ulrik Nyman, and Arne Skou. Widening the schedulability of hierarchical scheduling systems. In Proceedings of the 11th International Symposium on Formal Aspects of Component Software (FACS), Revised Selected Papers, volume 8997 of LNCS, pages 209–227. Springer, 2015.

[BDK+15b] Jalil Boudjadar, Alexandre David, Jin Hyun Kim, Kim Guldstrand Larsen, Marius Mikučionis, Arne Skou, Insup Lee, Linh Phan Xuan, and Ulrik Nyman. Quantitative schedulability analysis of continuous probability tasks in a hierarchical context. Springer International Publishing, 2015. To be appeared.

[BDL+06] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In Third International Conference on the Quantitative Evaluation of Systems (QEST), pages 125–126, 2006.

[BFH+01] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-cost reachability for priced timed automata. In Proceedings of the 4th International Workshop on Hybrid Systems: Computation and Control (HSCC), pages 147–161. Springer, 2001.

[BKD+] A.Jalil Boudjadar, Jin Hyun Kim, Alexandre David, Kim G. Larsen, Marius Mikučionis, Ulrik Nyman, Arne Skou, Insup Lee, and Lihn Thi Xuan Phan. Flexible framework for statistical schedulability analysis off probabilistic sporadic tasks. In 18th International Symposium of Real-Time Distributed Computing (ISORC). To be appeared.

[BKR09] Michael Backes, Boris Köpf, and Andrey Rybalchenko. Automatic discovery and quantification of information leaks. In S&P, pages 141–153. IEEE, 2009.

[BLMW15] Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wasowski. Quantifying information leakage of randomized protocols. Theor. Comput. Sci., 597:62–87, 2015.

[BLTW13] Fabrizio Biondi, Axel Legay, Louis-Marie Traonouez, and Andrzej Wasowski. QUAIL: A quantitative security analyzer for imperative code. In

Natasha Sharygina and Helmut Veith, editors, CAV, volume 8044 of LNCS, pages 702–707. Springer, 2013.

[BMR90] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In In Proceedings of the 11th Real-Time Systems Symposium, pages 182–190. IEEE Computer Society Press, 1990.

[BN93] Michèle Basseville and Igor V. Nikiforov. Detection of Abrupt Changes: Theory and Application. Prentice-Hall, Inc., 1993.

[CBF+11] Mikel Cordovilla, Frédéric Boniol, Julien Forget, Eric Noulard, and Claire Pagetti. Developing critical embedded systems on multicore architectures: the PRELUDE-SCHEDMCORE toolset. In Sébastien Faucou, Alan Burns, and Laurent George, editors, RTNS 2011, pages 107–116, 2011.

[CFO+11] Amedeo Cesta, Simone Fratini, Andrea Orlandini, Alberto Finzi, and Enrico Tronci. Flexible plan verification: Feasibility results. Fundam. Inform., 107(2-3):111–137, 2011.

[CKL+16] Mounir Chadli, Jin Hyun Kim, Axel Legay, Louis-Marie Traonouez, Stefan Naujokat, Bernhard Steffen, and Kim Guldstrand Larsen. A model-based framework for the specification and analysis of hierarchical scheduling systems. In Proceedings of the Joint 21st International Workshop on Formal Methods for Industrial Critical Systems and 16th International Workshop on Automated Verification of Critical Systems (FMICS-AVoCS), volume 9933 of LNCS, pages 133–141. Springer, 2016.

[CKN14] Tom Chothia, Yusuke Kawamoto, and Chris Novakovic. Leakwatch: Estimating information leakage from java programs. In Miroslaw Kutylowski and Jaideep Vaidya, editors, ESORICS, volume 8713 of LNCS, pages 219–236. Springer, 2014.

[CL00] Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In Proceedings of the 11th International Conference on Concurrency Theory (CONCUR), pages 138–152. Springer, 2000.

[CMR16] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Dynamic controllability of disjunctive temporal networks: Validation and synthesis of exe-

cutable strategies. In Proceedings of the 30th AAAI Conference on Artificial Intelligence, pages 3116–3122. AAAI Press, 2016.

[CMR17] Alessandro Cimatti, Andrea Micheli, and Marco Roveri. Validating domains and plans for temporal planning via encoding into infinite-state linear temporal logic. In Proceedings of the 31st AAAI Conference on Artificial Intelligence, pages 3547–3554. AAAI Press, 2017.

[CMS14] Rohit Chadha, Umang Mathur, and Stefan Schwoon. Computing information flow using symbolic model-checking. In Venkatesh Raman and S. P. Suresh, editors, FSTTCS, volume 29 of LIPIcs, pages 505–516. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[DDGL+13] Alexandre David, Dehui Du, Kim Guldstrand Larsen, Axel Legay, and Marius Mikučionis. Optimizing control strategy using statistical model checking. In NASA Formal Methods: Proceedings of the 5th International Symposium (NFM), pages 352–367. Springer, 2013.

[DDL+12] Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. In Proceedings of the First International Workshop on Hybrid Systems and Biology (HSB), volume 92 of EPTCS, pages 122–136, 2012.

[DLL+11] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical model checking for networks of priced timed automata. In Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS), volume 6919 of LNCS, pages 80–96. Springer, 2011.

[DLL+15] Alexandre David, KimG. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. International Journal on Software Tools for Technology Transfer, pages 1–19, 2015.

[DLLM12] Alexandre David, Kim Guldstrand Larsen, Axel Legay, and Marius Mikučionis. Schedulability of herschel-planck revisited using statistical model checking. In Proceedings of 5th International Symposium ISoLA, Part II, volume 7610 of LNCS, pages 293–307. Springer, 2012.

[FMC11] Nicolas Falliere, Liam O. Murchu, and Eric Chien. W32.Stuxnet dossier, 2011.

[Gra66] R. L. Graham. Bounds for certain multiprocessing anomalies. The Bell System Technical Journal, 45(9):1563–1581, Nov 1966.

[Gro08] Richard C. Gronback. Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley, 2008.

[Hen00] Thomas A. Henzinger. The Theory of Hybrid Automata, pages 265–292. Springer, 2000.

[JLM+12] Sven Jï£¡rges, Anna-Lena Lamprecht, Tiziana Margaria, Ina Schaefer, and Bernhard Steffen. A Constraint-based Variability Modeling Framework. International Journal on Software Tools for Technology Transfer (STTT), 14(5):511–530, 2012.

[KLL+15a] Jin Hyun Kim, Axel Legay, Kim G. Larsen, Marius Mikučionis, and Brian Nielsen. Resource-parameterized timing analysis of real-time systems. In Hardware and Software: Verification and Testing: Proceeding of the 11th International Haifa Verification Conference (HVC), pages 190–205. Springer, 2015.

[KLL+15b] Jin Hyun Kim, Axel Legay, Kim Guldstrand Larsen, Marius Mikučionis, and Brian Nielsen. Resource-parameterized timing analysis of real-time systems. In Nir Piterman, editor, HVC, volume 9434 of LNCS, pages 190–205. Springer, 2015.

[KLT+16a] Jin Hyun Kim, Axel Legay, Louis-Marie Traonouez, Abdeldjalil Boudjadar, Ulrik Nyman, Kim G. Larsen, Insup Lee, and Jin-Young Choi. Optimizing the resource requirements of hierarchical scheduling systems. SIGBED Rev., 13(3):41–48, August 2016.

[KLT+16b] Jin Hyun Kim, Axel Legay, Louis-Marie Traonouez, Abdeldjalil Boudjadar, Ulrik Nyman, Kim G. Larsen, Insup Lee, and Jin-Young Choi. Optimizing the resource requirements of hierarchical scheduling systems. SIGBED Review, 13(3):41–48, 2016.

[KZH+11] Joost-Pieter Katoen, Ivan S Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N Jansen. The ins and outs of the probabilistic model checker mrmc. Performance evaluation, 68(2):90–104, 2011.

[LDB10] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In Proceedings of the First International Conference on Runtime Verification (RV), volume 6418 of LNCS, pages 122–135. Springer, 2010.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM, 20(1):46–61, January 1973.

[LLG90] Douglas Locke, Lee Lucas, and John Goodenough. Generic avionics software specification. Technical Report CMU/SEI-90-TR-008, Software Engineering Institute, 1990.

[LNS13] Anna-Lena Lamprecht, Stefan Naujokat, and Ina Schaefer. Variability Management Beyond Feature Models. Computer, 46(11):48–54, 2013.

[LT16] Axel Legay and Louis-Marie Traonouez. Statistical model checking with change detection. Transactions on Foundations for Mastering Change I, 1:157–179, 2016.

[MCG13] Dorin Maxim and Liliana Cucu-Grosjean. Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters. In RTSS 2013 - IEEE Real-Time Systems Symposium, Vancouver, Canada, 2013.

[MEP07] Sorin Manolache, Petru Eles, and Zebo Peng. Analysis of monoprocessor systems. In Real-Time Applications with Stochastic Task Execution Times, pages 27–60. Springer Netherlands, 2007.

[Mok83] Aloysius Ka-Lau Mok. Fundamental design problems of distributed systems for the hard-real-time environment. PhD thesis, Massachusetts Institute of Technology, 1983.

[Mon06] Douglas C. Montgomery. Design and Analysis of Experiments. John Wiley & Sons, 2006.

[MS09] Tiziana Margaria and Bernhard Steffen. Business Process Modelling in the jABC: The One-Thing-Approach. In Handbook of Research on Business Process Modeling. IGI Global, 2009.

[MS10] Tiziana Margaria and Bernhard Steffen. Simplicity as a Driver for Agile Innovation. Computer, 43(6):90–92, 2010.

[MYPB14] Sibin Mohan, Man-Ki Yoon, Rodolfo Pellizzoni, and Rakesh Bobba. Real-time systems security through scheduler constraints. In ECRTS, pages 129–140. IEEE Computer Society, 2014.

[NLKS17] Stefan Naujokat, Michael Lybecait, Dawid Kopetzki, and Bernhard Steffen. CINCO: A Simplicity-Driven Approach to Full Generation of Domain-Specific Graphical Modeling Tools. Software Tools for Technology Transfer, 2017. to appear.

[NTI+14] Stefan Naujokat, Louis-Marie Traonouez, Malte Isberner, Bernhard Steffen, and Axel Legay. Domain-Specific Code Generator Modeling: A Case Study for Multi-faceted Concurrent Systems. In Proc. of the 6th Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation, Part I (ISoLA), number 8802 in LNCS, pages 463–480. Springer, 2014.

[ORC15] A. Oddi, R. Rasconi, and A. Cesta. A multi-objective large neighborhood search methodology for scheduling problems with energy costs. In 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), pages 453–460, Nov 2015.

[Pag54] E. S. Page. Continuous inspection schemes. Biometrika, 41(1/2):100–115, 1954.

[PLE+11] Linh T. X. Phan, Jaewoo Lee, Arvind Easwaran, Vinay Ramaswamy, Sanjian Chen, Insup Lee, and Oleg Sokolsky. CARTS: A tool for compositional analysis of real-time systems. SIGBED Rev., 8(1):62–63, March 2011.

[PPY+15] Rodolfo Pellizzoni, Neda Paryab, Man-Ki Yoon, Stanley Bak, Sibin Mohan, and Rakesh Bobba. A generalized model for preventing information leakage in hard real-time systems. In RTAS. IEEE, 2015.

[SA06] Joon Son and Jim Alves-Foss. Covert timing channel capacity of rate monotonic real-time scheduling algorithm in MLS systems. In Sanguthevar Rajasekaran, editor, IASTED, pages 13–18. IASTED/ACTA Press, 2006.

[SBPM08] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. EMF: Eclipse Modeling Framework (2nd Edition). Addison-Wesley, 2008.

[SEL08a] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In Euromicro Conference on Real-Time Systems, pages 181–190, July 2008.

[SEL08b] Insik Shin, Arvind Easwaran, and Insup Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In ECRTS, pages 181–190. IEEE Computer Society, 2008.

[SFC08] D. Smith, J. Frank, and W. Cushing. The anml language. In In ICAPS Poster session, 2008.

[SL03] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In Proceedings of the 24th IEEE International Real-Time Systems Symposium (RTSS), pages 2–13. IEEE Computer Society, 2003.

[SMD00] Sang Hyuk Son, Ravi Mukkamala, and Rasikan David. Integrating security and real-time requirements using covert channel capacity. IEEE Trans. Knowl. Data Eng., 12(6):865–879, 2000.

[SVA04] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In CAV, volume 3114, pages 202–215. Springer, 2004.

[SVA05a] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In CAV, volume 3576, pages 266–280. Springer, 2005.

[SVA05b] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In Quantitative Evaluation of Systems, 2005. Second International Conference on the, pages 251–252. IEEE, 2005.

[TDP12] Abhilash Thekkilakattil, Radu Dobrin, and Sasikumar Punnekkat. Probabilistic preemption control using frequency scaling for sporadic real-time tasks. In The 7th IEEE International Symposium on Industrial Embedded Systems, June 2012.

[VEB+16] Celina G. Val, Michael A. Enescu, Sam Bayless, William Aiello, and Alan J. Hu. Precisely measuring quantitative information flow: 10k lines of code and beyond. In EuroS&P. IEEE, 2016.

[VRS14] Venkatanathan Varadarajan, Thomas Ristenpart, and Michael M Swift. Scheduler-based defenses against cross-vm side-channels. In Usenix Security, pages 687–702, 2014.

[Wal45] A. Wald. Sequential Tests of Statistical Hypotheses. The Annals of Mathematical Statistics, 16(2):117–186, 1945.

[YMCS16] Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In RTAS, pages 1–12. IEEE, 2016.

[You05] Hakan L Younes. Verification and planning for stochastic processes with asynchronous events. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2005.

[You06] Håkan LS Younes. Error control for probabilistic model checking. In International Workshop on Verification, Model Checking, and Abstract Interpretation, pages 142–156. Springer, 2006.

[ZKG+08] Yuanfang Zhang, Donald K. Krecker, Christopher Gill, Chenyang Lu, and Gautam H. Thaker. Practical schedulability analysis for generalized sporadic tasks in distributed real-time systems. In Proceedings of the 2008 Euromicro Conference on Real-Time Systems, ECRTS '08, pages 223–232, Washington, DC, USA, 2008. IEEE Computer Society.

List of Figures