

Rigorous System-level Modeling and Performance Evaluation for Embedded System Design

Ayoub Nouri

▶ **To cite this version:**

Ayoub Nouri. Rigorous System-level Modeling and Performance Evaluation for Embedded System Design. Other [cs.OH]. Université Grenoble Alpes, 2015. English. NNT : 2015GREAM008 . tel-01148690

HAL Id: tel-01148690

https://theses.hal.science/tel-01148690

Submitted on 5 May 2015

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.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Ayoub Nouri

Thèse dirigée par **Saddek Bensalem**
et codirigée par **Marius Bozga**

préparée au sein **du laboratoire Verimag**
et de **l'École Doctorale Mathématique, Science et Technologie de l'Information, Informatique (MSTII)**

Rigorous System-level Modeling and Performance Evaluation for Embedded Systems Design

Thèse soutenue publiquement le **8 avril 2015**,
devant le jury composé de :

M. Jean Claude Fernandez
Professeur, Université de Grenoble, Président
M. Albert Cohen
Directeur de recherche, INRIA, Rapporteur
M. Radu Grosu
Professeur, Vienna University of Technology, Rapporteur
M. Kamel Barkaoui
Professeur, Conservatoire National des Arts et Métiers, Examinateur
M. Ahmed Bouajjani
Professeur, Université Paris Diderot, Examinateur
M. Saddek Bensalem
Professeur, Université de Grenoble, Directeur de thèse
M. Marius Bozga
Ingénieur de recherche – HDR, CNRS, Co-Directeur de thèse
M. Axel Legay
Chargé de recherche, INRIA, Co-Encadrant de thèse

Acknowledgments

Foremost, I would like to thank Saddek Bensalem for giving me the opportunity to join Verimag and to achieve my PhD work under his supervision. He offered me motivation, valuable research directions, and more importantly, a very nice working environment that helped me to accomplish this work.

I am deeply thankful to Marius Bozga, my co-advisor, for his availability, his valuable advice, and for the long and rich discussions, often beyond the work scope. Without his help, this work could never have been the same.

A great thank goes to Axel Legay for assisting and inspiring me in various parts of the work. Especially, the stochastic extension of the BIP framework and the statistical model checking technique.

I am very grateful to all the jury members for accepting to review this work and for their valuable feedback on the manuscript and the presentation.

I want also to thank people from CEA Grenoble, the LIALP group, for their collaboration on the HMAX case study, and professor Bernard Ycart from University of Grenoble for his help on statistical aspects.

I want to thank all my colleges at Verimag for the nice, healthy, and rich working environment, all the administrative staff for their help and assistance. A wink to the Verimag football team. I also thank Yassine Lacknech for giving me the opportunity to make an internship at Verimag in 2010.

Finally, I would like to thank my family : my parents, my brothers, my sister, and my in-laws, for their support and their affection. A special thank to my beloved wife for her unconditional support. Without her by my side this work could never have been achieved.

I dedicate this work to my family

Abstract

In the present work, we tackle the problem of modeling and evaluating performance in the context of embedded systems design. These have become essential for modern societies and experienced important evolution. Due to the growing demand on functionality and programmability, software solutions have gained in importance, although known to be less efficient than dedicated hardware. Consequently, considering performance has become a must, especially with the generalization of resource-constrained devices.

We present a rigorous and integrated approach for system-level performance modeling and analysis. The proposed method enables faithful high-level modeling, encompassing both functional and performance aspects, and allows for rapid and accurate quantitative performance evaluation. The approach is model-based and relies on the $\mathcal{S}$BIP formalism for stochastic component-based modeling and formal verification. We use statistical model checking for analyzing performance requirements and introduce a stochastic abstraction technique to enhance its scalability. Faithful high-level models are built by calibrating functional models with low-level performance information using automatic code generation and statistical inference.

We provide a tool-flow that automates most of the steps of the proposed approach and illustrate its use on a real-life case study for image processing. We consider the design and mapping of a parallel version of the HMAX models algorithm for object recognition on the STHORM many-cores platform. We explored timing aspects and the obtained results show not only the usability of the approach but also its pertinence for taking well-founded decisions in the context of system-level design.

Résumé

Les systèmes embarqués ont évolué d'une manière spectaculaire et sont devenus partie intégrante de notre quotidien. En réponse aux exigences grandissantes en termes de nombre de fonctionnalités et donc de flexibilité, les parties logicielles de ces systèmes se sont vues attribuer une place importante malgré leur manque d'efficacité, en comparaison aux solutions matérielles. Par ailleurs, vu la prolifération des systèmes nomades et à ressources limités, tenir compte de la performance est devenu indispensable pour bien les concevoir.

Dans cette thèse, nous proposons une démarche rigoureuse et intégrée pour la modélisation et l'évaluation de performance tôt dans le processus de conception. Cette méthode permet de construire des modèles, au niveau système, conformes aux spécifications fonctionnelles, et intégrant les contraintes non-fonctionnelles de l'environnement d'exécution. D'autre part, elle permet d'analyser quantitativement la performance de façon rapide et précise. Cette méthode est guidée par les modèles et se base sur le formalisme $\mathcal{S}$BIP que nous proposons pour la modélisation stochastique selon une approche formelle et par composants.

Pour construire des modèles conformes, nous partons de modèles purement fonctionnels utilisés pour générer automatiquement une implémentation distribuée, étant donnée une architecture matérielle cible et un schéma de répartition. Dans le but d'obtenir une description fidèle de la performance, nous avons conçu une technique d'inférence statistique qui produit une caractérisation probabiliste. Cette dernière est utilisée pour calibrer le modèle fonctionnel de départ. Afin d'évaluer la performance de ce modèle, nous nous basons sur du model checking statistique que nous améliorons à l'aide d'une technique d'abstraction.

Nous avons développé un flot de conception qui automatise la majorité des phases décrites ci-dessus. Ce flot a été appliqué à différentes études de cas, notamment à une application de reconnaissance d'image déployée sur la plateforme multi-cœurs STHORM.

Contents

1 Introduction **1**
- 1.1 Motivation and Methodology . . . 1
- 1.2 Embedded Systems Design . . . 3
  - 1.2.1 Design and Challenges . . . 3
  - 1.2.2 System-level Design . . . 5
  - 1.2.3 State of the Art . . . 8
- 1.3 Performance in System-level Design . . . 10
  - 1.3.1 Challenges . . . 11
  - 1.3.2 Performance Modeling Requirements . . . 12
  - 1.3.3 Performance Evaluation Requirements . . . 17
- 1.4 Contributions and Organization . . . 19
  - 1.4.1 System-level Modeling . . . 20
  - 1.4.2 System-level Verification . . . 21
  - 1.4.3 Integrated Performance Modeling and Analysis . . . 22

I Formalisms and Techniques **23**

2 Quantitative Analysis of Stochastic Models: A Background **25**
- 2.1 Stochastic Systems Modeling . . . 26
  - 2.1.1 Discrete Time Markov Chains . . . 26
  - 2.1.2 Markov Decision Process . . . 29
- 2.2 Requirements Formalization . . . 32
  - 2.2.1 Linear-time Temporal Logic . . . 32
- 2.3 Statistical Model Checking . . . 33
  - 2.3.1 Probabilistic Bounded LTL . . . 35
  - 2.3.2 Qualitative Analysis . . . 35
  - 2.3.3 Quantitative Analysis . . . 36
  - 2.3.4 Playing with Statistical Model Checking Algorithms . . . 37
- 2.4 Conclusions and Related Work . . . 37

3 Stochastic Component-based Modeling 41
3.1 Background on BIP . . . 42
3.1.1 Atomic Components . . . 42
3.1.2 Composition Operators . . . 45
3.2 $\mathcal{S}$BIP: A Stochastic Extension . . . 50
3.2.1 Stochastic Atomic Components . . . 50
3.2.2 Composition of Stochastic Components . . . 55
3.2.3 Purely Stochastic Semantics . . . 57
3.3 Performance Evaluation of a Multimedia SoC . . . 60
3.3.1 $\mathcal{S}$BIP Model of the Multimedia SoC . . . 61
3.3.2 QoS Requirements Evaluation . . . 64
3.4 $\mathcal{S}$BIP Expressiveness . . . 68
3.4.1 Modeling LMCs in $\mathcal{S}$BIP . . . 68
3.4.2 Modeling MDPs in $\mathcal{S}$BIP . . . 72
3.5 Conclusions . . . 77

4 Stochastic Abstraction 79
4.1 Preliminaries . . . 81
4.1.1 Additional Stochastic Models . . . 81
4.1.2 Probabilistic Learning Techniques . . . 81
4.2 Abstraction via Learning and Projection . . . 83
4.2.1 Abstraction Steps . . . 84
4.2.2 Correctness Statement . . . 86
4.3 Herman's Self Stabilizing Algorithm . . . 88
4.4 Conclusions and Related Work . . . 93

II Methods and Flow 97

5 Rigorous System-level Performance Modeling and Analysis 99
5.1 *ASTROLABE*: An Integrated Approach . . . 100
5.1.1 Answering General Design Challenges . . . 100
5.1.2 Answering Performance Specific Challenges . . . 101
5.2 Gathering Low-level Performance Details . . . 104
5.2.1 Automatic Implementation Generation . . . 104
5.2.2 Instrumentation . . . 105
5.3 Characterizing Performance Information . . . 105
5.4 Calibrating Functional Models . . . 108
5.4.1 Timing Information . . . 108
5.5 Conclusions . . . 113

6 Statistical Characterization of Performance 115
6.1 Statistical Inference . . . 115
6.2 Distribution Fitting . . . 117
6.2.1 Exploratory Analysis . . . 117
6.2.2 Parameters Estimation . . . 119
6.2.3 Evaluation of the Obtained Candidates . . . 121
6.3 Conclusions . . . 126

7 The *ASTROLABE* Tool-flow 127
7.1 Overview . . . 127
7.2 BIP2MCAPI Code Generator . . . 128
7.2.1 Process Generation . . . 129
7.2.2 Glue Code Generation . . . 130
7.2.3 Distributed Code Generation within BIP . . . 130
7.3 FitDist: A Distribution Fitting Tool . . . 131
7.4 Stochastic Abstraction Tool . . . 132
7.5 BIP$^{\mathcal{SMC}}$: An SMC Engine for $\mathcal{S}$BIP Models . . . 132
7.5.1 $\mathcal{S}$BIP Modeling Language . . . 133
7.5.2 Properties Specification Language . . . 134
7.5.3 Technical details and Tool Availability . . . 135
7.6 Integration within the BIP Design Flow . . . 136
7.6.1 DOL and DOL2BIP . . . 136
7.6.2 BIPWeaver and the BIP HW Components Library . . . 138
7.6.3 D-Finder . . . 138
7.7 Conclusions . . . 139

8 Image Recognition on Many-cores 141
8.1 Application Overview . . . 141
8.2 Hw Architecture Overview . . . 142
8.2.1 The STHORM Platform . . . 142
8.2.2 The Multi-core Communication API . . . 143
8.3 Performance Requirements Overview . . . 144
8.4 Functional and Performance Modeling . . . 145
8.4.1 High-Level Reconfigurable KPN Model . . . 145
8.4.2 Performance Characterization and Model Calibration . . . 149
8.5 Performance Evaluation . . . 156
8.6 Conclusions . . . 158

9 Conclusions and Perspectives 161

List of Figures 171

List of Tables 173

Bibliography 175

Chapter 1

Introduction

1.1 Motivation and Methodology

Computerized systems have become essential for modern societies. Embedded systems, in particular, have deeply impacted our daily lives and radically influenced our lifestyle. The important growth of transistors and microelectronics industries has contributed to democratize these systems which became ubiquitous. According to the ARTEMIS Strategic Research Agenda 2011 [1], it is estimated that there will be over 40 billion devices worldwide, that is, 5 to 10 embedded devices per person on earth by 2020.

Wireless communication capabilities offered by these systems have allowed peoples to get (inter-)connected everywhere, quickly, and easily, e.g., in cars, trains, planes. This has changed our perception of many concepts such as human relationships, where new possibilities have emerged. This can be clearly observed on the wide use of social networks and the emergence of Internet of Things (IoT). Emergence of participative models of democracy and governance, where citizens play a central role would have never been possible without embedded devices. Carrying out processing power changed our learning, working, and entertaining habits (allowing for instance teleworking). Abundance of sensors accompanying these devices, gave rise to new types of media that give an active role to citizens. For instance camera and social networks have contributed widely in the recent "arab spring".

Beyond individual and social scales, embedded systems are becoming essential for companies and even for governments and states. These represent an important leverage for innovation and competitivity for companies, in addition to creating new markets. Prosperity and growth of many industries are due to embedded systems as witnessed by the evolution of the automotive field, where embedded systems are gaining more importance, while

1. ftp://ftp.cordis.europa.eu/pub/technology-platforms/docs/sra-2011-book-page-by-page-9.pdf

mechanical-based solutions are stagnating. At the edge of great energetic challenges, energy efficiency became a must. Global initiatives at the level of states advocating more effective energy management techniques are increasing. Devices such as smart meters, monitors, sensors and more sophisticated setups such as micro-grids are hence sought. Embedded systems represent an opportunity for affordable health care systems. Recent advances in medical devices are various and ranging from medical imaging to pacemaker, and artificial heart. Domains benefiting from embedded systems assets, such as transportation, national security, and education are wide and steadily increasing. This evolution have contributed to draw a completely new lifestyle where mobility, speed and connectivity are the keywords.

The great impact of embedded systems on our everyday lives, come at the price of an increasing complexity to design them. More burden is put on designers that have to produce systems in less time with ever reducing costs. Designing mixed hardware/software systems that provide sophisticated services is inherently challenging. Additional constraints such as the shrinking time to market make it even harder. Besides, embedded systems are often used in critical setups involving human lives and wealth. Thus, ensuring their functional correctness is primordial.

Efficiency is becoming a real concern for modern embedded systems. As our reliance on these systems increases, so does the expectation that they will provide us with more mobility and ensure high-performance response. Embedded devices are mostly sibling mass consumption and often rely on batteries. Thus, cost optimization and energy management are of paramount importance. Moreover, such devices are increasingly integrating various functionality, hence they are required to provide high and often real-time performance. Classical views giving more importance to functional aspects are becoming obsolete and new design methods, equally capturing both aspects, are becoming a must.

This thesis aims at contributing to bridge the gap between the current state-of-the-art methods and techniques of embedded systems design and the growing challenges facing this area. Our main focus is on performance aspects, since these are still not well supported and represent a considerable hindrance towards substantial advances in this field. To accomplish our goal, we will start from concrete challenges and proceed from general to specific, that is, we will first consider general design challenges, then address performance issues. The main advantage of such an approach is consistency, since we are guarantied not to fall into contradictions between general design and specific performance questions. This way of proceeding will also help us to analyze existing answers for our problem and to build upon them. Analysis of challenges will lead us to identify the key requirements that need to be matched for an appropriate answer. Finally, to enable tackling this complex task, we will decompose our global objective in term of performance modeling and performance analysis.

1.2 Embedded Systems Design

1.2.1 Design and Challenges

Systems' design entails building software/hardware artifacts matching user defined specifications and requirements, which are often seen twofold:

- *Functional*, which are related to the expected functionality of the system. For example, an ATM is expected to deliver the specified amount of money when the used credit card is valid and when the specified amount of money is available.
- *Extra-functional*, that concern resources utilization, such as performance, cost, or security aspects. For the previous example for instance, the time to deliver money should be constant and lower that some bound.

The path leading to a potential design from initial specifications is the *design process*. A *design method* often decomposes the design process to several activities and provides a clear way these must be performed towards a valid artifact, that is, conforms to the initial specifications.

The evolution of embedded systems from centralized to more distributed settings and their deployment in more open and unpredictable environments have led embedded systems to be confronted with an increasing number of challenges. Designing such systems requires methods that account for extra-functional requirements concerning energy, timing, and memory while guarantying functional properties such as reactivity, reliability and robustness. Considering both aspects, i.e., functional and extra-functional, rises several difficulties at different levels, ranging from theoretical to more technical.

Sophisticated Functionality The wide acceptance of embedded systems and their success to improve our everyday lives, e.g., for communication, transportation, and health, induced a move towards new systems with more capabilities and increasing intelligence. These range from simple individual devices to widely distributed plans at the scale of cities and states. The IBM smarter planet is one among many initiatives that aims at using embedded systems to further improve the quality of life of human being. This initiative consists of deploying a huge number of embedded devices to manage vital resources, e.g., water, energy, in smarter way. Consequently, new kinds of systems, e.g., sensors network, Swarms of devices, which are often bio-inspired have emerged. These are basically networks of distributed, hybrid, and autonomous devices with adaptive capabilities and limited resources. Such systems imply many challenges varying from real-time constraints to distributed issues.

Critical Tasks In spite of their increasing complexity due to numerous and sophisticated functionality, embedded systems are increasingly used in

critical setups where human lives and wealth are involved. Embedded systems are for instance used to ensure safety, e.g., anti-lock braking systems (ABS) and air-bag systems. They are also used for industrial control, for temperature regulation in chemical and nuclear power plants, for traffic control, in addition to medical devices. These are often known as safety-critical systems and require rigorous methods of design and especially of verification to ensure their correctness.

Complex and Unpredictable Environment Embedded devices are reactive systems that continuously interact with their environment. They are mostly embedded on larger systems and often embody the control part. A big challenge in designing such system is to reconcile the physical part of the system, which is continuous and concurrent by nature with the computerized part, which is discrete and sequential. Furthermore, embedded systems behave in response to external stimuli perceived through sensors, and operate changes on the environment performed via actuators, following specified strategies. The environment has thus an important impact on the system behavior. Traditionally, embedded systems environments were believed to be well defined, however, this is not the case for modern systems, which are deployed in very complex and uncertain contexts. Account for environment uncertainties is hence quite important.

Heterogeneous Systems The number of functionality that a modern embedded device has to offer is increasingly important. Programmable hardware blocks are thus need, in addition to dedicated ones, to ensure flexibility. A mix of software and hardware components having different characteristics are thus used. Whereas, software algorithms relies on logical and sequential reasoning, hardware components have concurrent behavior and are based continuous notions. In such systems continuous and discrete time, synchronous and asynchronous communications, analog and digital components co-exist. Furthermore, do to the increasing complexity and diversity of functionality, the different components of an embedded systems are no more realized by a single team. Instead, various teams with different expertise are involved and some parts may be purchased as IPs from different suppliers. Ensuring interoperability of all these components is thus necessary.

Increasingly Computational Power The great move from single core processors to multi-cores and many-cores architectures due to Moore's Law (limit of integration) has brought an important processing power, e.g., multi-cores and many-cores architectures, for modern systems. However, new challenges have emerged in parallel to deal with these complex architectures. Efficiently programming such complex architectures and exploiting the computation power they provide is real concern.

Competition and Shrinking Time to Market The wide use of embedded devices and their success to metamorphose different application domains have induced a fast growth of the embedded systems market, which was valued at USD 140.32 billion in 2013, and is expected to reach USD 214.39 billion by 2020 (Grand View Research). An increasing number of players, from different horizons, such as Google or Microsoft, historically focused on software services, are showing more interest on this market. The competition is such that the cost of being late to the market is staggering as witnessed by phones or cars markets. Products delivery time, namely time to market, is thus drastically shrinking and costs optimization became very important.

Performance Embedded devices were traditionally used to perform dedicated tasks, often of industrial nature. They were designed using application specific non-programmable circuits. These ensure low power consumption, real-time requirements, low cost, and silicon efficiency. The major breakthrough of embedded devices in our everyday lives, has important consequences on the way their are designed. The increasing demand in term of functionality have a direct impact on the number and type of processing blocks in these devices. For instance, modern TVs are connected to the Internet and have to be able to decode several types of streams, smartphones provide various services ranging from calling to cameras and music/video players, and generally home appliances are providing diverse services varying from Internet access to home automation facilities. Moreover, because of mobility requirements, battery supplied devices are becoming the rule. Flexibility, and autonomy are thus essential. However, ensuring such contradictory goals is a real issue, since the former requirement entail using programmable components, which are known to be more requiring in term of resources, e.g. battery, and tougher to perform efficiently.

Embedded systems design is expected to be increasingly harder as they evolve towards more sophisticated settings with tighter connection between physical and cyber worlds. This requires to abandon ad hoc methods and to consider more disciplined, and holistic approaches taking into account both functional and extra-functional requirements. System-level design provides pertinent guidelines for such systematic approach.

1.2.2 System-level Design

System-level design [127] has emerged to provide a structured way to answer the growing embedded systems design challenges. In this setting, the design problem is organized as a sequence of phases, each involving several design alternatives. Realizing certain functionality can be actually performed in various manners soliciting different resources. Attaining a valid design requires to choose, at each step of the process, alternatives that match initial

requirements, among available choices. The amount of design possibilities is called the *design space*. This potentially has an infinite size, albeit only a finite set of alternatives is conform to initial specifications. For instance, the choice between wired and programmable logic is mainly related to the amount of flexibility required in the system (is the system going to evolve in future ?) and the complexity of the functionality to design (does it involve real-time constraints ? If any, what is the appropriate scheduling policy ?). Similarly, the choice of an appropriate communication infrastructure depends on the required functionality (point-to-point communication, etc.), but also on the amount of latency and overhead induced by this choice.

The central idea in system-level design is *high-level reasoning*, which tends to simplify the design problem and enhance the comprehension of designers. It is common to accomplish this principal following a *model-based approach*. Models allow for building simple representations of sophisticated concepts and thus help mastering complexity. They allow to explore the impacts of design alternatives before actually realizing the system, hence prevent important cost and reduce time. Given certain specifications, they allow for capturing different levels of details. Models encompassing fewer details are said to be *abstract or high-level*. Given an abstract model, introducing additional details is denoted *refinement*. Due to small amount of details, high-level models are obtained with less effort and are well-suited for fast analysis. Abstraction is clearly essential for high-level modeling. However, great difficulties accomplishing it remain unsolved, such as identifying the appropriate level of abstraction for trustworthy analysis.

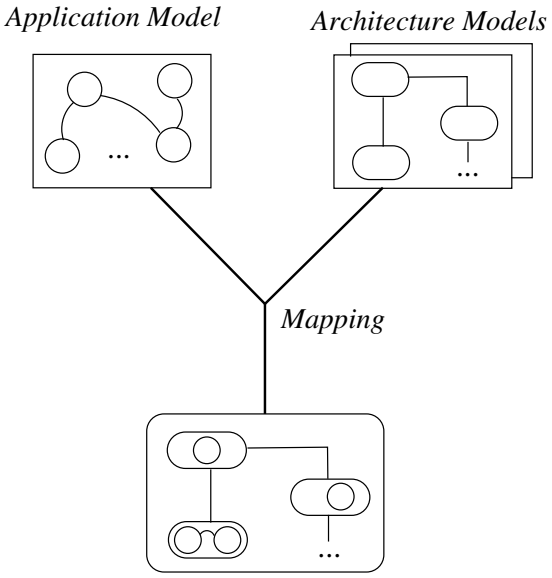


Figure 1.1: The Y-chart Scheme for mapping application to architecture.

When building models, *separation of concerns* is crucial as stipulated in system-level design. This involves *i*) the separation of computation and communication and *ii*) the separation of application from the architecture. The latter is particularly useful for the design of programmable systems, where it is required to distinguish the software application from the hardware architecture. Different strategies can be then investigated to map application functionality to architectures components, which is of great importance in the context of multi-core or many-cores architectures. This is often described as the Y-chart [128] method depicted in Figure 1.1. The second separation principal has been further developed and extended in the platform-based design approach [184]. This advocates using a parameterizable platform for a class of applications per domain to enhance components reuse and thus production and reduce cost.

At each step of the design process, making a choice implicitly implies to eliminate alternatives and thus to reduce the design space as illustrated in Figure 1.2. The highest importance in system-level design method is given to the earliest design steps. During these phases, the number of alternatives is the most important, thus decisions have the greatest impact on the rest of the process. For instance, bad decisions at this level, may lead to unfeasible design or to a design where some requirements cannot be met. Systematic *Design space exploration* methods are indispensable to investigate the design alternatives at different phases of the process. Such methods are required to be fast and accurate to enable well-founded decision at early stages.

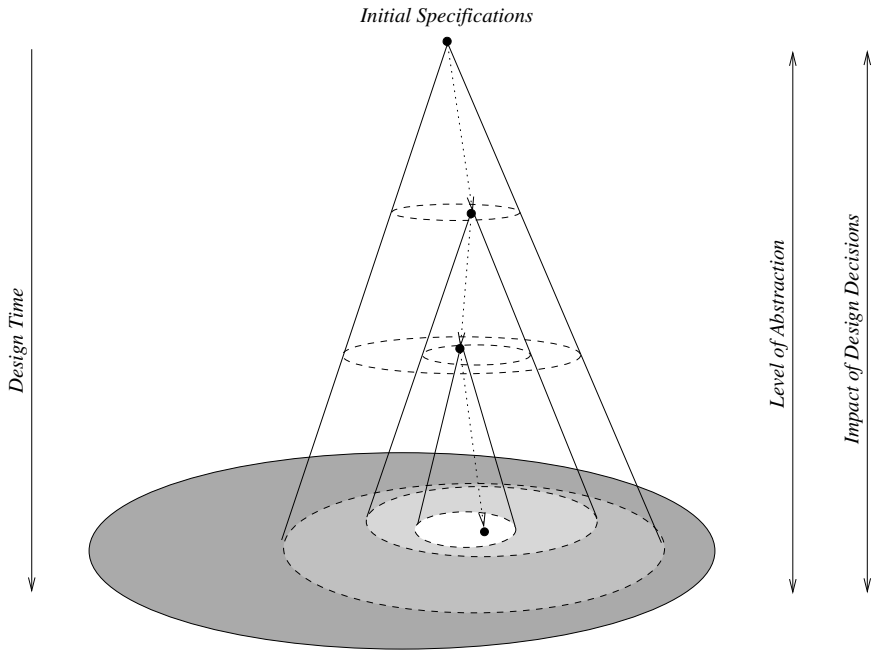


Figure 1.2: Schematic view of exploring the design space [197].

A generic scheme of design space exploration is presented in figure 1.3. It consists of an iterative process repeated for each design phase and potentially leading to next design phases. For the earliest phases, given initial specifications, high-level models capturing the main functionality in an abstract way are first built. These models represent different realizations alternatives of functionality of interest. Following the Y-chart pattern, separate models of application and architecture are produced. Analysis is then performed on each alternative to check conformance with the given requirements. The obtained analysis results are essential to decide which alternative is the most appropriate with respect to functional and extra-functional aspects.

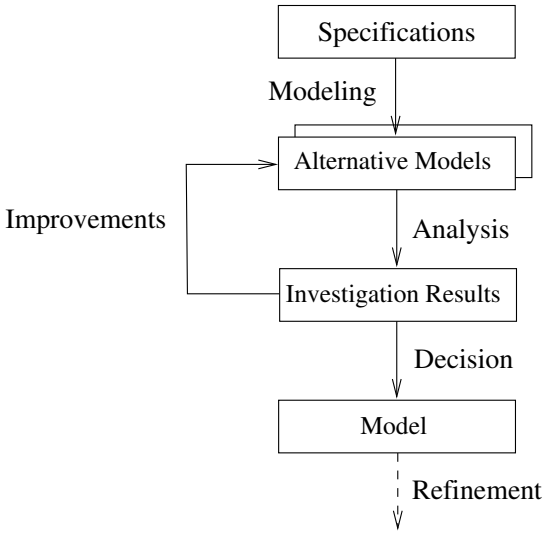


Figure 1.3: A generic process of design space exploration.

1.2.3 State of the Art

The generic process of design space exploration is widely adopted in several state-of-the-art tools following system-level design guidelines. For many decades, embedded systems designers were operating at very low-levels of abstraction, e.g., transistor-level, gate-level, or register transfer-level (RTL). Due to the increasing complexity and the exponential growth of functionality, rising the level of abstraction has become inevitable. Methods operating at higher-levels such as transaction-level modeling (TLM) [53] or system-level have thus emerged.

The shift of embedded systems towards more programmable and flexible settings has obliged designers to abandon traditional methodologies pushing initial specifications, through a sequence of transformations, to a dedicated wired logic implementation. It is worth to mention that this might still the case for the design of sub-components of systems.

Designing heterogeneous systems encompassing programmable and dedicated components generally falls within one of three configurations:

1. The first setting concerns architecture exploration, which is for instance useful to find the most appropriate architecture for a given domain of application, e.g., multimedia or communication.
2. Another possible configuration is when the architecture is known and the goal is to design and map software application to this architecture.
3. The third possible setting is the classical co-design situation where some functionality specification are given and the aim is to find the best partition into programmable and dedicated components.

These settings may coexist withing a single design process, e.g., at different phases of the process, or in independent processes for different application domains.

When confronted with these settings, different design approaches are possible, mainly in term of potential abstraction choices. For instance, the first setting requires to consider various applications in a specific domain that will constitute a benchmark for architectures exploration. To this end, abstracting functional details of applications is essential to ensure a fast exploration and a maximum of coverage. This is for example the case for the SPADE (System-level Performance Analysis and Design space Exploration) methodology [150], which relies on trace-driven simulation for architecture exploration. Trace-driven simulation consists of transforming models of applications into traces containing coarse-grained computation and communication operations. Similarly, the Artemis Workbench [168] offer the same functionality, although not limited to.

Partitioning initial specifications into programmable and dedicated components is among the most difficult and time consuming design activities because of the huge number of design alternatives. Only few methodologies providing assistance towards this goal exist. This is for instance the case of the POLIS [17] and the VCC methodologies, which start from high-level models that do not discriminate software and hardware components. An explicit partitioning process is performed to identify candidate components for software implementation. The methodologies were actually proposed in the context of automotive industry, often relying on single processor architectures. The increasing difficulty to partition functionality into hardware and software components has motivated the emergence of platform-based design approach [184] where a common denominator architecture is used for a given application domain. Such an architecture may be found using architecture exploration methodologies mentioned above.

The current state of embedded systems design mostly consider potential target architectures in addition to initial specifications, which correspond to the second setting discussed earlier. This may be following a platform-based approach, using human expertise, previous knowledge, or experience.

The considered target architecture may be completely specified or allow parameterization, e.g., cache size. In such configuration, embedded software design become very important. Moreover, with the emergence of multi-core and many-cores architectures, additional challenges such as finding the best parallelization and the optimal mapping of the application into architecture, figuring out the best communication mechanism to use, and finding software components parameters, e.g., queues size, are real concerns.

Most of the state of the art methodologies provide automated support to find one or several candidate mappings, generally taking into account functional and performance aspects. In this context, abstraction is more targeted to the architecture part of the system, although different possibilities are proposed in the literature. Several methodologies rely on libraries of generic hardware components at different levels of abstractions. For instance, the Artemis Workbench [168], DOL (Distributed Operation Layer) [198], and VISTA [159] use SystemC to model hardware components at different abstraction levels. Vista provides cycle-accurate TLM components to build virtual SoC architecture, whereas DOL offers high-level components. Artemis is based on the Sesame environment [66, 169] for architecture and application modeling. It provides hardware components capturing only performance information and not functional behavior. Other methodologies such as Metropolis [18] and MILAN [16] are based on meta-models.

Performance aspects are essential to make well-founded decisions in the different discussed settings. Integration of these information within high-level models is quite important for trustworthy design space exploration and consequently to a successful design.

Besides the important work in industry and academia proposing various design methods, several technical and theoretical problems remain unsolved. For instance, a big challenge in this context is building appropriate abstractions of application and architecture and their combination. Furthermore, performance aspects are still not very well understood, especially at system-level. As depicted in the next section, important challenges remains ahead for modeling and analyzing them faithfully at system-level.

1.3 Performance in System-level Design

For many years, functional aspects were in the center of system-level design methods, while extra-functional ones were considered as second-class citizens. Traditionally, ad hoc performance models decorrelated from functional behavior, built late in the design process and coupled with simple analysis techniques were the unique performance evaluation support.

As stated earlier, in modern systems such as wearable and portable devices, where a limited amount of resources is available and an increasing number of functionality is required, extra-functional aspects are becoming

equally important and not considering them may lead to poor results later in the design process. For instance, a smartphone that quickly losses energy or a multimedia device with a considerable response time (latency) is not going to sell even if it provides all required services. An equally important issue is to find trade-offs between different extra-functional requirements. For example, selecting a security strategy, involving cryptography, of some distributed setting which is monitored through portable devices must take into account autonomy issues.

Additional requirements concerning the models and the analysis techniques to use for system-level design of embedded systems are therefore needed. First, building high-level models that only capture the functional behavior of the system is no more sufficient. Taking into account extra-functional aspects, especially performance is a must for a successful design, which rises several questions about traditional models and their convenience. Second, given the importance of such aspects for well-founded decisions during design space exploration, classical performance evaluation techniques are no more sufficient.

While dealing with functional specifications has reached relatively mature state (a large body of theoretical and practical results), we still lack methods and tools to rigorously and systematically handling extra-functional requirements. These remain not well understood and still difficult to handle, especially at the earliest design phases. Dealing with these aspects at system-level brings additional difficulties which span from the specification phase to the analysis of the global system.

Our interest is to contribute advancing the state-of-the-art on system-level design. In particular, we aim at conceiving techniques and methods for better support of performance requirements following the system-level guidelines above. We approach this problem by first analyzing the current and sought challenges of dealing with performance at system-level. These challenges will guide us to identify the main requirements for building or using the appropriate methods and techniques to achieve our objective. Moreover, based on the previous analysis, we pinpoint modeling and analysis as the two main activities of a design process. Thus, in the next section, we will decompose the identification of requirements into identification of modeling requirements and identification of analysis requirements.

1.3.1 Challenges

The System Dimension. Because of their weak interaction with the physical world, classical computer systems are essentially designed based on strong abstractions of physical artifacts. For instance, building software applications to run on a computer, entails designing an algorithm for which physical time and memory complexities are evaluated, using complexity theory, with respect to abstract execution models, e.g., Turing machine. Such

abstractions are inappropriate for embedded systems because of the tight connection of the physical and cyber worlds. Performance of embedded systems is strongly related to the physical part of the system, e.g., execution of some functionality by specific architecture components; execution time of a Fourier transform on a processing unit, communication delay of a bus or a Network on Chip (NoC), amount of consumed energy or dissipated temperature induced by that function on the corresponding hardware. This shift from a program view to a system perspective gave rise to important theoretical and technical issues.

Contradictory Goals, Abstract Vs. Faithful. The modeling activity at system-level aims at enhancing designers comprehension and reducing effort by performing high-level reasoning. Thus, on one hand, one wants to deal with abstract models that minimize the modeling effort and the exploration time. On the other hand, these models are required to capture performance details in order to precisely reflect the reality and enable accurate reasoning about the whole system performance. This highlights the need of building good abstractions.

Availability in Early Design Phases. Because of the tight connection between cyber and physical worlds, performance aspects of embedded systems cannot be thought of independently of the hardware part of the system. However, the latter are rarely available in early design phases since physical realization of the hardware architecture come relatively late in the design process. This makes building high-level models encompassing faithful performance details quite challenging. Hence, only approximation or estimation of such details at different level of details is possible at system-level.

Variability. Performance details are characterized by their significant variability which obviously cannot be captured by point estimates. This fluctuation is mainly due to three reasons. First, the inputs (or the workload) are generally variable, albeit some systems are data-independent, that is, their performance is constant for various input data. Second, the impact of the environment, which is often unpredictable, is important. This underlines robustness requirements of embedded systems. The third reason is the inherent hardware components behavior, e.g., caches, memory contention, arbitration mechanisms, etc. These cannot be captured in details in early design phases because of the lack of detailed specification and the required high-level of abstraction.

1.3.2 Performance Modeling Requirements

The above challenges rise several natural questions with respect to the modeling process at system-level. The main question one have to answer is

How to build high-level models encompassing performance details in addition to functional behavior while ensuring faithfulness ? This question can be decomposed to several sub-questions that concern, on one hand the models to use: *i)* What is an appropriate characterization of performance details? *ii)* What is a well-suited model for capturing functional and performance aspects at system-level? On the other hand, question concerning the process to follow: *iii)* How to capture performance information in early design phases?

Let us first consider the first two questions, that is, *i)* and *ii)*. Generally two types of modeling approaches, providing orthogonal abstractions, exist, namely computational (machine-based, e.g., automata) and analytical (equation-based, e.g., transfer function) [109]. Because their are executable, computational models are often used to describe software systems, while analytical ones are more appropriate for hardware, since they capture more naturally concurrency and quantitative aspects. Whereas analytical models are inherently mathematically defined, computational models are not necessarily formal. Most of the time, these are obtained using modeling languages (textual or graphical), where the dynamic behavior is not formally specified, e.g., Java threads or Unified Modeling Languages (UML) [182]. Formal computational models in contrast, are the result of well-defined modeling languages, that is, with mathematically specified operational semantics, e.g., Petri nets [179].

In this thesis, a mathematically defined framework defining the semantics of a given modeling language is denoted as a formalism. Furthermore, we distinguish *modeling languages* used to catch systems functionality from those used to specify system requirements, e.g., good properties to satisfy or bad properties to avoid, denoted *property specification languages*. Back to the previous questions, based on the identified challenges, we impose the following constraints on modeling (respectively property specification) languages and system models (respectively system properties) to capture both functional and performance aspects at system-level.

Requirements on Modeling and Property Specification Languages

To deal with the above challenges, it is required to dispose of languages that are sufficiently expressive to cover the different aspects of software, hardware, and the interactions between them. These should also have clear semantics to enable rigorous reasoning and interoperability, in addition to providing support for quantitative aspects.

Expressiveness is the ability of a languages to provide intuitive, readable, and succinct mechanisms towards representing the different aspects of a system. To capture hardware and software aspects, a modeling language should provide means to express concurrent as well as sequential behaviors, continuous as well as discrete time, functional and extra-functional aspects. Another important ingredient concerns communications mechanisms which

maybe for instance synchronous or asynchronous. Modeling languages should also enable capturing uncertainties and variability through *stochastic* and/or *non-deterministic* support.

A natural way for building artifacts is incrementally and in a compositional fashion. The system is seen as a set of actors or components having certain behavior and properties which are assembled in a rigorous manner to achieve the global required behavior of the system. This entails *component-based modeling languages*.

Requirements on Models and Properties

Due to their central role in systems design, models must have clear interpretations which leaves no place for ambiguity. Models are hence required to be formal, that is, governed by clear semantics free of ambiguity, which is achieved by adopting a certain formalism. Furthermore, a model in system-level design is required to capture only the gist of the system behavior required in the corresponding phase, that is, with respect to taking decision in that stage or level. In contrast, models should also be faithful in that they reflect the real specifications of the system. Furthermore, models are required to be executable to enable fast prototyping and analysis.

Requirements on Performance Modeling Methods

The remaining question from the modeling perspective (*iii*) concerns the approach to use in order to capture performance details at system-level. First, it is worth mentioning that since complete physical details cannot be obtained in early design phases, such approach is expected to perform *estimation* from concrete or low-level hardware descriptions. Second, the used approach should be able to precisely capture performance variability while still abstract (not too verbose characterization). Finally, such method should be automatic (tool-supported) and fast to avoid creating bottlenecks that will eventually slow down the design space exploration. An important requirement is that the method keeps separated purely functional models from models containing in addition performance information (called *performance models* in this work) as shwon in Figure 1.4. This is important in order to ensure incremental design and separate analysis at different levels of abstractions.

State of the Art

Modeling and Property Specification Languages. Several representations were proposed in the literature to model application and architecture. The most common for architectures description are hardware description languages (HDLs). These enable modeling architecture behavior at different levels of abstractions. For example, VHDL and Verilog are often used for

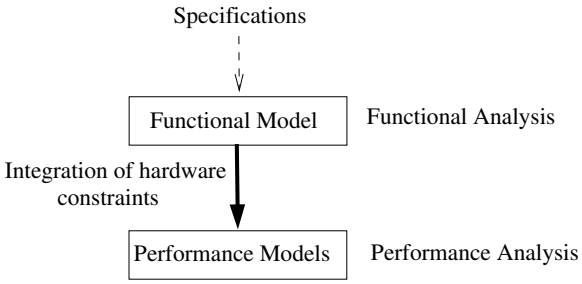


Figure 1.4: Building separate purely functional and performance models.

RTL descriptions and enable direct synthesis on ASICs or FPGAs, whereas SystemC [2] covers several abstraction levels (RTL, TLM, System-level). In addition to architecture, since based on C/C++, SystemC allows for modeling application software (with various communication mechanisms) as well. Architecture description languages (ADLs) allow for modeling systems architecture and application in higher levels of abstraction than HDLs. A well known example is AADL [88, 87].

In the literature, model of computations (MOCs) [58] are often used for modeling software/hardware systems and cover application as well as architecture. An important number of possibilities is available, Communicating Sequential Processes (CSP), Discrete Time, Continuous Time, Discrete Events, or Finite State Machine (FSM). The latter is for instance used as underlying model in the POLIS framework [17]. Other previously mentioned methodologies use Data Flow models such as synchronous Data flow (SDF) or Kahn Process Network (KPN) [95], which is the case of DOL [198], Artemis Workbench [168], and SPADE [150]. More flexible modeling framework relying on meta-models are also proposed. In this case, no restriction is imposed on the used model of computation but only a general semantics is provided to support several choices. This for instance the case for Metropolis [18] and MILAN [16] methodologies.

Property specification languages are only used in a formal analysis process as discussed later. Generally systems properties are specified as predicate formula often using temporal logic such as CTL [176, 62], LTL [171], or others [140, 149]. A famous example of such languages that has been standardized by IEEE in 2005 is PSL [84].

Methods for Capturing and Integrating Performance. State of art techniques for gathering low-level performance information in early design phases can be classified as follow. The most direct ways is to use documentations, e.g constructor data sheets [104]. These may be helpful to build models of certain components but more complicated to capture abstract per-

2. http://www.accellera.org/downloads/standards/systemc

formance information, especially their variation. Performance details may also be obtained from source/binary/object code, e.g static analysis or code inspection [45, 23]. This needs a (cross-)compiler and code profiling, although does not capture the dynamic behavior of the system. The most accurate and the most used technique relies on executable i.e high/low-level simulation or execution, albeit it is known to be time and resource consuming [17, 150, 158, 169].

Several frameworks for system-level design uses these techniques together with model calibration to improve the accuracy of high-level models. Model calibration, also referred to as back-annotation, is a well-known and widely-used technique which consists of adding specific details to a given model. However, it only received a little attention in the system-level design community and few frameworks consider and implement it in different ways.

In [170], the authors use low-level simulation and synthesis to calibrate architecture models in the context of the Sesame simulation framework [169]. The proposed techniques rely on instruction-set simulator (ISS) to calibrate programmable components and on automatic synthesis targeting FPGAs for dedicated ones. The goal is to build latency tables associated with architecture components models.

A system-level performance estimation method [158] for the MILAN framework [16] is proposed to calibrate parametrizable models of virtual SoC architecture using interpretive simulation techniques. Starting from initial performance values given by the designer, isolated simulation of specific application components mapped to specific architecture ones is performed. The obtained measures concern energy and latency and are characterized as average estimates. Later on, based on task graph, composite estimate are derived for energy and latency as well.

In the context of the BIP design flow [44], calibration is performed on two phases. First, hardware constraints concerning computation and communication delays are integrated, through refinement, at the level of the system model, that is the result of mapping application models into architecture. Application component are also calibrated with low-level performance details using tow techniques, namely Instruction Weight Tables (static) and Platform Dependent Code Generation (dynamic).

In [104], an approach is proposed for automatic code generation and calibration of compositional performance models. It relies on high and low-level simulations in addition to data sheets to obtain components performance parameters which are mainly used to characterize arrival and service curves in term of best-case and worst case execution time (WCET) since relying on Real Time Calculus [199].

An improvement of the VCC methodology using back-annotation of high-level behavioral models is proposed in [96]. Performance estimation is performed using statistical approach but only consider a single microprocessor. More specifically, the approach is based on linear regression analysis tech-

niques. Similarly, Scope [3] uses statistical regression analysis to perform power estimation of co-processors.

It is worth to observe that different performance estimation and calibration approaches provides completely different abstractions. Besides the level of detail of the initial models, the characterization approach of performance details (the type of the obtained characterization), the choice to calibrate either application or architecture models or both, and the decision to consider one or both models during the mapping step potentially produce different system model which might be appropriate for different design phases.

1.3.3 Performance Evaluation Requirements

Analysis of models encompassing functional and extra-functional, software and hardware aspects is challenging in different senses. Assuming that such models are already built, their size is expected to be important. In spite of the software and hardware aspects of real-life systems, formal models are often heavier to explore. Thus scalability is a real issue, especially when using rigorous analysis techniques. Analysis should not be intrusive to not alter the correct behavior of the model under analysis. The level of required abstraction for system-level analysis should be carefully chosen to not mask the relevant details for analysis. It is important that the used analysis technique provide quantitative results of performance. Moreover it should report accurate results, that is, approximation not too far from reality. Another important requirement about performance evaluation techniques is that they have to be fast to enable several analysis iterations in order to cover a maximum of design alternatives.

State of the Art

State-of-the-art performance analysis techniques at system-level can be broadly divided in two general families, pure simulation-based and formal approaches. Nonetheless, some works propose to combine techniques from both groups towards hybrid approaches.

Pure simulation-based techniques enable to simulate system components, e.g., independent hardware and software, or to co-simulate both parts at different abstraction levels: cycle-accurate using instruction-set simulator (ISS) for instance or higher levels such as functional simulations. Industrial tools often rely on simulation techniques for performance analysis, for instance, Seamless Hw/Sw Co-verification [3] or Synopsis System Studio [4]. Academic methodologies mostly base their performance analysis on simulation as well.

3. http://www.mentor.com/products/fv/seamless/

4. http://www.synopsys.com/Systems/BlockDesign/DigitalSignalProcessing/Pages/SystemStudio.aspx

This is for example the case of Artemis Workbench [168], SPADE [150], MILAN [16], and Metropolis [18]. Simulation-based approaches mainly suffer from long run-time and coverage issues. Some of the above methodologies propose trace-driven techniques to remedy to long simulation, which use co-simulation to build abstract event traces that are later used for lighter analysis such as in the Artemis Workbench and SPADE.

To deal with the increasing complexity and the critical aspects of modern embedded systems, it is required to rigorously verify that the built system matches the given specifications and requirements. Traditional methods consist mainly of testing and peer reviewing of code, which are widely used in industry [5] and are able to capture a wide class of errors. However, they suffer from several disadvantages such as manual or semi-automated procedures, coverage issues, and more importantly, they can only detect presence of errors and not their absence. Formal methods provide an automated alternative for system verification. Moreover, they offer mathematical guarantees for the absence (respectively presence) of a bad behavior (respectively good behavior). Formal methods for performance analysis can be classified into tow main categories with respect to the used formal models, namely based on computational models or on analytical ones.

Analytical-based. They mainly rely on the Real Time Calculus (RTC) method [199], which respectively characterizes workload and processing power as arrival and service curves. Arrival curves for example, capture upper and lower bounds of arrival time of a class of events. RTC was adopted and extended in several researches such as SymTA/S [107], MAST [100], and modular performance analysis (MPA) [55]. The latter is used within the DOL methodology [198] for system-level performance analysis. These methods are actually conceived to be constrained and often used to compute worst case scenarios.

Computational-based. They are primarily based on model checking (MC) [176, 62], which is a verification method that considers a formal model $\mathcal{M}$ of a system and formal representation of a requirement ϕ, and answers the question if $\mathcal{M}$ satisfies ϕ. As illustrated in Figure 1.5, it provides a yes or no answer and, when the property is not satisfied, it gives a counter-example as a witness. The model checking techniques exploits the formal definition of the inputs, i.e., $\mathcal{M}$ and ϕ, to perform systematic and exhaustive analysis. Given $\mathcal{M}$ and ϕ formalism, every state of $\mathcal{M}$ might be verified to satisfy or not ϕ. Exhaustive analysis constitutes the main limitation of MC techniques since it implies an exponential exploration, known as state-space explosion. Despite their advantages, MC methods are still difficult to achieve and hence not yet widely adopted, especially, for performance analysis. MetaMoc [69] is

5. For instance, between 30% and 50% of the total cost of software development is allocated to testing [15].

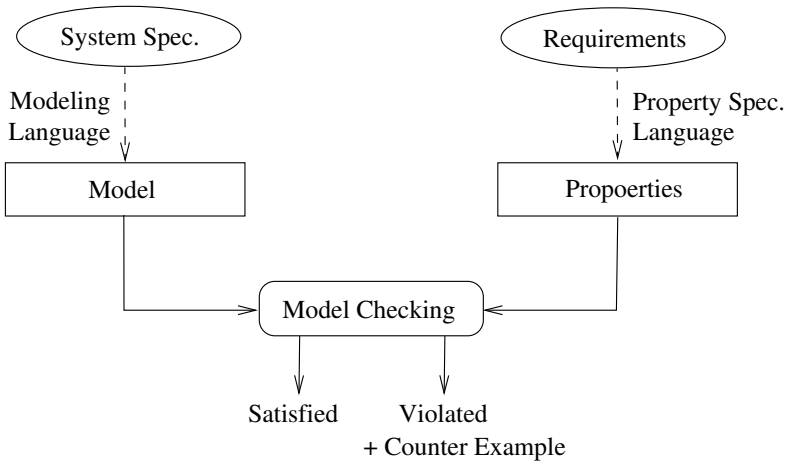


Figure 1.5: Illustration of the model checking verification method.

one among the tools that use MC for the analysis of embedded software. It is based on the UPPAAL model checker [30] and mainly targets Worst Case Execution Time (WCET) and schedulability analysis of hard real-time systems.

Combining simulation-based techniques and formal approaches could be a good strategy to bypass weaknesses of each method and strengthen their advantages. This has been actually suggested by some works in the literature such as in [135] where RTC [199] is combined with simulation to perform analysis with respect to the MPARM virtual platform [152], and in [40] where a hybrid method combining simulation with analytical models is proposed.

1.4 Contributions and Organization

Following the above analysis, we provide hereafter an overview of our answers, which consist of various contributions covering the different identified requirements. The presentation below is orthogonal to the global organization of the manuscript. We found it more pertinent to decompose the manuscript into two parts and to separate more generic contributions from more specific ones. In the first part, we present a general framework for quantitative analysis, formal component-based modeling, and automatic abstraction of stochastic systems, which may be used in different contexts. In the second part, we use this framework to conceive a rigorous and tool-supported method for system-level performance modeling and analysis in the context of embedded systems design.

The following overview follows the decomposition we made during the analysis phase above. It first presents our contributions in term of modeling of performance aspects at system-level, then depicts our contributions in

performance evaluation. Finally, it illustrates our global answer to the identified requirements, which is a rigorous, systematic and integrated method for performance modeling and analysis for systems-level design of embedded systems, called *ASTROLABE*. We opted for such a double decomposition to match our answers with the above identified requirements and to facilitate reading the manuscript.

1.4.1 System-level Modeling

Stochastic Component-based Formalism

We propose a component-based modeling language to handle system complexity. This is based on the BIP framework [28], which enables incremental system design starting from simple components which are later assembled together to build more complex functionality. This approach enables components reuse, hence reduces modeling time and enhance productivity. BIP has a well defined semantics for components modeling (behavior and interfaces) and for specifying coordination between them. It offers different communication mechanisms that are shown to be sufficiently expressive and enables user-defined scheduling policies. Moreover, BIP supports different timing models in addition to well-defined real-time semantics.

Our main contribution at this level, is the extension of this framework to support stochastic systems modeling. We mainly provide a syntactic extension of the language to enable modeling probabilistic behavior and a formal specification of the operational semantics of stochastic behavior. As we will expose in Chapter 3, the proposed stochastic component-based formalism denoted $\mathcal{S}$BIP, is shown to be well suited to model functional as well as performance aspects. The latter are captured as probability distributions or more sophisticated probabilistic models. The semantics of $\mathcal{S}$BIP, is sufficiently expressive to be used as single semantics driving the different design phases and to separately capture software and hardware models.

Code Generation

To enable gathering accurate performance information in early design phases, we rely on automatic code generation. Starting from application functional models, we generate concrete implementations targeting low-level models of architecture: virtual prototype, or physical implementation (FPGA or final chip), depending on the design phase. This enable accurate (since based on concrete execution) yet fast (automatic generation of implementation and deployment code) prototyping as shown in Chapter 5. This entails instrumentation of performance dimensions (time, energy, temperature, memory) of interest (obtained from systems requirements) and pertinent functionality to measure. Automatic code generation produces parallel

implementations for many-cores platforms, which answer the increasing complexity of programming these architecture challenges. An implementation of a code generator targeting the STHORM platform [155] by STMicroelectronics is presented in Chapter 7.

Statistical Characterization of Performance

To faithfully characterize performance details, we propose to use automatic learning techniques from concrete executions. This enables to capture the real performance characteristics of the application running on specific architecture with respect to some portioning. Moreover, we suggest learning probabilistic models of performance, e.g., probability distributions, to correctly catch variability. We believe that such models provide good abstraction of physical details without losing the gist. In Chapter 6, we detail how we use statistical inference algorithms to build such probabilistic performance characterizations. In Chapter 7, we provide tool support for the statistical inference procedure.

Model Calibration

The method we propose to build faithful performance models is to calibrate functional models (application and architecture), which are timeless and does not contain any information about energy consumption or temperature for instance, with probabilistic characterizations of performance. This back-annotation mechanism will produce stochastic models encompassing the functional behavior of the system in addition to the performance aspects at a good level of abstraction, that is appropriate for the earliest exploration phases as we will detail in Chapter 5.

1.4.2 System-level Verification

Performance Analysis

Our contribution at this level is to use a formal verification technique, namely statistical model checking (SMC) [110, 209], introduced in Chapter 2. Our proposal is a trade-off between purely simulation-based methods and analytical techniques, which consists of stochastic (Monte-Carlo) simulation and statistical tests. It combines benefits of both approaches, that is, the speed of simulation and the well-founded of analytical approaches. SMC only provides approximations which can be controlled by using user-defined level of confidence. Moreover, it allows for quantitative results, which are more appropriate for performance evaluation. To the best of our knowledge, this is the first time SMC is being used for performance evaluation of embedded systems. A second contribution in this context is the implementation of the BIP$^{\mathcal{SMC}}$ statistical model checker presented in Chapter 7.

Automatic Stochastic Abstraction

To answer the issues induced by analysis of software/hardware models (the important size of the model) and the challenges of performing specifically formal verification (state space explosion, important time), we propose in Chapter 4, a technique for automatically building models abstraction. Our idea is to perform abstraction with respect the property we want to verify on the model. The proposed technique is based on machine learning algorithms and enables to learn an abstraction from execution traces even in case of black-box models.

1.4.3 Integrated Performance Modeling and Analysis

The above contributions are used in a tool-supported method for performance modeling and analysis at system-level. The proposed approach is called *ASTROLABE* and is depicted in Chapter 5, the associated tool-flow is presented in Chapter 7. It provides a systematic and rigorous way to build faithful performance models since relying on the $\mathcal{S}$BIP formalism as a single semantics and by using code generation and statistical inference as shown in Figure 1.6. The second asset of the method is that it enables fast and accurate performance analysis using the statistical model checking and the stochastic abstraction techniques. In Chapter 8, the *ASTROLABE* approach and its associated tool-flow is used to design a real-life case-study consisting of the HMAX models algorithm [160] for image recognition deployed on the STHORM many-cores platform [155].

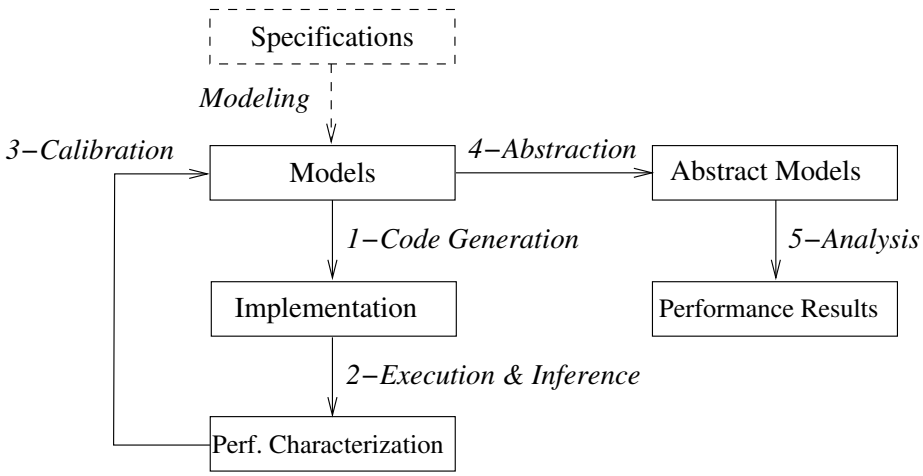


Figure 1.6: Overview of the *ASTROLABE* method for system-level performance modeling and analysis.

PART I

FORMALISMS AND TECHNIQUES

Chapter 2

Quantitative Analysis of Stochastic Models: A Background

In this first part of the thesis, we aim at providing a theoretical foundation upon which we will build our method for system-level performance modeling and analysis for many-cores embedded systems. We propose a set of formalisms and techniques for stochastic modeling and performance analysis at a high-level of abstraction. The part is composed of three chapters, where the first recalls general formalisms for stochastic systems modeling and probabilistic requirements specification. It also presents quantitative techniques for analyzing stochastic systems. In the second chapter, we introduce a component-based formalism for modeling stochastic systems and discuss its semantics and expressiveness. The third and last chapter of this first part is about abstraction of stochastic models. There, we propose a technique based on machine learning to automatically build abstract models in order to improve scalability and reduce analysis time.

Modeling performance requires rich formalisms that allow for capturing in the same time sophisticated functional behavior and complex performance information. In the context of embedded systems, modeling formalisms are even more important because of the inherent nature of these systems which evolve in unpredictable environments, are subject to unexpected situations, hence encompassing a high degree of uncertainty. Stochastic or probabilistic formalisms are thus needed to correctly and faithfully capture these behaviors. On the other hand, analyzing these models rigorously and efficiently is a challenging task due to the increasing complexity of modern systems. Quantitative analysis is even more difficult and less understood than classical qualitative analysis techniques. While several techniques exist, we still encounter difficulties performing such analysis especially at system-level.

This chapter recalls the main concepts of quantitative analysis of stochastic models following the model checking approach as stated in the introduc-

tion. It first recalls general stochastic modeling formalisms, namely Markov Chains and Markov Decision Processes. It then presents the Linear-time Temporal Logic and its probabilistic bounded variant as a mean for formalizing systems requirements. Finally, it introduces the Statistical Model Checking techniques for analysis of stochastic system models.

Throughout this dissertation, we will adopt a state-based view of stochastic processes unless differently stated. Classically, these are seen as sequences of random variables evolving over time. Moreover, we will only consider finite and discrete Markov models, i.e., Discrete Time Markov Chains (DTMCs) and Markov Decision Processes (MDPs) and do not discuss continuous time models such as Continuous Time Markov Chains (CTMCs). It is worth recalling that the state-based representation of stochastic processes entails two type of labeling, over states and over actions, as we will show allover the chapter. As a consequence, the notion of non-determinism may have different meanings accordingly, i.e., we may have non-determinism with respect to state labels or with respect to action labels. In this work, we are defining DTMCs to be only state labeled. Therefore, they are only concerned with non-deterministic state labels. In contrast, MDPs are assumed to have labels on both states and actions. Nonetheless, we will only consider non-deterministic actions as we will explain hereafter.

2.1 Stochastic Systems Modeling

We begin by giving a general background on stochastic models. We focus on a specific class of models called Markov Models. These have the particularity to be memoryless, that is, at each time, the decision to move to a next state only depends on the current state and does not take into account the whole history of the system evolution. We provide hereafter an overview of two well-known models, namely Discrete Time Markov Chains and more general ones called Markov Decision Processes which encompass non-determinism in addition to probabilistic behavior.

Let AP be a finite set of atomic propositions. We define the alphabet $\Sigma = 2^{AP}$ and denote the elements of Σ (all subsets of AP) as *symbols*. The empty symbol is denoted by τ. As usual, we denote by Σ^ω (respectively by Σ^*) the sets of infinite (respectively finite) words over Σ.

2.1.1 Discrete Time Markov Chains

Along the dissertation, we will use state-labeled Markov Chain, abbreviated to LMC to refer to Discrete Time Markov Chain (DTMC). The reader may find both notations.

Definition 2.1 (state-Labeled Markov Chain)**.** A state-Labeled Markov Chain $\mathcal{M}$ is a tuple $\langle S, \iota, \pi, \Sigma, L\rangle$ where,

- S is a finite and nonempty set of states,
- $\iota : S \to [0,1]$ is the *initial states distribution*, such that $\sum_{s \in S} \iota(s) = 1$,
- $\pi : S \times S \to [0,1]$ is the *transition probability function*, such that for all states $s \in S$, $\sum_{s' \in S} \pi(s,s') = 1$,
- Σ is an alphabet, and
- $L : S \to \Sigma$ is a state labeling function.

Note that this defines finite LMCs where S and AP are finite sets. The initial distribution $\iota(s)$ specifies the initial states of the system, i.e, where it starts evolving. The *transition probability function* π specifies, for each state $s \in S$, the probability $\pi(s,s')$ to move to a state $s' \in S$ by a single transition. For more convenience, we denote a transition from a state s to a state s' as $s \longrightarrow s'$. The *transition probability function* π is required to be a valid probability distribution, i.e, $\sum_{s' \in S} \pi(s,s') = 1$. In the discrete case, π may be identified by a matrix as shown in Example 2.1. Finally, the labeling function L assigns to each state a set of atomic propositions that are true in that state.

Given an LMC $\mathcal{M}$, we define $Post_{\mathcal{M}}(s) = \{s' \in S \mid \pi(s,s') > 0\}$ the set of immediate successors of a state s and $Pre_{\mathcal{M}}(s) = \{s' \in S \mid \pi(s',s) > 0\}$ the set of immediate predecessors of s. We also denote as $Det_{\mathcal{M}}(S)$ the set of states that have a single deterministic transition (a transition with probability 1), that is, $Det_{\mathcal{M}}(S) = \{s \in S \mid \exists s' \in Post_{\mathcal{M}}(s), \pi(s,s') = 1\}$.

Definition 2.2 (Deterministic LMCs)**.** A LMC $\mathcal{M} = \langle S, \iota, \pi, \Sigma, L \rangle$ is *deterministic* (DLMC) if and only if:

1. $\exists s_0 \in S$, such that $\iota(s_0) = 1$, and
2. $\forall s \in S$, $\forall \sigma \in \Sigma$ there exists at most one $s' \in S$, such that $\pi(s,s') > 0$ and $L(s') = \sigma$.

Given an LMC $\mathcal{M}$, a *path* is a possible behavior (infinite execution) of $\mathcal{M}$. A *trace* is the sequence of labels associated to the states of a given path.

Definition 2.3 (Paths and Traces of an LMC)**.** Let $\mathcal{M} = \langle S, \iota, \pi, \Sigma, L \rangle$ be an LMC. A path r of $\mathcal{M}$ is an infinite sequence of states $s_0 s_1 s_2 \ldots$ such that $\iota(s_0) > 0$ and $\pi(s_i, s_{i+1}) > 0$, for all $i \geq 0$. A trace σ associated to a path is the infinite word $\sigma_0 \sigma_1 \sigma_2 \ldots$ such that $\sigma_i = L(s_i)$ for all $i \geq 0$.

Remark that given an arbitrary LMC, we may identify traces using associated paths and not conversely, unless it is a DLMC. In the reminder of this chapter, we provide definitions for paths. Traces can be implicitly deduced.

We begin by defining two types of operations on paths, namely the suffix and the prefix of a path (respectively a trace). Given a path $r = s_0 s_1 s_2 \ldots$, the i^{th} suffix of r is a path starting at state s_i and denoted as $r[i..] = s_i s_{i+1} \ldots$. Conversely, the i^{th} prefix of r is a finite path starting at s_0, ending at state s_i, and denoted as $r[..i] = s_0 \ldots s_i$. A finite path $\hat{r}$ (respectively trace $\hat{\sigma}$) is any finite prefix of a path (respectively trace).

We denote by $Paths(\mathcal{M})$ (respectively $Traces(\mathcal{M})$) the set of all infinite paths (respectively traces) in $\mathcal{M}$ and by $Paths_{fin}(\mathcal{M})$ (respectively $Traces_{fin}(\mathcal{M})$) the set of all finite paths (respectively traces) in $\mathcal{M}$.

Probability Measure of a Markov Chain

The first motivation of building such rich models is to be able to question them to get valuable quantitative answers, that is probabilities of relevant events, e.g., the probability to reach a good state or to avoid a bad state. To this end, such models have to be associated with a well defined probability space which is basically a σ-algebra equipped with a probability measure (see [15] for a detailed description of this notions).

In order to define a probability space for a given LMC $\mathcal{M}$, we first need to specify a σ-algebra for it. In the literature, we generally consider the σ-algebra generated by the cylinder sets each consisting of infinite paths $r \in Paths(\mathcal{M})$ having a common prefix, i.e a finite path $\hat{r} \in Paths_{fin}(\mathcal{M})$.

Definition 2.4 (Cylinder Set of an LMC)**.** The cylinder set of $\hat{r} = s_0 \dots s_n \in Paths_{fin}(\mathcal{M})$ is $Cyl(\hat{r}) = \{r \in Paths(\mathcal{M}) \mid \exists i \in [0, n], \text{ where } r[..i] = \hat{r}\}$.

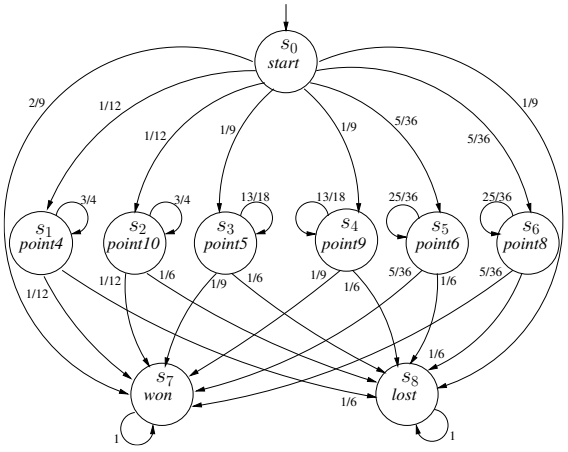
Definition 2.5. The σ-algebra associated with an LMC $\mathcal{M}$ is the smallest σ-algebra encompassing all the cylinder sets $Cyl(\hat{r})$ for all $\hat{r} \in Paths_{fin}(\mathcal{M})$. This induces a unique probability measure $P_{\mathcal{M}}$ on the σ-algebra of $\mathcal{M}$ where the probabilities of the cylinder sets are as follows. Given $\hat{r} = s_0 \dots s_n \in Paths_{fin}(\mathcal{M})$,

$$P_{\mathcal{M}}(Cyl(\hat{r})) = \iota(s_0) \cdot \prod_{0 \leq i < n} \pi(s_i, s_{i+1}).$$

Two LMCs $\mathcal{M}_1$ and $\mathcal{M}_2$ are said to be equivalent, denoted $\mathcal{M}_1 \approx \mathcal{M}_2$, if they have identical probability measures on all the traces, that is, $P_{\mathcal{M}_1} = P_{\mathcal{M}_2}$ (see [83] for more details).

Example 2.1. We consider the Craps Gambling Game introduced in [15]. In this game, a player starts by rolling two fair six-sided dice. The outcome of the two dice determines whether he wins or not. If the outcome is 7 or 11, the player wins. If the outcome is $2, 3$, or 12, the player looses. Otherwise, the dice are rolled again taking into account the previous outcome (called point). If the new outcome is 7, the player looses. If it is equal to point, he wins. For any other outcome, the dice are rolled again and the process continues until the player wins or looses.

The behavior of the craps gambling game is captured by an LMC $\mathcal{M}$. We graphically illustrate the corresponding model in Figure 2.1. A possible path of $\mathcal{M}$ is $r = s_0 s_1 s_1 s_7^{\omega} \in Paths(\mathcal{M})$, where the player wins after three trials. The associated trace σ with r is *start point4 point4 won*$^{\omega} \in Traces(\mathcal{M})$. Let $\hat{r} =$*start point4* be a finite path of $\mathcal{M}$. The cylinder set of $\hat{r}$ is $Cyl(\hat{r}) = \{$*start*



$S = \{s_0, s_1, , ..., s_8\}$

$L(s_0) = start, \ldots, L(s_8) = lost$

$\iota(s_0) = 1$

$$\pi = \begin{array}{c} \\ s_0 \\ s_1 \\ \vdots \\ s_8 \end{array} \begin{array}{c} \begin{array}{cccc} s_0 & s_1 & \cdots & s_8 \end{array} \\ \begin{pmatrix} 0 & \frac{1}{12} & \cdots & \frac{1}{9} \\ 0 & \frac{3}{4} & \cdots & \frac{1}{12} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \end{array}$$

Figure 2.1: An LMC model for the Craps Gambling Game.

$point4^k\ won^\omega | k \geq 1\} \cup \{start\ point4^k\ lost^\omega | k \geq 1\} \cup \{start\ point4^\omega\}$. The probability of the cylinder set of $\hat{r}$ is given by $P_{\mathcal{M}}(Cyl(\hat{r})) = \iota(\ start\ ) \cdot \pi(\ start,point4\ ) = 1 \cdot \frac{1}{12}$.

2.1.2 Markov Decision Process

Markov Decision Processes are more general models which encompass probabilistic and non-deterministic behavior.

Definition 2.6 (Markov Decision Process)**.** A Markov Decision Process (MDP) $\mathcal{M}$ is a tuple $\langle S, Act, \iota, \pi, \Sigma, L\rangle$ where,

- S is a finite nonempty set of states,
- Act is a finite set of action labels,
- $\iota : S \to [0, 1]$ is the *initial states distribution*, such that $\sum_{s\in S} \iota(s) = 1$,
- $\pi : S \times Act \times S \to [0, 1]$ is the *transition probability function*, such that for all states $s \in S$, and action $\varrho \in Act$, $\sum_{s'\in S} \pi(s, \varrho, s') \in \{0, 1\}$,
- Σ, is an alphabet and
- $L : S \to \Sigma$ is a labeling function that maps each state to a set of symbols satisfied in that state.

An action ϱ is enabled in a state s if and only if $\sum_{s'\in S} \pi(s, \varrho, s') = 1$.

Let $Act(s)$ denote the set of enabled actions in a state s. For any state $s \in S$, it is required that $Act(s) \neq \emptyset$. Each state s' for which $\pi(s, \varrho, s') > 0$ is called a ϱ-successor of s, and a transition from s to s' is written as $s \xrightarrow{\varrho} s'$ in this context. We denote by $Prob_{\mathcal{M}}(S)$ the set of states having a single enabled action, that is, $Prob_{\mathcal{M}}(S) = \{s \in S \mid |Act(s)| = 1\}$ and $Det_{\mathcal{M}}(S)$ the set of states that have a single outgoing deterministic transition, that is, $Det_{\mathcal{M}}(S) = \{s \in S \mid \exists \varrho\text{-successor } s', \pi(s, \varrho, s') = 1\}$.

Intuitively, the behavior of an MDP is as follow. Given ι the initial distribution, we probabilistically select an initial state s_0 [1]. To move from state s_0 to some other state, we have first to choose non-deterministically between enabled actions in $Act(s_0)$. Once the non-deterministic choice is taken, say $\varrho \in Act(s_0)$ has been selected, one of the ϱ-successor of s_0 is probabilistically selected according to the distribution $\pi(s_0, \varrho, .)$, where "." stands for any ϱ-successor from s_0.

Example 2.2 (Example of an MDP)**.** Consider the MDP depicted in Figure 2.2 where $S = \{s_0, s_1, s_2\}$ and s_0 is the unique initial state, that is $\iota(s_0) = 1$. The set of enabled actions are $Act(s_0) = \{\varrho, \varsigma\}, Act(s_1) = Act(s_2) = \{\vartheta\}$ with respective probabilities, $\pi(s_0, \varrho, s_1) = \frac{2}{3}, \pi(s_0, \varrho, s_0) = \frac{1}{3}, \pi(s_0, \varsigma, s_1) = \pi(s_0, \varsigma, s_2) = \frac{1}{2}, \pi(s_1, \vartheta, s_1) = \pi(s_2, \vartheta, s_0) = 1$. In this example, non-determinism exists between actions ϱ and ς in s_0. Remark that whatever the choice made between these actions, s_1 is reached with a non-null probability.

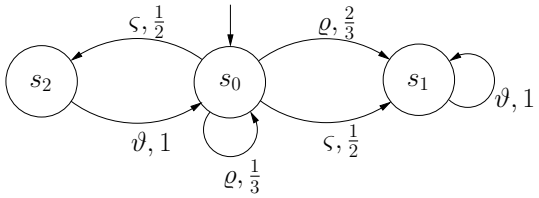


Figure 2.2: An MDP example.

Similarly to LMCs, a path of an MDP is a potential behavior. However, in this case, it consists of an alternating sequence of states and actions obtained by performing non-deterministic choices over actions followed by probabilistic choices over successors.

Definition 2.7 (Paths and Traces of an MDP)**.** Let $\mathcal{M} = \langle S, Act, \iota, \pi, \Sigma, L \rangle$ be an MDP. A path of $\mathcal{M}$ is an infinite alternating sequence of states and actions $r = s_0\varrho_1 s_1 \varrho_2 s_2 ...$ such that $\iota(s_0) > 0$ and $\pi(s_{i-1}, \varrho_i, s_i) > 0$ for all $i \geq 0$. A trace σ associated to a run $s_0\varrho_1 s_1 \varrho_2 s_2 ...$ is the infinite word $L(s_0)\varrho_1 L(s_1)\varrho_2 L(s_2)....$ A finite path (respectively a finite trace) is any finite prefix of a path (respectively of a trace). We denote by $Paths(\mathcal{M})$ (respectively $Traces(\mathcal{M})$) the set of all infinite paths (respectively traces) in $\mathcal{M}$ and by $Paths_{fin}(\mathcal{M})$ (respectively $Traces_{fin}(\mathcal{M})$) the set of all finite paths (respectively traces) in $\mathcal{M}$.

1. In the literature, we may find more general MDP models where several initial distributions are used. In such cases, a non-deterministic choice is performed to select which initial distribution to use. For the sake of simplicity, we restrict ourselves to one initial distribution.

Probability Measure of an MDP

In the case of MDPs, conversely to LMCs, asking a question about a probability of an event to happen is senseless because of non-determinism. More precisely, the probability of such an event will depend on the actions choices, which are performed non-deterministically, that is, they cannot be quantified. Non-determinism is by definition synonym of absence of knowledge, i.e, we do not have any information about how the choices are performed. Defining such a measure is thus not possible unless we fix a clear procedure to do it. This has to be quantifiable, i.e deterministic or probabilistic. Such a procedure is called scheduler or adversary. It defines a policy or a strategy to choose among enabled actions $\varrho \in Act(s)$ for any state $s \in S$. The scheduler does not interfere with the probabilistic choice following the action choice. In this dissertation, we will rely on *probabilistic memoryless schedulers* and we will refer to them by schedulers. They are formally defined as follows.

Definition 2.8 (Probabilistic Memoryless Scheduler)**.** Given an MDP $\mathcal{M} = \langle S, Act, \iota, \pi, \Sigma, L\rangle$, a *probabilistic memoryless scheduler* for $\mathcal{M}$ is a function $\mathscr{S} : S \times Act \to [0,1]$ such that for all $s \in S$, $\sum_{\varrho \in Act(s)} \mathscr{S}(s,\varrho) = 1$, that is a probability distribution over the enabled actions in s, and $\forall \varrho \in Act(s)$, $\mathscr{S}(s,\varrho) > 0$.

A scheduler where $\mathscr{S}(s,\varrho) \in \{0,1\}$ for all s, ϱ is called deterministic scheduler [2].

Given an MDP $\mathcal{M}$ and a Scheduler $\mathscr{S}$, the resulting behavior of $\mathcal{M}$ together with $\mathscr{S}$ is a Markov Chain. Non-determinism is resolved by using the scheduler that assigns a probability to every enabled actions in each state of the MDP. The behavior of $\mathcal{M}$ is thus reduced to a Markov Chain $\mathcal{M}^{\mathscr{S}}$ induced by the scheduler $\mathscr{S}$ as follows.

Definition 2.9 (Markov Chain Induced by a Scheduler)**.** Given an MDP $\mathcal{M}$ and a scheduler $\mathscr{S}$, the behavior induced by $\mathcal{M}$ together with $\mathscr{S}$ is a Markov Chain $\mathcal{M}^{\mathscr{S}} = \langle S, \iota, \pi', \Sigma, L\rangle$ as in Definition 2.1 where π' contains transitions of the form $s \longrightarrow s'$ corresponding to transitions $s \xrightarrow{\varrho} s' \in \pi$, where ϱ is the action selected by $\mathscr{S}$ in state s. The probabilities of such transitions are given by

$$\pi'(s,s') = \sum_{\varrho \in s \xrightarrow{\varrho} s'} (\mathscr{S}(s,\varrho) \cdot \pi(s,\varrho,s')).$$

Example 2.3 (LMC Induced by a Scheduler on the MDP of Example 2.2)**.** Let us consider a scheduler $\mathscr{S}$ that resolves the non-determinism in the MDP in Figure 2.2 by using the following probability distributions:

2. In the literature, we find different types of schedulers, with memory, memoryless, deterministic, and probabilistic. We refer the reader to [15] for a detailed survey.

- $\mathscr{S}(s_0, \varrho) = \frac{1}{3}$,
- $\mathscr{S}(s_0, \varsigma) = \frac{2}{3}$,
- $\mathscr{S}(s_1, \vartheta) = \mathscr{S}(s_2, \vartheta) = 1$.

The Markov Chain induced by such a scheduler is depicted in Figure 2.3. It is obtained by ignoring the set Act of actions (LMCs do not have transitions actions) and by keeping the same state space S, the same alphabet Σ, the same labeling function L, and the same initial state distribution ι. Moreover, a new transition probability function is computed given the scheduler $\mathscr{S}$ by applying Definition 2.9. For example, transition $s_0 \xrightarrow{\varsigma} s_2$ in Figure 2.2 is preserved without the transition label in the induced LMC (see Figure 2.3). The probability of this transition is $\frac{1}{3}$, obtained by multiplying $\frac{2}{3}$ (the scheduler value for this action) and $\frac{1}{2}$ (the corresponding probability in the MDP). Similarly, transition $s_0 \longrightarrow s_1$ in Figure 2.3 corresponds to transitions $s_0 \xrightarrow{\varrho} s_1$ and $s_0 \xrightarrow{\varsigma} s_1$ in the MDP. The probability of this transition is $\frac{5}{9}$, obtained by summing $(\frac{1}{3} \cdot \frac{2}{3})$ and $(\frac{2}{3} \cdot \frac{1}{2})$, where $\frac{1}{3}$ and $\frac{2}{3}$ are respectively the scheduler values for these actions.

Finally, one can easily check that the induced transition probability function is valid by computing the sum of the outgoing transitions probabilities in each state and equating it to 1. For instance, in the initial state s_0, we find that $\frac{1}{6} + \frac{11}{24} + \frac{3}{8} = 1$.

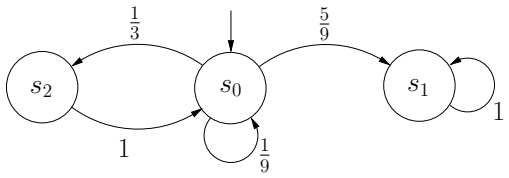


Figure 2.3: Markov chain induced by scheduler $\mathscr{S}$ on the MDP of Figure 2.2.

So far, we recalled two widely used formalisms for modeling stochastic systems behaviors. We will now present a quantitative analysis techniques for theses models. Before that, we first recall the Linear-time Temporal Logic (LTL) for formalizing requirements.

2.2 Requirements Formalization

2.2.1 Linear-time Temporal Logic

A Linear-time Temporal Logic (LTL) [171] formula φ built over a set of atomic propositions AP is defined by the following syntax:

$$\varphi := true \mid ap \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{N}\varphi \mid \varphi_1 \mathsf{U} \varphi_2 \qquad (ap \in AP)$$

N and U are respectively the next and until operators. Additional Boolean operators can be inferred using negation ($\neg$) and conjunction ($\wedge$).

Moreover, temporal operators such as G (*always*) and F (*eventually*) are defined as $\mathsf{F}\varphi \equiv true\ \mathsf{U}\varphi$ and $\mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\varphi$. LTL formulas are interpreted on infinite traces $\sigma = \sigma_0\sigma_1 \ldots \in \Sigma^\omega$ as follows:

- $\sigma \vDash true$;
- $\sigma \vDash ap$ iff $ap \in \sigma_0$
- $\sigma \vDash \neg\varphi$ iff $\sigma \nvDash \varphi$
- $\sigma \vDash \varphi_1 \wedge \varphi_2$ iff $\sigma \vDash \varphi_1$ and $\sigma \vDash \varphi_2$;
- $\sigma \vDash \mathsf{N}\varphi$ iff $\sigma[1..] \vDash \varphi$
- $\sigma \vDash \varphi_1 \mathsf{U} \varphi_2$ iff $\exists k \geq 0$ s.t. $\sigma[k..] \vDash \varphi_2$ and $\forall j \in [0, k[$ holds $\sigma[j..] \vDash \varphi_1$;

Definition 2.10. Given an LMC $\mathcal{M}$ and an LTL property φ, the probability for $\mathcal{M}$ to satisfy φ is denoted by $P(\mathcal{M} \vDash \varphi)$ and is given by the measure $P_{\mathcal{M}}\{\sigma \in \mathit{Traces}(\mathcal{M}) \mid \sigma \models \varphi\}$. In addition, we say that $\mathcal{M}$ satisfies φ denoted by $M \vDash \varphi$ iff $\forall \sigma \in \mathit{Traces}(\mathcal{M}), \sigma \vDash \varphi$. This is different from $P(\mathcal{M} \vDash \varphi) = 1$, which is known as *almost sure* model-checking [3].

2.3 Statistical Model Checking

The model checking approach was shown to be useful for the rigorous analysis of systems. As stated in the introduction, we are following a model-driven approach and we are relying on model checking for system-level verification of performance requirements. This implies the use quantitative techniques for the analysis of stochastic models.

During the last decades, there has been a great interest to extend classical model checking algorithms to cover the probabilistic setting. Probabilistic model checking has emerged in the early eighties [200] and has experienced many improvements that continue today [183, 59, 117, 113, 52, 14, 173, 67]. It has been successfully implemented in various tools, which reached certain degree of maturity [139, 126, 51], and has been applied in several case studies covering various application domains such as automotive [5], industrial process control [132], cloud computing [129], power management [161], avionics communication protocols [27], security [191], and biology [106], to mention but a few.

Given a stochastic model and a formal property, probabilistic model-checking aims to answer the following question: *what is the probability that the model satisfies the property of interest ?* There are two main categories of algorithms to solve the probabilistic model checking problem.

The first category, which is also the earliest, contains algorithms performing exhaustive exploration of the state space of the system. These algorithms rely upon numerical techniques, such as solving systems of linear equations or reformulating them as linear optimization problems. Due to their exhaustive nature, these algorithms provide accurate results. This is paradoxically

3. There might be an infinite trace of $\mathcal{M}$ that does satisfy the property φ and which has a null probability (See [13] for more details).

the source of their weakness: a known limitation of numerical techniques, is actually scalability. Numerical algorithms are not tractable when confronted with real-life applications having a large state space. To overcome this issue, there has been several proposals such as abstraction [172, 125], symmetry reduction [137], symbolic representation through special data structure such as multi-terminal binary decision diagrams (MTBDDs) [91].

The second category is more recent and emerged as an answer to the above challenges. It consists of *statistical model checking* (SMC) which relies on simulation and statistics to bypass the exhaustive exploration of the state space of the system. SMC explores only a sub-part of the state space and provides only an estimation. Using statistical techniques allows to generalize, under certain assumptions, the partial result (obtained on a sample of observations) to the whole system with a fixed confidence. Two formulations of the problem have been proposed by Younes and Simmons [209, 213] on one hand, and by Lassaigne and Peyronnet [142] on the other. The former propose to position the probability with respect to a given threshold without computing it, while the latter propose to estimate it. Whereas the first proposal relies on hypothesis testing and provides a qualitative answer, the second uses probability estimation and could be seen as quantitative.

In the following, we detail statistical model checking, which is the technique we are adopting in this work for system-level performance evaluation of many-cores embedded systems.

Formally, given a Markov Chain $\mathcal{M}$ and an LTL property φ, statistical model checking refers to a series of simulation-based techniques that can be used to answer two types of questions:

Qualitative: Is the probability for $\mathcal{M}$ to satisfy φ greater or equal (lower or equal) to a certain threshold θ ?

Quantitative: What is the probability for $\mathcal{M}$ to satisfy φ ?

General Setting. Let Y_i be a discrete random variable with a Bernoulli distribution of parameter p, i.e., $Y_i \sim B(p)$. Such a variable can only take two values 0 and 1 with $P(Y_i = 1) = p$ and $Pr(Y_i = 0) = 1 - p$. In the context of statistical model checking, each variable Y_i represents one simulation of the system model $\mathcal{M}$. The outcome for Y_i, denoted y_i, is 1 if the i^{th} simulation satisfies φ and 0 otherwise.

Remark that, in the above formulation, in order to be able to evaluate simulations outcomes y_i in a finite time, and hence to ensure the termination of the SMC procedure, one needs to consider bounded simulation traces. Thus, the formalism to be used to express system requirements has also to be bounded. In this case, we usually rely on bounded LTL (BLTL) [89], a variant of LTL with bounded temporal operators. Furthermore, in order to allow expressing probabilistic queries, a probabilistic variant of BLTL is usually used as depicted hereafter.

2.3.1 Probabilistic Bounded LTL

We use Probabilistic Bounded Linear Temporal Logic (PBLTL) as a formalism for describing probabilistic linear temporal properties of a considered system. As stated earlier, this is required to guarantee the termination of the SMC procedure and to allow expressing probabilistic requirements. The bounded fragment of LTL (denoted BLTL) restricts the use of the *until* operator U to its bounded variant U^i. BLTL formulas over the set of atomic propositions AP are syntactically defined as follows.

$$\varphi := true \mid ap \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid N^i\varphi \mid \varphi_1 \mathsf{U}^i \varphi_2 \qquad (ap \in AP)$$

Similarly to LTL, additional bounded temporal operators may be inferred G^i (*always*) and F^i (*eventually*). They are respectively defined as $\mathsf{F}^i\varphi \equiv true\ \mathsf{U}^i\varphi$ and $\mathsf{G}^i\varphi \equiv \neg\mathsf{F}^i\neg\varphi$. Moreover, we syntactically define PBLTL formula as BLTL formula φ encapsulated within probabilistic operator P, i.e. $P[\varphi]$. The P operator have two variants depending on the type of the query: when it is qualitative, we use $P_{\gtrless\theta}[\varphi]$, and when it is quantitative, we use $P_{=?}[\varphi]$.

The semantics of a BLTL formula is defined with respect to a finite trace $\sigma = \sigma_0\sigma_1 \cdots \in \Sigma^*$. It only differs from LTL for the bounded operator N^i and U^i which have respectively the following semantics:

- $\sigma \vDash \mathsf{N}^i\varphi$ iff $\sigma[1..] \cap \sigma[..i] \vDash \varphi$
- $\sigma \vDash \varphi_1 \mathsf{U}^i \varphi_2$ iff $\exists k \in [0, i]$ s.t. $\sigma[k..] \vDash \varphi_2$ and $\forall j \in [0, k[$, $\sigma[..j] \vDash \varphi_1$.

Definition 2.11. Given an LMC $\mathcal{M}$ and a BLTL property φ, the probability for $\mathcal{M}$ to satisfy φ denoted by $P(\mathcal{M} \vDash \varphi)$ is given as in Definition 2.10, that is, $P_{\mathcal{M}}\{\sigma \in Traces_{fin}(\mathcal{M}) \mid \sigma \models \varphi\}$.

Example 2.4. Given the Craps Gambling Game model in Figure 2.1, it is possible, for instance, to check the following P(B)LTL properties.

- The probability to win in one step: $P(true\ \mathsf{U}^1\ won) = 0.22$
- The probability to win in two steps: $P(true\ \mathsf{U}^2\ won) = 0.3$
- The probability to eventually loose: $P(\mathsf{F}\ lost) = 0.51$
- The probability to always win: $P(\mathsf{G}\ won) = 0$

2.3.2 Qualitative Analysis

The main approaches [209, 188] proposed to answer the qualitative question are based on *hypothesis testing*. Let $p = P(\mathcal{M} \models \varphi)$, to determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$ [4]. A simulation-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* of a test is determined by two

4. Note that we are illustrating the hypothesis testing for $p \geq \theta$, however it is similarly applicable for $p \leq \theta$.

parameters, (α, β), such that the probability of accepting K (respectively, H) when H (respectively, K) holds, called a Type-I error (respectively, a Type-II error) is less or equal to α (respectively, β). A test has ideal performance if the probability of the Type-I error (respectively, Type-II error) is exactly α (respectively, β). However, it is impossible to ensure a low probability for both types of errors simultaneously (see [209, 205] for details). A solution is to relax the test using an *indifference region* $[p_1, p_0]$ (with θ in $[p_1, p_0]$) and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$. Usually, we use an indifference region centered on θ by choosing a value δ such that $p_1 = \theta - \delta$ and $p_0 = \theta + \delta$.

We now briefly sketch a hypothesis testing algorithm that is called the *sequential probability ratio test (SPRT)* [205]. In SPRT, one has to choose two values A and B $(A > B)$ that ensure that the strength (α, β) of the test is respected. Let m be the number of observations that have been made so far. The test is based on the following quotient:

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{P(Y_i = y_i \mid p = p_1)}{P(Y_i = y_i \mid p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}},$$

where $d_m = \sum_{i=1}^{m} y_i$. The idea behind the test is to accept H_0 if $\frac{p_{1m}}{p_{0m}} \geq A$, and H_1 if $\frac{p_{1m}}{p_{0m}} \leq B$. The SPRT algorithm computes $\frac{p_{1m}}{p_{0m}}$ for successive values of m until either H_0 or H_1 is satisfied and it is shown to terminate with probability 1 [205]. Sequentially dealing with observations has the advantage of minimizing the number of required simulations. In his thesis [209], Younes proposed a logarithmic based SPRT algorithm that given p_0, p_1, α and β implements the sequential ratio testing procedure.

When one has to compare θ to 0 or 1, it is better to use *Single Sampling Plan* (SSP) (see [209, 144, 188] for details), another hypothesis testing algorithm whose number of simulations is pre-computed in advance. In general, this number is higher than the one needed by SPRT, but it is known to be optimal for the above mentioned values. More details about hypothesis testing algorithms and a comparison between SSP and SPRT can be found in [144].

2.3.3 Quantitative Analysis

In [110, 141] Peyronnet et al. propose an estimation procedure to compute the probability p for $\mathcal{M}$ to satisfy φ. Given a *precision* δ, Peyronnet's procedure, which we call PESTIM, computes a approximation p' such that $|p' - p| \leq \delta$ with *confidence* α, i.e., $P(|p' - p| \leq \delta) \geq 1 - \alpha$. The procedure is based on the *Chernoff-Hoeffding bound* [115].

Let $Y_1 \ldots Y_m$ be m discrete random variables with a Bernoulli distribution of parameter p associated with m simulations of the system. Recall that the outcome for each of the Y_i, denoted y_i, is 1 if the simulation satisfies φ and 0

otherwise. Let $p' = (\sum_{i=1}^{m} b_i)/m$, then Chernoff-Hoeffding bound [115] gives

$$P(|p' - p| > \delta) < 2e^{-\frac{m\delta^2}{4}}.$$

As a consequence, if we consider a number of simulations $m \geq \frac{4}{\delta^2} \log(\frac{2}{\alpha})$, then we are guaranteed that $P(|p' - p| \leq \delta) \geq 1 - \alpha$. Observe that if the value p' returned by PESTIM is such that $p' \geq \theta - \delta$, then $P(\mathcal{M} \models \varphi) \geq \theta$ with confidence $1 - \alpha$.

2.3.4 Playing with Statistical Model Checking Algorithms

The efficiency of the above algorithms is characterized by the number of simulations needed to obtain an answer. This number may change from executions to executions and can only be estimated (see [209] for an explanation). However, some generalities are known. For the qualitative case, it is known that, except for some situations, SPRT is always faster than SSP. PESTIM can also be used to solve the qualitative problem, but it is always slower than SSP [209]. If θ is unknown, then a good strategy is to estimate it using PESTIM with a low confidence and then validate the result with SPRT and a strong confidence.

2.4 Conclusions and Related Work

In this first chapter, we recalled a set of preliminary concepts aimed to be a background reference for the different notions introduced in the forthcoming chapters of the dissertation. Along the chapter, we tried to explain the main principles of statistical model checking, and to recall some widely used stochastic formalism for system modeling, namely, LMCs and MDPs. We also presented probabilistic bounded LTL as requirements specification formalism.

Statistical model checking has received an increasing interest during the last decade. A considerable amount of work in both theoretical and practical sides have been done around. It has been also used in different case studies in various application domains such as avionics communication protocols [27, 26], sensor networks [147, 146], multimedia applications [178], Microgrids [56], analog and mixed-signal circuits [60, 61, 206], subway control systems [86], energy aware buildings [73], wireless networks [50], satellite systems [156], and biology [63, 157, 101, 216]. In the sequel, we enumerate, without claiming to be exhaustive, a panel of related works in term of technical and theoretical contributions.

Besides various emerging full-fledged statistical model checkers, an increasing number of existing probabilistic model checkers have also adopted the statistical approach in addition to their classical numerical techniques.

Most of these tools implements both *probability estimation* and *hypothesis testing* procedures and mainly differ in term of the provided systems modeling/properties specification formalisms and slight implementations details. For instance, Prism [139], which is originally a symbolic model checker and relies on numerical techniques, considers *Discrete Time Markov Chains* (DTMCs), *Continuous Time Markov Chains* (CTMCs), *Markov Decision Process* (MDPs), *Probabilistic Automata* (PAs) [185, 186], and recently *Probabilistic Timed Automata* (PTAs) [138] for systems modeling. As for requirements specification, it accepts a variety of inputs such as PLTL, *Probabilistic Computation Tree Logic* (PCTL) [105], and *Continuous Stochastic Logic* (CSL) [12, 11]. Uppaal-smc [51] is a statistical model checking extension for the UPPAAL model checker. It accepts *Priced Timed Automata* (PTAs) [74] as a system modeling formalism and *Weighted Metric Temporal Logic* (WMTL) [131, 46] as properties specification language. Other tools like Vesta [190] supports, in addition to DTMCs and CTMCs, a rewrite-based specification languages, namely PMaude [134, 2] for modeling stochastic systems, and *Quantitative Temporal Expressions* (QuaTEx) [2] in addition to CSL for requirements specification. Cosmos [19] is another statistical model checker that considers *Generalized Semi-Markov Processes* (GSMPs) [98, 7] (specified as Generalized Stochastic Petri Net), a more general formalism not restricted to finite state space and to the Markovian assumption [5]. It relies on the *Hybrid Automata Stochastic Logic* (HASL) [20] for requirements specifications. Plasma Lab [119] is a modular and extensible statistical model checker that may be extended with external simulator and checkers. The default configuration accepts discrete-time models specified in the Prism format and requirements expressed in PBLTL. Ymer [210] is one of the first tools to implement sequential hypothesis testing algorithms. It considers GMSPs and CTMCs specified using an extension of the Prism input language and accepts both PCTL and CSL for requirements specification. MRMC [126] is originally an explicit-state model checker. It accepts classical DTMCs, CTMCs in addition to a rewards-enriched variants: *Discrete Time Markov Reward Models* (DMRMs) and *Continuous Time Markov Reward Models* (CMRMs) and *Continuous Time Markov Decision Processes* (CTMDPs). It accepts different logic such as PCTL,CSL, *Probabilistic Reward Computation Tree Logic* (PRCTL), and *Continuous Stochastic Reward Logic* (CSRL).

In [162], we present a new statistical model checking tool called BIP$^{\mathcal{SMC}}$, which is briefly discussed in Chapter 7 among others technical contributions. The tool accepts a component-based stochastic formalism called $\mathcal{S}$BIP having an MDP semantics for systems modeling. The latter is introduced in detail in the next chapter. For requirements specification, BIP$^{\mathcal{SMC}}$ considers for now PBLTL properties.

5. In a GSMP, the used probability distributions are not limited to memoryless distribution, e.g., Exponential.

It is worth mentioning that some of the above tools implement different variants of the hypothesis testing algorithm. For example, MRMC uses confidence intervals [214], while Vesta implements simple hypothesis testing [188] rather than the sequential one as in the case of Ymer, Prism, and Plasma Lab. We refer the reader to [117, 123] for a comparison between some statistical model checking tools and techniques.

A key advantage of statistical algorithms is that they are easy to parallelize, since they rely on independent samples. In his thesis [209], Younes addressed this question and proposed a parallel implementation of the sequential hypothesis testing procedure in Ymer. Other tools in the aforementioned list also provide distributed implementation such as [6, 119].

Far from the initial time-bounded properties verification, statistical model checking has evolved to cover properties with unbounded until operator and to verify steady states [189, 212, 85]. It has been also extended to handle black-box systems [188, 211]. In [121, 217], a Bayesian approach of both hypothesis testing and probability estimation procedures is proposed. It is shown that, in specific cases, the Bayesian hypothesis testing algorithm requires fewer observations compared to the classical SPRT algorithm [121].

Statistical model checking has also some disadvantages, such as providing only probabilistic guarantees with respect to requirements satisfaction. This is a real issue when using statistical model checking algorithms to verify safety-critical systems. However, in less constrained contexts such as verifying QoS requirements, this becomes less important. This is why we believe that SMC is more suited for performance evaluation. Another well-known limitation is that it cannot effectively handle rare events [181], that is, event with very small probabilities of occurrence. Several recent work tackle this issue, for instance, using coupling and importance sampling [21], importance sampling and cross-entropy optimization [64, 118], or using importance splitting [120]. Another potential hindrance towards applying SMC is non-determinism, as explained in the beginning of this chapter. Given an MDP, one can only compute the minimal and maximal probability bounds. Various solutions were proposed to remedy to this issue, such as partial order reduction [41] which rarely works, or the use of re-enforcement learning to find an optimal scheduler maximizing the probability of satisfying the property under consideration [108].

In the next chapter, we introduce a new stochastic modeling formalism called $\mathcal{S}$BIP. This enables building MDPs models in a component-based fashion. It also allows defining probabilistic schedulers on top of them in order to produce analyzable LMCs models using statistical model checking techniques, in particular through the BIP$^{\mathcal{SMC}}$ tool.

Chapter 3

Stochastic Component-based Modeling

In the previous chapter we introduced the Statistical Model Checking technique for quantitative analysis of stochastic systems. We also recalled some general stochastic formalisms and provided some basis on requirements formalization. More precisely, we saw that, given a stochastic system model and a set of formalized requirements (properties), Statistical Model Checking enables to estimate the probabilities for the considered stochastic model to satisfy the requirements of interest using simulation and statistical techniques.

Component-based modeling approach allows for building independent and reusable components that can be assembled together for different modeling purposes. Different parts of the whole system can be actually managed separately by different teams, which has the advantage to lighten the modeling burden and considerably reduce time. Modeling is henceforth seen as a matter of reconfiguring and reorganizing existing components, which is not always trivial and should not be neglected. Although, this allows systems designers to focus on *what* to model instead of *how* to model. From an enterprise point of view, more efficiency and flexibility is brought to the production process. In this context, we introduce a new formalism for modeling stochastic systems in a component-based fashion. This is built as an extension of the component-based modeling formalism BIP [22].

The remainder of this chapter is organized as follow. We first recall the main aspects of the BIP formalism, then detail the proposed stochastic extension called $\mathcal{S}$BIP. Our construction follows a recursive scheme, that is, in a first time, we extend basic building blocks (atomic components) with probabilities and show that their underlying semantics is an MDPs. Then, we tackle assembly of such stochastic building blocks using BIP composition operators, and show that they induce new components having the same

semantics. In order to enable Statistical Model Checking on these models, we show in the second part of the chapter, how to resolve non-determinism and hence obtain purely stochastic Markov Chains. We then illustrate the use of the $\mathcal{S}$BIP extension together with SMC on a real-life multimedia example. Finally, in the last part, we focus on the expressiveness of our construction. We show how it is straightforward to model LMCs as well as MDPs in $\mathcal{S}$BIP and discuss equivalence and state space size issues.

3.1 Background on BIP

BIP (*Behavior-Interaction-Priority*) [22] is a formal framework for building complex systems by coordinating the behavior of a set of *atomic components*. *Behavior* is defined as a labeled transition system extended with data. The coordination between components, also referred to as the *Glue*, is layered. The first layer describes the *interactions* between components, whereas, the second layer describes dynamic *priorities* between interactions and is used to express scheduling policies. BIP has a clean operational semantics that describes the behavior of a system model as the composition of the behaviors of its atomic components. This allows a direct relation between the underlying semantic model (labeled transition systems) and its implementation.

Before moving to BIP definitions, let us first recall Labeled Transition Systems (LTS), which represents the underlying semantics of atomic BIP components. In the literature, one can find several formulations of LTSs. In this dissertation, we adopt a definition where labels are action names on transitions.

Definition 3.1 (Labeled Transition System)**.** A labeled transition system is a tuple $(Q, Act, \rightarrow, Q^0)$, where

- Q is a set of states,
- Act is a set of action names, that is transition labels.
- $\rightarrow \subseteq Q \times Act \times Q$ is a set of labeled transitions. For convenience, if $(q, \kappa, q') \in \rightarrow$, we write $q \xrightarrow{\kappa} q'$,
- $Q^0 \subseteq Q$ is a set of initial states.

3.1.1 Atomic Components

Atomic components are the elementary building blocks for modeling a system. They are described as labeled transition systems extended with *variables* used to store local data. Transitions are steps, labeled by *ports*, from a control location to another. They have associated a guard and an update function, that are, respectively, a Boolean condition and a computation defined on local variables. Ports are action names generally used for synchronization with other components. States denote control locations at

which the components await for synchronization. In BIP, data and their related computation are written in C/C++ language. The syntax of an atomic component in BIP is formally defined as follow.

Given a set of variables $\mathcal{V}$, we denote $Bool(\mathcal{V})$ the set of Boolean conditions over $\mathcal{V}$, $Expr(\mathcal{V})$ the set of expressions over $\mathcal{V}$, and $Func(\mathcal{V})$ the set of functions over $\mathcal{V}$.

Definition 3.2 (Syntax of Atomic Components)**.** An atomic component is a labeled transition system extended with data $\mathcal{B} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$, where:

- $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a set of local variables,
- $(\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0)$ is a labeled transition system, with $\mathcal{L} = \{l_1, l_2, \ldots, l_k\}$ a set of control locations, $\mathcal{P}$ a set of communication ports, $\mathcal{T}$ is a set of transitions of the form (l, prt, g, f, l'), where $l, l' \in \mathcal{L}, prt \in \mathcal{P}, g \in Bool(\mathcal{V})$ is a guard, and $f \in Func(\mathcal{V})$ is a deterministic update function on a subset of $\mathcal{V}$. Finally, $l^0 \in \mathcal{L}$ is the initial location.

Let $\mathbb{D}$ be a finite universal domain. Given a set of variables $\mathcal{V}$, we define valuations for variables as functions $X : \mathcal{V} \to \mathbb{D}$ that associate each variable in $\mathcal{V}$ with a value in $\mathbb{D}$. We denote the set of valuations of variables in $\mathcal{V}$ as $\mathbb{D}^{\mathcal{V}} = \{X_0, X_1, \ldots\}$. Given a valuation $X \in \mathbb{D}^{\mathcal{V}}$ and an expression $e \in Expr(\mathcal{V})$, we denote by $e(X)$ the value of e under the valuation X. A set of variables $\mathcal{V}$ is initially associated with a default valuation $X_0 \in \mathbb{D}^{\mathcal{V}}$. We use small scripts x to denote valuation of single variables $v \in \mathcal{V}$.

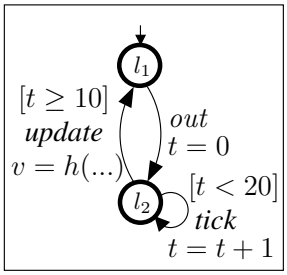


Figure 3.1: An example of a BIP atomic component.

Example 3.1. Figure 3.1 shows a graphical representation of a BIP atomic component. This contains two control locations, l_1 the initial location, and l_2, related through transitions labeled with ports *out, tick*, and *update*. From l_1, the *out* transition is always enabled [1] and leads to location l_2. This transition deterministically updates the variable $t = 0$. From l_2, two transitions are possible, *tick* and *update*. These respectively lead to l_2 and l_1 and are associated with guards $t < 20$ and $t \geq 10$. These guards state that the *tick* transition is enabled whenever $t < 20$ and that the *update* transition is

1. We will conventionally consider, throughout the dissertation, that omitted guards evaluate to *true*.

enabled whenever $t \geq 10$. Given that variable t is set to 0 on transition *out*, only the *tick* transition is first enabled. Executing *tick* increments the variable t until it reaches 10, where the *update* transition becomes also enabled. In such a situation (when several transitions are simultaneously enabled), a non-deterministic choice among the enabled transitions is performed, that is, each transition has an unknown likelihood to be selected. When *tick* is selected it continues incrementing t and remains in location l_2, whereas *update* moves to location l_1 and executes function h that updates variable v.

To summarize, within a BIP atomic component, for a given valuation of variables, a transition can be executed if and only if its associated guard evaluates to *true*. When several transitions are simultaneously enabled, a non-deterministic choice is performed to select one of them. Firing a transition implies an atomic execution of its internal computation f. Formally:

Definition 3.3 (Semantics of Atomic Components)**.** The semantics of an atomic component $\mathcal{B} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$ is an LTS $(Q, Act, \rightarrow, Q^0)$, such that

- $Q = \mathcal{L} \times \mathbb{D}^{\mathcal{V}}$ is the set of states,
- $Act = \mathcal{P}$ is the set of transitions labels,
- $\rightarrow$ is a set including transitions of the form $((l; X), prt, (l'; X'))$ such that, $g(X)$ evaluates to *true* and $X' = f(X)$ for some $\tau = (l, prt, g, f, l') \in \mathcal{T}$,
- $Q^0 = \{(l^0; X_0)\} \subseteq Q$ is the initial state.

Remark 3.1. The unique source of non-determinism at the level of atomic components is when several transitions are enabled simultaneously. Such non-determinism, like the one appearing in location l_2 in the previous example, may be resolved by using disjoint guards or priorities, as shown in the next section. These force a single transition to be enabled each time.

It is worth to mention that, in the BIP semantics, it is not permitted to have, from the same control location, several outgoing transitions labeled with the same port, unless having disjoint guards.

Example 3.2 (An Abstract Model of a Processing Unit)**.** We provide a second example to illustrate how to model real-life systems using BIP. We consider the processing unit of a video decoding system depicted farther in this chapter. The role of this unit within the whole system is to sequentially read input macro-blocks, then to decode them and write them to a buffer. An abstract view of the processing unit behavior is shown in Figure 3.2. It consists of a component having two states, namely, IDLE and PROCESS. The former models a state where the processing unit is waiting to read a macro-block, while the latter models the processing state of the unit.

Moving from one state to another is performed through the *read* and *write* transitions which respectively model reading a macro-block and writing it to a buffer. *Read* and *write* are actually the ports of the component that allow for composition with other components as we will see in the next section.

In both states of the component *tick* transitions are added to model time progress. In the IDLE state, this models the waiting time until a macro-block is available for read, whereas, in the PROCESS state, *tick* models the time for processing a macro-block, which is proportional to its size.

Note that in this example, there exists also non-deterministic choices between *tick* and *read* transitions at the IDLE state, and between *tick* and *write* transitions at the PROCESS state.

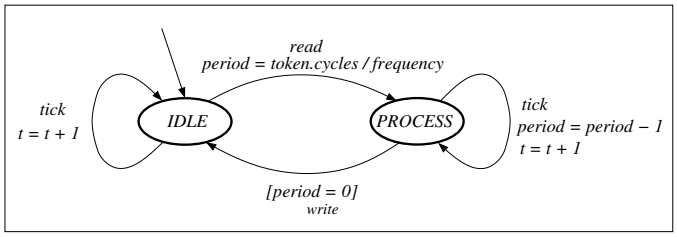


Figure 3.2: A BIP component of a processing unit of a video decoding system. The processing unit decodes macro-blocks at a pre-specified frequency. The processing time of each macro-block corresponds to its size.

Atomic components capture individual sub-behaviors of a more complex structure. These have to be assembled together in order to produce a desired global behavior, which is achieved by using compositional or *glue* operators. BIP provides several operators that allow composing and coordinating atomic components. The next section introduces these operators.

3.1.2 Composition Operators

In component-based design, systems are defined by assembling atomic components using composition operators. BIP offers a layered *glue* which provides mechanisms for coordinating components behaviors, namely *interactions* and *priorities*.

Interactions

The first layer of the BIP glue relies on *connectors*. These provide mechanisms to relate ports from different sub-components towards composition. They represent sets of interactions, that are, non-empty sets of ports that have to be jointly executed. For every such interaction, the underlying connector provides a guard and a data transfer function, that are, respectively, an enabling condition and an exchange of data across the involved ports.

For a model built from a set of component $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n$, where $\mathcal{B}_i = (\mathcal{L}_i, \mathcal{P}_i, \mathcal{T}_i, l_i^0, \mathcal{V}_i)$, we assume that their respective sets of ports and variables are pairwise disjoint, i.e. for any $i \neq j$ in $\{1 \ldots n\}$, we require that $\mathcal{P}_i \cap \mathcal{P}_j = \emptyset$ and $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$. Thus, we define the set $\mathcal{P} = \bigcup_{i=1}^{n} \mathcal{P}_i$ of all ports in the model

as well as the set $\mathcal{V} = \bigcup_{i=1}^{n} \mathcal{V}_i$ of all variables. An interaction is formally defined as follow.

Definition 3.4 (Interaction)**.** An interaction a is a triple (P_a, g_a, f_a) where $P_a \subseteq \mathcal{P}$ is a set of ports, g_a is a guard, and f_a is a data transfer function. We restrict P_a so that it contains at most one port of each component, therefore we denote $P_a = \{prt_i\}_{i \in I}$ with $prt_i \in \mathcal{P}_i$ and $I \subseteq \{1 \dots n\}$. The guard g_a and the data transfer function f_a are defined on the variables of the interacting components.

Given a set of interactions γ, the composition of components $\mathcal{B}_1, \dots, \mathcal{B}_n$ using γ is an atomic component in which, for a given valuation of variables, an interaction $a \in \gamma$ can be executed if its associated guard g_a evaluates to true and all its involved ports P_a are enabled. The execution of a is an atomic sequence of two micro-steps:

1. Execution of the interacting ports $\{prt_i\}_{i \in I} \in P_a$, which is a synchronization between the underlying components, with an exchange of data through the execution of the data transfer function f_a, followed by
2. Execution of internal update functions f_i associated with the respective transitions τ_i labeled with ports $\{prt_i\}_{i \in I} \in P_a$.

We denote by $f_1 \odot f_2$ the sequence of execution of function f_1 followed by the execution of function f_2. The composition operations is formally defined as follow.

Definition 3.5 (Composition)**.** The composition $\gamma(\mathcal{B}_1, \dots, \mathcal{B}_n)$ of n atomic components using the set of interactions γ, where $\mathcal{B}_i = (\mathcal{L}_i, \mathcal{P}_i, \mathcal{T}i, l_i^0, \mathcal{V}_i)$ is an atomic component $\mathcal{B} = (\mathcal{L}, \gamma, \mathcal{T}, l^0, \mathcal{V})$, where

- $\mathcal{L} = \mathcal{L}_1 \times \ldots \times \mathcal{L}_n$ is the set control locations,
- γ is the set of ports,
- $\mathcal{T}$ contains transitions of the form $\tau = ((l_1, \dots, l_n), a, G_a, F_a, (l'_1, \dots, l'_n))$ obtained by synchronization of sets of transitions $\{\tau_i = (l_i, prt_i, g_i, f_i, l'_i) \in \mathcal{T}_i\}_{i \in I}$ such that, $a = \{prt_i\}_{i \in I} \in \gamma$, $G_a = g_a \wedge \bigwedge_{i \in I} g_i$, $F_a = f_a \odot \bigcup_{i \in I} f_i$, and $l'_j = l_j$ if $j \notin I$.
- $l^0 = l_1^0 \times \ldots \times l_n^0$ is the initial control location,
- $\mathcal{V} = \bigcup_{i=1}^{n} \mathcal{V}_i$ is the resulting set of variables.

Example 3.3. Figure 3.3 shows the component *System* consisting of three interacting sub-components, *Sender*, *Buffer* and *Receiver* assembled using three connectors *io1, io2*, and *toc*. We graphically represent ports involved in interactions by boxes containing ports names. Furthermore, when such a port is associated with variables, they are represented in a second box near the port. Black bullets in the connectors ends state that the underlying synchronizations are of type *rendez-vous*, that is a strong syn-

chronization between the participating ports[2]. A *rendez-vous* connector yields one interaction. For instance, the *io1* connector gives the interaction $a = (P_a = \{out1, in2\}, f_a = (v_2 = v1), g_a$ evaluates to $true)$. Note that the data transfer function is optional as for the *toc* connector.

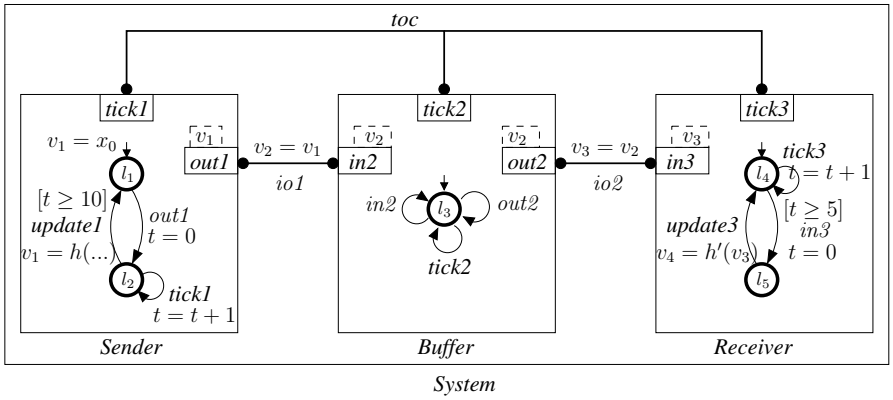


Figure 3.3: BIP example: Sender-Buffer-Receiver system.

The *System* component models a producer-consumer scheme through a buffer. The *Sender* produces new data values, represented by v_1, using the update function h, and write them to the *Buffer*. This is performed after 10 time units as forced by the *tick1* transition and the guard $t \geq 10$. Communication between the *Sender* and the *Buffer* is performed through the *io1* connector which provides a data transfer function $v_2 = v_1$, where v_2 is the *Buffer* local variable representing the buffer memory. Similarly, the *Receiver* component communicates with the *Buffer* through the *io2* connector. This ensures data transfer from the *Buffer* to the *Receiver* $(v_3 = v_2)$. The latter reads from the *Buffer* after 5 time units and uses the received data for some local computation $v_4 = h'(v_3)$. The *toc* connector synchronizes the *tick* transitions of the different components to enable overall time progress.

Non-determinism appears at this level when several interactions are enabled simultaneously. In the previous example for instance, from location l_4 of the *Receiver*, ports *tick3* and *in3*, involved in interactions *toc* and *io2* respectively, are both enabled when $t \geq 5$.

Given the composition semantics above, assembling deterministic components provides no guarantee that the produced component will be deterministic. Similarly, assembling non-deterministic ones does not imply necessarily that the obtained component is non-deterministic. This really depends on the used connectors and their respective guards. In addition to the use of disjoint guards, another mechanism to deal with non-determinism appearing at this level is to use priority rules introduced below.

2. BIP offers different types of connectors, e.g *broadcast* which are out of the scope of this dissertation. We refer the reader to [22] for more details.

Priorities

Priorities represent the second layer of the BIP glue. They provide means to coordinate the execution of interactions within a BIP system. Priorities are used to specify scheduling or similar arbitration policies between simultaneously enabled interactions. More concretely, priorities are rules, each consisting of an ordered pair of interactions associated with a condition. Formally:

Definition 3.6 (Priority)**.** A priority rule is a strict partial order over interactions $\prec \subset \gamma \times Bool(\mathcal{V}) \times \gamma$, where γ is the set of interactions and $Bool(\mathcal{V})$ is a set of Boolean conditions on variables $\mathcal{V}$. Given two interactions a and b, the priority rule (a, C, b) states that whenever the condition C holds, interaction a has less priority than interaction b. Given a priority rule (a, C, b), we write it as $a \prec_C b$ for more convenience.

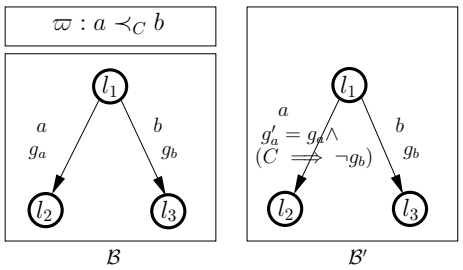


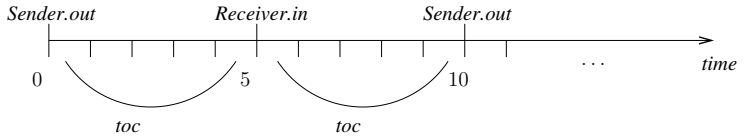
Figure 3.4: Priority restriction example.

We denote ϖ the set of priority rules or the priority model. Given a component $\mathcal{B} = (\mathcal{L}, \gamma, \mathcal{T}, l^0, \mathcal{V})$, we call *restriction* the operation of applying the priority model ϖ on $\mathcal{B}$, which results on disabling interactions having lower priorities. This is achieved by using guards transformations as follow. Consider the left component in Figure 3.4 which shows a component $\mathcal{B}$ having two interactions a and b from the control location l_1. Component $\mathcal{B}$ is associated with a priority layer ϖ consisting of a single rule, $a \prec_C b$ stating that whenever condition C holds, interaction b has the priority to execute over interaction a. Component $\mathcal{B}'$, shown in the right side of Figure 3.4, is the result of applying the priority model on $\mathcal{B}$. This is obtained by transforming the guard g_a in $\mathcal{B}$ to g'_a, where $g'_a = g_a \wedge (C \implies \neg g_b)$, that is by adding the complement of g_b to g_a whenever C holds. Restriction operation is formally defined as follow:

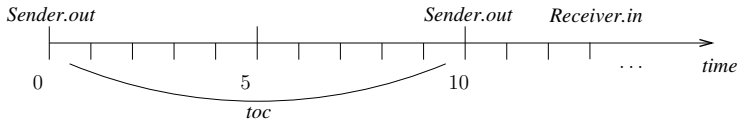
Definition 3.7 (Restriction)**.** Given an atomic component $\mathcal{B} = (\mathcal{L}, \gamma, \mathcal{T}, l^0, \mathcal{V})$ and a priority model ϖ, the restriction $\varpi(\mathcal{B})$ gives an atomic component $\mathcal{B}' = (\mathcal{L}, \gamma, \mathcal{T}', l^0, \mathcal{V})$ as in Definition 3.5, where $\mathcal{T}'$ contains transitions of the form $\tau = (l, a, g'_a, f_a, l')$ such that, $g'_a = g_a \wedge \bigwedge_{a \prec_C b \in \varpi} (C \implies \neg g_b)$.

Example 3.4 (Following Example 3.3)**.** In Example 3.3, the goal is to have a model acting as follow. Whenever the *Sender* writes data to the *Buffer*, the *Receiver* reads it before it gets overridden by a new data, since the *Buffer* has a single memory location. That is, we should never have successive writes nor reads. The expected behavior is shown in Figure 3.5a.

In the model presented in Figure 3.3, this is performed using time synchronization. The *Sender* is forced to write data after 10 time units while the *Receiver* reads it after 5 time units. In this way they are expected to alternate production and consumption. As previously mentioned, the model in Figure 3.3 exhibits non-determinism which actually alters the desired behavior. Consider a situation where the *Sender* is writing data to the *Buffer* through the *io1* connector. Meanwhile, the *Receiver* is waiting in location l_4. When $t \geq 5$, both *tick3* and *in3* become enabled. A non-deterministic choice is then performed among them. What may happen is that *tick3* is taken 10 consecutive times, which implies that *tick1* has also been taken 10 times (*ticks* are strongly synchronized using *toc* connector). This enables the *Sender* to write new data to the *Buffer* while the previous one is not yet consumed by the *Receiver* as shown in Figure 3.5b.



(a) Expected behavior of the *System* model



(b) *System* actual behavior

Figure 3.5: Behavior of the *System* component in Figure 3.3.

To resolve such a situation, one may change the *Buffer* model in such a way it does not allow successive reads nor writes which is very restrictive since it does not allow components reuse. Another alternative consists to use priorities to force *in* and *out* actions, when enabled, to execute over *tick*. A priority model consists thus on

$$\varpi : toc \prec_C io1, toc \prec_C io2$$

. The C condition in this case is simply $true$. The priority model ϖ stipulates that whenever the participating ports in $io1$ or $io2$ are enabled simultaneously with a *tick*, they will have the priority to execute. For instance, when

$t \geq 5$, both *tick3* and *in3* are enabled, but because of priorities, *in3* will be taken. Hence, we get the expected behavior shown in Figure 3.5a.

So far, we recalled the component-based formalism BIP and its associated semantics. We are now ready to build our stochastic construction as an extension of the previously introduced formalism.

3.2 $\mathcal{S}$BIP: A Stochastic Extension

We recall that our goal is to build stochastic models capturing performance information to enable quantitative analysis in early design phases. In this section, we introduce $\mathcal{S}$BIP, an extension of the BIP formalism above. This will constitute the foundation upon which we will build our approach for performance modeling and evaluation in system-level design context. $\mathcal{S}$BIP was first introduced in [32]. It allows (1) to specify stochastic aspects of individual components and (2) to provide a stochastic semantics for the parallel composition of components through interactions and priorities. We first introduce stochastic atomic components and define their underlying semantics. Then, we provide an MDP semantics for their parallel composition. Finally, we show how to obtain a purely stochastic LMC semantics towards analysis using SMC.

3.2.1 Stochastic Atomic Components

Syntactically, we add stochastic behavior at the level of atomic components by allowing the definition of probabilistic variables. These are, in contrast to deterministic variables, attached to given probability distributions and thus updated probabilistically. Probability distributions are defined as functions that associate possible valuations of probabilistic variables with some weight. Formally,

Definition 3.8 (Distributions of Probabilistic Variables)**.** Let $\mathbb{D}$ be a finite universal data domain. A probability distribution over $\mathbb{D}$ is a function $\mu : \mathbb{D} \to [0, 1]$ such that, $\sum_{x_i \in \mathbb{D}} \mu(x_i) = 1$ for all $x_i \in \mathbb{D}$.

Definition 3.9 (Syntax of Stochastic Atomic Components)**.** A stochastic atomic component is an atomic component extended with probabilistic variables $\mathcal{B}^s = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$, where

- $\mathcal{L} = \{l_1, l_2, \ldots, l_k\}$ is a set of control locations,
- $\mathcal{P}$ is a set of communication ports,
- $\mathcal{V} = \mathcal{V}^d \biguplus \mathcal{V}^p$, with $\mathcal{V}^d = \{v_1, \ldots, v_n\}$ the set of deterministic variables as in Definition 3.2 and $\mathcal{V}^p = \{v_1^p, \ldots, v_m^p\}$ the set of probabilistic variables attached to a set of probability distributions $\mu_{\mathcal{V}^p} = \{\mu_1, \ldots, \mu_m\}$.
- $\mathcal{T}$ is a set of transitions of the form $\tau = (l, prt, g, f, l')$, where $l, l' \in \mathcal{L}, prt \in \mathcal{P}, g$ is a guard over $Bool(\mathcal{V})$, and f is a pair (f^d, f^p), where

f^d is a deterministic update function on $\mathcal{V}$ as in Definition 3.2, and $f^p \subseteq \mathcal{V}^p$ is the subset of probabilistic variables to be updated on τ,
– l^0 is the initial location.

We define valuations of probabilistic variables as in the deterministic case. We denote the set of valuations of a set of probabilistic variables $\mathcal{V}^p$ as $\mathbb{D}^{\mathcal{V}^p}$. $\mathcal{V}^p$ is initially associated with a default valuation $X_0^p \in \mathbb{D}^{\mathcal{V}^p}$. Each probabilistic variable $v^p \in \mathcal{V}^p$ is attached to a probability distribution $\mu \in \mu_{\mathcal{V}^p}$, denoted as $v^p \sim \mu$. Note that the used probability distributions are discrete and assumed to be independent. We use small scripts x^{v^p} to denote valuations of single probabilistic variables $v^p \in \mathcal{V}^p$.

Semantics of Stochastic Atomic Components

The introduction of probabilistic variables at the level of atomic components engenders a probabilistic behavior over transitions. Let us consider the atomic component $\mathcal{B}^{\mathfrak{s}}$ in Figure 3.6a. We graphically represent update of probabilistic variables using $\triangleright$ near variables names. Component $\mathcal{B}^{\mathfrak{s}}$ has a unique transition going from location l_1 to location l_2 using port prt which updates the probabilistic variable v^p, defined over domain $\mathbb{D}^{v^p} \subseteq \mathbb{D}$, according to the distribution $\mu : \mathbb{D}^{v^p} \to [0, 1]$. Assuming the initial value of v^p is $x_0^{v^p}$, when executing $B^{\mathfrak{s}}$, there will be several possible transitions, having the same label prt, from state $(l_1, x_0^{v^p})$ to states $(l_2, x_i^{v^p})$ for all $x_i^{v^p} \in \mathbb{D}^{v^p}$ as illustrated in Figure 3.6b. According to the definition of probabilistic vari-



(a) A Stochastic Atomic Component $\mathcal{B}^{\mathfrak{s}}$ (b) Behavior of $\mathcal{B}^{\mathfrak{s}}$

Figure 3.6: Example of a stochastic atomic component $\mathcal{B}^{\mathfrak{s}}$ and its behavior.

ables, the probabilities of these transitions will then be given by μ. Note that when several probabilistic variables are updated, the resulting distribution on transitions will be the product of the distributions associated to each variable as will be shown in Example 3.5. We recall that these distributions are fixed during variables declaration, and are considered to be independent as stated in the beginning of this section.

Adapting the semantics of an atomic component in BIP as presented in Definition 3.3 to atomic components with probabilistic variables leads to be-

haviors that combine both stochastic and non-deterministic aspects. Indeed, even if transitions are either purely deterministic or purely stochastic [3], several transitions can be enabled in a given state. In such a case, the choice between them is non-deterministic as explained in the BIP background section.

Consider a stochastic component $\mathcal{B}^{\mathfrak{s}} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$. Given a state $(l; X^p; X)$ in $\mathcal{L} \times \mathbb{D}^{\mathcal{V}^p} \times \mathbb{D}^{\mathcal{V}^d}$, we denote by $\mathsf{Enabled}(l; X^p; X)$ the set of transitions in $\mathcal{T}$ that are enabled in that state, i.e. transitions $\tau = (l, prt, g, f, l') \in \mathcal{T}$, such that $g(X^p; X)$ is satisfied. Remark that the set $\mathsf{Enabled}(l; X^p; X)$ may have a cardinality $|\mathsf{Enabled}(l; X^p; X)| \geq 1$, as explained earlier. In the associated semantics of $\mathcal{B}^{\mathfrak{s}}$, a non-deterministic choice between ports in $\mathsf{Enabled}(l; X^p; X)$ is first performed. Then, probabilistic selection of the next state is done according to distributions attached to probabilistic variables updated on the transition labeled by the selected port. The semantics of a stochastic atomic component is thus an MDP as formally stated below:

Definition 3.10. The semantics of a stochastic atomic component $\mathcal{B}^{\mathfrak{s}} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$ is a Markov Decision Process $\mathcal{M} = \langle S, Act, \iota, \pi, \Sigma, L \rangle$, where

- $S = \mathcal{L} \times \mathbb{D}^{\mathcal{V}^p} \times \mathbb{D}^{\mathcal{V}^d}$ is the set of states,
- $Act = \mathcal{P}$ is the set of actions,
- $\iota(l^0) = 1$ is the initial probability distribution,
- Σ is the set of atomic proposition over $\mathcal{V}$,
- $L : S \to \Sigma$ is the state labeling function,
- $\pi : S \times Act \times S \to [0, 1]$ contains transition of the form $s \xrightarrow{prt} s'$ where $s = (l; X^p; X)$ and $s' = (l'; X'^p; X')$ for some $\tau = (l, prt, g, (f^d, f^p), l') \in \mathsf{Enabled}(l; X^p; X)$ where,

$$\begin{array}{ll} X'^p(v^p) = X^p(v^p) & \text{for all } v^p \notin f^p \\ X'^p(v^p) = x^{v^p} & \text{for each } v^p \in f^p, \text{ where } x^{v^p} \in \mathbb{D}^{v^p}, \end{array}$$

$(X''^p, X') = f^d(X'^p, X)$, and the probability to take a transition is defined using $\mu_{\mathcal{V}^p}$, the set of probability distributions attached to $\mathcal{V}^p$ as follow:

$$\pi(s, prt, s') = \prod_{v^p \in f^p} \mu^{v^p}(x^{v^p})$$

Let $\mathcal{B}^{\mathfrak{s}}$ be a stochastic atomic component and $\mathcal{M}$ be the associated MDP. A state $s \in S$ is seen as the combination of a control location $l \in \mathcal{L}$ with the valuations of the probabilistic and deterministic variables, respectively X^p and X. We write $s = (l; X^p; X)$. The set of enabled actions in some state $Act(s)$ corresponds to the set of ports labeling transitions $\tau \in \mathsf{Enabled}(l; X^p; X)$, where $s = (l; X^p; X)$. That is, $Act(s) = \{prt \in \mathcal{P} \mid \tau = (l, prt, g, (f^d, f^p), l') \in \mathsf{Enabled}(l; X^p; X)\}$.

3. In the sense that they are associated or not with probabilistic variables.

From now on, we will use $Act(s)$ to denote the set of ports enabled in some state s of a stochastic atomic component and ports labels to denote transitions. We will also adopt the prt-successor notation from the MDP definition in Chapter 2.

In each state $s \in S$, we select a port $prt \in Act(s)$ non-deterministically. We recall that all variables $v^p \in \mathcal{V}^p$ are defined over domains $\mathbb{D}^{v^p} \subseteq \mathbb{D}^{\mathcal{V}^p}$ and are associated with distributions $v^p \sim \mu^{v^p}$. A prt-successor state $s' = (l; X'^p; X')$ is obtained by:

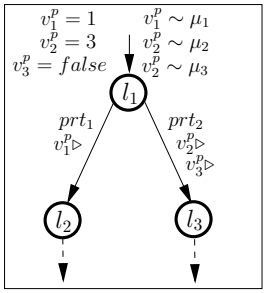
1. Keeping valuations of non-updated probabilistic variables on prt unchanged, i.e., $\forall v^p \notin f^p, X'^p(v^p) = X^p(v^p)$,
2. Assigning new valuations $x^{v^p} \in \mathbb{D}^{v^p}$ to updated ones, i.e., $\forall v^p \in f^p, X'^p(v^p) = x^{v^p}$, and
3. Updating variables according to f^d, i.e., $(X''^p, X') = f^d(X'^p, X)$.

The probability of a transition $s \xrightarrow{prt} s'$ is the product of the distributions μ^{v^p} of all $v^p \in f^p$, that is $\prod_{v^p \in f^p} \mu^{v^p}(x^{v^p})$.

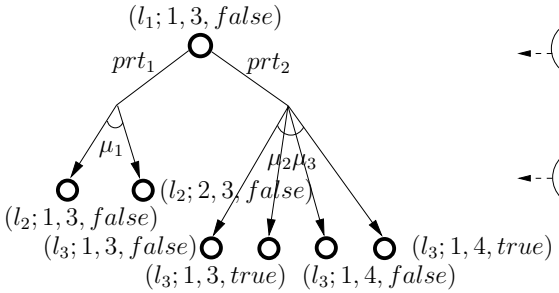
Example 3.5. Consider a stochastic atomic component $\mathcal{B}^5$ having three locations l_1, l_2, and l_3 and two enabled transitions prt_1 and prt_2 from l_1 to l_2 and l_3 respectively, as shown in Figure 3.7a. Transition prt_1 is associated with a probabilistic variable $v_1^p \sim \mu_1$ and prt_2 is associated with two probabilistic variables $v_2^p \sim \mu_2$ and $v_3^p \sim \mu_3$. Moreover, v_1^p is defined over domain $\mathbb{D}_1 = \{1, 2\}$, and v_2^p and v_3^p are respectively defined over domains $\mathbb{D}_2 = \{3, 4\}$ and $\mathbb{D}_3 = \{true, false\}$. The probabilities of variables valuations are given by the associated distributions.

In the underlying MDP semantics depicted in Figure 3.7c, we find the enabled actions prt_1 and prt_2, each unfolded with respect to its associated probability distribution as defined in the $\mathcal{B}^5$ component. For instance, we identify two transitions labeled prt_1 having probabilities $\mu_1(1)$ and $\mu_1(2)$. For prt_2, since its is associated with two probabilistic variables in $\mathcal{B}^5$, we obtain four transitions labeled prt_2, due to the product $\mu_2\mu_3$, having the probabilities $\mu_2(3)\mu_3(true)$, $\mu_2(3)\mu_3(false)$, $\mu_2(4)\mu_3(true)$, and $\mu_2(4)\mu_3(false)$.

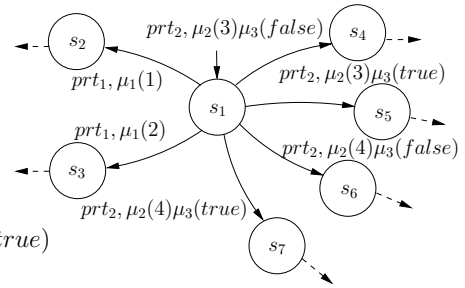
Example 3.6 (Part of a Stochastic Multimedia Stream Source)**.** In this example, we model the stochastic behavior of an MPEG2-coded video stream source. This will actually be part of the input stream model for the video decoding system initially introduced in Example 3.2. In this sub-model of the input stream source, we only consider frames generation. An MPEG2-coded stream is composed of coded video frames of three types, namely I,P, and B [133]. These are organized following a predefined Group Of Pictures (GOP) pattern: $IBBPBBPBBPBB, IBBPBBPBBPBB, \ldots$ Figure 3.8 shows an $\mathcal{S}$BIP component modeling this behavior, where frames sizes are produced in a probabilistic fashion following three different probability distributions (μ_i, μ_p, μ_b), each corresponding to a frame type.



(a) A Stochastic BIP Component $\mathcal{B}^5$



(b) Underlying Behavior of $\mathcal{B}^5$



(c) The MDP semantics of $\mathcal{B}^5$

Figure 3.7: Example of a stochastic atomic component $\mathcal{B}^5$ and its underlying semantics when several transitions are simultaneously enabled.

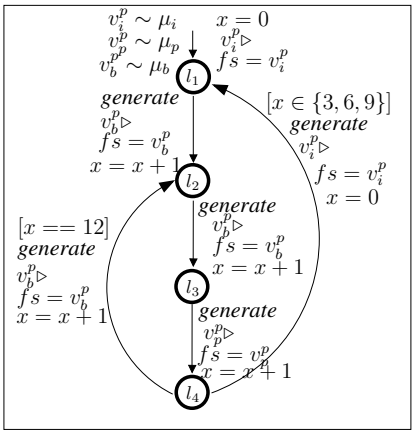


Figure 3.8: Stochastic Multimedia Stream Source model: frames generator component.

Till now, we saw that the semantics of an $\mathcal{S}$BIP component is defined as a Markov Decision Process. In the next section we provide definitions for assembling stochastic components using BIP composition operators.

3.2.2 Composition of Stochastic Components

In this section we define the semantics of the parallel composition of stochastic atomic components. In this context, compositional operators are unchanged, that is, we adopt the same definitions of interactions (Definition 3.4) and priorities (Definition 3.6). For the sake of simplicity, we restrict data transfer functions on interactions to be deterministic. Thus, probabilistic and deterministic variables are transferred in a deterministic way.

When considering a system with n stochastic components $\mathcal{B}_i^{\mathfrak{s}} = (\mathcal{L}_i, \mathcal{P}_i, \mathcal{T}_i, l_i^0, \mathcal{V}_i)$ and a set of interactions γ, the construction of the product component $\mathcal{B}^{\mathfrak{s}} = \gamma(\mathcal{B}_1^{\mathfrak{s}}, \ldots, \mathcal{B}_n^{\mathfrak{s}})$ is performed as in BIP. Formally:

Definition 3.11. (Composition of Stochastic Components) Given a set of interactions γ, the composition of n stochastic components $\gamma(\mathcal{B}_1^{\mathfrak{s}}, \ldots, \mathcal{B}_n^{\mathfrak{s}})$, where $\mathcal{B}_i^{\mathfrak{s}} = (\mathcal{L}_i, \mathcal{P}_i, \mathcal{T}_i, l_i^0, \mathcal{V}_i)$ is a stochastic atomic component $\mathcal{B}^{\mathfrak{s}} = (\mathcal{L}, \gamma, \mathcal{T}, l^0, \mathcal{V})$ where,

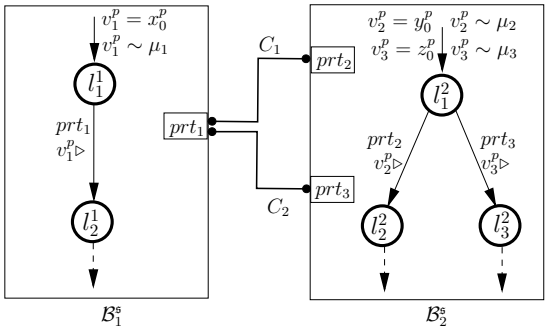
- $\mathcal{L} = \mathcal{L}_1 \times \ldots \times \mathcal{L}_n$, is a set of control locations,
- γ is the set of ports,
- $\mathcal{T}$ contains transitions of the form $\tau = ((l_1, \ldots, l_n), a, G_a, (F_a^d, F_a^p), (l_1', \ldots, l_n'))$ obtained by synchronization of sets of transitions $\{\tau_i = (l_i, prt_i, g_i, f_i, l_i') \in \mathcal{T}_i\}_{i \in I}$ such that,
  - $a = \{prt_i\}_{i \in I} \in \gamma$,
  - $G_a = g_a \wedge \bigwedge_{i \in I} g_i$,
  - $F_a^d = f_a^d \odot \bigcup_{i \in I} f_i^d$,
  - $F_a^p = \bigcup_{i \in I} f_i^p$, and
  - $l_j' = l_j$ if $j \notin I$,
- $l^0 = l_1^0 \times \ldots \times l_n^0$,
- $\mathcal{V} = \bigcup_{i=1}^n \mathcal{V}_i$, is the set of deterministic and probabilistic variables.

Restriction operation over stochastic atomic components is performed using priority rules in Definition 3.6. It consists of the same guards transformation operation presented in Definition 3.7. Thus, it produces a new stochastic component.

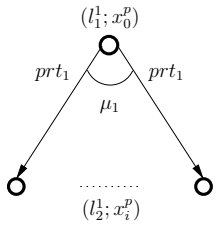
By construction, as assembling stochastic atomic components using composition operators (composition and restriction) produces a stochastic atomic component, its semantics is a Markov Decision Process given by Definition 3.10.

Example 3.7. Consider the stochastic BIP components $\mathcal{B}_1^{\mathfrak{s}}$ and $\mathcal{B}_2^{\mathfrak{s}}$ given in Figure 3.9a. $\mathcal{B}_1^{\mathfrak{s}}$ has a single probabilistic variable v_1^p, defined over domain $\mathbb{D}_1$ and attached to distribution μ_1, and a single transition from location l_1^1 to location l_2^1 using port prt_1, where v_1^p is updated. In location l_1^1, the variable v_1^p is assumed to have value $x_0^p \in \mathbb{D}_1$. $\mathcal{B}_2^{\mathfrak{s}}$ has two probabilistic variables v_2^p and v_3^p respectively defined over domains $\mathbb{D}_2$ and $\mathbb{D}_3$, to which are attached distributions μ_2 and μ_3 respectively. $\mathcal{B}_2^{\mathfrak{s}}$ has two transitions: a

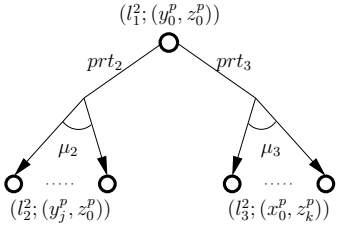
transition from location l_1^2 to location l_2^2 using port prt_2, where v_2^p is updated, and a transition from location l_1^2 to location l_3^2 using port prt_3, where v_3^p is updated. In location l_1^2, the variables v_2^p and v_3^p are assumed to initially have value y_0^p and z_0^p in $\mathbb{D}_2$ and $\mathbb{D}_3$ respectively. Let $\gamma = \{a = \{prt_1, prt_2\}, b = \{prt_1, prt_3\}\}$ be the set of interactions resulting from connectors C_1 and C_2 respectively. Interactions a and b have the same priority since we are not specifying any priority rules. The semantics of the component $\mathcal{B}^{\mathfrak{s}} = \gamma(\mathcal{B}_1^{\mathfrak{s}}, \mathcal{B}_2^{\mathfrak{s}})$ is given in Figure 3.9e.



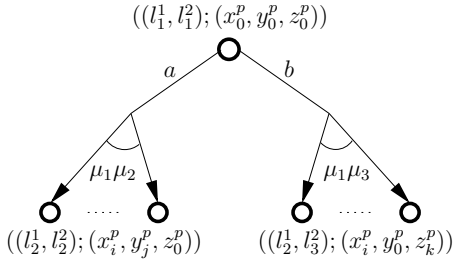
(a) Composition of two $\mathcal{S}$BIP components



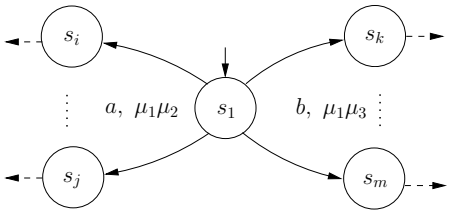
(b) Underlying Behavior of $\mathcal{B}_1^{\mathfrak{s}}$



(c) Underlying Behavior of $\mathcal{B}_2^{\mathfrak{s}}$



(d) Underlying Behavior of $\mathcal{B}^{\mathfrak{s}} = \gamma(\mathcal{B}_1^{\mathfrak{s}}, \mathcal{B}_2^{\mathfrak{s}})$, with $\gamma = \{a = \{prt_1, prt_2\}, b = \{prt_1, prt_3\}\}$



(e) Associated MDP to $\mathcal{B}^{\mathfrak{s}} = \gamma(\mathcal{B}_1^{\mathfrak{s}}, \mathcal{B}_2^{\mathfrak{s}})$

Figure 3.9: Illustration of the stochastic semantics of composition in $\mathcal{S}$BIP.

In state $s_1 = ((l_1^1, l_1^2); (x_0^p, y_0^p, z_0^p))$ of the composition, a non-deterministic choice between interactions a and b is performed. After choosing the inter-

action, the corresponding transition is taken, updating the corresponding probabilistic variables with the associated distributions. As an example, the probability of going to state $((l_2^1, l_2^2); (x_i^p, y_j^p, z_0^p))$, where $x_i^p \in \mathbb{D}_1$ and $y_j^p \in \mathbb{D}_2$, after non-deterministically choosing interaction a is $\mu_1(x_i^p)\mu_2(y_j^p)$, while the probability of going to state $((l_2^1, l_3^2); (x_i^p, y_0^p, z_k^p))$, $x_i^p \in \mathbb{D}_1$ and $z_k^p \in \mathbb{D}_3$, after non-deterministically choosing interaction b is $\mu_1(x_i^p)\mu_3(z_k^p)$.

Example 3.8 (Stochastic Multimedia Stream Source)**.** This example depicts the full stochastic multimedia stream source model started in Example 3.6. The underlying model is a composition of three components consisting of the *Frames Generator* (see Example 3.6), a *Splitter*, and a *Transfer Media* component. The generated output stream is made of macro-blocks which are going to be stored in the input buffer of the decoding unit. The number of bits (the size) of every macro-block, together with a bit-rate parameter of the transfer media, determine the arrival time of the macro-block to the buffer. This number of bits is specific for each macro-block type (w.r.t frame types) and follows a specific probability distribution (μ_{mb}) parametrized by frame sizes.

Figure 3.10 shows the $\mathcal{S}$BIP component modeling this behavior. Frames are produced following the GOP pattern as explained earlier in Example 3.6. Each frame is split into 330 micro-blocks, having probabilistically distributed sizes, by the *Splitter* component. Each macro-block is then sent for storage through the *Transfer Media* component which model the arrival time of macro-blocks to the buffer.

The components are assembled together using two connectors $C1$ and $C2$ providing data transfer facilities. The first connector enables delivering frames to the *Splitter*, whereas the second ensures the transfer of the generated macro-blocks to the *Transfer Media* component. Finally, it is worth to mention that the *transfer* and *tick* ports of the *Transfer Media* component are made available as interfaces for the *Source* component to communicate with the rest of the video decoding system model presented later in the chapter.

Stochastic atomic components as well as their composition have an MDP semantics that encompasses non-deterministic and probabilistic decisions. However, to be able to perform quantitative analysis such as SMC on these models, it is required to resolve any non-determinism. In the following, we show how to reduce $\mathcal{S}$BIP models to produce a purely stochastic semantics.

3.2.3 Purely Stochastic Semantics

Building rich stochastic performance models and using SMC to quantitatively analyze them is the main motivation behind the proposed stochastic extension. To this end, one needs to produce a purely stochastic behavior where only probabilistic choices are possible. Given the $\mathcal{S}$BIP construction

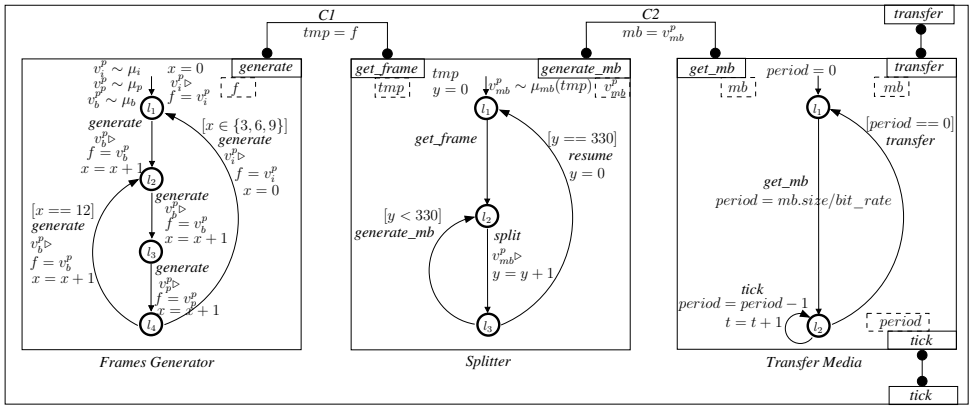


Figure 3.10: $\mathcal{S}$BIP component of an MPEG2 Stream Source.

above, we propose to use probabilistic schedulers in order to produce a purely stochastic LMC semantics of $\mathcal{S}$BIP models. We first define the semantics of a purely stochastic component. Then, adapt the definition of schedulers in Chapter 2 and provide definition of the induced LMC in this context.

In chapter 2, we introduced schedulers and shown that it yields Markov Chain semantics, when applied on MDPs. We build upon chapter 2 definitions to define schedulers for $\mathcal{S}$BIP models. We recall that non-determinism occurs in $\mathcal{S}$BIP models, whenever several interactions are simultaneously enabled. A scheduler, in this context, provides probability distributions to select among enabled interactions in global states as in Definition 2.8. Given a stochastic atomic component and a scheduler $\mathscr{S}$, the produced behavior is purely stochastic. It allows probabilistic choices between transitions in $Act(s)$ instead of non-deterministic choices. The resulting underlying behavior is thus a Markov Chain as given by Definition 2.9.

Let $\mathcal{B}^{\mathbf{s}} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$ be a stochastic atomic component and $\mathcal{M} = \langle S, Act, \iota, \pi, \Sigma, L\rangle$ the associated MDP semantics. In the Markov Chain $\mathcal{M}^{\mathscr{S}} = \langle S, \iota, \pi', \Sigma, L\rangle$ induced by the scheduler $\mathscr{S}$ on $\mathcal{B}^{\mathbf{s}}$, the probability of taking transition $s \longrightarrow s' \in \pi'$ corresponding to transition $s \xrightarrow{prt} s' \in \pi$, where $s = (l; X^p; X)$ and $s' = (l; X'^p; X') \in S$ and $\tau = (l, prt, g, (f^d, f^p), l') \in Act(s)$ is computed as follows. A probabilistic choice over enabled transitions in $Act(s)$ using the distribution provided by $\mathscr{S}$ is performed first, followed by a probabilistic choice using probability distributions attached to probabilistic variables as earlier, that is,

$$\pi'(s, s') = \sum_{s \xrightarrow{prt} s'} (\mathscr{S}(s, prt) \cdot \prod_{v^p \in f^p} \mu^{v^p}(x^{v^p}))$$

Example 3.9 (Induced Markov Chain by a Uniform Scheduler on the MDP Figure 3.7c)**.** Consider the $\mathcal{S}$BIP model and its associated MDP in Example 3.5 where non-determinism appears only on state $s_1 = (l_1; 1, 3, false)$,

where $Act(s_1) = \{prt_1, prt_2\}$. In order to resolve this non-determinism and produce a fully stochastic behavior, we may use a scheduler $\mathscr{S}$ that uniformly selects among enabled actions in each state. Hence, $\forall s \in S$, each $prt \in Act(s)$ has an equal probability to be selected. In this case,

$$\mathscr{S}(s_1, prt_i) = \frac{1}{|\mathsf{Enabled}(s_1)|} = \frac{1}{2}, \text{ where } 1 \leq i \leq 2.$$

Figure 3.11 depicts the induced Markov Chain. In this example, the difference between the MDP and the induced LMC is at the level of probabilities of transitions (no structural transformation has been operated). These are obtained by multiplying the probabilities of the MDP transition by the scheduler value, that is, $\frac{1}{2}$.

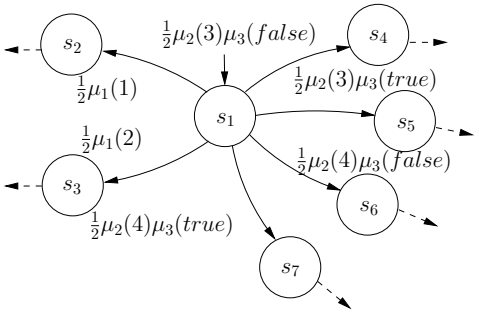


Figure 3.11: LMC induced by a uniform scheduler $\mathscr{S}$ on the MDP Figure 3.7c.

It is worth mentioning that defining probabilistic schedulers for $\mathcal{S}$BIP models that encompass non-deterministic behavior requires a bit of care. Let us consider a component $\mathcal{B}^{\mathfrak{s}}$ and its associated MDP semantics $\mathcal{M}$. Given a state $s \in S$, where three actions are possible $\{\varrho, \varsigma, \vartheta\}$, it is not sufficient to define a single probability distribution over them, but we need to specify a probability distribution for each possible $Act(s)$. Actually, the non-determinism implies that we don't have any apriori knowledge about which actions will be enabled each time, that is, $Act(s) \in \{\{\varrho, \varsigma, \vartheta\}, \{\varrho, \varsigma\}, \{\varrho, \vartheta\}, \{\varsigma, \vartheta\}, \{\varrho\}, \{\varsigma\}\}$, or $\{\vartheta\}$. This is tedious to do for each state of the model especially in for industrial-size models. In the current state of this work, we stick to uniform distributions which implicitly takes into account the number of enabled actions in $Act(s)$, i.e. $\forall \varrho \in Act(s), \pi(s, \varrho, s') = \frac{1}{|Act(s)|}$. Details on investigating possible representation of probabilistic schedulers with others probability distributions in $\mathcal{S}$BIP are discussed as perspective work in the conclusion chapter of the manuscript.

Formalism	*Composition*	*Semantics*
BIP	$\varpi\gamma(\mathcal{B}_1,\ldots,\mathcal{B}_n) \rightsquigarrow \mathcal{B}$	$\mathcal{B} \rightarrow$ LTS
$\mathcal{S}$BIP	$\varpi\gamma(\mathcal{B}_1^s,\ldots,\mathcal{B}_n^s) \rightsquigarrow \mathcal{B}^s$	$\mathcal{B}^s \rightarrow$MDP $\dashrightarrow_{\mathscr{S}}$ LMC

Table 3.1: Summary of the stochastic extension of the BIP formalism.

3.3 Performance Evaluation of a Multimedia SoC

Before going further with the $\mathcal{S}$BIP extension, it is important to consider a real-life example in order to concertize the previously developed theory for component-based stochastic systems modeling and analysis. To this end, we consider an abstraction of a multimedia SoC shown in Figure 3.12.

In the abstract SoC model, an input coded video stream is assumed to be initially stored in the SoC memory. It is then transferred, with a constant bit-rate, to an input buffer in terms of stream objects, such as macro-blocks. A pipeline of functional units process the input stream in a sequential fashion. Processed items are then temporarily stored in an output (also called play-out) buffer before being displayed by a video player. In this example, we assume that the multimedia SoC contains a single processing unit consisting of an MPEG2 video decoder.

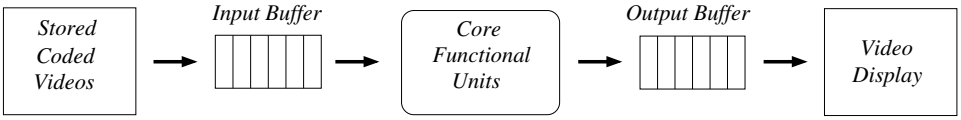


Figure 3.12: An abstract view of a multimedia SoC. The functional units of the decoder can include variable length decoding, motion compensation, etc.

The main challenge when designing such systems is to ensure a good trade-off between quality of service (QoS), usually expressed as soft real-time constraints, and resources usage, in order to keep the cost of the final product as low as possible. This is known as resource constrained or best-effort design. Such trade-off implies to tolerate certain quality degradation which will not jeopardize the final system. In the multimedia literature [207], acceptable video degradation is usually specified as: *less than two consecutive frame loss within 30 frames*, *less than* 17 *aggregate frame loss within* 100 *frames*, etc. Moreover, such systems exhibit an important uncertainty due to the high variability present in the input multimedia stream, in terms of number and complexity of items that arrive per unit time to the system, which justify a stochastic characterization.

Our goal in this example is to build a component-based stochastic model of the multimedia SoC (using the $\mathcal{S}$BIP formalism) and to analyze it (using Statistical Model Checking) towards reducing resource usage while guaran-

tying a tolerable loss in video quality. We will mainly focus on the first specification of tolerable video degradation, i.e., *less than two consecutive frame loss within 30 frames*, while trying to reduce the buffer sizes. A Frame loss is seen at the player end. The latter starts reading decoded items after an *initial delay* (also called *play-out delay*). Reads are then performed periodically from the output buffer. A loss happens when the player do not find the required items to display in a specific period. This is denoted as *buffer underflow* and can be interpreted as a deadline miss, that is, the decoding unit failed to provide the required items in time. Figure 3.13 illustrates such behavior.

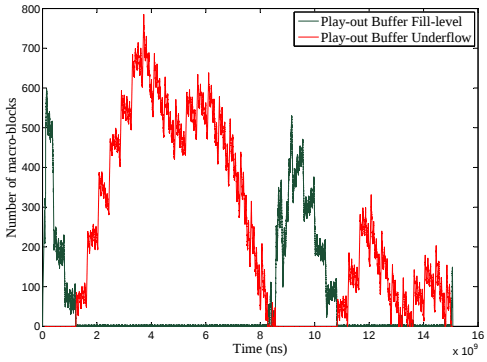


Figure 3.13: Evolution of the play-out buffer fill-level over time. It also shows the buffer underflow evolution. This is obtained by simulating the $\mathcal{S}$BIP model below with an initial delay equal to 75 ms.

Our goal is to estimate the probability that buffer underflow (denoted U) is less than two consecutive frames in 30 frames, given the following parameters of the multimedia SoC:

- A set of video clips of certain bit-rate (r) and resolution,
- The maximum frequency of the MPEG2 decoding unit (f),
- The consumption rate of the player device (c),
- The start-up values for the initial delay (d), input buffer size (ibs), and play-out buffer size (obs).

3.3.1 $\mathcal{S}$BIP Model of the Multimedia SoC

The $\mathcal{S}$BIP model of the multimedia SoC is obtained in a straightforward manner from the description above. It has a component for each part of the SoC shown in Figure 3.12. This model captures the stochastic behavior of the system in the following way. The coded object's arrival time to the input buffer follows defined probability distributions which are constructed from a given set of video clips (See Figure 3.14). Thus, the decoded object's arrival to the output buffer is also probabilistic, which leads to probabilistic

buffer underflow. Next, we explain how to estimate the probability of certain amount of buffer underflow.

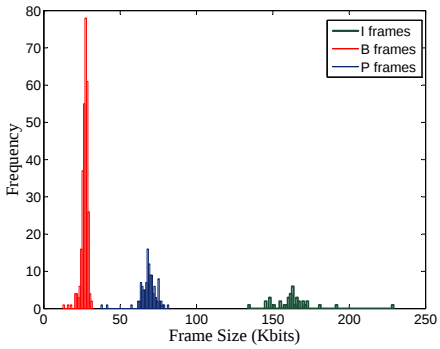


Figure 3.14: Frequency distributions of I,P, and B frames. I frames are larger in size but less frequent than the B and P frames.

Figure 3.15 shows the $\mathcal{S}$BIP model of the multimedia SoC, essentially performing video decoding tasks. It consists of three functional components, namely, Source, Decoder, and Player. The Decoder and the Source components were introduced in Example 3.2 (see Section 3.1) and Example 3.8 (see Section 3.2) respectively. The former is deterministic and models MPEG2 macro-blocks decoding, while the latter is stochastic and is made of three different components, respectively corresponding to GOP frames production, frames splitting into macro-blocks, and stochastic transfer time.

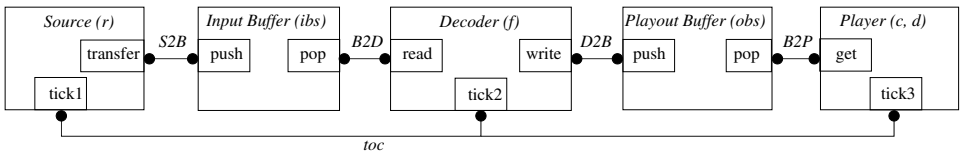


Figure 3.15: $\mathcal{S}$BIP model of the multimedia SoC with a single functional unit consisting of an MPEG2 decoder.

Functional components communicate through the Input Buffer and the Play-out Buffer, represented, in the model, as deterministic atomic components. Figure 3.16 shows a generic $\mathcal{S}$BIP model of a buffer component. It maintains an array of macro-blocks parametrized by a maximum capacity and updated through push/pop primitives executed when interacting with other components accordingly.

The SoC's components are assembled, in the usual way in $\mathcal{S}$BIP, using connectors $S2B, B2D, D2B, B2P$ which enable macro-blocks transfer between functional components and buffers. An additional *toc* connector is used to synchronize timed components (Source, Decoder, and Player). This explicitly models the progress of the absolute (global) time.

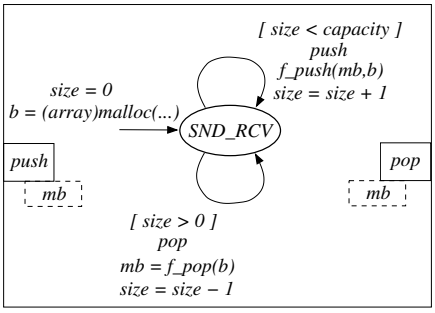


Figure 3.16: A generic $\mathcal{S}$BIP model of a buffer component. It has a capacity parameter which fixes the upper size limit and may be instantiated as input or output buffer.

The Player component models the display of the stream of decoded macro-blocks. After an initial play-out delay d, it starts reading macro-blocks from the play-out buffer at a constant rate c, as shown in Figure 3.17. A buffer underflow occurs whenever the requested number of macro-blocks is not available in the buffer. In this case, the request is postponed for the next iteration and the underflow is accumulated. For example, if the current buffer underflow is 2, then, at the next request, the Player seeks 3 macro-blocks. If the buffer is still empty, the underflow became 3. Else, if the play-out buffer has (at least) 3 items, then all three items are read at once and the buffer underflow is reset to 0, etc.

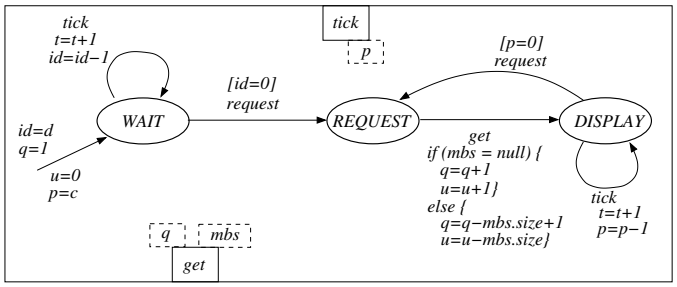


Figure 3.17: $\mathcal{S}$BIP model of the Player component. It has two input parameters, id and c corresponding respectively to the initial delay and the display rate.

It is important to remark that the designed components of the multimedia SoC encompass non-determinism. This is the case for the SoC components introduced in the previous sections (Source, and Decoder). For instance, we recall that the MPEG2 decoder component depicted in Example 3.2 has non-determinism between *read*, *write*, and *tick* transitions. Other components in the model, such as the Player above, have similar non-deterministic choices.

In order to be able to use SMC for analysis, we use priorities and rely on

a uniform scheduler to resolve such non-deterministic behavior. For instance the following priority model is used to enable components progress:

$$\varpi : toc < S2B,\ toc < B2D,\ toc < D2B,\ toc < B2P.$$

This gives high priority to decode macro-blocks and to transfer them, when enabled, over time progress.

3.3.2 QoS Requirements Evaluation

To evaluate QoS requirements of the multimedia SoC, we apply statistical model checking on the $\mathcal{S}$BIP model presented above. Before presenting the experiment details, it is worth to report some important observations with respect to the SoC model operation.

We previously stated that play-out buffer underflow is observed at the Player end, and that this component has two parameters c and d, respectively defining the initial delay after which it has to start reading from the buffer, and the periodicity of reading. Both parameters, together with the decoder frequency and the arrival time of macro-blocks to the input buffer, have an impact on buffer underflow. Actually, when the initial delay is high, and when both the decoder and the arrival time are not too slow, the play-out buffer will accumulate an important number of macro-blocks before the player starts reading [4], which reduces the probability of occurrence of underflow later on. In the opposite scenario, the probability of having buffer underflow becomes higher.

In the following experiments, the decoder frequency, the macro-blocks arrival time to the input buffer [5], as well as the periodicity of the Player are considered to be fixed. We will try to tune the remaining parameters, such that to obtain a good trade-off between the play-out buffer size and the probability of having two consecutive frames loss within 30 frames.

In order to estimate the probability of this requirement, we use an Observer which monitors the occurrence of the frames loss. This will run in parallel with the SoC components and reacts to relevant events related to the property of interest, that is, play-out buffer pops. We exploit BIP expressiveness to encode the observer behavior as an atomic component which will be connected to the $B2P$ connector between the play-out buffer and the Player. As shown in Figure 3.18, the Observer has three states: OK, PARTIAL, and FAIL. The FAIL state capture the fact that two consecutive frames within 30 have been lost. The component starts in the OK state. When a single frame is lost, the observer moves from state OK to PARTIAL. Later, if there is an additional frame loss, within the same window of 30 frames,

4. A very high initial delay requires an big play-out buffer size and may induce a buffer overflow. In this experiment, we are only focusing on buffers underflow.

5. This is defined by the bit-rate of the considered video clips.

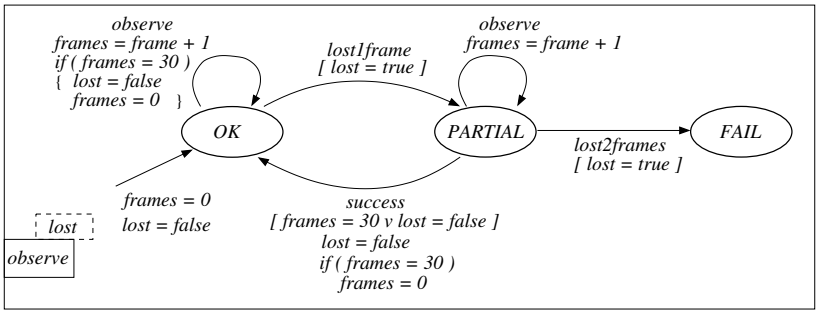


Figure 3.18: $\mathcal{S}$BIP model of the *Observer* component. It models the QoS property to be verified. The variable *frame* counts the number of frames to check if two consecutive loss occurs within a second (i.e. within 30 frames). The read port of the Observer is synchronized with the read port of the Player. A variable *lost* is associated with the read port records a frame loss.

the Observer moves to the FAIL state. Otherwise, when no additional losses happen within that widow, it moves back to the OK state.

Analysis and Results

This section sketches QoS requirement probabilities estimated using the statistical model checking technique. The experiments were conducted for low bit-rates and low resolution clips ($352 * 240$) obtained from an open source[6]. The bit-rate of the input videos is 1.5 Mbits per second and the frame output rate is 30 frames per seconds (fps). We used an MPEG2 implementation optimized for speed[7]. The MPEG2 source was annotated to get the number of bits corresponding to each compressed macro-block. The execution cycles for each macro-block is obtained from the processor simulator *SimpleScalar*[8]. We chose the video files `cact.m2v, mobile.m2v,` and `tennis.m2v` for our experiments.

Recall that we used the observer component to monitor events corresponding to two consecutive frames loss within 30 frames (which is equivalent to 1 second given the frame output rate of 30 fps). As explained earlier, this raises a flag denoted *lost* whenever it catches that event. Thus, we used the following BLTL property to query the SoC model with respect to the probability to never get two consecutive frames loss within a second.

$$\phi = \mathsf{G}^t(\neg lost)$$

We choose $t = 1500000$ steps, such that to display all the macro-blocks of each video clip. Note that all the considered clips are composed of 148500

6. ftp://ftp.tek.com/tv/test/streams/Element/index.html
7. http://libmpeg2.sourceforge.net
8. http://www.simplescalar.com/

macro-blocks, that is, 450 frames. Finally, we applied SMC with inputs the SoC model, the BLTL property ϕ and the confidence parameters $\delta = 5\ 10^{-2}$ and $\alpha = \beta = 10^{-2}$. We did measure the probability of ϕ for different play-out delay values and observed the play-out buffer fill-level evolution. We repeated the same experiment for the three video clips selected previously.

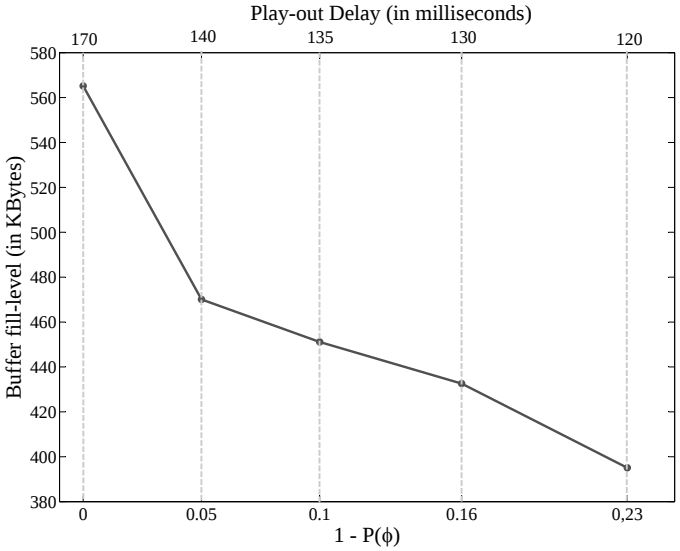


Figure 3.19: Play-out buffer fill-level and probability of ϕ for various play-out delay values for `cact.m2v`. For this video clip, the processor frequency is set to 109 Mhz.

Figures 3.19, 3.20, and 3.21 plots the results of the play-out buffer fill-level for various play-out delay values, and corresponding probabilistic bounds. In these figures, we report, in the X axis, the probability of eventually having two consecutive frames loss withing a second, i.e., $(1 - P(\phi))$. The play-out buffer fill-level and the play-out delay evolution are respectively depicted in the Y and Z axis (in the top).

Let us first focus on the impact of the play-out delay on the probability of ϕ. As expected, one can see that increasing the play-out delay reduces the probability of eventual underflow $(1 - P(\phi))$ for the different video clips. With respect to the buffer fill-level, we observe that it reduces substantially even for a small decrease of probabilistic value. For instance, there could be a buffer size reduction of 40% for an increase in the value of the probabilistic bound from 0 to 0.2 (Figures 3.20). In fact the buffer savings can be larger if we compare the buffer size required for no underflow and the memory required for the QoS property to be always true. It is worth mentioning that these results confirm our claim that, for tolerable degradation in video quality, output buffer size could be significantly reduced compared to the buffer size required for playing lossless videos.

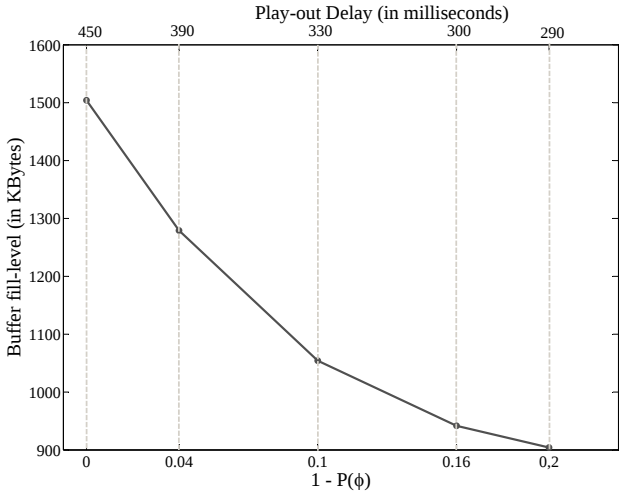


Figure 3.20: Play-out buffer fill-level and probability of ϕ for various play-out delay values for `mobile.m2v`. For this video clip, the processor frequency is set to 94 Mhz.

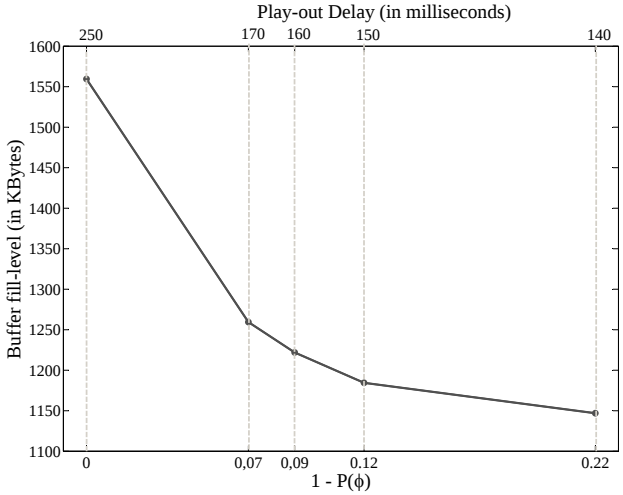


Figure 3.21: Play-out buffer fill-level and probability of ϕ for various play-out delay values for `tennis.m2v`. For this video clip, the processor frequency is set to 94 Mhz.

We report that for each probability estimation, the number of traces statistical model checking required ranged from 44 to 1345 and that for each trace, the method took around 6 to 8 seconds to verify the property, that is, in the worst situation it took around 3 hours to give the final verdict. The experiments were performed using BIP$^{\mathcal{SMC}}$, a statistical model checking engine for $\mathcal{S}$BIP models. This will be introduced with more details in Chapter 7, later in the dissertation.

In this example, we used the $\mathcal{S}$BIP formalism to model an abstraction of a multimedia SoC performing MPEG2 video decoding. We were able to capture the stochastic behavior of the SoC and to model it using stochastic components with realistic probability distributions of macro-blocks arrival time. These were obtained from a software MPEG2 decoder executed on the *SimpleScalar* simulator for concrete video clips. In the second part of the dissertation, in Chapter 6, we present a systematic approach for learning probability distributions characterizing performance information at system-level. QoS requirements analysis were performed using statistical model checking techniques. We were able to find good trade-offs between buffer sizes and acceptable video degradation in small time.

We illustrated the combined use of the $\mathcal{S}$BIP formalism and the statistical model checking technique to capture systems stochastic behavior and to perform quantitative analysis on a real-life example. Next, we show how the $\mathcal{S}$BIP formalism is able to capture any MDP or LMC behavior in a systematic way.

3.4 $\mathcal{S}$BIP Expressiveness

So far, we introduced the $\mathcal{S}$BIP stochastic extension for the BIP formalism and we showed that its underlying semantics is either an MDP or an LMC when using schedulers. In this section, we are interested in evaluating the expressiveness of our stochastic formalism. $\mathcal{S}$BIP inherits BIP expressiveness which is shown, not only theoretically [37, 38] but also in practice [29, 24, 23], to be sufficiently expressive to model a variety of systems in different application domains. In the sequel, we show how the $\mathcal{S}$BIP extension enables to capture more general stochastic models such as MDPs and LMCs.

3.4.1 Modeling LMCs in $\mathcal{S}$BIP

Given an LMC $\mathcal{M} = \langle S, \iota, \pi, \Sigma, L \rangle$, we build the corresponding stochastic atomic component $\mathcal{B}^{\mathfrak{s}} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$ by defining its set of control locations from the set of states of $\mathcal{M}$, where the initial location is simply the initial state of the LMC. The set of ports of $\mathcal{B}^{\mathfrak{s}}$ consists of two ports $\{sample, move\}$. The set of variables $\mathcal{V}$ is induced from the LMC states and their labels. The distributions of the probabilistic variables $\mathcal{V}^p$ are obtained from the probability transition function of the LMC. Finally, two type of transitions will compose the set $\mathcal{T}$ as formally defined hereafter.

Definition 3.12 (Syntactic Transformation of LMCs to $\mathcal{S}$BIP)**.** Given an LMC $\mathcal{M} = \langle S, \iota, \pi, \Sigma, L \rangle$, we define the syntactic transformation from $\mathcal{M}$ to a stochastic atomic component $\mathcal{B}^{\mathfrak{s}} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$ as follows:

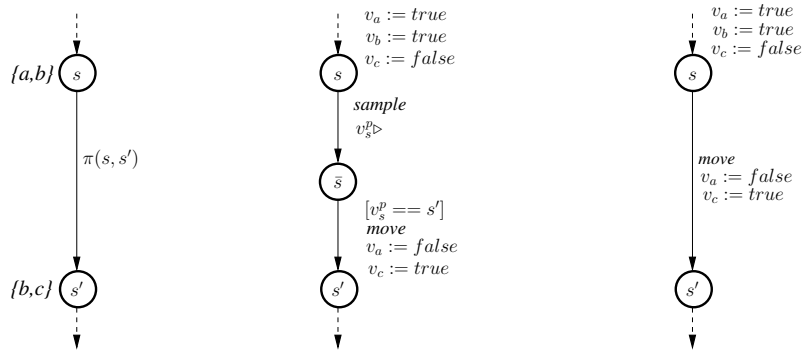
- $\mathcal{L} = S \cup \{\bar{s} \text{ for each } s \notin Det_{\mathcal{M}}(S)\}$,
- $l^0 = s_0$, such that $\iota(s_0) = 1$,

- $\mathcal{P} = \{sample, move\}$ is the set of ports,
- $\mathcal{V} = \mathcal{V}^d \cup \mathcal{V}^p$, where $\mathcal{V}^d = \{v_{ap} \mid ap \in AP\}$ and $\mathcal{V}^p = \{v_s^p \mid s \notin Det_{\mathcal{M}}(S)$ for each s, $\mu_s : S \to [0,1]$ and $\mu_s(s') = \pi(s,s')\}$,
- $\mathcal{T}$ contains transition of the form $(s, prt, g, (f^d, f^p), s')$, where $s, s' \in \mathcal{L}$, $prt \in \mathcal{P}$, g is defined over $Bool(\mathcal{V})$, and f^d and f^p are respectively the deterministic and probabilistic update functions, such that,

$$\frac{s \longrightarrow s' \in \pi,\ \pi(s,s') < 1}{(s, sample, true, (\emptyset, v_s^p), \bar{s}),\ (\bar{s}, move, v_s^p == s', (\rtimes(s,s'), \emptyset), s') \in \mathcal{T}} \quad (3.1)$$

$$\frac{s \longrightarrow s' \in \pi,\ \pi(s,s') = 1}{(s, move, true, (\rtimes(s,s'), \emptyset), s') \in \mathcal{T}} \quad (3.2)$$

where $\rtimes(s,s')$ is a function that assigns deterministic variables with their corresponding valuations in each state as follow: for all $s, s' \in S$, $\rtimes(s,s') = \{v_{ap} := x \mid (ap \in L(s) \wedge x = false) \vee (ap \in L(s') \wedge x = true)\}$.



(a) An LMC transition (b) $\mathcal{B}^{\mathbf{s}}$ transition (Rule 3.1) (c) $\mathcal{B}^{\mathbf{s}}$ transition (Rule 3.2)

Figure 3.22: Transformation rules of LMCs to $\mathcal{S}$BIP model.

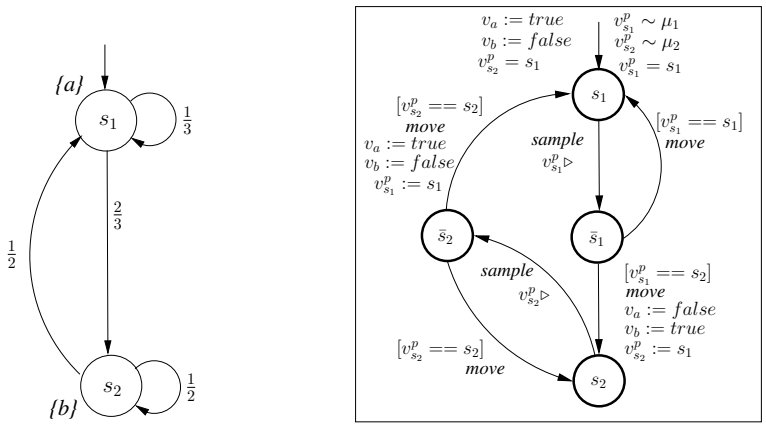
Intuitively, this transformation states that each transition $s \longrightarrow s' \in \pi$ having probability $\pi(s,s') < 1$ is associated, in the corresponding $\mathcal{B}^{\mathbf{s}}$ component, with two transitions as shown by rule (3.1), where:

- The first is a stochastic transition $(s, sample, true, (\emptyset, v_s^p), \bar{s})$ from location s to $\bar{s}$, where v_s^p is a probabilistic variable attached to a distribution μ_s obtained from π such as $\mu_s(s') = \pi(s,s')$.
- The second transition $(\bar{s}, move, v_s^p == s', (\rtimes(s,s'), \emptyset), s')$ is deterministic. It leads to location s' given a guard that checks if $v_s^p == s'$, that is, corresponding to the target state.

The second case considered by the transformation concerns transitions $s \longrightarrow s'$, where $\pi(s,s') = 1$. In such cases, a unique deterministic transition $(s, move, true, (\rtimes(s,s'), \emptyset), s')$ is produced in $\mathcal{B}^{\mathbf{s}}$ as specified by rule (3.2).

Remark 3.2 (LMCs with Several Initial States). Note that in the above definition, we only mentioned the case where $\iota(s_0) = 1$. An LMC may have a more general initial distribution than $\iota(s_0) = 1$, i.e., $\iota(s_i) = [p_i]$ for different states $s_i \in S$ and were $\sum_i p_i = 1$. In such a case, we use a transformation similar to rule (3.1). Concretely, this requires adding two control locations denoted l and $\bar{l}$, where l will be the initial control location of the corresponding $\mathcal{S}$BIP component, and a probabilistic variables v_l^p following the ι initial distribution. Then, we will have a *sample* transition from l to $\bar{l}$ which samples v_l^p, and several *move* transitions from $\bar{l}$ to initial states s_i having respective guards $[v_l^p == s_i]$.

Example 3.10 ($\mathcal{S}$BIP model of an LMC)**.** Let us consider the LMC in Figure 3.23a consisting of two states, s_1, the initial state, having label a and s_2 having label b. From s_1, two transitions are possible, leading to s_1 with probability $\frac{1}{3}$ and to s_2 with probability $\frac{2}{3}$. Similarly, from s_2, it is possible to stay in the same state with probability $\frac{1}{2}$ or to move back to s_1 with the same probability.



(a) An LMC Example (b) Corresponding stochastic atomic component

Figure 3.23: An example of transformation of an LMC to an $\mathcal{S}$BIP model.

We apply the transformation rules (3.1) and (3.2) above to build the corresponding stochastic component shown in figure 3.24b. It consists of a single stochastic component having four control locations $s_1, \bar{s}_1, s_2$, and $\bar{s}_2$, where s_1 is the initial control location. In the obtained component, probabilistic variables $v^p_{s_1} \sim \mu_1$ and $v^p_{s_2} \sim \mu_2$ model the probability transition function in states s_1 and s_2 of the LMC. These are defined over domains $\mathbb{D}_1 = \mathbb{D}_2 = \{s_1, s_2\}$. Moreover, deterministic Boolean variables v_a and v_b are used to model the labeling of the LMC states as follow. In the initial control location, we assign $v_a = true,\ v_b = false$, and $v^p_{s_1} = v^p_{s_2} = s_1$.

From location s_1, a *sample* transition associated with $v^p_{s_1}$ leads to $\bar{s}_1$ from which a *move* transition to location s_1 or s_2 is performed depending on the outcome of $v^p_{s_1}$ (s_1 or s_2). When moving to s_2, variables v_a is set to $false$, v_b is set to $true$, and $v^p_{s_2}$ is set to s_1. The same behavior is replicated from location s_2, that is, a *sample* transition associated this time with $v^p_{s_2}$ and leading to $\bar{s}_2$ is performed. This is followed by a *move* transition to s_1 or s_2 depending on the valuation of $v^p_{s_2}$ (s_1 or s_2). When moving to location s_1, v_a is set to $true$, v_b is set to $false$, and $v^p_{s_1}$ is set to s_1.

The $\mathcal{S}$BIP component $\mathcal{B}^{\mathfrak{s}}$ obtained by the construction in Definition 3.12 is an LMC $\mathcal{M}^{\mathcal{B}^{\mathfrak{s}}}$. While in general $\mathcal{S}$BIP models have an MDP semantics, relying on disjoint guards to capture the original model purely stochastic behavior, prevents any non-determinism and produces an LMC semantics.

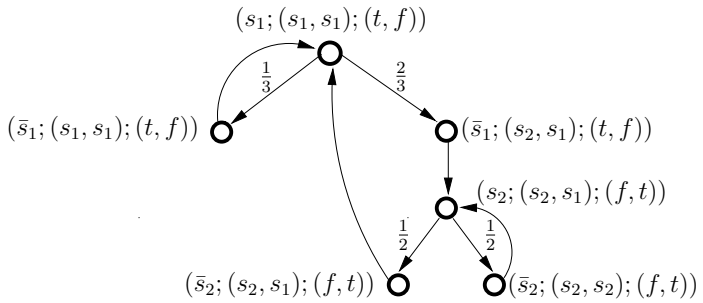
Obviously, the state space engendered by this component is bigger than the original state space of $\mathcal{M}$. This is due to locations $\{\bar{s} \in \mathcal{L}\}$, added for each state $s \notin Det_{\mathcal{M}}(S)$. Therefore, the state space of $\mathcal{M}^{\mathcal{B}^{\mathfrak{s}}}$ grows linearly when compared to the original state space. Nevertheless, $\mathcal{M}^{\mathcal{B}^{\mathfrak{s}}}$ is equivalent to $\mathcal{M}$, when observing the relevant set of atomic propositions.

To prove the statement above, let us first take a look to the paths generated by $\mathcal{M}^{\mathcal{B}^{\mathfrak{s}}}$. These are of the form $r = s \longrightarrow \bar{s} \longrightarrow s' \longrightarrow \ldots$, where $s = (l; X^p; X)$. The corresponding traces are of the form $\sigma = (X^p; X) \longrightarrow (X'^p; X) \longrightarrow (X'^p, X') \longrightarrow \ldots$, that is, a succession of two steps, the first is a sampling of probabilistic variables and the second is a deterministic update. Whereas probabilities of probabilistic steps are given by $\pi(s, s')$, the transition probability function of $\mathcal{M}$ as specified by the transformation above, deterministic steps have a probability one, hence they do not contribute on the probability of the generated transition.

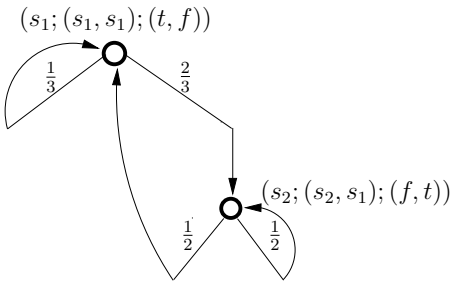
When considering Example 3.10 above, one should note that *sample* transitions from s to $\bar{s}$ induces two transitions (since $v^p_{s_1}$ is defined over $\mathbb{D}_1$). For instance, the *sample* transitions from s_1 to $\bar{s}_1$ induces transitions from state $(s_1; (s_1, s_1); (t, f))$ to states $(\bar{s}_1; (s_1, s_1); (t, f))$ and $(\bar{s}_1; (s_2, s_1); (t, f))$ respectively with probabilities $\frac{1}{3}$ and $\frac{2}{3}$. Similarly, the one from s_2 to $\bar{s}_2$ generates two transitions from state $(s_2; (s_2, s_1); (f, t))$ to states $(\bar{s}_2; (s_2, s_1); (f, t))$ and $(\bar{s}_2; (s_2, s_2); (f, t))$ with the same probability $\frac{1}{2}$. This behavior is shown in Figure 3.24a, which gives a clear idea about the type of traces that the $\mathcal{S}$BIP component may generate. For instance,

- $\sigma_1 = ((s_1, s_1); (t, f)) \xrightarrow{\frac{2}{3}} ((s_2, s_1); (t, f)) \longrightarrow ((s_2, s_1); (f, t)) \longrightarrow \ldots$, when $v^p_{s_1}$ evaluates to s_2 on transition *sample* from s_1 to $\bar{s}_1$, and
- $\sigma_2 = ((s_1, s_1); (t, f)) \xrightarrow{\frac{1}{3}} ((s_1, s_1); (t, f)) \longrightarrow ((s_1, s_1); (t, f)) \longrightarrow \ldots$, when $v^p_{s_1}$ evaluates to s_1 on the same transition.

Coming back to the equivalence statement. Given the generated traces, it suffices to specify the set of observable atomic propositions to be the one of $\mathcal{M}$. This is because building the corresponding stochastic atomic



(a) Behavior of the $\mathcal{S}$BIP component in Figure 3.24b.



(b) Collapsed behavior of the $\mathcal{S}$BIP component in Figure 3.24b.

Figure 3.24: Behavior of the obtained $\mathcal{S}$BIP component from an LMC.

component entails additional labels (represented by probabilistic variables), which are not relevant to observe. Furthermore, we do not observe states induced by locations $\{\bar{s} \in \mathcal{L}\}$. Concretely, given a trace $\sigma = (X^p; X) \longrightarrow (X'^p; X) \longrightarrow (X'^p, X') \longrightarrow \ldots \in Traces(\mathcal{M}^{\mathcal{B}^{\mathfrak{s}}})$, we only observe those labels corresponding to the state labels of $\mathcal{M}$ on locations $\mathcal{L} \setminus \{\bar{s} \in \mathcal{L}\}$. The obtained traces are thus $\sigma' = X \longrightarrow X' \longrightarrow \ldots$, where each transition have a probability $\pi(s, s')$ since the ignored deterministic steps does not contribute to the final probability. For instance, in the case of traces σ_1 and σ_2 shown earlier, we get $\sigma'_1 = (t, f) \xrightarrow{\frac{2}{3}} (f, t) \longrightarrow \ldots$ and $\sigma'_2 = (t, f) \xrightarrow{\frac{1}{3}} (t, f) \longrightarrow \ldots$, which are respectively equivalent to traces $\sigma_1^{\mathcal{M}} = a \xrightarrow{\frac{2}{3}} b \longrightarrow \ldots$ and $\sigma_2^{\mathcal{M}} = a \xrightarrow{\frac{1}{3}} a \longrightarrow \ldots$ generated by $\mathcal{M}$, the original LMC.

3.4.2 Modeling MDPs in $\mathcal{S}$BIP

In this section we show that $\mathcal{S}BIP$ models can also capture general MDP models. The transformation has the same philosophy as for LMCs, albeit slightly different because of non-determinism, which engenders in this case an additional step corresponding to the non-deterministic choice. Thus, given an MDP $\mathcal{M} = \langle S, Act, \iota, \pi, \Sigma, L \rangle$, the set of locations of the corresponding $\mathcal{S}BIP$ component $\mathcal{B}^{\mathfrak{s}} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$ includes $\{\tilde{s}\}$ in addition

to S and $\{\bar{s}\}$ previously. Moreover, locations, are identified using actions names when several actions are enabled from a given state. The set of ports of the component is obtained by the set Act of $\mathcal{M}$ plus additional $\overline{Act}$ and $\{sample\}$. Deterministic variables are obtained from the state labels as for LMCs, while probabilistic ones are induced from probabilistic distributions associated with the MDP actions. Finally, $\mathcal{B}^{\mathfrak{s}}$ transitions are of three types, corresponding respectively to a non-deterministic step (labeled with $\bar{\varrho} \in \overline{Act}$), followed by a probabilistic step (labeled with $sample$), and finally the actual move to the successor state (labeled with $\varrho \in Act$). Formally,

Definition 3.13 (Syntactic Transformation of MDPs to $\mathcal{S}$BIP)**.** Given an MDP $\mathcal{M} = \langle S, Act, \iota, \pi, \Sigma, L\rangle$, we define the syntactic transformation from $\mathcal{M}$ to a stochastic atomic component $\mathcal{B}^{\mathfrak{s}} = (\mathcal{L}, \mathcal{P}, \mathcal{T}, l^0, \mathcal{V})$ as follows:

- $\mathcal{L} = S \cup \{(\bar{s} \times Act) \text{ for each } s \notin Prob_{\mathcal{M}}(S)\} \cup \{(\tilde{s} \times Act) \text{ for each } s \notin Det_{\mathcal{M}}(S)\}$,
- $l^0 = s_0$, such that $\iota(s_0) = 1$,
- $\mathcal{P} = Act \cup \overline{Act} \cup \{sample\}$ is the set of ports,
- $\mathcal{V} = \mathcal{V}^d \cup \mathcal{V}^p$, where $\mathcal{V}^d = \{v_{ap} \text{ for each } ap \in AP\}$ and $\mathcal{V}^p = \{v^p_{s,\varrho} \text{ for each } s \notin Prob_{\mathcal{M}}(S) \text{ and } s \notin Det_{\mathcal{M}}(S) \mid \forall s' \ \varrho\text{-successor of } s,\ \mu_s : S \subset \mathbb{D} \to [0,1] \text{ and } \mu_{s,\varrho}(s') = \pi(s,\varrho,s')\} \cup \{v^p_s \text{ for each } s \in Prob_{\mathcal{M}}(S) \text{ and } s \notin Det_{\mathcal{M}}(S) \mid \forall s' \ \varrho\text{-successor of } s,\ \mu_s : S \subset \mathbb{D} \to [0,1] \text{ and } \mu_s(s') = \pi(s,\varrho,s')\}$,
- $\mathcal{T}$ contains transition of the form $(s, prt, g, (f^d, f^p), s')$, where $s, s' \in \mathcal{L}$, $prt \in \mathcal{P}$, g is defined over $Bool(\mathcal{V})$, and f^d and f^p are respectively the deterministic and probabilistic update functions, such that,

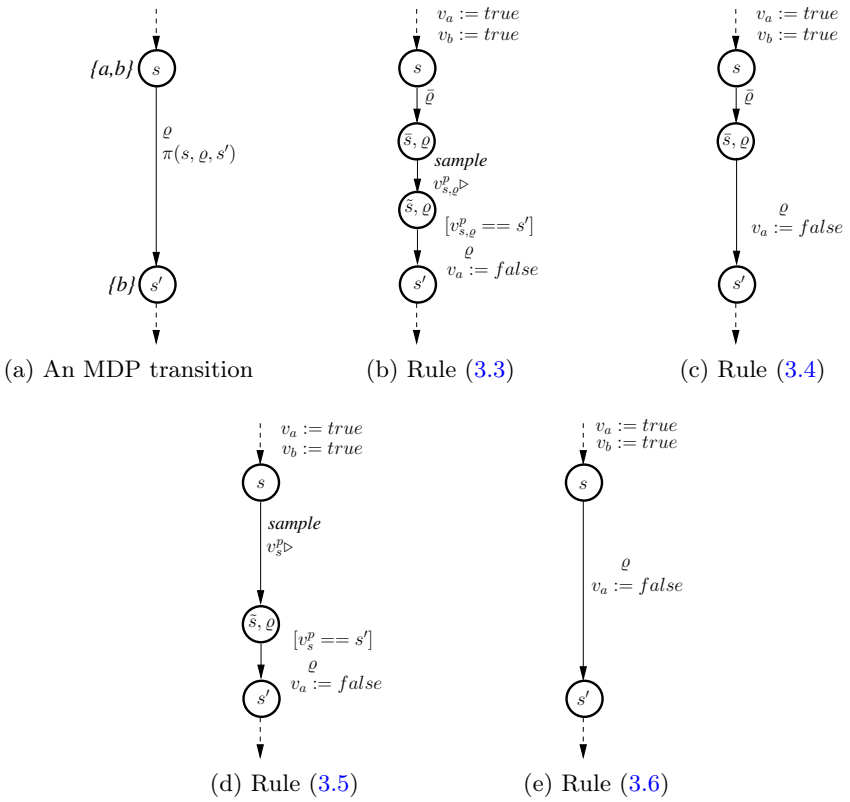
$$\frac{s \xrightarrow{\varrho} s' \in \pi, \text{ such that } |Act(s)| > 1 \text{ and } \pi(s,\varrho,s') < 1}{\begin{array}{r}(s, \bar{\varrho}, true, \emptyset, (\bar{s}, \varrho)), ((\bar{s}, \varrho), sample, true, (\emptyset, v^p_{s,\varrho}), (\tilde{s}, \varrho)),\\ ((\tilde{s}, \varrho), \varrho, v^p_{s,\varrho} == s', (\rtimes(s,s'), \emptyset), s') \in \mathcal{T}\end{array}} \tag{3.3}$$

$$\frac{s \xrightarrow{\varrho} s' \in \pi, \text{ such that } |Act(s)| > 1 \text{ and } \pi(s,\varrho,s') = 1}{(s, \bar{\varrho}, true, \emptyset, (\bar{s}, \varrho)), ((\bar{s}, \varrho), \varrho, true, (\rtimes(s,s'), \emptyset), s') \in \mathcal{T}} \tag{3.4}$$

$$\frac{s \xrightarrow{\varrho} s' \in \pi, \text{ such that } |Act(s)| = 1 \text{ and } \pi(s,\varrho,s') < 1}{\begin{array}{r}(s, sample, true, (\emptyset, v^p_s), (\tilde{s}, \varrho)),\\ ((\tilde{s}, \varrho), \varrho, v^p_s == s', (\emptyset, \rtimes(s,s')), s') \in \mathcal{T}\end{array}} \tag{3.5}$$

$$\frac{s \xrightarrow{\varrho} s' \in \pi, \text{ such that } |Act(s)| = 1 \text{ and } \pi(s,\varrho,s') = 1}{(s, \varrho, true, (\rtimes(s,s'), \emptyset), s') \in \mathcal{T}} \tag{3.6}$$

In the previous definition, one can distinguish two general cases with respect to the type of the transition to deal with in the given MDP. The first situation is when the transition label is the unique enabled action ϱ from state s, that is, $Act(s) = \{\varrho\}$. This corresponds to Rules (3.5) and (3.6) illustrated respectively in Figures 3.25d and 3.25e. In such case, we have almost the same rules for building an LMC transition (see Figure 3.22c and 3.22b), since there is no non-determinism. With the difference of the



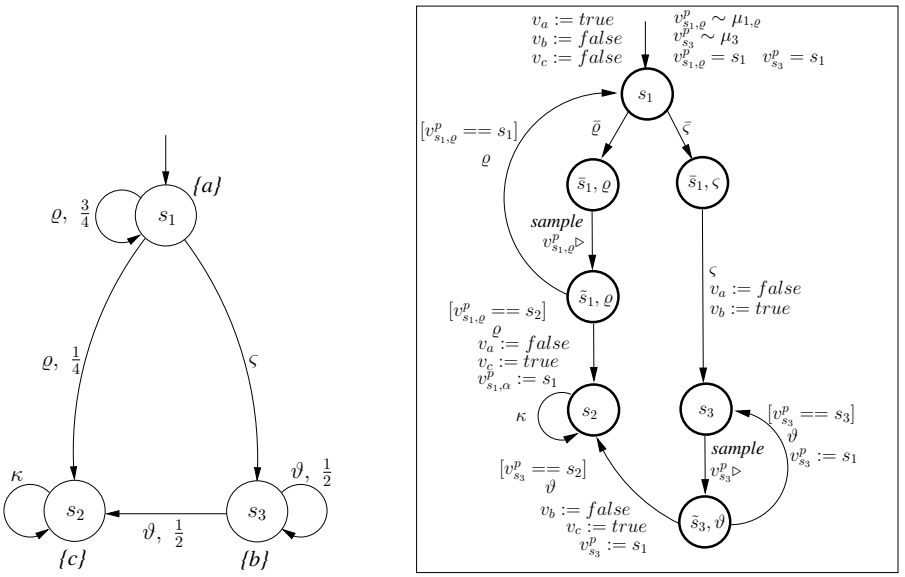
(a) An MDP transition (b) Rule (3.3) (c) Rule (3.4)

(d) Rule (3.5) (e) Rule (3.6)

Figure 3.25: Transformation rules of MDPs to $\mathcal{S}$BIP model.

ports names here given by the MDP actions. The second case is when the transition is labeled with an action, which is one among many other enabled from the corresponding state s, that is, $|Act(s)| > 1$. In this case, there are two possible situations. Depending whether the transition have a probability one (deterministic) or not, Rule (3.3) or (3.4) is applied. The first, have a non-deterministic step from s to $(\bar{s}, \varrho)$ followed by a probabilistic step from $(\bar{s}, \varrho)$ to $(\tilde{s}, \varrho)$, and finally a deterministic step from $(\tilde{s}, \varrho)$ to s', while the second does not have a probabilistic step and moves directly from $(\bar{s}, \varrho)$ to s' as shown in Figures 3.25b and 3.25c.

Example 3.11 (Transforming MDPs to $\mathcal{SBIP}$ Components)**.** Figure 3.26a shows an MDP $\mathcal{M}$ consisting of states $S = \{s_1, s_2, s_3\}$ labeled respectively by atomic propositions a, c, and b. Form s_1, the initial state, a non-deterministic choice exists between actions ϱ and ς, i.e., $Act(s_1) = \{\varrho, \varsigma\}$, where $\pi(s_1, \varrho, s_1) = \frac{3}{4}$, $\pi(s_1, \varrho, s_2) = \frac{1}{4}$ and ,$\pi(s_1, \varsigma, s_3) = 1$. From states s_2 and s_3, only purely probabilistic choice exists, i.e., $Act(s_2) = \{\kappa\}$ and $Act(s_3) = \{\vartheta\}$, where $\pi(s_2, \kappa, s_2) = 1$, $\pi(s_3, \vartheta, s_3) = \frac{1}{2}$, and $\pi(s_3, \vartheta, s_2) = \frac{1}{2}$.



(a) An MDP Example

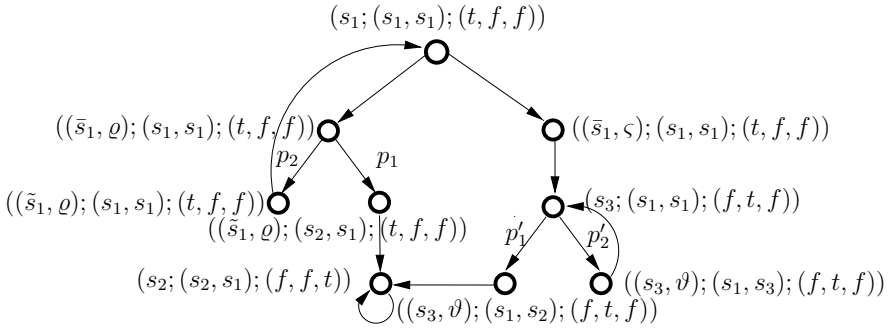
(b) Corresponding stochastic atomic component

Figure 3.26: An example of transformation of an MDP to an $\mathcal{S}$BIP model.

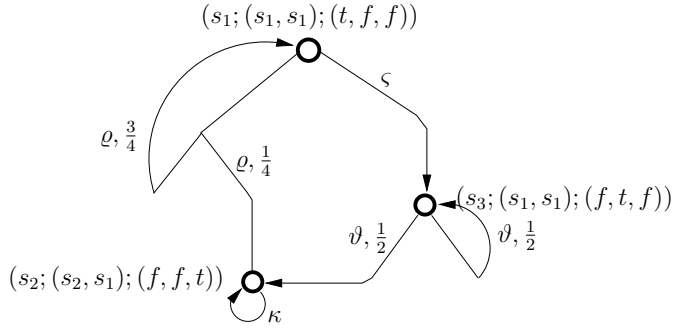
To build the $\mathcal{S}$BIP component corresponding to $\mathcal{M}$, we proceed stepwise. First, deterministic variables v_a, v_b, v_c are obtained directly from labels a, b, c of $\mathcal{M}$. Moreover, we will have two probabilistic variables $v^p_{s_1,\varrho} \sim \mu_{1,\varrho}$ corresponding to action ϱ from s_1, and $v^p_{s_3} \sim \mu_3$ corresponding to action ϑ from s_3. Note that the probabilistic variable $v^p_{s_1,\varrho}$ is identified using the action name in addition to the state name because $s_1 \notin Prob_{\mathcal{M}}(S)$, whereas, variable $v^p_{s_3}$ is only identified using the state name since $s_3 \in Prob_{\mathcal{M}}(S)$. Let us also remark that states $s_2 \in Prob_{\mathcal{M}}(S)$ and $s_1, s_2 \in Det_{\mathcal{M}}(S)$. The probabilistic variable $v^p_{s_1,\varrho}$ is defined over domains $\mathbb{D}_{1,\varrho}$ and $v^p_{s_3}$ is defined over domain $\mathbb{D}_3$. Transitions labeled with ϱ from s_1 are transformed using Rule (3.3) since $|Act(s_1)| > 1$ and $\pi(s_1, \varrho, s_1/s_2) < 1$ as shown in Figure 3.26b. In the same figure we can see that the transition $s_1 \xrightarrow{\varsigma} s_3$ is transformed using Rule (3.4) because $|Act(s_1)| > 1$ and $\pi(s_1, \varsigma, s_3) = 1$, transitions labeled with ϑ from s_3 are transformed using Rule (3.5) because $|Act(s_3)| = 1$ and $\pi(s_3, \vartheta, s_3/s_2) < 1$. Finally transition $s_2 \xrightarrow{\kappa} s_2$ is transformed using Rule (3.6) since $|Act(s_2)| = 1$ and $\pi(s_2, \kappa, s_2) = 1$.

As expected, the built $\mathcal{S}$BIP component $\mathcal{B}^5$ has an MDP semantics $\mathcal{M}^{\mathcal{B}^5}$. Similarly to LMCs, capturing MDPs behavior results on a bigger state space. Definition 3.13 shows that, besides locations corresponding to the original MDP states, additional locations are added for each state, in case of non-determinism as stipulated by the construction rules. Rule (3.3) induces three

locations for each state in the MDP enabling non-deterministic and probabilistic choices, while Rule (3.6) induces one location for each state enabling purely deterministic choice. Finally, Rules (3.4) and (3.5) induce two locations for each state in the MDP. All these cases engender a stochastic atomic component with linearly bigger state space. Nonetheless, several generated states does not bring useful information and thus will be ignored as for LMCs. Concretely, we only consider states induced by locations $\mathcal{L} \setminus \{\bar{s} \cup \tilde{s}\}$. Furthermore, we only observe deterministic variables corresponding to the MDP labels. Hence, we can follow the same procedure for LMCs and observe that $\mathcal{M}^{\mathcal{B}^{\mathfrak{s}}}$ is structurally equivalent to the original MDP $\mathcal{M}$.



(a) Corresponding stochastic atomic component behavior.



(b) Corresponding stochastic atomic component collapsed behavior.

Figure 3.27: Behavior of the obtained $\mathcal{S}$BIP component from an MDP.

For instance, when considering Example 3.11, the induced behavior of $\mathcal{B}^{\mathfrak{s}}$ in Figure 3.26b is given in Figure 3.27a. This shows the potential traces that may be generated when executing $\mathcal{B}^{\mathfrak{s}}$. For instance, given a run $(s_1;(s_1,s_1);(t,f,f)) \longrightarrow ((\bar{s}_1,\varrho);(s_1,s_1);(t,f,f)) \xrightarrow{\frac{3}{4}} ((\tilde{s}_1,\varrho);(s_1,s_1);(t,f,f)) \longrightarrow (s_1;(s_1,s_1);(t,f,f)) \longrightarrow \ldots$, the associated trace when observing all the states and variables is $\sigma = ((s_1,s_1);(t,f,f)) \longrightarrow ((s_1,s_1);(t,f,f)) \xrightarrow{\frac{3}{4}} ((s_1,s_1);(t,f,f)) \longrightarrow ((s_1,s_1);(t,f,f)) \longrightarrow \ldots$, whereas, if we only observe the spec-

ified set of states and variables we obtain $\sigma' = (t, f, f) \xrightarrow{\frac{3}{4}} (t, f, f) \longrightarrow \ldots$, which is equivalent to the trace $\sigma^{\mathcal{M}} = a \xrightarrow{\frac{3}{4}} a \longrightarrow \ldots$ of the original MDP.

3.5 Conclusions

The proposed stochastic extension of the BIP formalism enables building stochastic models in a component-based fashion, that is, in an iterative and incremental way as explained in [24]. It allows for capturing non-deterministic as well as purely stochastic behaviors by using priorities and schedulers.

We draw the reader's attention that in this work, we are not concerned with finding an optimal scheduler given certain requirements as in [108]. This is a separate problem that has its own challenges and is out of the scope of this dissertation. Our concern here is, to build a purely stochastic $\mathcal{S}$BIP model, and to use SMC to analyze it with respect to different properties of interest.

As stated in the beginning of the previous section, our stochastic extension inherits BIP expressiveness. We have shown how straightforward it is to model MDPs or LMCs using the $\mathcal{S}$BIP formalism. This have been already used in many case studies covering a multitude of domains such as randomized distributed protocols [165], automotive [147, 146], avionics [25, 27], many-cores embedded systems [163], multimedia [178], etc. Although, we recognize some limitations and possible amelioration discussed in more details in the conclusion of the dissertation. For instance,

- The size of the induced state space when modeling MDPs or LMCS in $\mathcal{S}$BIP is significant. However, this is not an issue in the context of this work since analysis using SMC do not require to build the whole state space and store it. Furthermore, in the next chapter, we propose a technique aiming to reduce state space size and hence to improve the performance of SMC.
- Uniform schedulers to resolve non-determinism is the default choice we made and the easiest to implement. Other alternatives enabling the use of different probability distributions is discussed in the perspective section of the conclusion of the dissertation.
- Currently, $\mathcal{S}$BIP allows only to capture discrete time models. Nonetheless, it may be extended to cover continuous time by allowing continuous data domains for example. We speculate that this will have a Generalized Semi Markov Process (GSMP) semantics.
- For now, we capture MDPs and LMCs behaviors in a single BIP component. What would be interesting is to capture it over several $\mathcal{S}$BIP components. This is quite challenging because it will need to synthesize the corresponding glue.

From the application perspective, one may ask where to find the probability distributions to be used in the model under construction. The origin of such distributions depends on the application domain of the system to be modeled. For instance, this work focuses on the design of manycore embedded systems. In Chapter 6, we present an approach based on statistical inference to learn probability distributions for the $\mathcal{S}$BIP model. Before reaching that point, in the next chapter, we introduce an abstraction technique aiming to improve statistical model checking scalability.

Chapter 4

Stochastic Abstraction

In the previous chapters, we recalled the probabilistic model checking (PMC) setting and we saw that it is recognized among the most successful approaches for quantitative analysis of stochastic models. We discussed the two main techniques that implement it (numerical and statistical) with more focus on SMC, which we are adopting in this work for system-level performance evaluation. These are shown to be more efficient handling bigger models, since relying on simulation and statistics.

In spite of its wide use and acceptance, probabilistic model checking techniques still encounters significant difficulties when used on real-life systems. First, the stochastic modeling of these systems might be extremely cumbersome. Actually, high expertise is generally required to produce any kind of meaningful formal model. For stochastic models, besides functional aspects, they must include stochastic information in form of probabilities. These are hardly available and usually incomprehensible by an average system designer.

Second, whenever such stochastic models exist, they can be very detailed and contain too much information than actually needed for analysis purposes. This is usually the case when stochastic system models are automatically generated from higher level descriptions, e.g., as part of various design flows [22]. For instance, in the case of SMC, Monte-Carlo simulation becomes problematic as individual simulation time (time to obtain a single execution trace) could be very long. Henceforth, it could not be possible to obtain any but only a limited number of traces and consequently, prevent the use of SMC techniques [1]. Moreover, it is worth mentioning that for verification of system-level properties, the observation of any such trace in detail is rarely needed. Most of the time, such properties are expressed in terms of few observable actions/states of the system while the remaining are completely irrelevant and can be safely ignored.

1. Similarly, numerical techniques could not be applied in such settings because exhaustive exploration will not be feasible due to size of the generated models.

In this chapter, we aim at improving the general applicability of probabilistic model checking, especially the SMC technique which will be used in the next part of the dissertation for system-level performance analysis. To this end, we propose the combined use of projection and machine learning techniques to automatically construct faithful abstractions of system models and therefore to overcome the issues discussed earlier. To decide on the appropriate level of abstraction to produce, we suggest using the properties under verification as a guide.

Nowadays, machine learning is an active field of research and learning algorithms are constantly developed and improved in order to address new challenges and new classes of problems (see [202] for a recent survey on grammatical inference). In our context, learning is combined with projection as follows. Given a property of interest and a (usually large) sample of partial traces generated from a concrete system (or system model), we first use projection to restrict the amount of visible information on traces to the minimum required to evaluate the property and then, use learning to construct an abstract, probabilistic model which conforms to the abstracted sample set. Under some additional restrictions detailed later in this chapter, the resulting model is a sound abstraction of the concrete model, in the sense that the probability to satisfy the property is approximately the same in the learned and the original model. Hence, it can be used to correctly predict/generate the entire abstract behavior of the system, in particular, as an input model for SMC.

The above approach has multiple benefits. First of all, the sample set of traces can be generated directly from an existing black-box implementation of the system, as opposed to a concrete detailed model. In many practical situations, such detailed system models simply do not exist and the cost for building them using reverse-engineering could be prohibitive. In such cases, learning provides an effective, automated way to obtain a model and to get some valuable insight on the system behavior. The use of projection is also mostly beneficial. In most of the cases, the complexity of the learning algorithms as well as the complexity of the resulting models are directly correlated to the number of distinct observations of traces. Moreover, under normal considerations, a larger alphabet requires a larger size for the sample set. Intuitively, the more complex the final model is, the more traces are needed to learn it correctly. Nevertheless, one should mention that a bit of care is needed to meaningfully combine projection and learning. That is, projection may change a deterministic model into a non-deterministic one, and henceforth has an impact on the learning algorithms needed for it.

We start the chapter by a brief reminder of some probabilistic modeling and learning techniques. Then, we detail the joint use of projection and learning as a method to compute an appropriate system abstraction for analysis. The convergence and the correctness of the proposed method are then discussed and assumptions are clearly stated. We finally show the concrete

use of the method on Herman's Self Stabilizing Algorithm with an increasing number of processes, before concluding the chapter and presenting a bench of related works.

4.1 Preliminaries

We first recall and define some important notions for the forthcoming development. We mainly recall the concept of determinism in the context of LMCs. We also provide a definition for probabilistic automata as an alternative model for stochastic systems. These are also the models that most of the learning algorithms were designed for. Finally, we give an overview of probabilistic learning techniques.

4.1.1 Additional Stochastic Models

Probabilistic Finite Automata

Probabilistic finite automata (PFA) represent an alternative for modeling probabilistic systems. They are defined similarly to LMC with the difference of having termination states.

Definition 4.1. (Probabilistic Finite Automata) The probability transition function π is now defined on $S \times S \cup \{\$\}$ and $\pi(s, \$)$ stands for the probability to terminate execution at state s. Henceforth, the associated notions of paths and traces correspond to finite paths and finite traces for an LMC. The probability of a finite path $r = s_0 s_1 ... s_n$ of a PFA is

$$P(r) = \iota(s_0) \cdot (\prod_{i=0}^{n-1} \pi(s_i, s_{i+1})) \cdot \pi(s_n, \$).$$

Deterministic PFA are defined similarly to DLMCs and are denoted as DPFA.

4.1.2 Probabilistic Learning Techniques

Learning probability distributions over traces is a hard problem [76] with potential applications in a wide range of domains, far beyond formal verification. Many methods have been proposed in the research literature and are continuously improved and challenged on learning research competitions [202]. The family of state merging techniques is one of the most successful nowadays. Intuitively, these techniques proceed by first constructing some large automata-based representation of the set of input traces and then progressively compacting them, by merging states, into a smaller automaton, while preserving as much as possible trace occurrence frequencies/probabilities. Different algorithms in this family can learn either DPFA models [54, 78, 77] or general PFA models [193, 180, 81].

The AAlergia Algorithm

AAlergia [153] is a state merging algorithm that exclusively learn deterministic models. It is a variant of the Alergia algorithm [54] proposed by Carrasco and Oncina in early nineties. Given a sample of traces, the algorithm proceeds in three steps, as depicted in Figure 4.1.

Step 1: it builds an intermediate representation, named *Frequency Prefix Tree Acceptor* (FPTA), which is a restricted form of DPFA that represents all the traces in the input sample and their corresponding frequencies (number of occurrence in the sample).

Step 2: based on a compatibility criterion parametrized by α_A (see [153] for details), it iteratively merges states of the FPTA having the same labels and similar probability distributions until reaching a compact DPFA.

Step 3: it transforms the obtained DPFA into a DLMC model by normalizing the probabilities of termination.

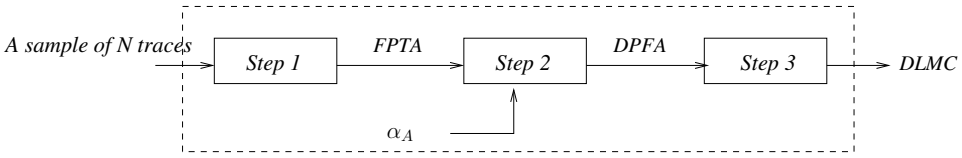


Figure 4.1: Main steps of the AAlergia algorithm.

AAlergia is proven to converge to the correct model in the limit [2] if the input traces are generated randomly, and with random lengths, from an LMC model [153]. A first consequence concerns verification on DLMCs and ensures that, in the limit, a given LTL property will hold on the original and the learned model with approximately the same probability. This result is partially extended to LMC. That is, for arbitrary Markov chain models, the algorithm might not converge to the good model in general. In the case of input traces from a non-deterministic LMC model (which moreover, does not have an equivalent deterministic representation), as the sample size increases, AAlergia will build a sequence of DLMCs (usually, of increasing size) tending to approximate the original model. It is proven that, in the limit, these learned DLMC models provide an increasingly better approximation for the initial (prefix) behavior, and hence preserve the satisfaction of bounded LTL properties.

2. When the sample size tends to be infinite, the learned model converges to the original system. This is known as *identification in the limit* and is widely used in grammatical inference [99].

4.2 Abstraction via Learning and Projection

The verification problem in the stochastic setting amounts to compute $P(\mathcal{M} \models \varphi)$ for an LMC model $\mathcal{M}$ and an LTL property φ. In several cases, $\mathcal{M}$ is not explicitly known, that is, it could be a black-box probabilistic system which can be executed arbitrarily many times in order to produce arbitrarily long traces. As stated in the introduction of the chapter, producing any stochastic system model is inherently difficult. Moreover, deciding on the appropriate level of abstraction of the model, that is, the amount of details to model/ignore, is an open question and is quite challenging. Finally, the verification complexity is proportional to the produced model size.

Due to these reasons, we would like to avoid the verification of φ on the original model $\mathcal{M}$. Instead, we would like to perform it on a smaller, abstract model $\mathcal{M}^\sharp$ which preserves the satisfaction probability of φ, that is, $P(\mathcal{M} \models \varphi) = P(\mathcal{M}^\sharp \models \varphi)$. We propose hereafter a method to compute such an abstraction $\mathcal{M}^\sharp$ by combining learning and a projection operator on traces parametrized by the property φ. The idea is based on the observation that, when checking a model against a property, only a subset of the propositions is really relevant. In fact, only the atomic propositions mentioned explicitly in the property are useful while the others can be safely ignored.

The proposed approach is depicted in Figure 4.2. It consists of initially generating a finite set of random finite traces T (with random lengths) from $\mathcal{M}$ (the sampling operation). In a second step, a projection is applied on traces T in order to restrict the atomic propositions to the ones needed for the evaluation of the property φ. The projection is guided by the property φ as detailed below. Third, the set of projected traces is used as an input to a learning algorithm. For the experiments, we will use the AAlergia algorithm [153], however, any other algorithm could be used. The output of the learning step denoted $\mathcal{M}^\sharp$ on Figure 4.2 will be used to evaluate the property of interest φ.

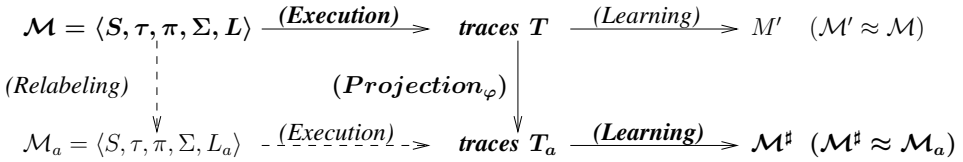


Figure 4.2: Illustration of our approach for learning abstract models.

It is worthwhile to mention that, in our approach, the sampling step (which could be time consuming depending on the used sample size) is done only once, while the following steps (projection and learning) are repeated given different properties[3]. Our approach ensures a significant time reduc-

3. This is not always need as will be shown later. Basically, we redo projection only for those classes of properties that have different sets of symbols.

tion with respect to applying SMC directly on the black-box system since it generally requires trace re-sampling every time. In [188], a method is proposed to avoid re-sampling in a slightly different setting, but it raises confidence issues as discussed in the related work section at the end of the chapter. In addition, some SMC algorithms, besides their termination guarantee, might potentially need a huge number/length of traces depending on the required confidence level.

The proposed approach is sound under some conditions. Note that a projection may potentially introduce non-deterministic behavior at the level of traces. Consequently, we need to distinguish several cases. The first one is when the traces are generated from a DLMC and the projection operation does not introduce any non-determinism. In this case any learning algorithm, for instance AAlergia, works. Another case is when the traces are generated from an LMC and/or the projection introduces non-determinism. This case is divided into two sub-cases depending on the type of non-determinism. If the non-deterministic model has an equivalent deterministic one, then any learning algorithm can be used. Otherwise, one needs to use learning algorithms capable to learn non-deterministic models such as [193, 81].

Next, we detail the main steps of the approach and illustrate them on the Craps Gambling Game example. The correctness is formally established by Theorem 4.1.

4.2.1 Abstraction Steps

In this subsection, we illustrate the different steps of the abstraction approach on the Craps Gambling Game model presented in Chapter 2. We first define the projection operation and depict the learning phase using the AAlergia algorithm. We then show the learned models obtained for two example properties. Finally, we provide verification results on the obtained models using SMC.

Projection

The projection operation is defined on sample traces generated from the stochastic system, so as to reduce the number of labels and henceforth, later on, the number of states in the learned model. In this work, we introduce a first syntactic definition of a projection operator. It intuitively consists of ignoring the atomic propositions that are not relevant to the property under verification as formally defined below.

Definition 4.2. Given a property φ, we define $V_\varphi \subseteq AP$ the support of φ as the set of atomic propositions occurring explicitly in φ.

Definition 4.3. The projection operator $\mathcal{P}_\varphi : \Sigma^* \to \Sigma^*$ is defined as

$$\mathcal{P}_\varphi(\sigma_0\sigma_1...\sigma_n) = \sigma'_0\sigma'_1...\sigma'_n$$

where $\sigma'_i = \sigma_i \cap V_\varphi$ for all $i \in [0, n]$.

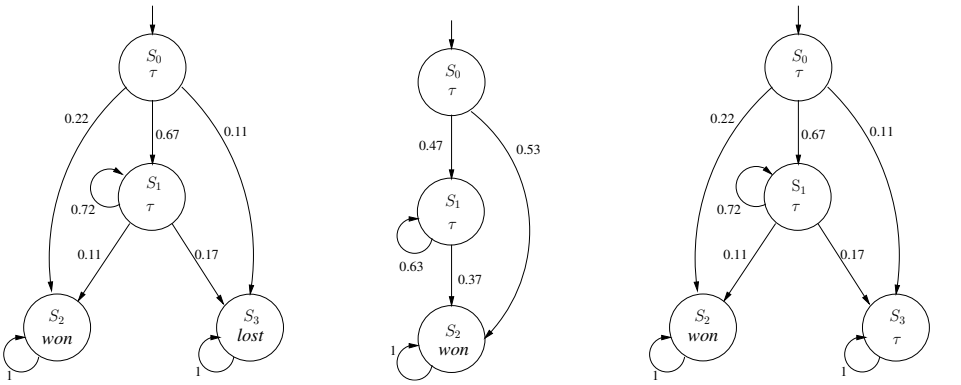
Given the definitions above, one can easily see that for two different properties φ_1 and φ_2, the projection operation may produce the same set of traces, if they have the same support, that $V_{\varphi_1} = V_{\varphi_2}$. We define a class of properties as the set of properties that have the same support. For such properties, the projection and learning steps are required once. Furthermore, if the support of φ_1 is only a subset of φ_2 ($V_{\varphi_1} \subset V_{\varphi_2}$), then the learned model given φ_2 may be used to verify both properties.

Example 4.1 (Projection on the Craps Gambling Game Traces)**.** Given a set T of traces generated from the Craps Gambling Game model in Figure 2.1 and the properties $\varphi_1 = \mathsf{F}\ won$ and $\varphi_2 = \mathsf{F}\ (won \vee lost)$, we apply Definitions 4.2 and 4.3 respectively to compute the corresponding supports $V_{\varphi_1} = \{won\}$, $V_{\varphi_2} = \{won, lost\}$ and the sets of projected traces T_{a_1} and T_{a_2} below.

- $T = \{$*start won, start lost lost, start won won won won won won won won won, start point5, start point10 point10 point10 point10 point10, start point9 point9,* $\ldots\}$;
- $T_{a_1} = \{\tau$ *won,* $\tau\ \tau\ \tau$, τ *won won won won won won won won won,* $\tau\ \tau$, $\tau\ \tau\ \tau\ \tau\ \tau\ \tau$, $\tau\ \tau\ \tau$, $\ldots\}$;
- $T_{a_2} = \{\tau$ *won,* τ *lost lost,* τ *won won won won won won won won won,* $\tau\ \tau$, $\tau\ \tau\ \tau\ \tau\ \tau\ \tau$, $\tau\ \tau\ \tau$, $\ldots\}$

Learning

We briefly illustrate the learning phase using AAlergia on the running example. Figure 4.3 shows three abstract models of the Craps Gambling Game obtained from the set T of 5000 traces generated from the model in Figure 2.1 (partially presented in Example 4.1). One can note from this figure the important reduction of the sizes of the obtained models with respect to the original one. Figure 4.3a shows the model learned by AAlergia taking as input the set T_{a_2}, that is, with respect to property $\varphi_2 = \mathsf{F}\ (won \vee lost)$. Figure 4.3b is obtained by applying AAlergia on the set T_{a_1}, that is, projected with respect to $\varphi_1 = \mathsf{F}\ won$. Remark that this model is not equivalent but only an approximation of the original model in Figure 2.1. That is, in the latter there exists some non null probability to never reach the *won* state, whereas, in the learned model the *won* state is reachable with probability 1. This approximation could however improve if a larger set of traces is used for learning as stated in the previous section. Finally, the third learned model shown in Figure 4.3c is equally obtained from T_{a_1} but when using an algorithm able to learn non-deterministic models such as the one in [193].



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 4.3: Learned Abstract Models of the Craps Gambling Game. 4.3a and 4.3b are obtained using AAlergia, while 4.3c is obtained using [193]. For all of them, we used 5000 traces generated from the model in Figure 2.1 and performed projection with respect to properties φ_1 and φ_2.

Abstract Model Analysis

Once the model is learned, we evaluate the property used for projection on the obtained abstraction. Table 4.1 provides results of verifying the property $\varphi_1 = \mathsf{F}\ won$ on the obtained Craps Gambling Game models. It shows that the model in Figure 4.3a exhibits similar probability to the original Craps Gambling Game model, whereas the one in Figure 4.3b shows a different result. The reason is that the projection performed in this case introduced non-determinism in the input sample. In addition, it seems that in this case there is no equivalent deterministic model that could be learned by AAlergia. Finally, note that the model in Figure 4.3a is obtained with respect to φ_2 having the support $V_{\varphi_2} = \{won, lost\}$ and that we were able to use it to verify φ_1 since $V_{\varphi_1} = \{won\} \subseteq V_{\varphi_2}$ as explained earlier.

Models	$P(\varphi_1)$
Scenario 1 (Figure 4.3a)	0.485
Scenario 2 (Figure 4.3b)	1
Original Model (Figure 2.1)	0.493

Table 4.1: Verification results of φ_1 on the original and the abstract Craps Gambling Game models using SMC (PESTIM).

4.2.2 Correctness Statement

The correctness of our approach is formally stated as follows.

Theorem 4.1. *Let $\mathcal{M}$ be an LMC model and let φ be an LTL property. Let $\mathcal{M}^\sharp$ be the learned model from a sample set of size n of traces generated from $\mathcal{M}$ and projected according to φ as in Definition 4.3. Then, in the limit, $\mathcal{M}^\sharp$ is a correct abstraction for the verification of φ, that is*

$$P(\lim_{n\to\infty} P(\mathcal{M} \models \varphi) = P(\mathcal{M}^\sharp \models \varphi)) = 1$$

if either

i) φ belongs to the bounded fragment BLTL and the learning algorithm converges for DLMC models, or

ii) the learning algorithm converges for arbitrary LMC models.

Proof. First, let us remark that $\mathcal{M}^\sharp$ is constructed as illustrated by the thick line in Figure 5.1. Let us moreover observe that any sample set of projected traces T_a obtained from $\mathcal{M}$ is equally obtained from $\mathcal{M}_a$, that is, from the "abstracted" version of $\mathcal{M}$ where only the labeling function has changed from L into L_a by taking $L_a(s) = L(s) \cap V_\varphi$, for all $s \in S$. In other words, the left-hand side of the diagram shown in Figure 5.1 commutes. Henceforth, $\mathcal{M}$ and $\mathcal{M}_a$ are identical with respect to the satisfaction of φ. The underlying set of runs and their associated probabilities are the same in $\mathcal{M}$ and $\mathcal{M}_a$. As the atomic propositions occurring in φ are preserved by relabeling, it obviously holds that $P(\mathcal{M} \models \varphi) = P(\mathcal{M}_a \models \varphi)$.

Moreover, learning from the sample set T_a leads eventually to $\mathcal{M}_a$. That is, under particular restrictions specific to the learning algorithms and limit conditions, the learned model $\mathcal{M}^\sharp$ will be an equivalent representation of $\mathcal{M}_a$, that is, $\mathcal{M}^\sharp \approx \mathcal{M}_a$. We distinguish two cases depending on the learning algorithm:

i) In the case of deterministic models learning (e.g., AAlergia), the learned model $\mathcal{M}^\sharp$ is provable equivalent only for a deterministic input model $\mathcal{M}_a$. But, in addition, for the general case, this models is also providing good approximations for the initial (prefix) behavior of $\mathcal{M}_a$ and hence preserve the probability of satisfaction for properties in the BLTL fragment (see Theorem 3 in [153]). Thereof, by using AAlergia or a similar learning algorithm, it holds that $P(\lim_{n\to\infty} P(\mathcal{M}_a \models \varphi) = P(\mathcal{M}^\sharp \models \varphi)) = 1$ whenever φ belongs to BLTL.

ii) In the general case of non-deterministic models learning, it is guaranteed in the limit that $\mathcal{M}_a \approx \mathcal{M}^\sharp$. Thereof, one can safely conclude that $P(\lim_{n\to\infty} P(\mathcal{M}_a \models \varphi) = P(\mathcal{M}^\sharp \models \varphi)) = 1$ for any φ.

Henceforth, in both cases it holds that

$$P(\lim_{n\to\infty} P(\mathcal{M} \models \varphi) = P(\mathcal{M}^\sharp \models \varphi)) = 1 \tag{4.1}$$

□

Table 4.2 summarizes the key results of the abstraction approach. The $\star$ symbol identifies the case where the used learning algorithm is only able to

learn DLMCs (such as AAlergia) and when $\mathcal{M}$ is non-deterministic (either inherently or because of projection) and φ is an LTL property. In general, equality 4.1 does not hold in this scenario. However, when there exists an equivalent deterministic model to $\mathcal{M}$, this may hold.

	Learning Algorithm			
	Converge to a DLMC $\mathcal{M}^\sharp$			*Converge to an LMC* $\mathcal{M}^\sharp$
$\mathcal{M}$	DLMC	*LMC*		(D)LMC
φ	(B)LTL	BLTL	LTL	(B)LTL
Equality 4.1	✓	✓	$\star$	✓

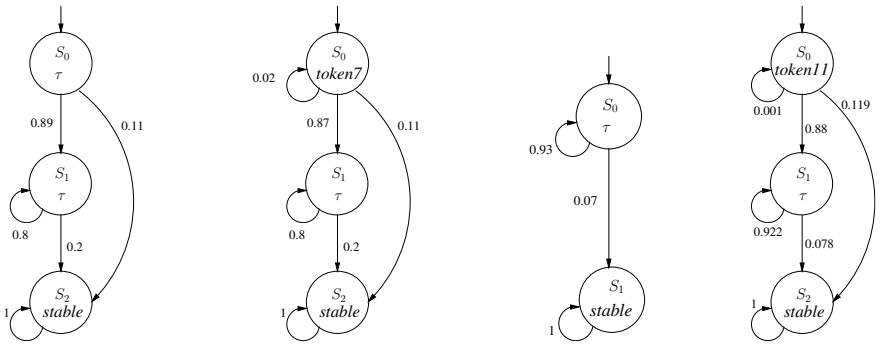
Table 4.2: Summary of the possible settings and the corresponding results.

4.3 Herman's Self Stabilizing Algorithm

To experiment our approach, we use the Herman's Self Stabilizing Protocol [111]. The goal of such a protocol is to perform fault tolerance by enabling a distributed system starting in an arbitrary state to converge to a legitimate one in a finite time. Given a token ring network where the processes are indexed from 1 to M (M must be odd) and ordered anticlockwise, the algorithm operates synchronously. Processes can possess tokens, which circulate in one direction around the ring. At every step, each process with a token tosses a coin. Depending on the outcome, it decides to keep it or to pass it on to the right neighbor. When a process holds two tokens, they are eliminated. Thus, the number of tokens remains odd. The network is said to be stable if exactly one process has a token. Once such a configuration is reached, the token circulates forever, fairly, around the ring.

We apply our abstraction approach to several configurations of the protocol ($M = \{7, 11, 19, 21\}$). Note that as the number of processes increases, the state space becomes very large and makes the verification quite heavy even using simulation-based methods such as SMC. We use AAlergia for learning and show that we are able to reduce the state space while still accurate for several properties. We consider the bounded properties $\varphi^L = P(true\ \mathsf{U}^L\ stable)$ and $\psi^L_M = P(tokenM\ \mathsf{U}^L\ stable)$ where M is the number of processes in the network and L is a bound. The first property states that the protocol reaches the *stable* state in L steps whatever the intermediate ones are. The second specifies in addition that the protocol directly moves from M tokens to the *stable* state (1 token), that is, all the states before *stable* are *tokenM*. We first apply the projection on the traces generated from the different configurations using the properties supports $V_{\varphi^L} = \{stable\}$ and $V_{\psi^L_M} = \{tokenM, stable\}$. Then, we use AAlergia to learn the corresponding models shown in Figure 4.4. The models obtained

for $M = \{19, 21\}$ are similar to $M = 11$. Finally, we verify the obtained abstractions with respect to different instances of the properties φ^L and ψ^L_M, that is, for various values of L and M.



(a) Given φ^L (b) Given ψ^L_7 (c) Given φ^L (d) Given ψ^L_{11}

Figure 4.4: Abstract Herman's Self Stabilizing Protocol models learned using AAlergia with respect to φ and ψ, for $M = 7$ (a,b) and $M = 11$ (c,d).

Table 4.3 summarizes the learned models characteristics, AAlergia performance when combined with projection, and verification results using PRISM [139]. In this table, the first two columns list the used configurations and their corresponding sizes. The third column depicts the properties under consideration. Information about the learning process are then detailed: α_A is the AAlergia compatibility criterion parameter, *Size* is the learned model size, and *Time* is the learning time in seconds. The last part concerns the comparison of the original and the learned model in term of properties probabilities and verification time. The verification part relies on the PESTIM algorithm which is parametrized by two confidence parameters δ and α.

The results in Table 4.3 point out two important facts. The first is the drastic reduction of the learned models sizes and SMC time compared to the original Herman's Self Stabilizing protocol model. Figure 4.5 summarizes the SMC time of φ^{10} for the learned and the original models when increasing the number of processes M. Figure 4.5 and the Table 4.3 allow us to see how big is the SMC time of the original model with respect to the abstract one. Figure 4.5 shows in addition the learning time which is also far below the SMC time of the original model for $M > 11$. Moreover, one can see that the time to learn plus the time to verify the abstract model is below the SMC time of the original model for $M > 11$, which confirms the pertinence of our approach for big models. For instance, for $M = 19$, learning takes about 83 seconds and verification takes about 0.307 seconds while verifying the original model takes about 13 hours. Furthermore, since the sampling step is done only once in our approach, its time impact is reduced when considering many properties. The second fact shown in Table 4.3 is that,

	Size	*Prop.*	*Learning*			*SMC*				
						δ, α	*Learned Model*		*Original Model*	
			α_A	*Size*	*Time(s)*		*Prob.*	*Time(s)*	*Prob.*	*Time*
$M = 7$	2^7	φ^{10}	$[2^{-9}, 2^0]$	3	69.70	$10^{-2}, 10^{-1}$ $10^{-2}, 10^{-2}$	0.874 0.880	0.180 0.320	0.874 0.873	3.40 *s* 5.44 *s*
		ψ^{30}	$[2^{-6}, 2^6]$	3	45.98	$10^{-2}, 10^{-1}$ $10^{-2}, 10^{-2}$	0.112 0.109	0.050 0.111	0.112 0.111	0.93 *s* 1.51 *s*
		ϕ	$[2^{-8}, 2^0]$	4	167.50	–	0.160	0.005	0.167	0.02 *s*
		Sample Size = 5000								
$M = 11$	2^{11}	φ^{10}	$[2^{-4}, 2^6]$	2	54.67	$10^{-2}, 10^{-1}$ $10^{-2}, 10^{-2}$	0.517 0.518	0.250 0.440	0.543 0.543	33.1 *s* 58.3 *s*
		ψ^{30}	$[2^{-6}, 2^6]$	3	60.22	$10^{-2}, 10^{-1}$ $10^{-2}, 10^{-2}$	0.011 0.012	0.039 0.070	0.012 0.011	12.1 *s* 21.7 *s*
		Sample Size = 5000								
$M = 19$	2^{19}	φ^{10}	$[2^{-4}, 2^6]$	2	82.95	$10^{-2}, 10^{-1}$ $10^{-2}, 10^{-2}$	0.197 0.191	0.180 0.307	0.148 0.151	8.1 *h* 13.3 *h*
		ψ^{30}	$[2^{-6}, 2^6]$	3	172.58	$10^{-2}, 10^{-1}$ $10^{-2}, 10^{-2}$	0.000 0.000	0.040 0.074	0.0001 0.0008	5.7 *h* 10.1 *h*
		Sample Size = 10000								
$M = 21$	2^{21}	φ^{10}	$[2^{-10}, 2^0]$	3	253.71	$10^{-2}, 10^{-1}$ $10^{-2}, 10^{-2}$	0.169 0.163	0.355 0.616	0.172 –	34 *h* > 5 *d*
		Sample Size = 10000								

Table 4.3: Abstraction and verification results of φ^{10} and ψ^{30} using PESTIM.

besides this reduction, the models are quite accurate in terms of probability measures as clearly shown in Figures 4.6, 4.7a, and 4.7b as well. These figures show the verification results of φ^L (for different L) on the original protocol versus the learned model for all the considered configurations. For example, for $M = 21$, the probability to satisfy φ^{10} is 0.172 for the original model and 0.169 for the learned one, that $3.\,10^{-3}$ of inaccuracy.

It is worth to mention that the results for $M = 21$ shown in Figure 4.7b are obtained using the PESTIM algorithm with confidence $\delta = 5.10^{-2}, \alpha = 5.10^{-3}$. That is, with less confidence when compared to the results in Table 4.3. This choice is made to reduce the experiments time since the goal of the figure is to show the global behavior of the learned model with respect to the original one. It was sufficient in this case to take low confidence level to get very similar shapes.

In addition to PESTIM, we used the SPRT technique to validate with more confidence the results of the property $\psi_M^L = P(tokenM \; \mathsf{U}^L \; token1) \geq \theta$ for $M = 7, 11$. We fixed the confidence parameters to $\alpha = \beta = 10^{-3}$ and $\delta = 10^{-3}$. Table 4.4 shows the verification results and performance of the SMC algorithm (verification time and number of traces) for different L values. Note that for this experiment, we used the same model learned previously. In this table, θ is the probability range to satisfy ψ_M^L, *Traces* is the number of traces used by SPRT, and *Time* is the SMC time. This

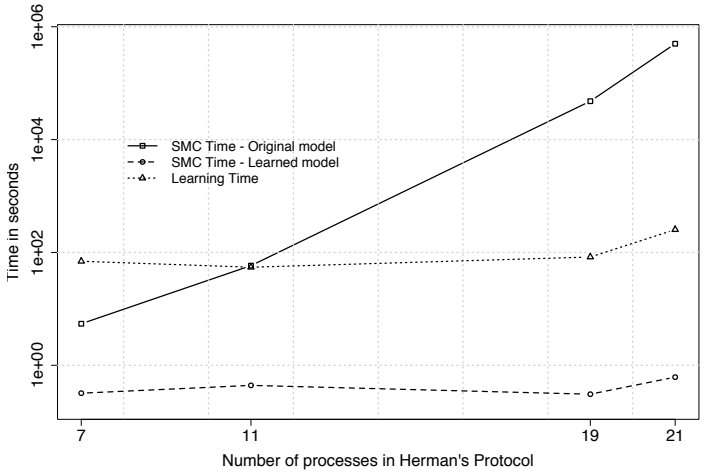


Figure 4.5: Statistical Model Checking Time of φ^{10}: original vs. abstract Herman's Self Stabilizing Protocol model. Obtained using PESTIM algorithm with $\delta = 10^{-2}, \alpha = 10^{-2}$.

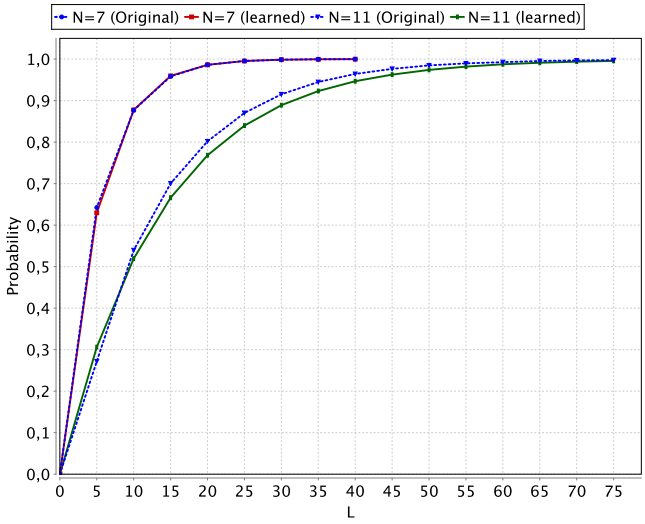
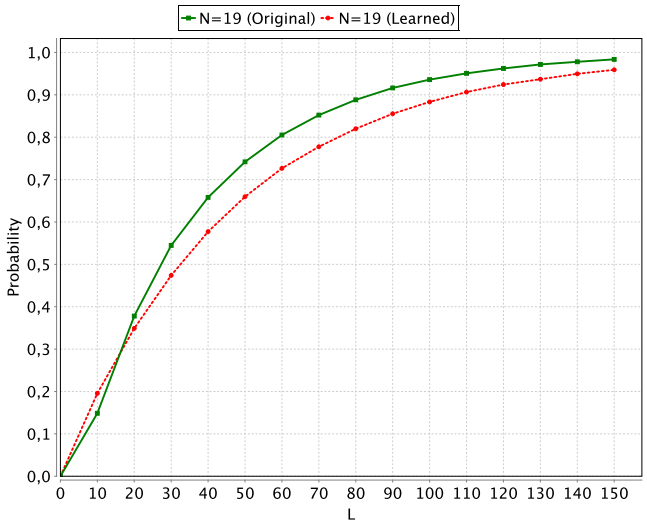


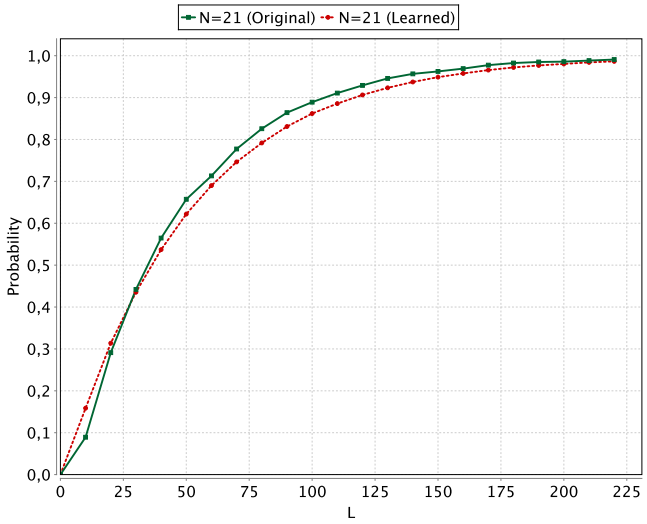
Figure 4.6: φ^L verification results using PESTIM for $M = \{7, 11\}$.

table confirms the observation made in the previous experiment, that is, the reduction of the SMC time when using the abstract model while the probability estimation is still accurate.

Results on Unbounded LTL using Numerical PMC We did an additional experiment on the Herman's Self Stabilizing protocol with $M = 7$ in order to investigate the usability of this instance of the approach for un-



(a) $M = 19$.



(b) $M = 21$.

Figure 4.7: φ^L verification results using PESTIM for $M = \{19, 21\}$.

bounded LTL properties (all the considered properties so far were bounded). We considered the property $\phi = P(\mathsf{N}(token5\ \mathsf{U}\ stable))$ which queries the model with respect to the probability to reach a stable state directly from a state with 5 tokens. N is the *Next* operator and is used for technical reasons. Since the initial state of the Herman's Protocol with $M = 7$ has 7 tokens, we use *Next* to avoid this state. Otherwise, the probability of ϕ will be 0.

The learned abstract model corresponding the this experiment is shown in Figure 4.8 and the verification results are depicted in Table 4.3. The

	L	*Original Model*			*Learned Model*		
		θ	*Traces*	*Time(s)*	θ	*Traces*	*Time(s)*
$M = 7$	$L = 1$	$[0.109, 0.110[$	622018	25.643	$[0.107, 0.108[$	588357	1.363
	$L = 30$	$[0.111, 0.112[$	622834	25.749	$[0.108, 0.109[$	533885	1.282
	$L = 65$	$[0.111, 0.112[$	651434	26.756	$[0.108, 0.109[$	476883	1.118
$M = 11$	$L = 1$	$[0.011, 0.012[$	147178	85.135	$[0.012, 0.013[$	163600	0.411
	$L = 30$	$[0.011, 0.012[$	105362	60.206	$[0.013, 0.014[$	098493	0.262
	$L = 65$	$[0.011, 0.012[$	137469	80.648	$[0.013, 0.014[$	248300	0.564

Table 4.4: Statistical Model Checking of ψ_M^L on the original and the learned Herman's Self Stabilizing Protocol models. Obtained using the SPRT algorithm with $\alpha = \beta = 10^{-3}, \delta = 10^{-3}$.

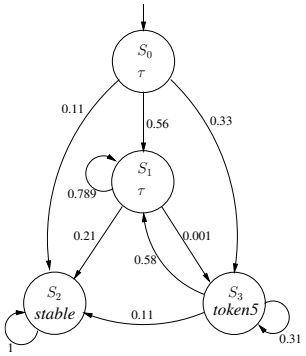


Figure 4.8: Abstract Herman's Self Stabilizing Protocol model with $M = 7$ using the AAlergia algorithm. Obtained with respect to property ϕ.

obtained results show that the probability of satisfying ϕ is almost the same for the abstract and the original protocol models. Concretely, we obtained 0.160 for the abstract model and 0.167 for the learned one. This is possible (to check unbounded LTL properties on the abstract model with a good accuracy) because, in this case, there exist an equivalent deterministic model for the original Herman's Self Stabilizing protocol (which encompasses non-determinism) that AAlergia was able to capture. For the verification part, we used numerical probabilistic model checking algorithm implemented within PRISM to show that the proposed abstraction method is also beneficial for this category of algorithms.

4.4 Conclusions and Related Work

Conclusions and Future Work

Reducing the PMC time, and more specifically the SMC time, of a given LTL property on large stochastic models is the primary benefit of our abstrac-

tion approach. This gain is achieved through the combined use of projection on traces and machine learning. Projection is performed by considering the support of the property of interest, that is, the set of symbols explicitly appearing in that property, while learning automatically infers a compact model using state merging techniques. The proposed approach could be instantiated with any learning algorithm, under the assumption that it respects the conditions discussed earlier. It produces accurate models preserving the probability of satisfying the property of interest. Experimental results show that (1) verifying the property of interest on the abstract model is faster than doing it on the original one, and (2) that the estimation of the probability of satisfying the properties is accurate with respect to the one obtained on the original system.

In the current stage of work, the proposed projection definition is quite simple. It allowed us to instantiate our methodology and to implement it for validation. As future work, we are planning to improve it to obtain coarser abstractions, yet preserving the probability of the underlying property (as opposed to a class of properties currently). This could be potentially achieved by taking into account the semantics of the LTL operators. We shall also apply the approach to other real-life systems and consider using other algorithms able to learn non-deterministic models. An important extension would be to adopt it for learning $\mathcal{S}$BIP components and their corresponding glue. Furthermore, the proposed approach is only applicable to discrete stochastic systems. An interesting direction to investigate is its extension to continuous systems such as continuous time Markov chains or probabilistic timed automata. We provide hereafter some pertinent works that may help obtaining such an extension.

Related Work

In this section, we first provide an overview of studies that applied learning techniques for systems verification in general. These are not necessarily state merging, nor in the limit identification techniques. For more details, we refer the reader to the literature survey by Martin Leucker [148]. With reference to the future research directions mentioned above, we present a bench of algorithms that allow for learning non-deterministic behavior of systems in addition to probabilities. In the same perspective, we introduce other algorithms for learning stochastic continuous-time models such as CTMCs and GSMPs.

Pena et al. propose to use machine learning for the purpose of state reduction in incompletely specified finite state machines (for which for all states and all inputs, the next state or outputs are unknown) [167]. The main motivation of this work is that (a) it is proven that the state reduction in incompletely specified finite state machines is NP-complete, and (b) the exact and heuristic existing solutions are time consuming. The authors ap-

proach the problem using learning techniques to obtain a minimal system for given input/output mappings. Based on Angluin's L* algorithm, which computes the minimal DFA in polynomial time, the authors propose a learning technique to obtain an equivalent FSM to the given input.

Cobleigh et al. propose to use learning for assumptions generation in assume-guarantee style compositional verification to address state space explosion problem in verification of systems [65]. Verification is achieved in an incremental fashion, that is, an assumption that characterizes a certain component C context is first generated and combined with a property, then checked with respect to C. If the assumption is satisfied by C and on the rest of the system as well, it is proven that the property is verified on the whole system. This iterative process is shown to terminate. In fact, the algorithm converges to an assumption that is necessary and sufficient for the property to hold on the underlying system.

Peled et al. propose to combine model checking, testing, and learning to automatically check properties of systems whose structure is unknown [166]. This paper motivates black-box checking where a user performs acceptance tests and does not have access to the design, nor to the internal structure of the system. The authors, however, conclude that the complexity of their algorithms could be reduced if an abstract model of the system would be available. Additionally, the authors point out the need to take into account the property of interest to tackle verification complexity.

In the same context, (of black-box systems), we also mention the work by Sen et al. which propose an SMC algorithm to verify this kind of system [188]. In this work, systems are assumed to be uncontrolled, that is, traces cannot be generated on demand. The approach cannot guarantee correct answers within required error bounds. It computes instead a p-value as a confidence measure. In contrast to our approach, [188] uses pre-generated traces as direct input to their SMC algorithm. This raises confidence issues but makes it faster since no learning is performed.

Habermehl et al. propose to use learning for regular model checking infinite-state systems [102]. The learning technique enables generating a sample of reachable configurations of the given system. Methods for inference of regular languages are used to generalize the sample towards obtaining the full reachability set or an over-approximation where the property of interest holds.

In the first part of the chapter, we briefly mentioned some algorithms for learning non-deterministic models such as PFAs. These are all following a state merging approach. Here, we introduce additional algorithms using a different technique and detail both categories.

Recently, Mao et al. proposed a new algorithm called IOAlergia to learn MDPs [154] from sample traces, following the same assumptions and procedure in AAlergia [153]. IOAlergia is an adaptation of the Alergia algorithm [54] in the context of non-deterministic models, specifically for de-

terministic labeled MDPS. Stolcke and Omohundro propose a Bayesian algorithm for learning the parameters and the structure of Hidden Markov Models (HMMs) [177] [4] following a state merging fashion [194, 193]. In this work, similarly to Alergia, an initial specific representation consisting of the training input data is first built. Then, based on Bayesian posterior probability, a more general model is derived by merging states. Ron et al. propose an algorithm for learning a subclass of acyclic PFAs [180]. This work also uses state merging starting from an initial representation of input traces. The merge criterion is based on frequencist statistics in contrast to the previous Bayesian approach. The above mentioned algorithms are all proven to learn the correct target model in the limit.

In contrast to the previous merging-based algorithms, more recent work [81, 103] proposes to learn more powerful models [5] such as Multiplicity Automata (MA). These works performs identification of underlying rational languages residuals and iteratively solve equations over them.

The previously mentioned works essentially focus on discrete-time models. However, others work, albeit few, exists for learning continuous-time models. For instance, Verwer et al. propose to learn Timed Automata (TA) from timed systems observations [203]. The algorithm start by a prefix tree and performs merging and splitting based on hypothesis testing and well-known statistical tests. This work learns models in the deterministic context, in the sense of no probabilistic behavior. In the following, we present two algorithms for learning stochastic continuous-timed models. Sen et al. proposed an algorithm to learn CTMCs models from sample traces [187]. In this work, the authors follow the state merging approach by providing a variant of the Alergia algorithm. In [79], Pedri et al. propose an algorithm following the same strategy to learn more general continuous-time models, namely GSMPs. This uses new statistical clocks estimators, in addition to the state compatibility criterion of Alergia.

So far, we presented a set of formalisms and techniques for both probabilistic modeling and analysis of stochastic systems. This first part of the dissertation could be seen as a stochastic framework for component-based stochastic modeling, automatic stochastic abstraction through machine learning, and stochastic model analysis using probabilistic model checking and more specifically, statistical model checking.

In the next part, we will introduce a method for dealing with performance aspects at system-level in the context of embedded systems design. There, we will be using our stochastic framework to build functional high-level models encompassing performance details and to analyze them with respect to global extra-functional requirements.

4. These could be seen as PFAs where the strings generation is performed according to probability distributions of state symbols.

5. For more details about the classification of the above mentioned models in term of expressiveness, we refer the reader to [202].

PART II

METHODS AND FLOW

Chapter 5

Rigorous System-level Performance Modeling and Analysis

The first part of the manuscript has set up most of the foundations of the present work. It depicted our contributions in terms of formalism for stochastic modeling of systems, that is, the $\mathcal{S}$BIP component-based formalism, in addition to techniques for probabilistic analysis of these systems and their abstraction. In this second part, we will use these ingredients to construct a rigorous approach for system-level performance modeling and analysis of embedded systems. The part is split into four chapters. The first introduces the *ASTROLABE* approach and its different steps. In the second chapter, we give a detailed view on the method we propose for statistical characterization of performance data. The third chapter presents a tool-flow implementation for the approach and finally in the fourth chapter, we illustrate the proposed approach and tool-flow on a real-life case study for image recognition.

The increasing complexity of modern embedded systems together with the growing competition and the time-to-market constraints, forced designers to consider more elaborated and systematic ways for system design. Although several design methods were proposed in the last decades, a tangible lack of support for performance aspects remains. Extra-functional aspects in general are becoming as important as functional ones. As discussed in the introduction, performance has become very important due to the proliferation of electronic and multimedia devices in our everyday lives. These are expected to ensure several functionalities, that is, they must be designed in a flexible and programmable manner. At the same time, they must be efficient and provide good performance while delivering services.

In this chapter we propose an approach for modeling and analyzing performance in early design phases. We first explain the choices we made to conceive this method with respect to the identified challenges for embedded systems design in general and for dealing with performance at system-level.

We then depict the different steps of the method in the following sections.

5.1 *ASTROLABE*: An Integrated Approach

5.1.1 Answering General Design Challenges

Rigorous design approaches, in contrast to empirical approaches, specify systematic and methodical ways to transform a set of specifications or ideas to a concrete artifact or to an unambiguous design which is realizable in a straightforward manner, e.g., a net-list for circuits design, a plan of a building. To be successful, a design approach must decompose an initial problem to a set of humanly tractable sub-problems addressed in *iterative and incremental* fashion [192]. Iterative is intended in the sense of repetition while incremental means to consolidate results when moving from a step to another, that is, preserving previous results and improving them in the next steps.

As explained in the introduction, in addition to the *divide and conquer* approach, *system-level design* allows to handle systems complexity by enabling reasoning at high-level of abstraction. Moreover, *model-driven* approaches help representing and capturing the considered concepts and anticipating potential errors. Besides, models helps realizing the previous aspects as they may be initially abstract then refined in an incremental manner. Several iterations may be also necessary to converge to good refinement. Furthermore, rigor imposes to rely on unambiguous models, which have clear semantics and that are not subject to divergent interpretations. This may be applied using *formal models and analysis techniques*.

Design space exploration is among the most important activities in system-level design. It aims at finding the most appropriate design alternatives matching the initial requirements. This must be performed following the previous guidelines, i.e., iteratively and incrementally as illustrated in Figure 1.3. System-level design also advocates *separation of concerns*, which distinguishes between communication and computation on one hand, and between application and architecture on the other hand. This is actually a realization of the more general divide and conquer principle. From this perspective, the *Y-chart pattern* proposed in [17] is a concrete implementation of this principle. It consists of considering an application model, several target architecture models, several mappings, and trying to figure out the best configuration with respect to given requirements, generally concerning performance.

Following these guidelines, we propose a rigorous approach that enables performance modeling and analysis during early systems design phases:

– It is rigorous because it relies on formal methods for modeling, model transformations, abstractions, and analysis.

- It is model-based because it is centered around the $\mathcal{S}$BIP formal modeling language which guides the different design activities.
- It operates at system-level because of the high-level of abstraction enabled by the used formalism and abstraction techniques.
- It is iterative as it proposes a limited number of steps to repeat until reaching an optimal design. These steps consists mainly on modeling and verification steps as detailed later.
- It is incremental since moving from an iteration to another preserves properties as guaranteed by the used formalism.
- It also applies the divide and conquer principle through the Y-chart pattern. It explores different models of applications and architecture, in addition to several mapping.

5.1.2 Answering Performance Specific Challenges

As discussed in the introduction, extra-functional aspect, especially performance, have become as important as functional ones, and taking them into account in the earliest design phases is a must for successful designs. Hence, we need rigorous methods to characterize them faithfully, to build models that integrate them in addition to functional behavior, and to analyze them.

Performance details are related to the physical part of the system, that is, the execution of application functions on architecture components, e.g., execution time of a specific functions, say a Fourier transformation, by a processing unit, communication delay of a bus or a NoC used by some component, amount of consumed energy or dissipated temperature induced by performing a specific functionality. Nonetheless, such details are rarely available in early design phases, which makes the process of building high-level performance models quite challenging.

Moreover, following the system-level philosophy, one wants to deal with abstract models that minimize modeling effort and enable fast exploration. These models are required to capture low-level performance details in order to precisely reflect the reality and enable accurate reasoning about the whole system performance. These contradictory goals raise several natural questions. How to capture performance information in early design phases ? What kind of formalism is appropriate to characterize them ? And, how to integrate them in the abstract system model ?

The above challenges maybe classified into modeling and analysis challenges. The proposed approach is thought in term of these steps which constitute its main activities. One great modeling challenge is to enable building abstract models while being faithful. Ensuring these contradictory goals clearly requires trade-offs. Moreover, these models must correctly capture performance variability. The approach has also to enable an easy way to produce distributed models for many-core platforms. For the analysis part, contradictory objectives are also to satisfy. It is required to enable accurate

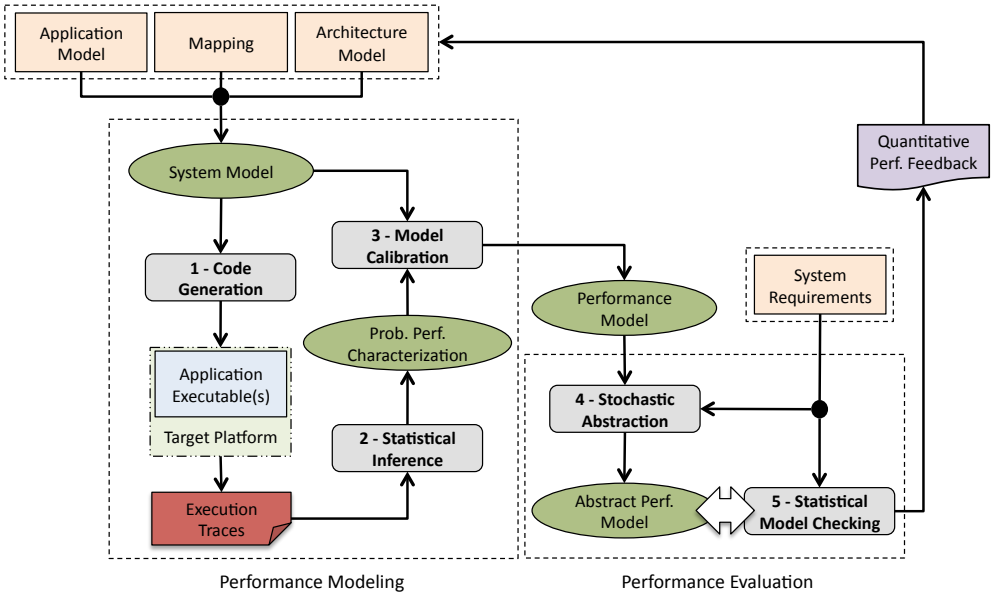


Figure 5.1: The *ASTROLABE* approach for building and analyzing faithful high-level performance models.

analysis while being as fast as possible to permit rapid iteration. Another challenge from this perspective is the scalability of the analysis technique, which has to handle real-life systems models.

To answer these challenges, we conceived the *ASTROLABE* approach illustrated in Figure 5.1. In line with the Y-chart scheme, it takes as input an application model, one or several architecture models, and one or several mappings of the application functionality to the architecture components. It also accepts a set of requirements, which are formalized in temporal logic. We mainly focus on performance requirements, albeit functional requirements are also accepted. The considered input application and architecture models are purely functional and are built in the BIP formalism. These may be obtained automatically through refinement from higher level specifications [22] or provided directly by the designer.

To enable iterative and incremental construction, the proposed approach sets up a closed-loop schema where analysis feed-backs are used to refine the input models. A macro view of Figure 5.1 shows that the approach is made of tow main activities, a performance modeling followed by a performance evaluation step. The former aims at producing a performance model that is in an appropriate level of abstraction, while the latter aims at performing accurate and fast analysis. Together, these ensure trustworthy and rigorous design space exploration. Each one of these two activities is composed of sub-tasks orchestrated towards implementing the global objectives of the approach. The performance modeling phase is conceived following a meet-

in-the-middle scheme to ensure the targeted abstraction trade-off, whereas the analysis part is top-down as explained hereafter.

A Meet-in-the-Middle Modeling Solution

Top-Down Trajectory. The top-down branch of the modeling phase consists of generating a distributed implementation of the application and an associated deployment strategy on the target architecture with respect to the input mapping. This take advantage of the component-based modeling formalism we are adopting. This task is twofold, first a functional systems model consisting of the combination of the application with the architecture with respect to the given mapping is automatically built [44]. Then, automatic code generation is performed to produce a concrete implementation of the application functionality and the corresponding deployment schema with respect to the mapping on the target platform given some runtime support. The target platform could be, depending on the design phase, an already existing board, a virtual prototype, or an Instruction-Set Simulator (ISS). The code generation step produces instrumented code with respect to the input requirements. This specifies the performance dimension to estimate.

Bottom-Up Trajectory. The bottom-up branch of the modeling phase is performed on the traces obtained by execution of generated code on the target platform. This consists of first learning a probabilistic characterization of the specified performance aspects. As we will explain later in detail, this relies on a statistical inference procedure. This choice is made to capture performance variability in a probabilistic manner. The second step in this bottom-up part is the calibration of functional system models. This will augment the BIP system model with performance information.

The meet-in-the-middle solution for performance modeling produces $\mathcal{S}$BIP models encompassing functional behavior and performance details. We claim that such an approach produces models with a good level of abstraction, which are sufficiently faithful for the earliest phases of the design.

A Formal Analysis Approach

In the *ASTROLABE* approach, we propose to use formal verification as means to quantitatively analyze the obtained model with respect to the input requirements. This is enabled because of the formal semantics of the $\mathcal{S}$BIP formalism. We rely on statistical model checking since it combines simulation and statistical techniques and thus provides a good trade-off between speed and accuracy. For scalability issues, we will use our stochastic abstraction technique proposed in Chapter 4 to compute smaller models for faster analysis.

The remainder of the chapter describes the different steps of performance

modeling. Performance evaluation techniques has been already introduced and discussed in the first part of the manuscript. We first present the code generation technique for gathering realistic performance information. Then, we depict the statistical inference step to derive probabilistic characterization of these information. Finally, we illustrate how to calibrate functional models with probabilistic performance data in the case of timing information.

5.2 Gathering Low-level Performance Details

This phase aims at generating a concrete runnable implementation of the application part of the system. The goal is to run this implementation on the target hardware architecture in order to extract performance measures. This has to be performed automatically to avoid bug injection and quickly to enable rapid exploration. Furthermore, since we target multi and many-cores architectures the generated implementation should be distributed with respect to a given mapping and the deployment must be straightforward. In order to enable measurements, the generated code must be instrumented with respect to the targeted performance dimensions.

5.2.1 Automatic Implementation Generation

The BIP framework encompasses different code generation back-ends for centralized and distributed targets [42]. In the context of many-core architectures, we mainly consider the application model and the mapping, in addition to the performance requirements to instrument the generated implementation accordingly. For the sake of simplicity, we consider application model in BIP with restricted form of interaction and coordination, e.g., send/receive, KPN BIP models for which producing distributed implementation is straightforward and natural. Often in these models we distinguish between computational components, e.g., processes and threads, and communication components, e.g., FIFOs and shared memory. There are basically two steps in order to produce implementations from BIP models.

Computation/Communication Objects Generation

In this phase, each computational BIP component is systematically transformed to a process. The notion of process is used in an abstract meaning. Its concrete interpretation depends on the target runtime, e.g. POSIX threads. The behavior of each generated process consists of the corresponding BIP component automaton where synchronizations are transformed to communication primitives calls provided by the target runtime. Communication objects are usually shared memory objects, e.g. fifo channels. This are similarly generated given the target runtime.

Deployment/Glue Code Generation

This produces code that maps the generated objects on the target hardware architecture. That is, the processes to specific processing units and the communication objects to memory.

Remark that the above process is strongly related to the target runtime and that we only provide a generic description of the main steps. Additional details are presented in Chapter 8 where we consider STHORM [155] as a target architecture and an implementation of the MCAPI[1] standard as the underlying runtime.

5.2.2 Instrumentation

Instrumentation consists of annotating the generated implementation with specific function calls for measuring performance, e.g., timing, temperature. At this level, both communication and computation functionality are handled similarly. Each communication routine or computation functionality is enclosed withing specific annotations that specify the desired segment to measure its time or energy consumption for instance. This is currently done manually and depend on the provided support of the target runtime.

5.3 Characterizing Performance Information

Depending on the observed performance dimension, collected data may reflect timing behavior of computation or communication, temperature dissipation, energy consumption, or memory utilization. These performance aspects are necessary together with functional behavior to enable trustworthy system analysis. Our goal is to characterize these details in order to obtain a faithful and abstract representation of performance data.

An important characteristic that must be considered is variability. Performance behavior is ideally deterministic, although due to several reasons, it is affected by some fluctuation which should be taken into consideration for trustworthy analysis. Variable workloads are among these reasons, albeit, there exist systems which have constant performance with respect to variable inputs. Another source of fluctuation is the environment where the system will be deployed. For example, when considering temperature as a performance metric, the role of the environment is preponderant since it directly impacts components temperature. Furthermore, some mechanisms such as caches and arbitration impact performance evolution.

In general, variability induced by the above reasons may be deterministically characterized. However, besides being in contradiction with the system-level design guidelines, this is generally unfeasible. Precisely characterizing the environment is unfeasible unless the system is designed for very specific

1. www.multicore-association.org/workgroup/mcapi.php

missions and targeted to a well known and controlled environment. Similar arguments apply for the input data. Moreover, this requires to have detailed specifications, which are rarely available in early design phases. Assuming that such details are available, this will induce a considerable understanding and interpretation efforts, and consequently huge models. This eventually leads to an ad hoc approach which is tedious and error prone. We recall that the goal is to build faithful high-level models. Abstraction of details is thus a must. For instance, low-level arbitration and conflict resolution mechanisms cannot be modeled in details during early design phases.

Our proposal is to consider that performance evolves probabilistically. We argue that this approach provides very natural abstractions. Let us consider a coin tossing experiment. In such a setting, it is common to consider that the coin have a probabilistic outcome. However, this is an abstract view of reality. Actually, if we are able to precisely characterize the coin, e.g., its weight, the environment where the experiment is happening, e.g., wind speed and direction, and the flipping power, we would be able to precisely compute the outcome of the experiment using physics laws. Probabilistic modeling is a natural choice to abstract details either because we don't care about them at the moment, or because we are not able to handle all of them, e.g., we don't have access to details, it takes too much time, they are too complex.

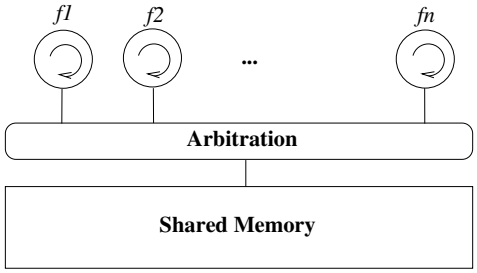


Figure 5.2: Memory access and arbitration mechanism.

Let us consider a situation where the goal is to characterize the execution time of a function (or a process) f running periodically on a specific processing unit. The latter is assumed to execute only f, that is, no scheduling (context swap) overhead is induced. However, data to be processed by f is stored in a shared memory which is accessible by different processing units executing similar functions as shown in Figure 5.2. Thus, the execution time of f will vary depending on the number of processes and the way they are accessing the shared memory each time. Formally, if at time t, f tries to access the memory alone, the execution time will be x_1. If P is the periodicity of f, and at time $t + P$ f accesses the memory with y concurrent processes (in competition), the execution time will be $x_2 = x_1 + \epsilon$, where ϵ is an arbitration overhead which depends on the number of processes y. Figure 5.3 below illustrates this setting.

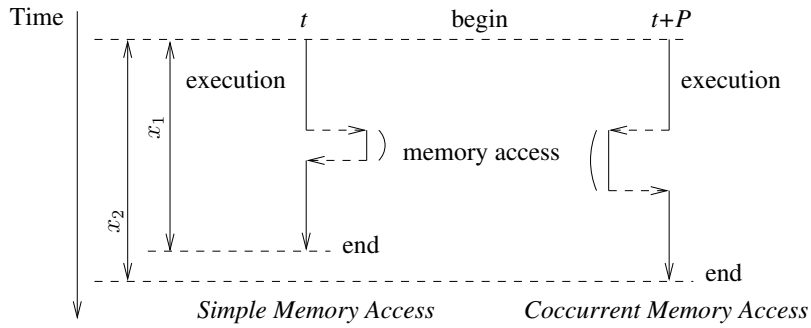


Figure 5.3: Impact of arbitration mechanism on execution time.

Characterizing such behaviors probabilistically avoids modeling arbitration mechanisms precisely and enables capturing the context in which the considered function is evolving, denoted *interference*, faithfully and in an abstract fashion. Again, it is possible to build a detailed model of interference, where the execution time evolves with respect to the number of processes accessing the shared memory and given the used arbitration mechanism but this is not our objective. This maybe need for refining the abstract model.

Probabilistic modeling provides various abstraction possibilities, ranging from simple point estimate, e.g., mean, variance, to more sophisticated models, e.g., probability density function, stochastic processes. This depends on the level of details required to resolve a given problem but also on the used method to build such models. In our case, we aim to summarize performance details obtained from concrete executions while being faithful. Such procedure may be seen as learning general characteristic of some process, assumed to be probabilistic, by only observing some of its manifestations, e.g, running data. This bottom up approach, in contrast to building probabilistic models from specifications, is often called *Statistical Inference*.

We use *Distribution Fitting*, which will be detailed in Chapter 6, together with and overview of statistical inference, to probabilistically characterize performance information. In distribution fitting, we consider probability distributions as potential model for the data, i.e, it is only possible to characterize the data as a probability distribution. It is based on the three following steps detailed in Chapter 6:

1. Exploratory Analysis: in this first step, we explore the data in term of its shape to get a first idea on possible standard probability distributions that may match.
2. Parameters Estimation: once we have some candidate distributions, we estimate their parameters using known estimation techniques.
3. Evaluation: finally, we evaluate the obtained fits using established statistical tests.

5.4 Calibrating Functional Models

The calibration process aims to augment functional BIP models with performance information, learned using distribution fitting. This transformation produces stochastic BIP models that enables probabilistic analysis. The inferred probabilistic characterizations are used as probability distributions related to the $\mathcal{S}$BIP probabilistic variables that are added to model the desired performance dimensions.

We focus on calibration of functional application models. The reason for this choice is to avoid building architecture components in the very early design phases. Moreover, we are positioning this work in the setting where we are given a target architecture as part the system specifications. The main question is thus how to map the different application functionalities into the target architecture. However, we claim that the proposed approach can be easily extended to handle architecture models as well. A previous work in the context embedded systems design in BIP [44] proposed a library of architecture components. We can rely on this work to calibrate architecture components in a similar manner.

5.4.1 Timing Information

Functional models are generally untimed or limited to functional use of time, e.g., timeouts specification. They only concern the functional behavior of the system. Calibrating such models with timing information has two main consequences. First, it produces a timed model out of a "zero-time" or untimed functional model. Second, since the timing information is probabilistically characterized, this will engender a stochastic model. The result is thus a new model, denoted *performance model*, that encompasses in one hand the functional behavior of the system and in the other hand the stochastic characterization of timing behavior.

Timing information may be of two types, *Computation time* or *Communication time*. The first is the result of executing some application functionality on certain architecture component. Application functionality may be a software process running on a programmable processing unit or designed as dedicated hardware component. Communication time is induced by software or hardware components that solicit communication resources such buses, NoCs, etc. These are handled differently when calibrating functional models.

In order to introduce computation time of a specific function within a component, a probabilistic variable related to the corresponding learned distribution is first added then the learned computation time behavior is injected trough a waiting time using a *tick* interaction. Communication time are slightly more complicated since are decomposed into begin and end actions as shown in the remainder of this section.

Computation Time

Let $\mathcal{B}$ be the functional BIP model of the component to calibrate. In this model, f is the function that induces a computation time when executed on a target architecture. We call μ_f the learned probability distribution of the computation time of f obtained using the distribution fitting approach, and c_f related to μ_f the probabilistic variable that models this timing behavior.

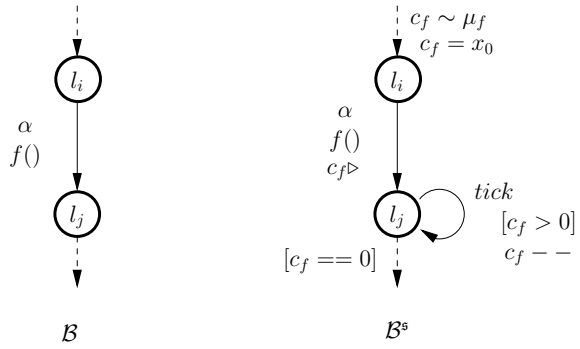


Figure 5.4: Calibration of functional models with computation time.

The calibration of component $\mathcal{B}$ with computation time, gives rise to a stochastic component $\mathcal{B}^5$ as shown in Figure 5.4. First, variable c_f is related to μ_f and correctly initialized as specified in Chapter 3. The transformation mainly concerns the transition α that calls function f in component $\mathcal{B}$. This is transformed in component $\mathcal{B}^5$ as follows. A sampling step ($c_f\triangleright$) that probabilistically updates c_f is introduced on α. The value of c_f specifies the amount of time to be spent as computation time induced by f. For timing aspects, we currently use time transitions called *tick* that models discrete time progress in BIP. In the *tick* transition, the variable c_f is decremented as to model time evolution. Guards are then used to prevent firing the next transition before the sampled time has completely elapsed. Therefore, each time f is called, a certain amount of time, modeling the computation time of f on the hardware architecture, is spent by the component $\mathcal{B}^5$.

Communication Time

Calibrating components with communication time is slightly more tricky than calibrating them with computation time, because communication involves more components. Communication naturally happens between two or more components. Hence, we nedd to take into account the coordination between communicating components.

In this setting, we decompose each communication action (*write* and *read*), to *begin* and *end*: *begin_ read/end_ read* and *begin_ write/end_ write*.

We can then interpose a *tick* transition between the beginning and the ending of each action as shown in Figure 5.5.

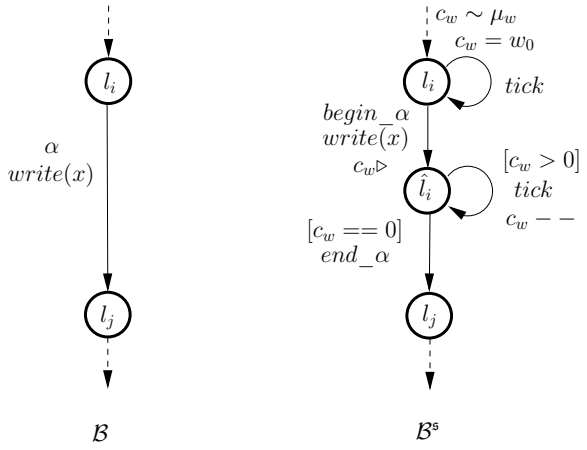


Figure 5.5: Calibration of functional model with communication time.

Given a transition α from l_i to l_j that calls a communication primitive, say *write* for instance, a probabilistic variable (c_w in Figure 5.5), related to the corresponding learned probability distribution (μ_w in the same figure) and modeling the communication time to integrate, is introduced as for the computation time. A $begin_\alpha$ transition leading from l_i to an additional location $\hat{l}_i$ is added. In this transition the corresponding probabilistic variable is sampled ($c_w\triangleright$). Time elapsing is again modeled using *tick* transition as earlier. Finally, the end_α transition leads from location $\hat{l}_i$ to l_j when $c_w = 0$ as specified by the associated guard in Figure 5.5.

Let us consider the case where communication is occurring between two components, namely *Producer* and *Consumer*, through a FIFO *Buffer* as shown in Figure 5.6. In this example, the *Producer* writes data into the *Buffer* (when it is not full) using the *write,push* interaction and the *Consumer* read data from it (when it is not empty) using the *read,pop* interaction. Remark that, in this example, the three components are untimed.

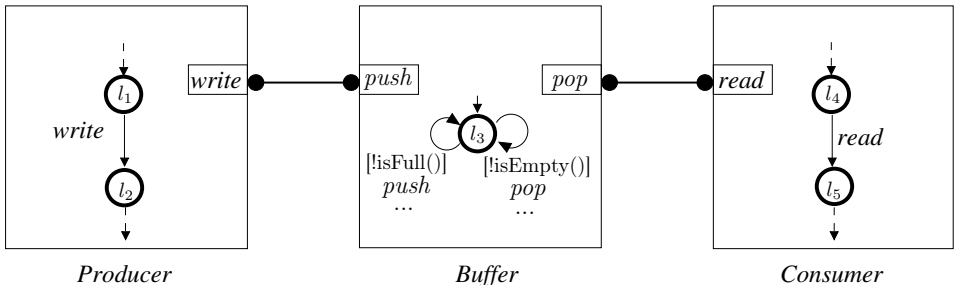


Figure 5.6: Example of communication through a fifo buffer.

Given two probability distributions corresponding to the writing and reading time into/from the buffer, we will calibrate this model to introduce communication timing aspects. We will only calibrate the *Producer* and *Consumer* components and keep the *Buffer* untimed. This is Because we generally learn the communication time from the initiating processes perspective (in this case *Producer* and *Consumer*). In such a setting, applying the transformation illustrated in Figure 5.5, requires to modify the *push* and *pop* actions of the *Buffer* as follows, *begin_push*, *end_push*, and *begin_pop*, *end_pop*. It is then straightforward to see that *begin* actions (respectively *end* actions) are synchronized together for all components as illustrated in Figure 5.7, which show the obtained model after calibration.

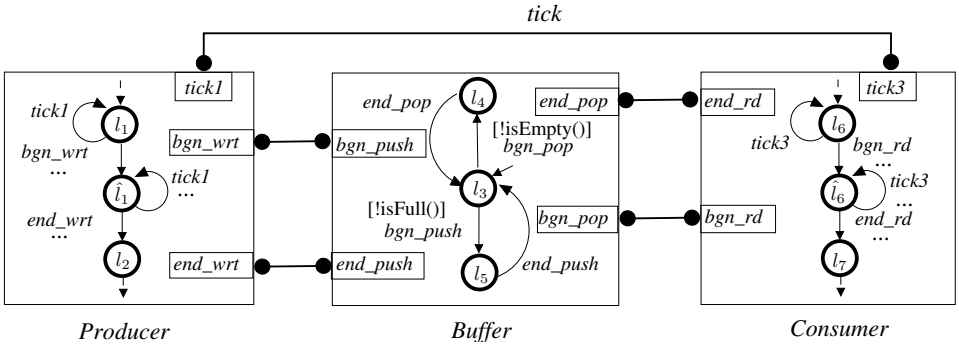


Figure 5.7: Calibration of the functional model in Figure 5.6.

The obtained execution sequence when applying the transformation illustrated in Figure 5.5 is the following. When the *Producer* wants to send some data, the interaction *begin_write,begin_push* is executed. This is only possible when the *Buffer* have sufficient free memory. Otherwise, the *Producer* is blocked (*tick1* transition in l_1 is executed until some memory is freed). When *begin_write,begin_push* is executed, the corresponding probabilistic variable is sampled and the *tick1* transition in $\hat{l}_1$ becomes enabled. During this time, the *Consumer* is blocked until writing has completed. The *tick3* transition on l_6 enables correct time progress through the *tick* interaction. Omitting this transition engenders a deadlock (same as *tick1* transition in l_1). The sampled time elapses in $\hat{l}_1$ through the *tick* interaction. When it is done, the *end_write,end_push* is performed, which enables the *Consumer* to read following the same scheme. That is, *begin_read,begin_pop*, *tick*, then *end_read,end_pop*. A complete typical execution sequence is thus [2]

$$begin_write, begin_push \rightarrow tick^m \rightarrow end_write, end_push \rightarrow$$

$$begin_read, begin_pop \rightarrow tick^m \rightarrow end_read, end_pop.$$

2. The used notations are as follow. $action_1, action_2$ depict an interaction, that is the strong synchronization of the involved actions (see Chapter 3 for more details). The $\rightarrow$ stand for a sequence of actions. Finally, $tick^m$ means that the action is performed m time.

Remark 5.1 (Read/Write Blocking Time). It is important to remark from this example that the blocking time when reading or writing data is already captured by the functional part of the model. The *push* (respectively the *pop*) action of the *Buffer* component is blocking if the buffer is full (respectively empty). We have to take this into account when learning communication time distributions in order to correctly model it. To avoid modeling blocking time redundantly, we have to learn only the communication time that does not include blocking parts. Such observation is quite important and enables to precisely instrument the pertinent parts of the generated code to observe right timing aspects.

Observe that for the communication time, we didn't used the same calibration technique as for the computation time, that is, independently calibrating components. The latter approach will not actually work correctly in the case of communication as shown below.

Following the computation time calibration approach, illustrated in Figure 5.4, will consist to calibrate the *Producer* by the time to write to the buffer and the *Consumer* by the time to read from the buffer as shown in Figure 5.8. What is expected is that the producer performs the *write* action, then a certain time modeling the communication elapses before the consumer can perform *read* from the buffer. Then, some time modeling the reading from the buffer will elapse. That is,

$$write, push \rightarrow tick^m \rightarrow read, pop \rightarrow tick^m$$

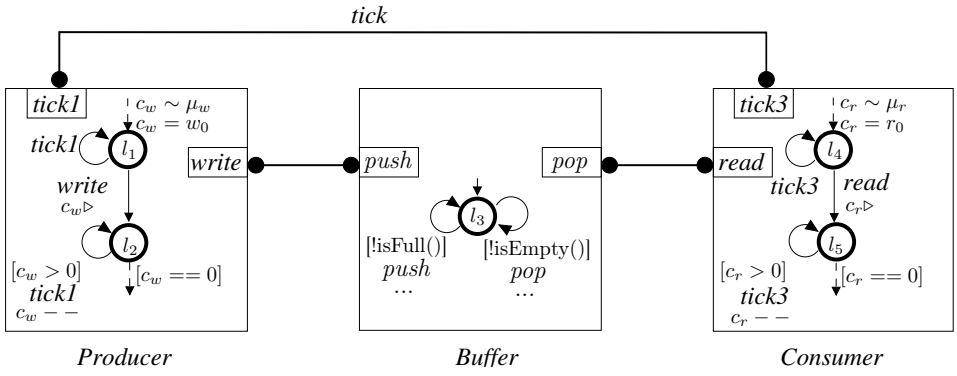


Figure 5.8: Bad calibration of the functional model in Figure 5.6.

However, if we transform the *Producer* and the *Consumer* as in Figure 5.4, what will happen when running the system is the following. When the *Producer* wants to send some data, the interaction $write, push$ is performed (when the buffer is full or empty, the same reasoning as previously is applied). As a consequence, in the *Producer* component, the *tick1* transition

in l_2 becomes enabled. Similarly, the *pop* action in the *Buffer* component becomes enabled since the *push* action is performed instantaneously. We recall that this component is kept untimed. The *Consumer* component is thus enabled to read data. The interaction $read, pop$ is assumed to become enabled whenever *pop* is enabled. As specified in Chapter 3, the interaction involving *tick* transitions is often given the lowest priority to enable system progress deterministically. Hence, the interaction $read, pop$ has a higher priority and is thus performed. The obtained execution sequence is thus

$$write, push \rightarrow read, pop \rightarrow tick^m.$$

This sequence implies that the communication time (or at least a part of it) induced by writing data and the one induced by reading data (or at least a part of it) are spent in parallel, which is not the desired behavior. Thus, the previous approach is not appropriate to calibrate components with communication time.

To correctly represent time in BIP, all the timed components, having *tick* transitions, have to be correctly synchronized to enable overall time progress. A bit of care is needed to build such representations since bad synchronization of timed components lead inevitably to deadlocks as discussed for the l_6 location in the previous example. It is possible to rely on the real-time capabilities of BIP [1] instead of discrete tick transitions to capture time evolution. This enable using built-in clock variables and transition firing with respect to specified deadlines instead of guards. It is worth to mention that we are learning continuous probability distributions, while the the $\mathcal{S}$BIP semantics introduced in Chapter 3 assumes finite models, hence finite data domains. We are thus performing a discretization step of the obtained probability distributions to match the $\mathcal{S}$BIP semantics.

5.5 Conclusions

In this chapter we presented the *ASTROLABE* approach for rigorous performance modeling and analysis for system-level design of embedded systems. The proposed approach combines several activities which are organized together to answers various design challenges. The latter have been identified as general ones and to more specific to performance aspects, which is the primary goal of this work. The approach consists of first building abstract and faithful performance models. These are then analyzed with respect to performance requirements in order to obtain quantitative feed-backs which are used to refine initial models.

Starting from purely functional models of application and architecture, the modeling step first generates a distributed implementation for a target hardware architecture. The generated implementation is instrumented with

respect to given requirements to report on performance metrics. Execution traces obtained by running this implementation on the target platform are then statistically analyzed to learn a probabilistic characterization of performance behavior. Finally, the latter are injected into functional models through a calibration step to obtain stochastic performance models encompassing functional and performance behavior.

The analysis phase takes as input the stochastic performance model and the formalized requirements and operates an abstraction step to reduce the model size. This is followed by a statistical model checking step that checks if the considered model that represents a given design alternative verify the given performance requirements or the expected properties. The results are then used to refine the initial models or to modify the input mapping.

The described process enables vertical evolution through models refinement but also horizontal exploration of different performance dimensions for a given configuration, that is, a specific mapping with a fixed set of parameters, e.g., FIFOs sizes. In the above description we mainly focused on timing. Other performance aspects such as energy or temperature, can be similarly handled in the approach by introducing probabilistic variables modeling temperature evolution or energy consumption. This will constitute the subject of future work and further discussions in the conclusion. Before, we detail in the next chapter the distribution fitting technique we use for probabilistically characterizing performance information.

Chapter 6

Statistical Characterization of Performance

In the previous chapter, we presented the *ASTROLABE* approach and overview its different steps. We briefly discussed how to statistically characterize performance information obtained from concrete executions. In this chapter, we present in more detail the general statistical inference procedure and introduce specifically distribution fitting, the method we use to learn probability distributions that characterize low-level performance data.

6.1 Statistical Inference

Probabilistic modeling often consists of deriving probabilistic characterizations from specifications of an artifact. Another possible approach, in this context, is when we are given data obtained from running the artifact (this may be for instance polling data, or data from a dice experiment). In this case, inferential analysis, that is bottom up trajectory must be taken to build probabilistic model. Statistical inference is thus the process of probabilistically characterizing the behavior of an artifact from data. In our case, data concern performance information obtained from concrete execution or low-level simulation of functional models on a target hardware platform.

From this perspective, data is assumed to be generated by a stochastic process for which the governing law is unknown. Our goal is to infer such a law from a subset of observations, called a sample, since the whole population is generally not available. Formally, given $x_1, ..., x_n$ a set of observations, we assume that there exist $X_1, ..., X_n$ *independent and identically distributed (iid)* random variables such that x_i is a possible realization of X_i. Independence is to be understood in the sense that one outcome of a random experiment does not affect the outcome of another. Identically distributed random variables basically means that they all follow the same probability

distribution $D(\omega)$ where $\omega \in \Theta$ is the set of parameters of the distribution defined over the space Θ.

One should pay attention to the independence assumption above since this will enable accurate generalizations of the inference results. Note that the goal is not only to characterize the available data. The most important is to be able to generalize the result to the generating process. That is, to conclude that the generating process follows some probability distribution. Concretely, the independence assumption states that observations are obtained randomly. Two possible configurations are generally possible:

- The first is when an experiment is conceived with the aim to observe a specific phenomenon. In such a case, independence is easy to guarantee since the procedure is completely controlled.
- The second case is when we perform observations on a process which is not under our control (or partially controlled), e.g simulation or execution of a system.

We are in the latter setting, where independence cannot be assumed but must be checked. Several ways exists to check independence albeit not always easy to understand. One can, for example, use specific plots, e.g., Lag plot, which require expertise for interpretation or rely on existing statistical tests such as Box-Pierce [47], Ljung-Box [151], and runs test [1].

Several inference approaches giving different abstractions are available. These learning procedures may be automatic or not and may produce various probabilistic models ranging from probability distributions to Markov or more general stochastic models. In section 4.4, we detailed several automatic techniques based on machine learning that allow to learn such probabilistic models. These could be used in this context to characterize performance information.

It is worth to mention that adopting a method or another, in addition to being dependent on the desired abstraction, also depends on the available data which reflect the artifact dynamics under certain circumstances. Data may show a certain structure, that is, dependency on other factors. For instance, the execution time of a certain function may depend on its inputs, the communication delay may depend on the number of process that solicit the communication media at that time, the temperature of a component at a time t depends on its temperature at time $t-1$ but may also depend on the temperature of others components or the ambient temperature. This suggest a state-based characterization of the data or an analytical formula that capture this relationship. The latter are for instance obtained using time series methods such as regression analysis approaches [8, 96, 175, 3] that characterize data as a random variable called dependent obtained as a linear or non-linear relations of a set of other variables called independent and some random error, called noise.

1. http://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm

In the present work, we introduce a technique that enable learning probability distribution of performance data. This is motivated by the question we asked in Chapter 3 on how to get probability distributions for $\mathcal{S}$BIP models. Moreover, our focus is on systems which are data-independent. As we will see later in the case study presented in Chapter 8, the system is conceived such that different input data do not affect execution time for example. We also assume that the impact of the environment can be neglected since we are mainly interested on timing aspects (no temperature or energy for now). We are thus left with the effect of internal mechanisms such as arbitration, caches, interference, etc. As stated earlier, our ambition to capture them in an abstract way, motivates us to consider them as stochastic processes.

6.2 Distribution Fitting

Distribution fitting is a special case of a more general approach known as model fitting [143], where the target model is a probability distribution. As stated earlier, the idea is to fit a good probabilistic model to the performance data obtained from concrete executions. In general, this could be a probability distribution, e.g., Exponential, or a more sophisticated model such as a combination of distributions, a regression, or a Markov model. We consider probability distributions as potential model and use *Distribution Fitting* [143, 204] as to probabilistically characterize the performance data. Given execution traces, that is, a set of observations of the performance metric, distribution fitting allows to statistically learn the best probability distribution that fits the data.

We propose a three-steps process to fit a probability distribution to a set of observations. It consists of an exploratory analysis step, followed by parameters estimation, and finally the evaluation of the obtained fit. In the following, we detail the different steps of this fitting process and illustrate them on a data set of size $n = 500$ observations generated from a Normal probability distribution with parameters $\omega = [\mu = 6, \sigma = 10]$.

6.2.1 Exploratory Analysis

In this first step, one aims at identifying a set of candidate distributions that may potentially fit the data. This essentially relies on the shape of the data and its similarity to a known distribution. Such exploration may be performed qualitatively using well-known plots such as histograms or Box-Whisker plots, and/or quantitatively using summary parameters of the data such as mean, median, variance, symmetry, skewness [2], number of modes [3], etc (see [82] for more details).

2. This reflects the existence of a heavy tail. Data may be left skewed or right skewed.

3. A mode can be identified as a prominent peak in the histogram. Data can be unimodal, bimodal or multimodal.

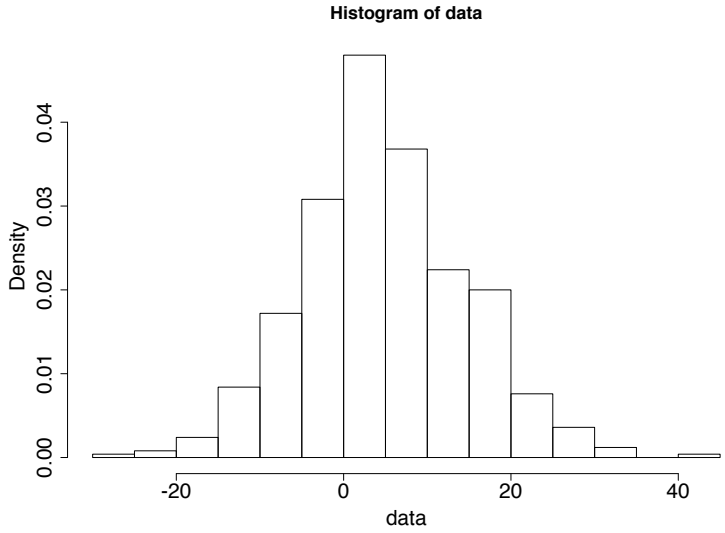


Figure 6.1: Histogram of 500 Normally distributed observations.

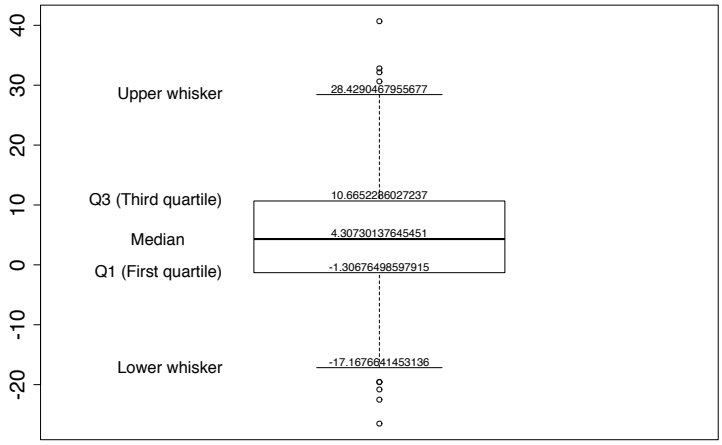


Figure 6.2: Box-Whisker Plot of 500 Normally distributed observations.

Figures 6.1 and 6.2 present respectively a histogram and a Box-Whisker plot of the 500 Normally distributed observations. The histogram shows that the data is unimodal, that is, having a single mode, and is symmetric. The Box-Whisker plot summarizes the data as median, quartiles (Q1 and Q3), and whiskers. It confirms the symmetry of the data since the quartiles are equidistant from the median. During this phase, one would also verify the independence assumption using the statistical tests or the plots mentioned

earlier. A straightforward and direct check is possible through a Lag plot as the one shown in Figure 6.3. The data is uniformly spread and no clear shape is emerging then there is potentially no dependency in the data.

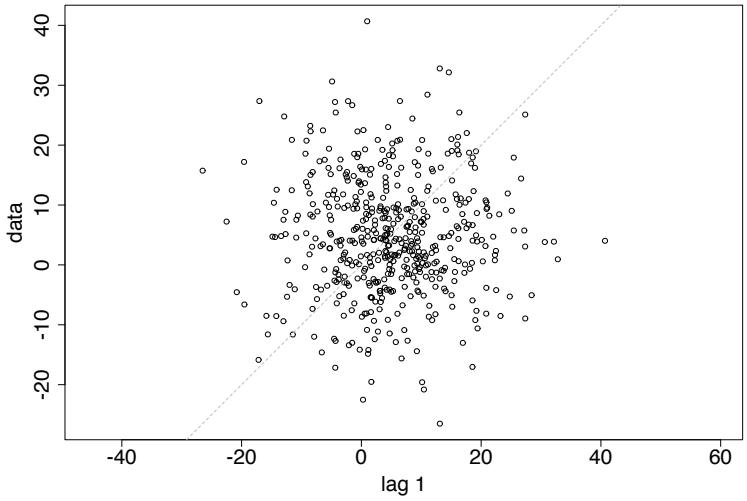


Figure 6.3: Lag Plot of 500 Normally distributed observations.

6.2.2 Parameters Estimation

The goal of this step is to estimate, from the data, the parameters of the candidate distributions identified during the previous step. To this end, one may use one of the methods detailed below [68, 130]. It is important to stress the fact that each candidate distribution actually represent a family of distributions $\{D(\omega) \mid \omega \in \Theta\}$, with respect to a possible rang of parameters. Note also that ω may represent a vector of parameters.

Maximum of Likelihood Estimate (MLE) In this approach, the vector of parameters estimators is denoted ω_{MLE} and obtained by selecting the distribution parameters that maximize a likelihood function. Formally, given a finite data sample $x_1, ..., x_n$ as introduced earlier, ω_{MLE} is the maximum likelihood estimate of ω if

$$\omega_{MLE} = Argmax_{\omega \in \Theta}\ P(x_1, ..., x_n \mid \omega)$$

which is equivalent to say that

$$P(x_1, ..., x_n \mid \omega_{MLE}) = max_{\omega \in \Theta}\ P(x_1, ..., x_n \mid \omega).$$

The likelihood function in this case is $P(x_1, ..., x_n \mid \omega) = \prod_{i=1}^{n} P(x_i \mid \omega)$, since x_i are independent observations. The intuition behind this approach

is that the probability to obtain the observations x_i when selecting the good parameters ω is expected to be the highest, otherwise (when the selected parameters are not good) it is low. It is worth to mention that MLE estimates might be not unique or even, in some cases, not exist.

For the considered 500 observations, using the MLE method we obtain $\omega_{LME} = [\mu_{MLE} = 4.82, \sigma_{MLE} = 9.78]$. We also report a standard error (SE) of the estimates: 0.44 for the mean and 0.31 for the standard deviation. This reflect the marge of error due to data fluctuation. When fixing a confidence level of 95% for instance, we can provide a confidence interval for the estimations [82]. For the mean for example, this interval is equal to [4]

$$5.74 \pm (1.96 \times 0.44) \simeq [3.96, 5.68].$$

This meas that if we generate several data sets of size 500 from a Normal distribution with $\mu = 6$ and $\sigma = 10$, then 95% of these sets will have a mean estimator within this interval.

Moments Matching Estimate (MME) We denote the vector of parameters estimators as ω_{MME}. In this approach, the idea is to resolve a system of m independent linear equations. These are obtained with respect to the different moments of the considered iid random variables X_i, where m is the number of parameters of the underlying distributions. That is, it is required to have exactly [5] as many moments as the number of parameters of the distributions. In the case of the Normal distribution for instance, we have only two parameters (μ, σ). Then, we will have a linear system composed of two equations to resolve. Concretely, the vector of estimators is $\omega_{MME} = [\mu_{MME}, \sigma_{MME}]$.

The moments m of a random variable following a certain probability distribution corresponds to the expectation values $E[X_i^m]$ of that variable. For instance, for a Normal distribution, the first moment is simply the mean, i.e., $E[X_i] = \mu$ and the second moment is $E[(X_i - \mu)^2] = \sigma^2$. Based on the *weak law of large numbers* [8], in the case of $n = 500$ normally distributed observations, the system of equations is

$$\begin{aligned} \mu_{MME} &= \frac{1}{n}\sum_{i=1}^{n} X_i \\ \sigma^2_{MME} &= \frac{1}{n}\sum_{i=1}^{n}(X_i - \mu_{MME})^2 \end{aligned}$$

That is, we are simply estimating the mean of the population by the mean of the considered sample and similarly for the standard deviation. The estimated values are $\omega_{MME} = [\mu_{MME} = 4.73, \sigma_{MME} = 9.17]$. This approach

4. The general formula to compute a confidence interval (CI) for the mean, under the assumption that the data is drawn from a Normal distribution, is $CI = PE \pm (1.96 \times SE)$ for a 95% confidence level and where PE is the point estimate obtained using MLE for instance.

5. In generalized method of moments (GMM), we may have more equations than the parameters to estimate. This induce the existence of several possible solutions.

do not provide a standard error, that is it does not enable to compute a confidence interval.

Percentile Matching Estimate (PME) This approach is similar to the previous one. It consists of matching arbitrary selected percentile values instead of the random variable moments in the previous approach. As in the MME method, it is required to resolve a system of linear equations, where the number of equations is equivalent to the number of parameters to estimate. The vector of parameters estimators is denoted ω_{PME} in this case. Concretely, the idea is to equate the Cumulative Distribution Function (CDF) (which is a function of ω, the distribution parameters) of the fitted distribution, denoted $F(x)$, to the selected percentiles, denoted $g_k,\ k = 1 \dots p$. Formally,

$$F(\pi_{g_k} \mid \omega) = g_k, \text{ for } k = 1 \dots p$$

where π_{g_k} is the values of the g_k^{th} percentile. The estimation of ω is obviously based on the data sample. That is, we may write the above equation as

$$F(\hat{\pi}_{g_k} \mid \omega_{PME}) = g_k, \text{ for } k = 1 \dots p$$

where $\hat{\pi}_{g_k}$ is the values of the g_k^{th} percentile observed from the data set.

For our example data set of size $n = 500$ which is drawn from a Normal distribution $N(6, 10)$, we selected the 1^{st} and 3^{rd} quantiles (Q_1 and Q_3), that is, matching the upper and lower border of the box shown in Figure 6.2. These match respectively 25% and 75% of the data. The obtained estimations of $\omega_{PME} = [\mu_{PME} = 4.73, \sigma_{MPE} = 9.17]$.

Another estimation method, denoted Maximum Goodness-of-fit Estimate (MGE), may be also used. This tries to maximize the goodness-of-fit statistics that we will detail in the next section. It is worth mentioning that besides the MLE method, all the other ones does not provide the possibility to compute a confidence interval for their estimate. In Figure 6.4, we show four fits to the 500 normally distributed observations using the above methods. We took the Normal distribution as potential candidate.

6.2.3 Evaluation of the Obtained Candidates

This third step aims to evaluate the obtained candidate distributions and to select the best that fits the data. The most direct approach to this end is to use plots like the density function (see Figure 6.4 for the 500 Normally distributed observations for example), Q-Q plot, P-P plot, or Cumulative Distribution Functions (CDF) to visually compare the candidate distributions with respect to the data. Figures 6.5 and 6.6 show respectively the Q-Q plot and the CDFs of the fitted Normal distributions to the 500 observations example.

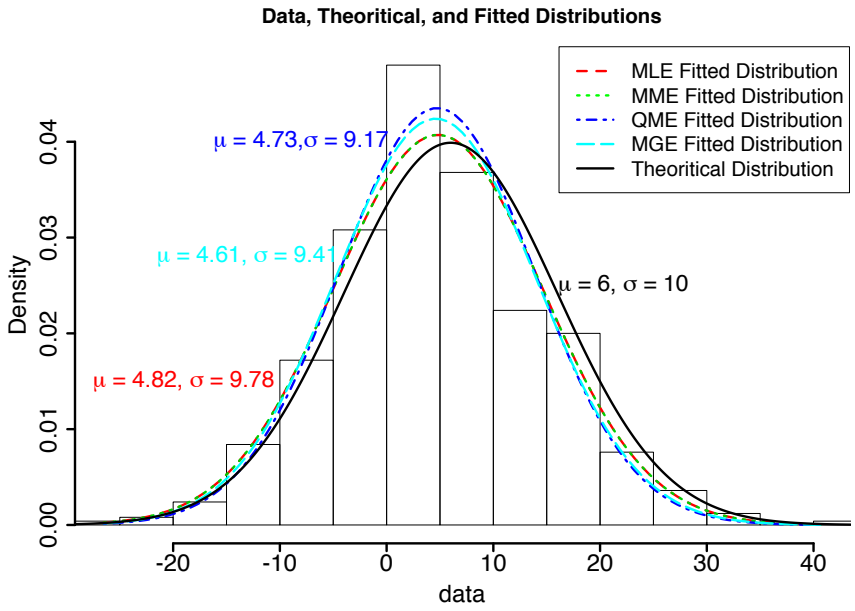


Figure 6.4: Fitted Normal distributions obtained using the MLE, MME, QME, MGE methods. The MLE and MME density functions are superposed and are the closet to the original Normal distribution.

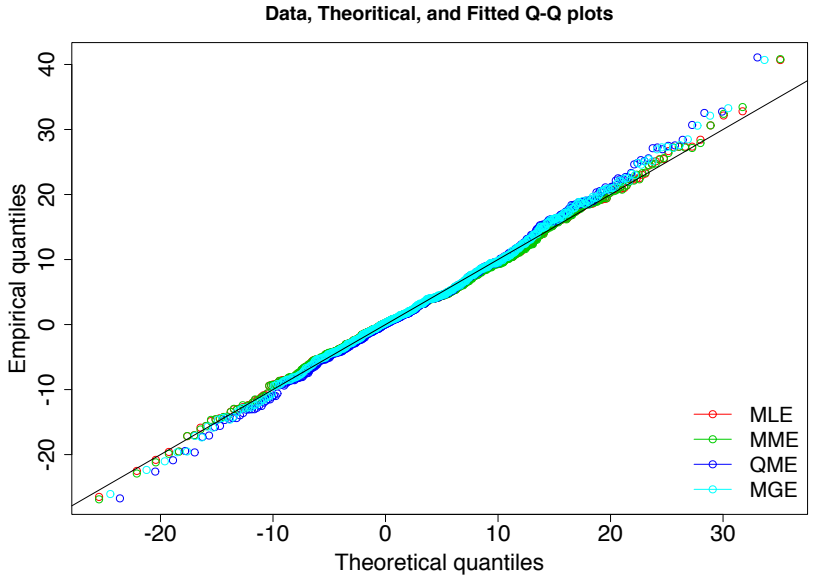


Figure 6.5: Q-Q plot of the different fits obtained by MLE, MME, QME, and MGE estimation methods.

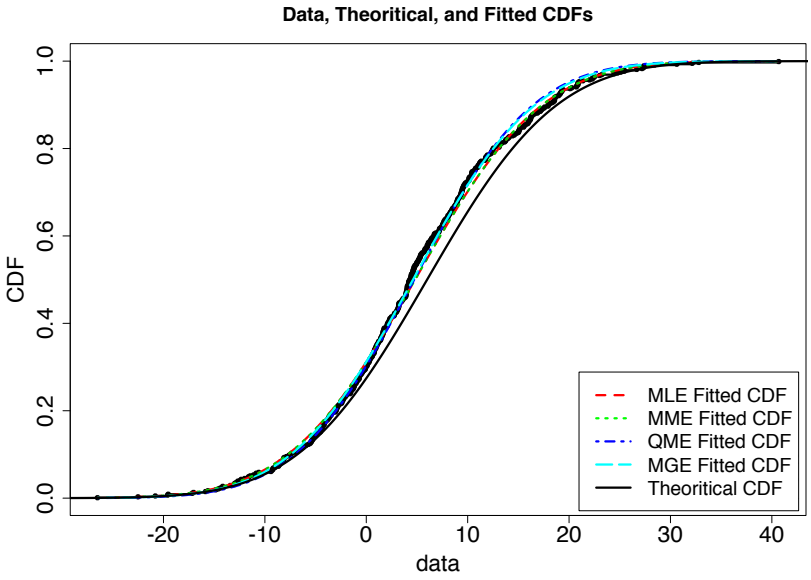


Figure 6.6: Comparison of the Cumulative Distribution Functions (CDFs) of the Normal fits obtained by using MLE, MME, QME, GME on the 500 Normally distributed observations.

Using plots requires human interpretation, which may be subjective and change from a person to another. It is thus better to rely on numerical measures or scores that quantify how close or far a candidate fit is from the data. This is usually performed using a *goodness-of-fit test* which is based on well-known statistics especially designed for this task, such as Kolmogorov-Smirnov (K-S), Anderson-Darling (A-D), and Carmer-Von Mises (C-VM) [204, 130] in the case of continuous distributions, or on the χ^2 test [68, 8] when the fitted distribution is discrete. We illustrate hereafter the main intuition of the statistics used in the continuous case since these are our target.

Goodness-of-fit test A goodness-of-fit test is a special case of a hypothesis test where the goal is to give a measure of how well the null hypothesis H_0 is compatible with the considered data, without using an alternative hypothesis H_1 [68] (as was the case for example in Chapter 2). This may be performed by constructing a test static whose value reflects the agreement degree between the null hypothesis H_0 and the data. In this case, the hypothesis H_0 states that observed data is coming from a population governed by a candidate distribution (obtained in the previous steps).

Let us consider again the coin tossing experiment. Assume one tosses the coin N times and obtains h heads and $t = N - h$ tails. Let us consider the hypothesis H_0 as "*the coin is unbiased or fair*", that is the probabilities for heads and tails are equal. Our goal is to quantify to what extent the

observed h and t for certain N are consistent with H_0. In this case, one can for instance use the number of heads h as a test statistics (h is a random variable). This is known to follow a binomial distribution, $h \sim B(N, p)$, where N is the number of tosses and p is the probability to obtain a head outcome in each toss. When the coin is assumed to be fair, that is, H_0 is true, the probability $p = 0.5$. Given these, we can compute the probability to observe h heads in N tosses using the following formula

$$f(h, N, p) = P(h) = C_h^N p^h (1-p)^{N-h}. \tag{6.1}$$

We take $N = 20$ tosses and we assume we observed $h = 17$ heads. Given $N = 20$ and $p = 0.5$, we know that the expected value for h is $E[h] = Np = 10$ (expectation of a Binomial random variable). We can observe a clear difference between the expected value of number of heads for this configuration and what we actually observed. In order to quantify the level of this discrepancy, one may give the probability to obtain such results or more extreme given the hypothesis H_0 is true, that is,

$$P(\text{to obtain the observed results or more extreme} \mid H_0 \text{ is true}),$$

which is known as *p-value* [8]. In this case, the observed result is $h = 17$ and more extreme results would be $h = 18, 19, 20$ but also $h = 0, 1, 2, 3$ (symmetrical values). Using equation 6.1, and by summing the probabilities for each h above, one obtain $P = 0.0026$. When specifying a significance level α, we may take a decision with respect to hypothesis H_0 as follow.

- If the *p-value* is lower than α, we conclude that it is very unlikely to get the observed data when the null hypothesis were actually true, and hence we reject H_0.
- If the *p-value* is greater than α, we conclude that it is likely to get the observed data even if the null hypothesis were true, and hence we do not reject H_0.

In general, we specify $\alpha = 0.05$ or 0.01. In this example, for both α, the hypothesis H_0 is rejected since $0.0026 < 0.01$. That is, given the observations of 17 heads in 20 tosses, we reject the hypothesis that the used coin is fair.

The test statistics presented above were specifically designed to test if a given sample was drawn from a certain distribution. They use different constructions for testing such hypothesis. For instance, the Kolmogorov-Smirnov [130] statistic is based on the measure of the difference between the theoretical and the empirical cumulative distribution function. Similarly the Anderson-Darling [130] and the Carmer-Von Mises [204] statistics are based on difference measure of the cumulative distributions function. However, they perform weighted and quadratic tests. The For example, the Anderson-Darling statistic gives more weight to the information on the tail.

For the 500 normally distributed observations, the obtained evaluation results using the above methods are presented in Table 6.1. We observe

that the K-S statistic is lower than the significance level $\alpha = 0.05$ fixed in this case, and suggests to reject the Normal fit, while both A-D and C-VM are greater. We thus accept the hypothesis that the data is drawn from a Normal process.

	Kolmogorov-Smirnov	Carmer-Von Mises	Anderson-Darling
MLE	0.0355	0.1090	0.5778
MME	0.0355	0.1090	0.5778
QME	0.0331	0.0852	0.8180
MGE	0.0295	0.0678	0.5954

Table 6.1: Results of the goodness-of-fit tests of the fitted distributions obtained using the previous estimation methods on the 500 considered observations with a significance level $\alpha = 0.05$.

The last phase of the distribution fitting process is quite important since it allows to select the best distribution that fits the data. In addition to the aforementioned test statistics, other tools may be of use, like *Bayesian Information Criterion* (BIC) and *Akaike Information Criterion* (AIC) [130] which provide a way to quantitatively compare the candidate models based on the likelihood function and the number of parameters used in the model. The latter is mainly aimed to prevent over fitting issues, that is, to choose a model (in our case a distribution) with an important number of parameters. For instance, for the 500 Normally distributed observations, we are considering the AIC and BIC scores are shown in Table 6.2 below.

	BIC score	AIC score
MLE	3713.4	3705.0
MME	3713.4	3705.0
QME	3718.1	3709.6
MGE	3715.3	3706.9

Table 6.2: BIC and AIC scores for the four previously illustrated parameters estimation methods on the 500 normally distributed observations.

By crossing the quantitative results with the obtained plots, it is clear that the Normal distribution is a good fit for the data (which is expected in this example, because we initially generated these observations using a Normal distribution). Moreover, we may use the estimation obtained by the MLE and MME methods since they are the closest to the original parameters. In the general case, where the original parameters are unknown, we choose the fit which is accepted by a maximum number of tests, having great p-values, and lower BIC/AIC scores.

It is important to mention that in some cases, a pre-processing phase of the data may be required before performing the fitting. For instance, data may need scaling, outliers analysis (we will give an example about outliers in the next chapter), and/or log transformations. Such requirements are usually detected during the exploratory analysis phase.

6.3 Conclusions

In this chapter, we presented the general statistical inference setting and introduced a distribution fitting process in order to characterize performance information in a probabilistic fashion. The proposed process is composed of three steps potentially leading to a probability distribution that fits the input data. We recall that for distribution fitting to work properly, it is required that the data do not show any dependency. This assumption must be checked during the first exploratory analysis step, which also enables to identify candidate distributions that characterize the data. The second step in the process is to estimate the parameters of the identified distributions. Finally, an evaluation step of the obtained distributions and estimated parameters is performed.

In the next chapter, we will present a tool that assists designers to accomplish this process, as part of a complete tool-flow covering most of the steps of the *ASTROLABE* approach.

Chapter 7

The *ASTROLABE* Tool-flow

In the previous chapters of this part, we presented *ASTROLABE*, a rigorous method for performance modeling and evaluation for early phases of system-level design. We mainly depicted the approach philosophy and its different steps, that is, code generation and instrumentation together with statistical inference and model calibration for building faithful performance models, followed by stochastic abstraction and statistical model checking for performance evaluation.

Design automation has become essential for modern companies to be proactive and to be able to bring quick answers to rapidly evolving markets. Besides time reduction, automation reduces manual tasks which are error prone and hence reduces debugging and testing efforts and tends to increase designer efficiency. The *ASTROLABE* approach is in line with this view in that it is conceived to enable a maximum of automation. This chapter describes a tool-flow set-up to automate and materialize the proposed approach. The flow is composed of several interconnected tools that automate most of the approach tasks, that is:

- Automatic generation of distributed implementations,
- Assisting designer to perform distribution fitting,
- Automatically learning stochastic abstraction for faster analysis,
- Statistical model checking the obtained models with respect to performance requirements.

Model instrumentation and calibration are still performed manually at the current state.

7.1 Overview

The tool-flow is centered on the BIP and the $\mathcal{S}$BIP models. All the manipulated models are either BIP models or stochastic ones expressed in $\mathcal{S}$BIP. Figure 7.1 shows the different tools used in the different phases of

the approach. For code generation, we implemented the BIP2MCAPI tool that, given BIP representations and a mapping, generates an executable code and a deployment scheme targeting the MCAPI runtime implementation for the STHORM many-cores platform, which will be described in detail in the next chapter. Model instrumentation to observe specific performance dimensions is still done manually at this state of the work. It is performed after code generation and does not appear explicitly in the figure. FitDist is a distribution fitting tool which, given a set of execution traces, enables checking independence, and applies the steps presented in the previous chapter to learn the best probability distribution that fits the input data. For stochastic abstraction, we rely on existing implementation of the AAlergia algorithm. We present the projection part (see section 4.2.1) we built on top of it. The model calibration step is still performed manually as stated earlier. Finally, the statistical model checking phase is based on the BIP$^{\mathcal{SMC}}$ engine we built for $\mathcal{S}$BIP models.

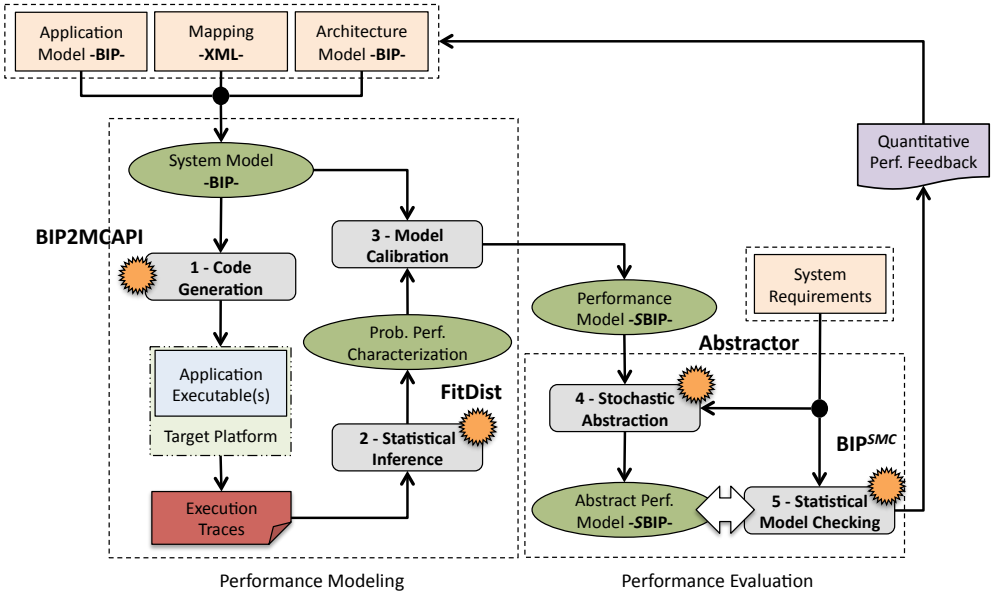


Figure 7.1: Illustration of the *ASTROLABE* tool-flow.

7.2 BIP2MCAPI Code Generator

This tool generates a distributed C implementation of the input BIP system model. The tool only generates code for the application part of the system, since the architecture components will be replaced by a physical or a virtual platform for execution or simulation/emulation. This implementation targets the MCAPI runtime, that is, it uses provided primitives for communication and processes management for instance. The considered

MCAPI implementation is provided by CEA [1] in the context of the SMECY project [2] and is specifically designed for the STHORM many-cores platform, which is designed by STMicroelectronics [3]. Given the functional BIP model, a set of design parameters, e.g., buffers sizes, and a mapping, this step consists to generate the low-level running C/MCAPI code for STHORM platform. This is performed in two main steps: (1) Object generation, that is processes, queues, and shared objects, and (2) Deployment/Glue code generation, which maps the generated objects on the target platform.

7.2.1 Process Generation

During this step, each BIP component is transformed to a C/MCAPI process. A challenging point at this level is to generate small set of process local variables to fit the small amount of available memory per process stack on STHORM. The generated processes are basically composed of two parts:

- **Initialization:** in this part, BIP components interfaces (ports) are transformed into equivalent MCAPI endpoints. The generated process endpoints are initialized, opened, and connected to other processes in this same block. Moreover, endpoints memory attributes are generated to allocate FIFO buffers and shared objects (given their sizes) and to map them into specific memory locations. MCAPI implementation for STHORM provides several possibility to map buffers into specific target memory as shown in Table 7.1. Buffers could be mapped either in $L3$, $L2$ or $L1$ memory using the attributes shown in the table. For $L1$ memory, it depends on the sender/receiver processes location. If these run in the same cluster, the buffer is allocated in the $L1$ memory of that cluster. Otherwise, it is mapped in the $L1$ of the cluster where the sender (respectively the receiver) runs.

Target Memory	*Sender Attribute*	*Receiver Attribute*
$L3$	*Remote*	*Remote*
$L2$	*Shared*	*Shared*
Receiver $L1$	*Shared*	*Local*
Sender $L1$	*Local*	*Shared*

Table 7.1: MCAPI endpoints memory attributes.

- **Behavior:** in this part, BIP component behavior is transformed to equivalent C code with MCAPI primitives calls. It consists essentially on an infinite while loop with several steps (using Switch/Case statements) reproducing the BIP automata behavior. In the generated code,

1. Commisariat à l'energie atomique et aux energies alternatives, http://www.cea.fr
2. http://www.artemis-ia.eu/project/25-smecy.html
3. http://www.st.com

all BIP synchronizations are transformed to C/MCAPI *Read/Write* primitives.

7.2.2 Glue Code Generation

This code is composed of two parts as well: a generic and a generated part. The generic part is parametrized by the amount of memory to be allocated for each stack of the generated processes. It is meant to allocate the generated processes memory and to launch them on the specified hardware location (given the mapping). The generated part is the one that specifies the mapping of the generated objects on the target platform. This is obtained based on the input mapping. In addition, parametric Makefiles are automatically generated to compile and run the generated C/MCAPI code on the STHORM test-board.

7.2.3 Distributed Code Generation within BIP

The BIP tool-set [4], provides several code generators producing distributed implementations for different targets platforms. The default code generator targets classical computers and enables simulation, analysis, and formal verification. This is often based on POSIX threads and a scheduler that implements the BIP coordination semantics. When the coordination between components is not complex, it is possible to get rid of the scheduler. For instance, code generation of distributed implementation of BIP models using asynchronous message passing is presented in [42].

For many-cores platforms, a code generator targeting the MPARM virtual platform and its runtime [152] was presented in [44]. Similarly, a work targeting Kalray [5] MPPA platform [75] is under development. It is also possible to generate distributed implementations targeting sensors networks [145] from BIP models.

In [35], we presented an algorithm and a prototype implementation of a distributed scheduling algorithm for probabilistic components with non-deterministic interactions. This work is based on the idea introduced in [124] for stochastic Petri nets. The proposed implementation is based on the Java *compare-and-swap* primitive and uses shared memory. It does not require any additional support e.g., an operating system. This prototype implementation only supports a subset of the BIP semantics (it does not consider priorities for instance) and is not yet integrated within the BIP tool-set.

4. http://www-verimag.imag.fr/BIP-Tools,93.html
5. http://www.kalrayinc.com/

7.3 FitDist: A Distribution Fitting Tool

As stated in the previous chapter, the distribution fitting process is quite difficult and requires expertise and thus human intervention. The aim of the DistFit tool is to simplify this process by providing assistance to the designer. The tool covers the three steps of the process, that is, exploratory analysis, parameters estimation, and evaluation of the fit as shown in Figure 7.2. In the first step, the tool performs quantitative tests, e.g., Box-Pierce and draws several plots, e.g., ACF and Lag Plots to check independence of data. If dependency is detected, the tool stops. The tool does not yet provide support for this setting. The designer has to perform more advanced explorations (see perspective section in the conclusion), and potentially, data treatment such as scaling or outliers analysis for instance.

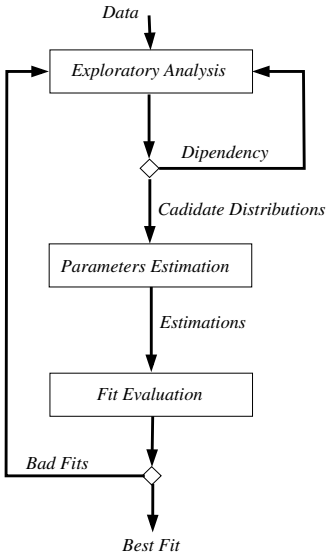


Figure 7.2: DistFit Tool: the different steps, inputs, and outputs.

When the independence assumption is verified, other plots such as histogram, Box-Whisker plots are drawn to enable the designer to identify candidate distributions based on the shape of the data. For instance, the tool draws the Cullen-Fray diagram that positions the data given its skewness and kurtosis with respect to a family of standards distributions. The tool then tries to estimate the parameters of the candidate distributions using the different methods shown in the previous chapter. It then performs a comparison between the obtained evaluations results as in the previous chapter, e.g., comparisons of CDFs, QQ-plots, and provides quantitative results of goodness-of-fit tests and AIC/BIC scores. Although the tool can select a distribution based on the latter information, it is preferable that the designer chooses by combining this information with the graphical results.

The tool is built in the R [6] statistical environment [196]. It uses a specific package called *fitdistrplus*, that offers various facilities for fitting a distribution to a set of input data [80], in addition to the *lawstat* package[116] for specific statistical tests.

7.4 Stochastic Abstraction Tool

As explained in Chapter 4, the stochastic abstraction is mainly based on machine learning algorithms. The illustrated results are based on the AAlergia algorithm [153], for which a Matlab implementation is provided [7]. We built upon this existing work and implemented the projection phase of the stochastic abstraction in Python. We created a set of scripts that, given a set of input strings and the support of an LTL property ϕ, i.e., the set of symbols explicitly appearing in ϕ, produces a set of projected strings by replacing the irrelevant symbols in the strings by a special symbol.

7.5 BIP$^{\mathcal{SMC}}$: An SMC Engine for $\mathcal{S}$BIP Models

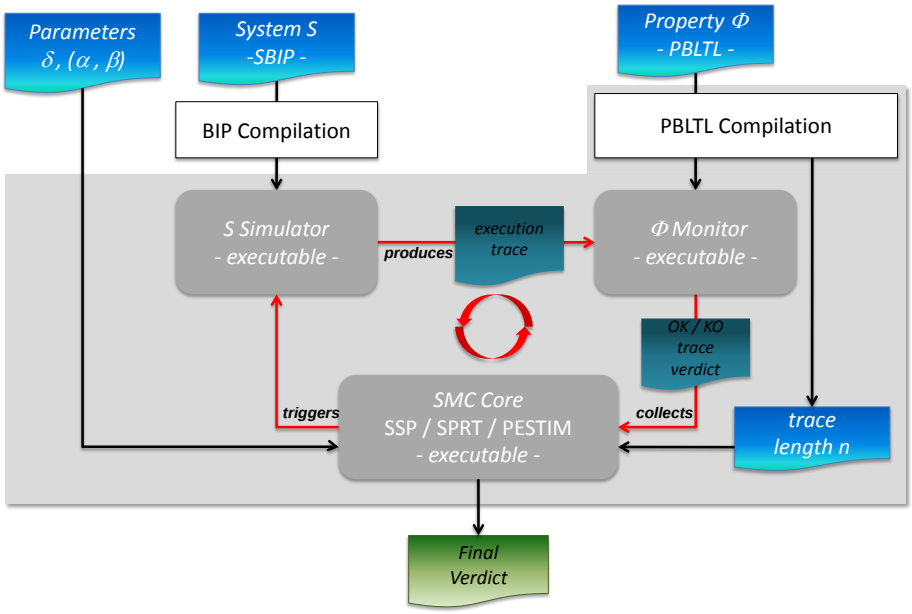


Figure 7.3: BIP$^{\mathcal{SMC}}$ tool architecture: diagram of the main modules.

The BIP$^{\mathcal{SMC}}$ tool implements several statistical testing algorithms for stochastic systems verification, namely, Single Sampling Plan (*SSP*), Sim-

6. http://www.r-project.org/
7. http://mi.cs.aau.dk/code/aalergia/

ple Probability Ratio Test (*SPRT*) [205, 209], and Probability Estimation (*PESTIM*) [110]. Figure 7.3 shows the most important modules of the tool and how they interact in order to perform statistical model checking. The tool takes as inputs a stochastic model description in $\mathcal{S}$BIP format, a PBLTL property to check, and a series of confidence parameters needed by the statistical test. During an initial phase, the tool performs a syntactic validation of the PBLTL formula through a parser module. Then, it builds an executable model and a monitor for the property under verification. Next, it will iteratively trigger the stochastic BIP engine to generate independent execution traces which are monitored to produce local verdicts. This procedure is repeated until a global decision can be taken by the SMC core module (that implements the statistical algorithms). As our approach relies on SMC and since it considers bounded LTL properties, we are guaranteed that the procedure will eventually terminate.

7.5.1 $\mathcal{S}$BIP Modeling Language

We consider the following example of binary signal generator to illustrate the $\mathcal{S}$BIP modeling language. Figure 7.4 shows a DTMC model of the generator and its corresponding $\mathcal{S}$BIP model. The textual description of this component is given hereafter.

```
/* Declares an atomic component */
atomic type binary_generator

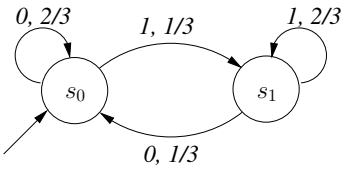
  data int v0, v1   // Declares probabilistic variables
  data distribution_t d0 // Declares probabilistic distributions
  ...

  export port Port print_0() // Declares and exports ports
  ...
  place l0, l1, l0', l1' // Declares BIP locations

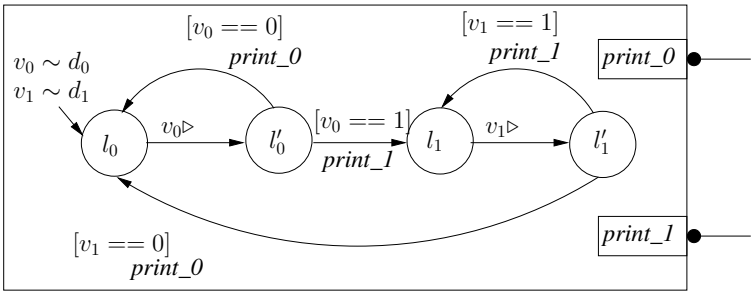
  initial to l0 do {
    d0 = init_distribution('distribution0.txt'); // Initialize d0
    ... }
  ...

  on internal from l0 to l0' do { // Transition from l0 to l0'
    v0 = select(d0); // Update v0 w.r.t. d0
  }

  /* Transition from l0' to l0 */
  on print_0 from l0' to l0 provided (v0 == 0)
  /* Transition from l0' to l1 */
  on print_1 from l0' to l1 provided (v0 == 1)
  ...
end
```



(a) A DTMC model for binary signal generation.



(b) Graphical Representation of the corresponding $\mathcal{S}$BIP model.

Figure 7.4: Example of a DTMC model and its specification in $\mathcal{S}$BIP.

Besides empirical discrete distributions, the $\mathcal{S}$BIP modeling language allows using standard distributions, such as *Uniform*, *Normal*, *Exponential*, etc. For a *Uniform* distribution, *select* function could be called without initialization phase and by providing it with interval bounds as parameters. For instance, $select(100, 500)$ will uniformly sample values in the interval $[100, 500]$.

In addition to probabilistic helper functions, the tool provides tracing capabilities that are needed to monitor state variables involved in the property to check. In the previous example, assume that the $v0$ variable is subject to verification, then the following function call should be used in order to monitor it:

```
trace_i(‘‘binary_generator.v0’’, v0);
```

7.5.2 Properties Specification Language

The properties specification language over stochastic systems in BIP$^{\mathcal{SMC}}$ is a probabilistic variant of bounded Linear-time Temporal Logic. Using this language it is possible to formulate two type of queries on a given system:

- Qualitative queries : $\mathbf{P} \geq \theta\ [\phi]$, where $\theta \in [0, 1]$ is a probability threshold and ϕ is a bounded LTL formula, also called path formula.
- Quantitative queries : $\mathbf{P} =?\ [\phi]$ where ϕ is a path formula.

Note that it is possible through those queries to either ask for the actual probability of a property ϕ to hold on a system (using the second type of

queries) or to determine if the property satisfies some threshold θ (using the first type of queries).

Path formulas, in $\mathcal{S}$BIP, are defined using four bounded temporal operators namely, Next ($\mathsf{N}\{l\}\ \psi$), Until ($\psi_1 \mathsf{U}\{l\}\ \psi_2$), Eventually ($\mathsf{F}\{l\}\ \psi$), and Always ($\mathsf{G}\{l\}\ \psi$) where l is an integer value that specify the length of the trace to consider and ψ, ψ_1, ψ_2 are called state formulas, that is, Boolean predicates evaluated on system state. For example, the following PBLTL formula

$$\mathbf{P} =?[\mathsf{G}\{1000\}(abs(Master.tm - Slave.ts) \leq 160)]$$

asks *"what is the probability that the absolute value of the difference between variable tm and variable ts is always under the bound* 160 *?"*.

In this example, the path formula is

$$\mathsf{G}\{1000\}(abs(Master.tm - Slave.ts) \leq 160)$$

and the state formula is $abs(Master.tm - Slave.ts) \leq 160$. Variables names are always specified as *component_name.variable_name*. Note that $\mathcal{S}$BIP gives the possibility to use built-in predefined mathematical functions in state formulas. For the example above, $abs()$ function is called to compute the absolute value of the difference between tm and ts.

Monitoring LTL Properties

For applying statistical model checking on stochastic systems it is mandatory to be able to evaluate the BLTL property under consideration on system execution traces. Indeed, this monitoring operation shall generate binary observations $x_i = \{0, 1\}$ (single trace verdict) which are requested by the statistical algorithms to provide a global verdict that concerns the whole system (final verdict). In theory, monitoring consists to check if some word (labeling the current execution trace) belongs to the language generated by some automaton encoding the property. Actually, there exist an important research literature about the efficient transformation from LTL to Buchi [93, 208] or alternating [201] automata. Some of these works cover bounded LTL [89, 94].

In BIP$^{\mathcal{SMC}}$, we syntactically restricted BLTL to a fragment where the temporal operators cannot be nested. This simplification restricts the definition to a finite number of automata patterns that covers all property classes. Moreover, this fragment has been expressive enough to cover all properties of interest in practical applications. Furthermore, it is always possible to enrich this set with additional patterns, as needed.

7.5.3 Technical details and Tool Availability

BIP$^{\mathcal{SMC}}$ is fully developed in Java programming language. It uses JEP 2.4.1 library [8] (under GPL license) for parsing and evaluating mathemati-

8. http://www.singularsys.com/jep/

cal expressions and ANTLR 3.2[9] for PBLTL parsing and monitoring. At this stage, BIP$^{\mathcal{SMC}}$ only runs on GNU/Linux operating systems since it is coupled to BIP. The first release of the tool only works on command line mode. We are working on a graphical user interface that will be available in the next release. The model checker can be downloaded from http://www-verimag.imag.fr/Statistical-Model-Checking.html, where you can find additional resources on how to install it and use it with the BIP framework.

7.6 Integration within the BIP Design Flow

The presented tool-flow completes a set of already existing tools within the BIP design flow. This offers a rigorous approach for tackling embedded systems design challenges. As shown in Figure 7.5, it offers an ensemble of tools for formal modeling, model transformations, formal verification, and code generation. We present hereafter the one which are in direct relation with the present work.

7.6.1 DOL and DOL2BIP

DOL stands for Distributed Operation Layer [198]. It is a framework devoted to the specification and analysis of mixed hardware/software systems. DOL provides languages for the representation of particular classes of applications software, multi-processor architectures and their mappings. In DOL, application software is defined using a variant of Kahn process network [95] model. It consists of a set of deterministic, sequential processes (in C) communicating asynchronously through FIFO channels. The hardware architecture is described as interconnections of computational and communication resources such as processors, buses and memories. The mapping associates application software components to resources of the hardware architecture, that is, processes to processors and FIFO channels to memories.

DOL2BIP [44] is a tool that transforms DOL specifications into BIP models. Figure 7.6 shows a KPN model composed of six processes, namely *Config, Splitter, Joiner*, and three *Worker* instances. The communication between these processes is based on blocking *Read/Write* primitives on FIFO channels following the DOL semantics. In this example, the initial step consists of configuring (data size, data type, etc.) the *Worker* processes. It is performed by the *Config* process. Generic processes are considered to enable the processing of different data types/sizes. After configuration, the data to be processed is split and sent by the *Splitter* to the workers that run in parallel. Finally, the results is collected by the *Joiner*.

9. http://www.antlr3.org/

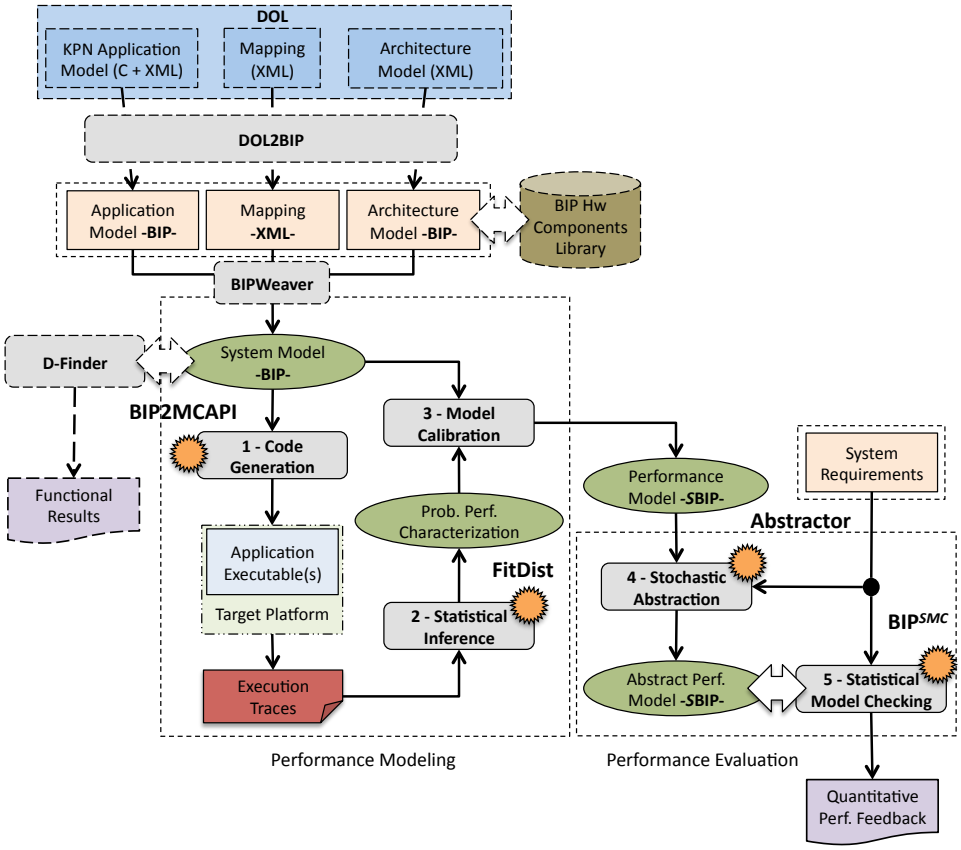


Figure 7.5: The *ASTROLABE* tool-flow extended with several tools: DOL, DOL2BIP, BIPWeaver, the BIP Hw library, and D-Finder.

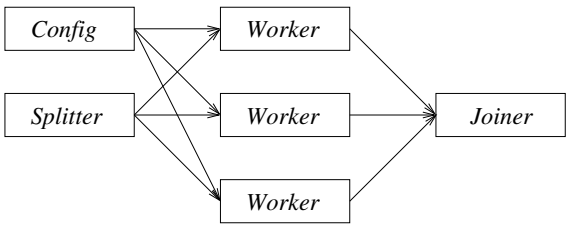


Figure 7.6: A KPN model example consisting of a split/join schema: A splitter, Several parallel workers, and a joiner process.

Given the KPN model in Figure 7.6, the generation of the corresponding BIP representation is straightforward. As shown in Figure 7.7, each process is transformed to an atomic BIP component modeled as an extended automaton following the BIP formalism. Additional components modeling communication through FIFOs are explicitly introduced. In each BIP component corresponding to a DOL process, the DOL *Read/Write* calls are transformed

to equivalent BIP interactions (ports) synchronized with FIFO components that provide *push/pop* primitives. Parametrized FIFO components (with a maximum capacity) are explicitly inserted for each inter-process communication. The connection of the processing components (*Splitter, Worker, Joiner*) with the FIFO components is made through BIP render-vous connectors that models strong synchronization.

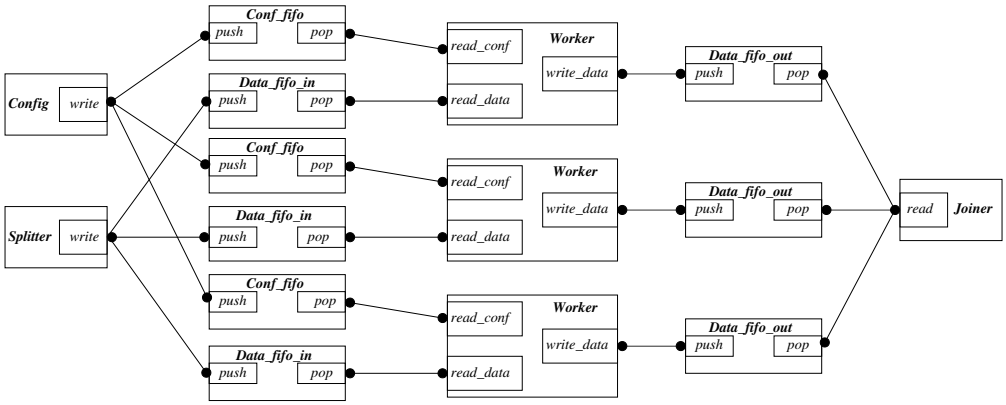


Figure 7.7: The BIP model of the *Worker* example Figure 7.6. Additional FIFOs components are introduced between communicating processes.

7.6.2 BIPWeaver and the BIP HW Components Library

The BIPWeaver tool [43, 45] generates the full model (called system model) of the application software mapped into the hardware architecture components with respect to the input mapping. The inputs to the tool are the BIP application model, the library of BIP HW components, the architecture description in XML format, and the discretion of the mapping also in XML. The tool operates in several steps. It first validates the input files especially the XML descriptions with respect to given schema. Then, it loads HW component corresponding the architecture description from library of BIP components. These range from communication infrastructure, to processing and scheduling components. Finally given the mapping, connectors are synthesized to relate application components to the instantiated architecture components.

7.6.3 D-Finder

D-Finder [10] [33, 34] is a compositional verification tool that enable functional verification of BIP models, that is checking safety properties and dead-

10. http://www-verimag.imag.fr/DFinder.html

lock freedom. It uses an abstraction technique based on invariant and reachability analysis to avoid exhaustive and costly verification. To this end, it enables an efficient generation of components and communication invariant. The tool has been used to verify several case-studies and has been recently extended to invariant generation of timed models [10].

7.7 Conclusions

In this chapter, we presented a tool-flow implementation associated with the *ASTROLABE* approach introduced in Chapter 5. The goal of this tool-flow is to enable rigorous and especially automatic performance modeling and analysis at system-level. We mainly presented a code generator for the STHORM platform and the MCAPI runtime, denoted BIP2MCAPI, an assistance tool for distribution fitting, an automatic way to build stochastic abstractions, and a statistical model checker, namely BIP$^{\mathcal{SMC}}$.

The BIP$^{\mathcal{SMC}}$ model checker consists of 22 Java classes summing up to $\sim$ 3500 lines of code (loc), the BIP2MCAPI code generator consists of 7 Java classes summing up to $\sim$ 1740 loc, and the DistFit tool is one R file that contains about 650 loc. As stated earlier, the stochastic abstraction tool is based on an existing Matlab implementation of the AAlergia algorithm. We implemented on top of it a set of Python scripts for the projection part which is about 130 loc.

In the next and last chapter of this part, we present a case study that illustrates the use of the *ASTROLABE* approach and the associated tool-flow. The case study consists of an image recognition application running on a many-cores platform.

Chapter 8

Image Recognition on Many-cores

In this chapter we illustrate the use of the *ASTROLABE* approach and its associated tool-flow for the design of an embedded system for image recognition and contour detection. The application part of this system consists of the HMAX models algorithm for object recognition that operates on several steps as detailed hereafter. In this case study, the architecture part is completely specified and consists of a many-cores platform, namely STHORM. With reference to the possible design configurations (see Section 1.2.3 in Introduction), this falls within the second setting, where the goal is to efficiently design and map the application into the specified architecture. For this case study, we will be considering timing as the main efficiency criterion.

8.1 Application Overview

HMAX models algorithm [160] is a hierarchical computational model of object recognition which attempts to mimic the rapid object recognition of human brain. Hierarchical approaches to generic object recognition have become increasingly popular over the years, they indeed have been shown to consistently outperform flat single-template (holistic) object recognition systems on a variety of object recognition task. Recognition typically involves the computation of a set of target features at one step, and their combination in the next step. A combination of target features at one step is called a layer, and can be modeled by a 3D array of units which collectively represent the activity of set of features (F) at a given location in a 2D input grid. HMAX starts with an image layer of gray scale pixels (a single feature layer) and successively computes higher layers, alternating "S" and "C" layers:

- Simple ("S") layers apply local filters that compute higher-order features by combining different types of units in the previous layer.
- Complex ("C") layers increase invariance by pooling units of the same type in the previous layer over limited ranges. At the same time, the

number of units is reduced by sub-sampling.

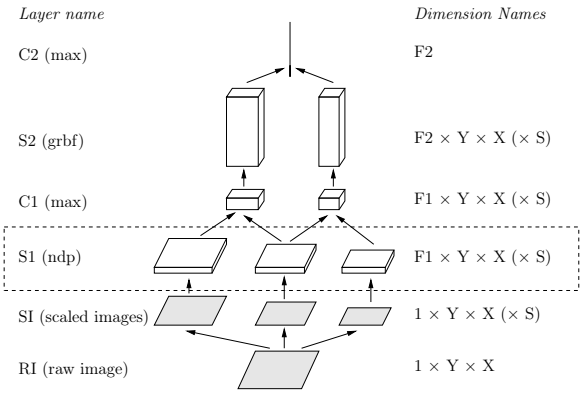


Figure 8.1: HMAX models algorithm overview.

In the present case study, we only focus on the first layer of the HMAX Models algorithm (see Figure 8.1) as it is the most computationally intensive. In a pre-processing phase, the input raw image is converted to grayscale input (only one input feature: intensity at pixel level) and the image is then sub-sampled at several resolutions, 12 scales in our case[1]. For the S1 layer, a battery of three 2D-Gabor filters is applied to the sub-sampled images and then for C1 layer, the spatial max of computed filters across two successive scales is taken. Figure 8.2 shows an example of an input image (8.2a), an example image obtained after the pre-processing and the sub-sampling phases, scale 180 (8.2b), and the outputs of the S1 layer (three direction with respect to three 2D-Gabor filters, 8.2c 8.2d, 8.2e). In this application, parallelism can be exploited at several levels. First, at layer level, where independent features can be computed simultaneously. Second, at pixel level, that is, the computation of contribution to a feature may be distributed among computing resources.

8.2 Hw Architecture Overview

8.2.1 The STHORM Platform

STHORM [155] is a many-cores system consisting of a host processor and a many-core fabric. The host processor is a dual-core ARM cortex A9. The STHORM fabric comprises computing clusters, inter-connected via a high-performance fully-asynchronous (2D mesh structure) network-on-chip (NoC), which provides communication with high, scalable bandwidth. Each cluster aggregates a multi-core computing engine, called ENCore, and a clus-

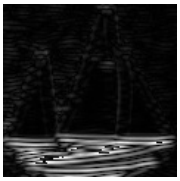
1. 256, 214, 180, 152, 128, 106, 90, 76, 64, 52, 44, 38



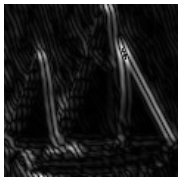
(a) Input image



(b) Sampled gray scale image



(c) Output image direction 1



(d) Output image direction 2



(e) Output image direction 3

Figure 8.2: Example of input and output images of the HMAX S1 layer.

ter controller (CC). The ENCore embeds a set of tightly-coupled processor elements (PE) which are customizable 32-bit STxP70-v4 RISC processors from ST Microelectronics. On the STHORM test-board used in our experiments, the fabric comprises 4 clusters, with 16 PEs each. The PEs in one cluster share a multi-banked level-1 (L1) data memory of 256 KBytes. The banks of the L1 memory can be accessed in parallel in one processor cycle. Each PE has its private instruction cache with a size of 16 KBytes.

The CC consists of a cluster core (STxP70-v4), a multi-channel advanced DMA engine, and specialized hardware for synchronization. The latter two are accessible also by the PEs. The CC interconnects with two interfaces: one to the ENCore and one to the asynchronous NoC. All clusters share 1 MByte of level-2 (L2) memory, accessible via the NoC. The access time is several tens of cycles. A DDR3 level-3 (L3) memory is available off-chip (1 GByte); this memory has a large size, however its access time and bandwidth are much slower than the ones of the on-chip memory.

In summary, due to area and power constraints, the fast memory available on the chip is scarce. Furthermore, the host processor and the cores on the fabric have a different instruction-set-architecture. These two aspects make efficient programming on STHORM a challenging task.

8.2.2 The Multi-core Communication API

The main objective of the MCAPI implementation on STHORM is to offer a homogeneous programming interface for the entire platform (fabric and host). This uniform representation covering the entire platform can perfectly integrate the full design flow targeting STHORM, which significantly

eases code generation and analysis. The current MCAPI implementation features five domains: one for each cluster on the fabric, and one for the host as shown in Figure 8.3.

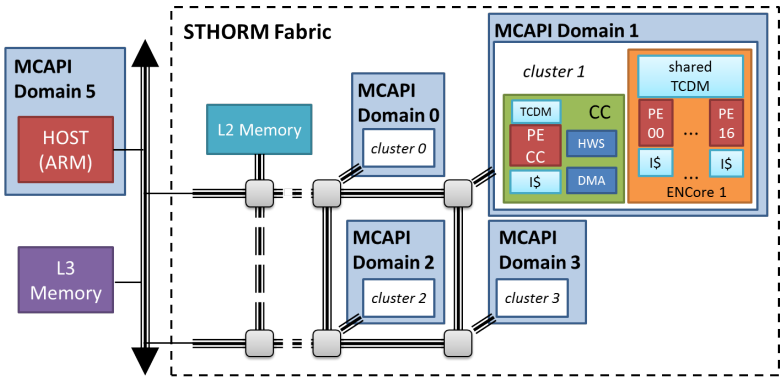


Figure 8.3: MCAPI domains on STHORM.

The five domains naturally reflect the STHORM hardware organization. Furthermore, in each cluster-domain each MCAPI node is mapped on a PE. The host-domain has only a single node corresponding to the ARM dual-core processor. The host node is responsible for the deployment of the entire execution, as it is the main entry point of an application. The MCAPI initialization of the host node automatically loads the fabric binary code into the L2 memory, and starts the fabric MCAPI nodes. The MCAPI implementation totally hides from application the complexity dynamic loading of of binary and symbols dependency solution. A single semantic for the entire STHORM platform is exposed for homogeneous programming.

The channels are implemented using FIFO buffers allocated in any of the memory levels available on STHORM. The size of the allocated buffer and its memory placement can be set by using endpoint attributes, as specified in the MCAPI standard. The sending endpoint and the receiving endpoint must have consistent attributes definition for the creation of a channel. No MCAPI node is mapped on the CCs, meaning that they are not directly visible to the programmer. Instead, they are used internally in the MCAPI implementation to support the various communication mechanisms and synchronizations. DMA engines are also hidden from the programmer, but used by the implementation to transfer data between memory levels and domains.

8.3 Performance Requirements Overview

As stated in the beginning of this chapter, we will mainly focus on the timing dimension of the system, which will constitute the efficiency criterion. We are interested on the overall execution time and the time to process single

lines of the input image. More precisely, we will compute the probabilities that the overall execution time is always lower than a given bound Δ and that the variability in the processing time of successive lines is always bounded by Ψ. To this end, we specify the above requirements in BLTL as follows:

1. *Overall execution time*: $\phi_1 = \mathsf{G}^I(t < \Delta)$, where t is the monitored overall execution time.
2. *Variability of the time of processing successive lines*: $\phi_2 = \mathsf{G}^I(|tl| < \Psi)$, where tl is the difference between the processing time of successive lines.

8.4 Functional and Performance Modeling

8.4.1 High-Level Reconfigurable KPN Model

We developed a parametric KPN model for the S1 layer of HMAX in DOL. The model is based on the worker pattern presented previously. It uses a certain number of reconfigurable processes for implementing the 2D-Gabor filtering and image splitting/joining. Every image is handled by one *processing group* consisting of one *Splitter*, one or more *Gabor* (*Worker*) and one *Joiner*, connected through blocking FIFO channels as illustrated in Figure 8.4. This model exploits parallelism both at image level, as different images are processed in parallel by different processing groups and at pixel level, as different stripes of the image are processed in parallel by different *Gabor* processes. Moreover, parallelism is exploited between pure computation on *Gabor* processes and data transfer from/to main memory by *Splitter*/*Joiner* processes.

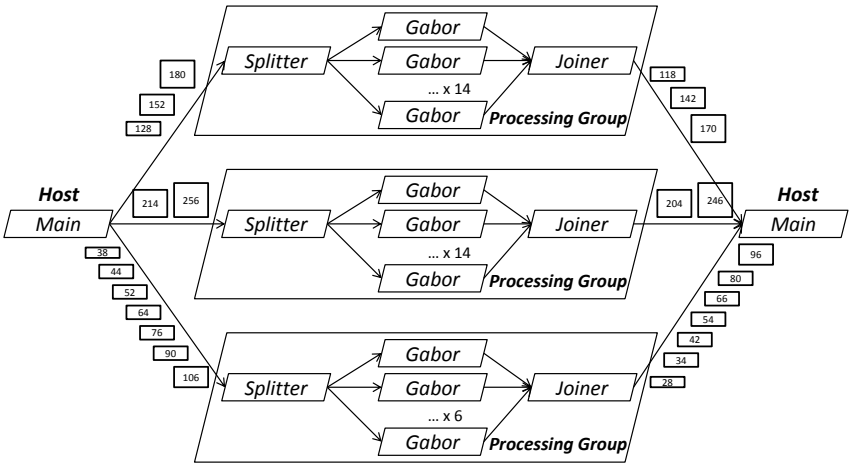


Figure 8.4: An abstract view of the S1 layer of HMAX models algorithm.

The computation of the entire S1 layer is coordinated by a single main process. Several image scales are handled concurrently. That is, the twelve scaled images are statically pre-allocated and mapped on different processing groups. For every image scale, the processing is pipelined as follows. Initially, the main process sends the first $10 + P$ lines to the corresponding processing group, where $P \geq 0$ is an integer parameter called *line pressure* that specifies the pipelining rate. In normal regime, one input line is sent and one output line is received, for every filter rotation (that is, actually three output lines). Finally, once all the input image has been sent, the main process receives P more output lines. At this point, the processing group is ready (empty) and can be reconfigured to restart computation for another image scale.

```
/** DOL Gabor Process **/
void gabor_init()
{ status = CONFIG; }

void gabor_fire(){
  if(status == CONFIG){
    /* Read configuration parameters
    from input configuration FIFO */
    read(CONF_FIFO, &config);
    /* Update number of steps */
    step = config.step;
    /* Update status */
    status = EXEC; }
  if(status == EXEC){
    /* Read data from input data FIFO */
    read(DATA_FIFO_IN, &data);
    /* Execute computation function */
    compute_filter(&data);
    /* Write data to output FIFO */
    write(DATA_FIFO_OUT, data);
    /* Update number of steps */
    step--;
    /* Update status */
    if(step == 0) status = CONFIG;
  }
}
```

Within the processing group, the *Splitter* receives input images, line by line from the main process. Every line is split into a number of equal length (and overlapping) fragments, one for every *Gabor* process, and sent to these processes. *Gabor* processes implement the computation of the 2D-filter itself (A DOL code sample of the *Gabor* process is given above). Filter size is fixed to 11×11 in the case study. Hence, *Gabor* processes need to accumulate 11

line fragments in order to perform computation. Henceforth, they maintain and compute the result operating on an internal "sliding" window of 11 line fragments. Finally, the resulting fragments are sent further to the *Joiner*, which packs them into complete output lines and sends them to the main process.

In this experiment, for the sake of simplicity, we only consider one image scale (256×256), that is, one processing group will be actually used as shown in Figure 8.5. Besides, reducing the model size, this restriction relaxes data-dependency since a single input size is considered. It is worth mentioning that, given one image scale as input, each process will always handle the same workload (amount of data) which increases the statistical learning confidence.

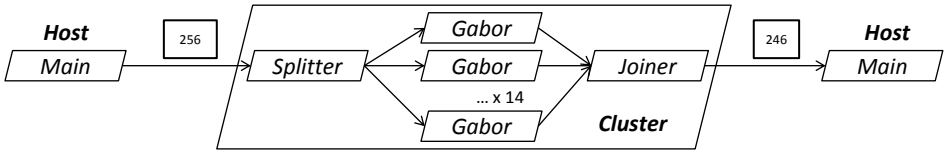


Figure 8.5: The simplified model of the S1 layer of HMAX models algorithm.

Given the KPN model above, we generate a BIP model using the DOL2BIP tool described in the previous section. We depict in Figure 8.6 the obtained BIP component for the *Gabor* process, which consists of an automaton that reproduces the behavior described earlier. That is, a configuration step in l_0 given the configuration data (this specifies for instance the number of fragment to process denoted steps in Figure 8.6), followed by several iterations (from l_1 to l_4, which depends on the number of steps to perform) consisting of reading a fragment from the input queue DATA_FIFO_IN, computing the filer, and writing the results into the output queue DATA_FIFO_OUT. The DOL2BIP transformation produces in addition FIFO components, as described earlier. Figure 8.7 shows such a component.

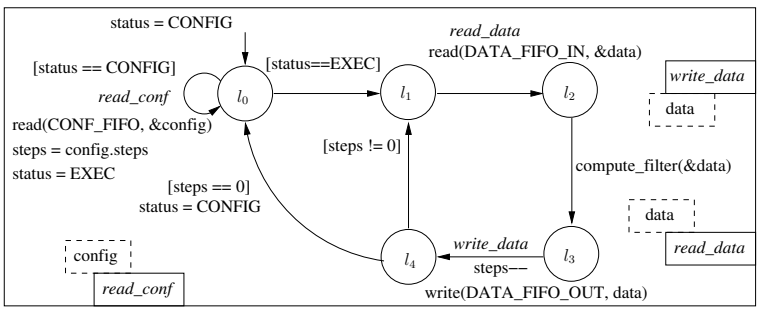


Figure 8.6: A functional BIP model of the *Gabor* process.

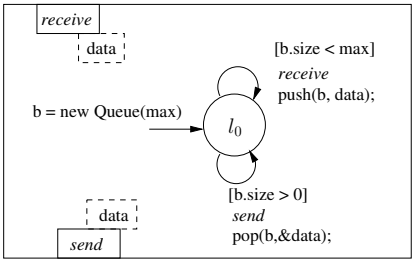


Figure 8.7: A BIP model of a FIFO buffer.

Once we generate the functional BIP model of the application, we produce a distributed implementation using the BIP2MCAPI code generator tool. Note that in this case study we are not producing a complete system model (composed of the application mapped to the architecture) as described in in Chapter 5. Since we are given the target platform, we are not considering a model of the architecture but only a model of the application and a mapping. A sample of generated code, corresponding to the *Gabor* process, is shown below. The produced code is instrumented in order to observe execution and communication time of each process. In this case study, we rely on a physical STHORM test-board in order to gather low-level performance data. The generated implementation is therefore deployed and executed and the corresponding performance traces can be directly produced.

```
void gabor_ins_execute(void* args) {
/* Initialization part */
...
/* Behavior part */
while(Wlcontinue){
 switch(BIP_CTRL_LOC){
  case L0 : {
   if (status == CONFIG){ ...
    status = EXEC; BIP_CTRL_LOC = S0; }
   if (status == EXEC) BIP_CTRL_LOC = S1;
   break; }
  case L1 : {
   mcapi_pktchan_recv(h_WORKER_read_data, (void**)&mcapi_buffer,
   &mcapi_received, &mcapi_status);
   if((mcapi_received != size) || (mcapi_status != MCAPI_SUCCESS))
    ERR_RAISE("FAIL TO READ DATA");
   memcpy(data, mcapi_buffer, mcapi_received);
   size = mcapi_received;
   mcapi_pktchan_release(mcapi_buffer, &mcapi_status);
   if(mcapi_status != MCAPI_SUCCESS) ERR_RAISE("FAIL TO RELEASE CHANNEL");
   BIP_CTRL_LOC=S2 ;
   break; }
  case L2 : {
   compute(&data); BIP_CTRL_LOC=S3;
   break; }
```

```
  case L3 : {
   mcapi_pktchan_send(h_WORKER_write_data, data, size, &mcapi_status);
   if(mcapi_status != MCAPI_SUCCESS) ERR_RAISE("FAIL TO SEND DATA");
   step--; BIP_CTRL_LOC=S4;
   break; }
   ...
  }
 }
}
```

8.4.2 Performance Characterization and Model Calibration

Now that we dispose of execution traces that concern the performance dimension of interest, that is time, we apply the distribution fitting process to learn probability distributions that characterizes these data. We illustrate the different steps of the process on the execution time of the *Gabor* process. *Exploratory Analysis* is first performed to observe if the data provide any clues to resemble to a standard probability distribution. Runs and Box plots are initially used to observe the data shape and evolution. The corresponding

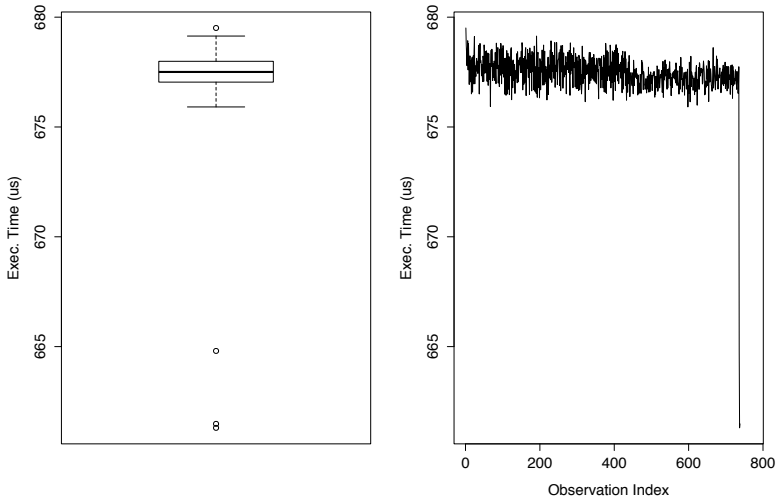


Figure 8.8: Box-Whisker and Runs plots of the *Gabor* execution time. Extreme observations correspond to the end of the process execution.

plots are presented in Figure 8.8, which reveal the presence of very extreme values clearly deviating from the other observations, these are considered as outliers [2] and are eliminated from the sample under study. Figure 8.9 shows the same plots after removing outliers.

2. http://www.itl.nist.gov/div898/handbook/eda/section3/eda35h.htm

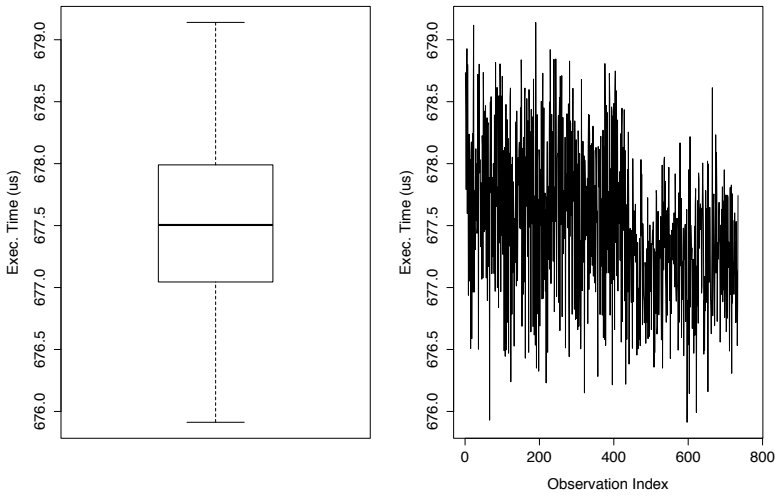


Figure 8.9: Box-Whisker and Runs plots of the *Gabor* execution time observations after outlier elimination (observation are more compact).

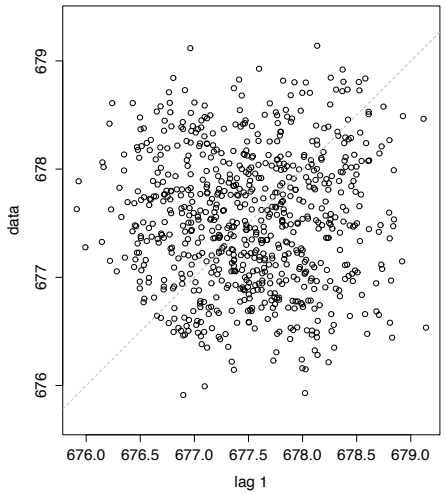


Figure 8.10: Lag plot of the *Gabor* execution time observations. It shows random spread of the data (no specific shape is appearing).

To check if the data observations are independent, we first use the Lag plot. This draws the observations x_i in the y-axis and x_j in the x-axis, where $i - j$ is the fixed lag. For instance, Figure 8.10 shows the Lag plot of the *Worker* execution time with lag equal to 1. The figure clearly shows a random dispersion of the observations. To get more confidence, we used the

Ljung-Box and the Box-Pierce tests at significance level of 0.05. These gave respectively 0.0531 and 0.0533 as p-values which confirms the independence assumption.

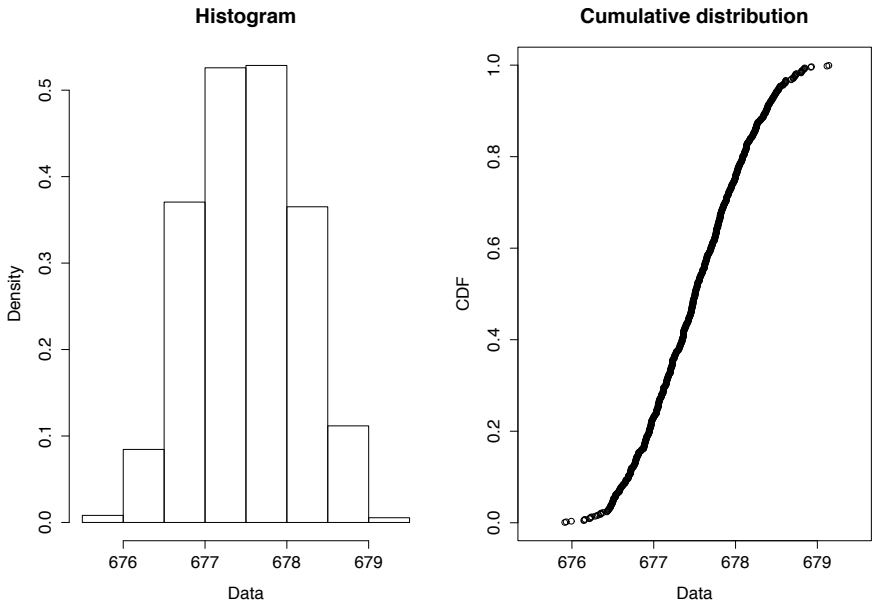


Figure 8.11: Histogram and CDF of the *Gabor* execution time observations.

Finally, we use the histogram and the CDF shown in Figure 8.11 to observe the shape of the data. One can see out of this figure that the data is uni-modal and symmetric which means that it may be potentially generated from bell-curved process. We use the Cullen and Fray graph illustrated in Figure 8.12 to get more insight with respect to the Skewness and the Kurtosis of the data. The figure shows that the observations are seemingly Normal, however, according to the same figure, Lognormal, Gamma, and Beta distributions are also good candidates.

The second step in the distribution fitting process is to fit the candidate distributions to the data, that is, *Parameters Estimation* from the data. Note that all the identified candidates have the same number of parameters, which is 2. Concretely, the Normal and the Lognormal distributions are defined in terms of a mean μ and standard deviation σ. The Beta distribution has two shape parameters α and $\beta$and the Gamma distribution has a shape parameter k and the scale parameter θ. Moreover, the Beta distribution is defined over $[0, 1]$, thus a pre-processing of the data is first required before the parameters estimation. Since it does not provide any advantage with respect to the other candidates, like using less parameters, we are not going to consider it for the fitting. We are thus left with the Normal, the Lognormal, and the Gamma candidates. Since the Normal and the Lognormal are

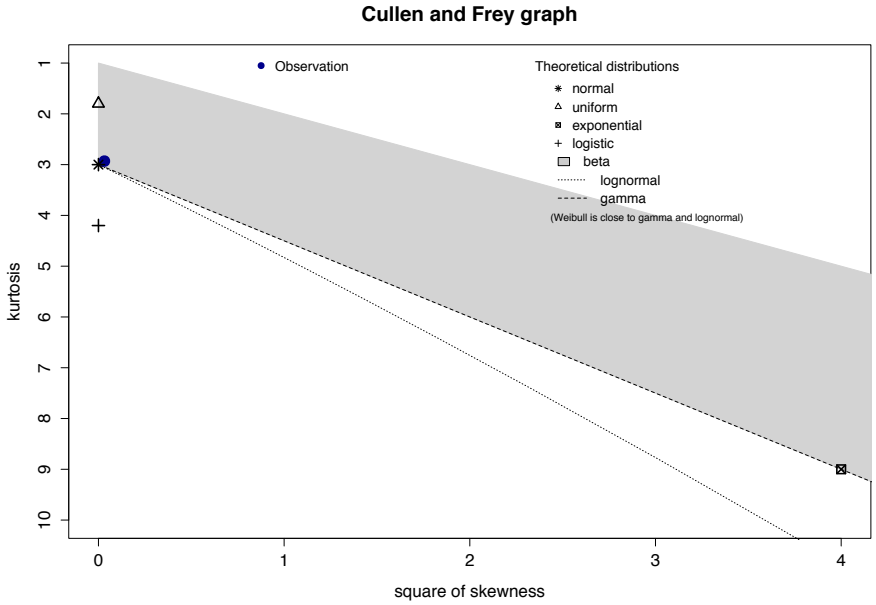


Figure 8.12: Cullen and Frey graph for the *Gabor* execution time.

quite similar [3], we only consider the Normal and the Gamma distributions as pertinent candidates for the fitting process. Observe that the above choices are mostly subjective and are mainly based on human interpretation.

We now move to estimate the parameters of the retained candidates. To this end, we used the *method of moment* (MME) described in the previous chapter, which gives the following estimates:

- For the Normal candidate: $\mu = 677.52\mu s$ and $\sigma = 0.626\mu s$.
- For the Gamma candidate: $k = 1172721.38\mu s$ and $\theta = 1730.909\mu s$

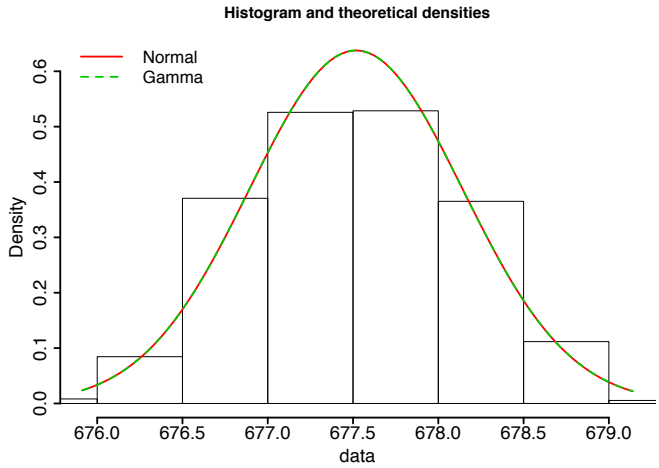
The final step in this fitting process is to *Evaluate the Obtained Fits* through a goodness-of-fit test. We used the different test statistics described in the previous chapter, that is, Kolmogorv-Smirnov (K-S), Carmer-Von Mises (C-VM), and Anderson-Darling (A-D), for which the results are given in Table 8.1.

	K-S	C-VM	A-D	AIC	BIC
Normal	0.0331	0.1931	1.342	1398.534	1407.731
Gamma	0.0330	0.1930	1.340	1398.517	1407.714

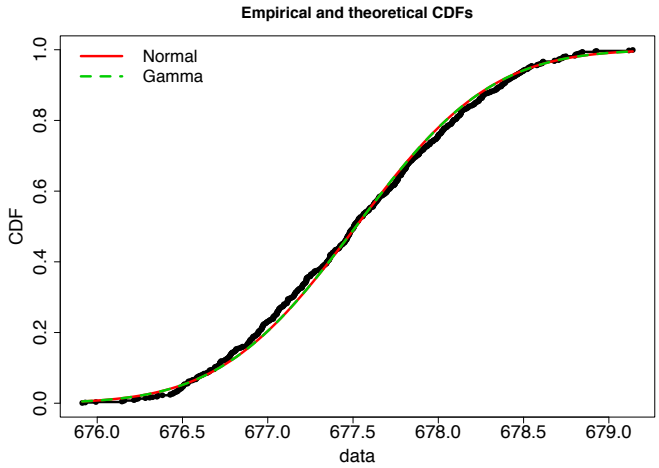
Table 8.1: Goodness of fit evaluation results for the Normal and the Gamma distributions.

3. They can be obtained from each other using a Log transformation.

The goodness-of-fit tests were performed at a significance level of 0.05. In Table 8.1, two results out of three (Carmer-Von Mises (C-VM) and Anderson-Darling (A-D)) show that both candidates provide good fits, since the obtained p-values in each case are greater that 0.05. Thus the we cannot reject the hypothesis stating that the data is normally distributed with $\mu = 677.52\mu s$ and $\sigma = 0.626\mu s$, neither the hypothesis stating that the data follows a Gamma distribution with $k = 1172721.38$ and $\theta = 1730.909$. In Table 8.1, we give in addition the obtained AIC and the BIC scores, which are very similar in this case and do not help us to select. We rely on additional plots to visually compare the two candidates. In Figure 8.13, we show



(a) Comparison of PDFs

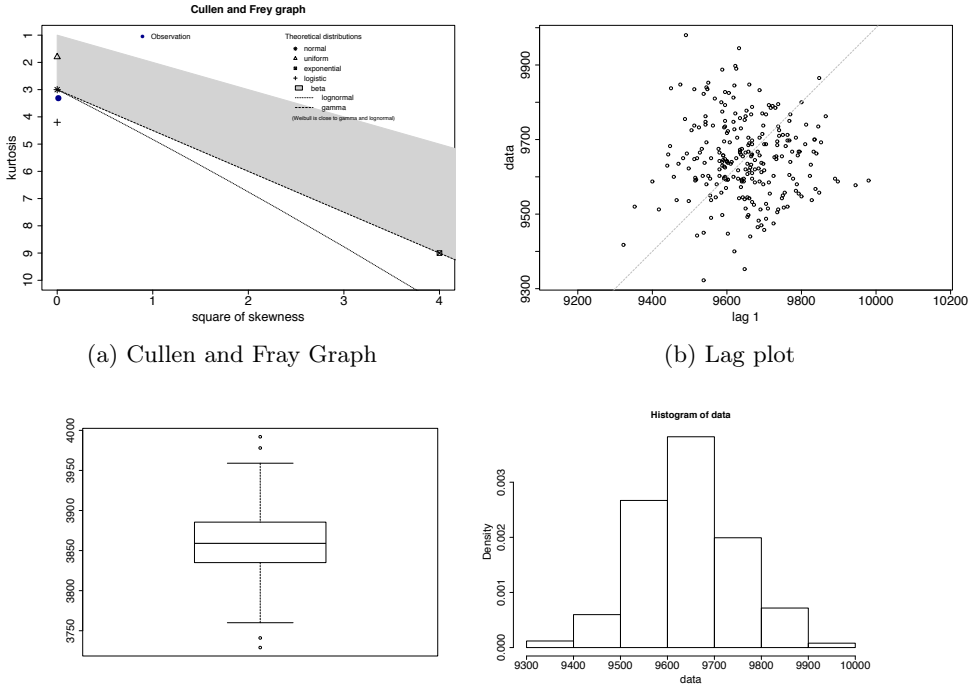


(b) Comparison of CDFs

Figure 8.13: Graphical comparison between the Normal and the Gamma fits. The obtained curves for both fits are very similar.

a comparison between the obtained candidates in terms of their probability density functions (PDFs) and cumulative distribution functions (CDFs). The two fits provide very similar fits to the observations and it is almost impossible to distinguish between them. In such a situation, the decision is left to the designer. For instance, in our case, we choose the Normal distribution since it provides interesting properties and is easier to implement.

We applied the same steps for the processes communication times and learned similar distributions. In Figures 8.14 and 8.15 we show plots illustrating the fitting process: a Cullen and Fray Graph (8.14a), a Lag plot (8.14b), a Box-Whisker plot (8.14c), and a histogram (8.14d) for the reading time of the *Splitter* process. These shows that the data are basically Normal. The parameters estimation using the MME method provides that $\mu = 9649.83$ and $\sigma = 105.92$. The goodness-of-fit evaluation at 0.05 significance level gives the following: using K-S, 0.0592, using C-VM, 0.0861, and using A-D, 0.4619. These results provide no clues against our hypothesis that the data is normally distributed with the estimated parameters μ and σ. Figure 8.15 shows a graphical summary of the obtained Normal fit (PDF, CDF, Q-Q plot, and P-P plot). The learned communication time only concern the data transfer time and does not include the blocking time as it is already captured in functional part of the model as explained in Section 5.4.



(a) Cullen and Fray Graph

(b) Lag plot

(c) Box-Whisker plot

(d) Histogram

Figure 8.14: Exploratory Analysis for *Splitter* reading time.

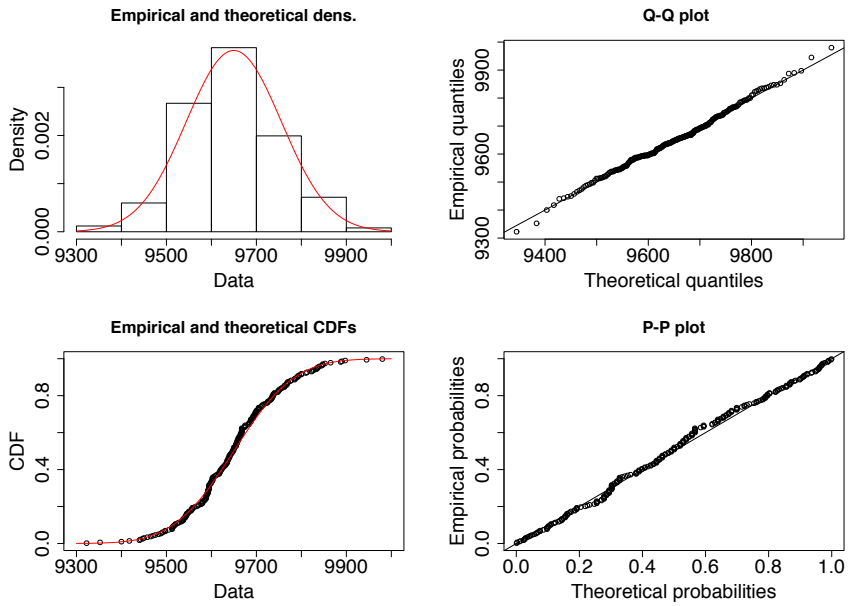


Figure 8.15: Normal Fit Summary for *Splitter* reading time.

It is worthwhile mentioning that, in this case study, only the *Gabor* process computation time, that is, the *compute_filter* function was considered since we observed that it is the most time consuming. For the other processes (*Splitter* and *Joiner*), only the communication time was characterized. These are actually doing very light computation consisting of cutting and assembling data. We also draw the reader attention to the fact that the learning of the computation time behavior of the *Gabor* process is performed only for one process. Since all the running *Gabors* in the system are instances of the same component which run the same *compute_filter* function and execute on identical cores, it is safe to calibrate them with the same probability distribution. This actually an advantage of using probabilistic characterization.

Once all the performance aspects of interest are characterized (in this case, we recall that we only consider time), calibration is performed by annotating the functional BIP model using the learned distributions as illustrated in Chapter 5. Figure 8.16 shows an example of calibrated BIP component. This corresponds to the *Gabor* process in Figure 8.6, which is augmented with timing information that concerns communication, i.e read and write time, and the computation time of the *compute_filter* function, which represents the core computation part of the process. Three different probability distributions where learned to this end, namely μ_r for the read time, μ_w for the writing time, and μ_c for the computation time. Note that the calibration phase implies also to modify the FIFO components, as we did in Chapter 5, that is, by decomposing the push and pop actions into begin and end.

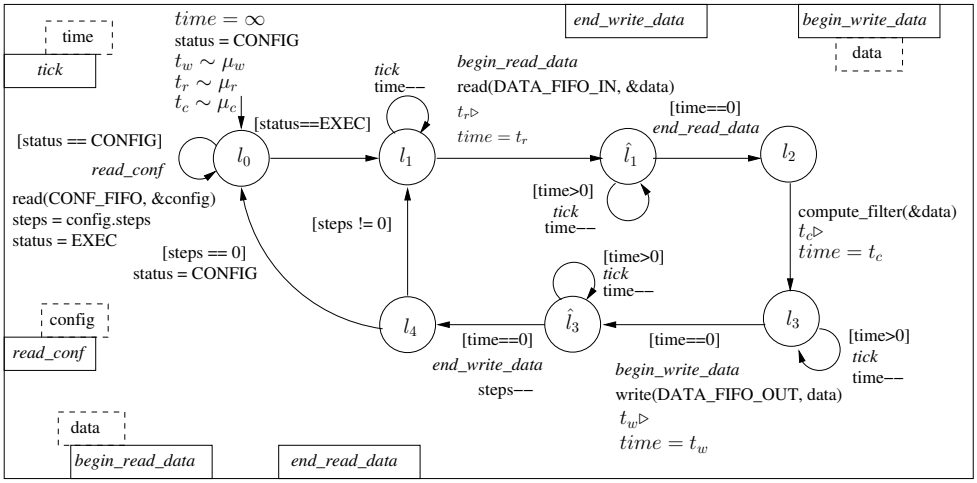


Figure 8.16: The calibrated model of the *Gabor* process. An $\mathcal{S}$BIP model that represents a part of the obtained performance model of the designed system.

8.5 Performance Evaluation

Before using SMC to check the system-level timing requirements, we wanted to validate the calibrated model with respect to the actual implementation. To this end, we compared the overall execution time obtained by 1) running the generated HMAX implementation on the test-board and 2) simulating the calibrated $\mathcal{S}$BIP model. We observed that the time on the model is about 20% lower than what we obtained on the test-board. This result is expected since the calibrated model does not take into account all the implementation delays. For instance, the splitting and joining time were not introduced in the model. Moreover, high-level models are generally more optimistic due to abstraction.

Now that we built a high-level performance model that encompasses the functional behavior and which is calibrated with timing information, it can be used for performance evaluation using SMC. We recall that the obtained model is an $\mathcal{S}$BIP model where non-determinism is resolved using priorities on one hand, and uniform schedulers on the other (see Chapter 3 for more details). We use the SPRT algorithm implemented within The BIP$^{\mathcal{SMC}}$ statistical model checker with confidence parameters $\alpha = \beta = 0.001$ and $\delta = 0.05$. We checked the aforementioned performance requirements, i.e., $\phi_1 = \mathsf{G}^l(t < \Delta)$ and $\phi_2 = \mathsf{G}^l(|tl| < \Psi)$, for different pipelining rate $P \in \{0, 2\}$. For this case study, we have used arbitrary FIFOs sizes as follows: *Main-Splitter*= 10 KB, *Splitter-Worker*= 112 B, *Worker-Joiner*= 336 B, and *Joiner-Main*= 30 KB. Note that such a choice might be also explored using our approach.

Δ (ms)	572.75	572.79	572.8	572.83	572.85	572.87	572.89	572.91	572.95
$P(\phi_1)$	0	0.2	0.28	0.57	0.75	0.91	0.98	0.99	1
Traces	66	457	1513	1110	488	954	171	89	66

Table 8.2: $P(\phi_1)$ and number of used SMC traces when varying Δ $(P = 0)$.

Table 8.2 shows the probability evolution of ϕ_1 for different Δ and the corresponding required SMC traces. This first analysis is performed with no pipelining, i.e., $P = 0$. One can for instance conclude out of this table that the overall execution time is always lower than $572.95ms$ with probability 1. What would be interesting to verify is for instance a trade-off between the obtained timing bound and the used resources in terms of memory utilization. That is, when decreasing the FIFOs sizes, how the overall execution time is impacted? This is an ongoing work for the moment.

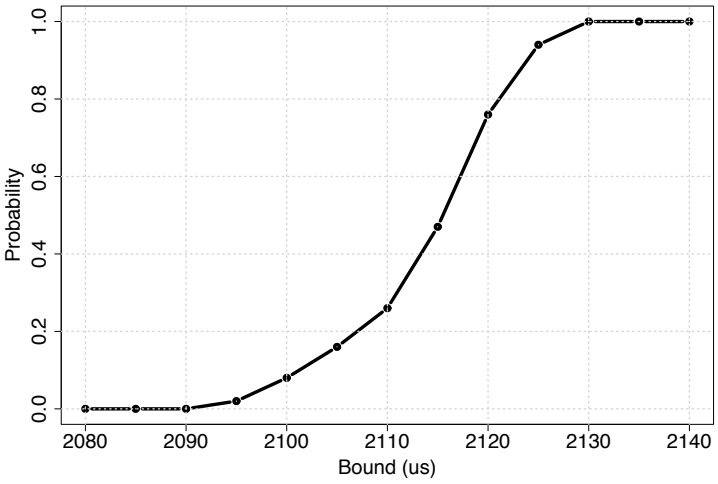
In Figure 8.17, Tables 8.3 and 8.4, we present two results of verifying ϕ_2 when varying the bound Ψ. The curve on 8.17a is obtained with no pipelining, i.e., $(P = 0)$, while the one on 8.17b is obtained with $P = 2$. The two curves have similar evolution with a small difference in the bounds. The second curve $(P = 2)$ has actually greater values, which reflect more variation. We recall that when $P = 0$, all the processes are perfectly synchronized which yields to small variation over line processing time. Using $P > 0$ leads to greater variation since it somehow alter this synchronization. We finally mention that the SMC time was relatively small given the model size: 5 hours in average for each property. The BIP model has 47 BIP components and $\sim$ 6000 lines of code. Components have in average 20 control locations and 10 integer variables each, which induces a big state space.

Ψ (μs)	2095	2100	2110	2115	2120	2125	2130
$P(\phi_2)$	0	0.1	0.24	0.47	0.76	0.94	1
Traces	66	65	255	1036	2371	410	66

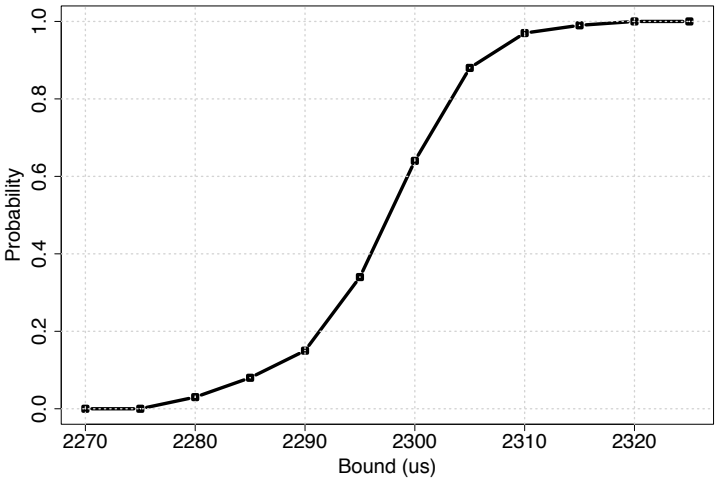
Table 8.3: $P(\phi_2)$ and number of used SMC traces when varying Ψ $(P = 0)$.

Ψ (μs)	2280	2285	2290	2300	2305	2310	2315	2320
$P(\phi_2)$	0	0.1	0.14	0.64	0.89	0.98	0.99	1
Traces	66	418	394	752	853	206	65	66

Table 8.4: $P(\phi_2)$ and number of used SMC traces when varying Ψ $(P = 2)$.



(a) $P = 0$



(b) $P = 2$

Figure 8.17: Probabilities of ϕ_2 for $P \in \{0, 2\}$.

8.6 Conclusions

This chapter covered a case study that concerns an image recognition application running on a many-cores platform. The application consists of the HMAX models algorithm for objects recognition, which is originally a sequential algorithm. The goal was to design a parallel version of this application and to efficiently deploy it on the target many-cores platform, that is in this case, STHORM. We used the *ASTROLABE* tool-flow to this end. First,

we built a KPN model of the application following a join/split schema. Using the DOL2BIP tool, we obtained a functional BIP model, which was used to generate a distributed implementation of the application and to deploy it on STHORM given a certain mapping. We then performed statistical learning to characterize timing information and to derive a performance model, which was finally analyzed using SMC against the timing requirements.

Due to time and technical issues, we were not able to explore several aspects of the system. Actually, the STHORM project was unfortunately shut-down by STMicroelectronics and we were only able to experiment on a test-board provided by CEA for limited use. We hence performed only timing analysis for a single mapping and on a sub-part of the system. What would be interesting is to explore the full version, that is, processing all the input image scales and to analyze trade-offs between time and memory, especially because memory is scarce on the fabric side of the platform. Exploration could also cover several mappings and various images allocation settings. A preliminary work in this direction can be found in a technical report [164]. It is worth mentioning that for this study, we did not use the abstraction technique introduced in Chapter 4, since the obtained performance model was affordable using bare SMC algorithms.

Chapter 9

Conclusions and Perspectives

Conclusions

The main objective of the present work is to enable better support of performance aspects in system-level design of many-core embedded systems. As exposed along the manuscript, our belief is that such support can only be possible within a holistic and rigorous approach that provides systematic and tool-supported steps performed in an iterative and incremental fashion. Our goal was thus to conceive an approach that follows these guide-lines and that eventually leads to a correct and efficient implementation of a given system. To precisely identify how to handle performance at system-level, we analyzed the existing and sought challenges in this area. We were able to identify modeling and analysis as essential activities and to specify requirements that must be satisfied to answer the identified challenges.

We proposed the $\mathcal{S}$BIP stochastic component-based modeling language and its associated formal semantics as answer to rigorousness, incrementality, and expressiveness requirements. The proposed modeling language is seen as the spine of the conceived approach. It enables to capture different aspects of embedded systems, especially performance and induced variability, through the proposed stochastic semantics that allows for modeling probabilistic and non-deterministic behaviors, i.e., MDPs. Designing following a component-based fashion enables natural thinking and building of systems and helps mastering complexity. Furthermore, formal semantics allows for rigorous and automated reasoning about the designed system.

On top of the $\mathcal{S}$BIP formalism, we built a probabilistic model checker, namely BIP$^{\mathcal{SMC}}$, in order to perform quantitative analysis, which is of paramount importance for performance evaluation. This tool uses a mix of simulation and statistical-based approaches, which provide a good trade-off between accuracy, rapidity, and scalablity requirements. We presented an example of using the $\mathcal{S}$BIP formalism and the BIP$^{\mathcal{SMC}}$ tool for verifying

a multimedia system. As for any formal analysis technique, SMC suffers from state space explosion issues. To bypass this hindrance, we proposed an abstraction technique that computes a smaller stochastic model given a certain requirement. The suggested technique uses projection with respect to a given property to keep only the relevant behavior of the system, and machine learning to build a compact representation. The method was shown to converge in the limit to a small yet accurate representation of the original system.

The above contributions, which were presented in the first part of the manuscript, besides being an important part of the theoretical foundation of the proposed performance modeling and analysis approach, they represent a generic and a formal framework for rigorous modeling and quantitative analysis of stochastic systems. They may be used in a different context than performance modeling and analysis at system-level. This is the reason we choose to present them in a separate part.

An important challenge that was identified during the analysis of requirements for system-level performance modeling is to figure out the appropriate level of abstraction of the system model to build. To produce a trustworthy design, such models must capture a sufficient amount of information that concern functional and performance aspects of the system, that is, they must be faithful. Given this requirement, we conceived a modeling approach that aims at producing performance models offering good trade-offs between the level of abstraction and the degree of faithfulness required in early design phases. The proposed method follows a meet-in-the-middle scheme and is composed of three main tasks, namely, code generation, distribution fitting, and models calibration. The idea of the approach is to combine behavioral information coming from high-level functional models and low-level performance details obtained from real executions or low-level simulations.

In spite of being part of the performance modeling approach, the code generation contribution answers the requirement of quickly programming and software prototyping on many-cores platforms. The goal of this task is to generate a distributed implementations and the associated deployment of a given high-level functional model on a target many-cores platform. Such implementation allows for concrete executions and thus for observing real performance evolution, which is enabled through code instrumentation with respect to the performance dimension of interest. In order to faithfully capture such information and to characterize it, we opted for probabilistic modeling which enables to capture performance fluctuations and allows for operating natural abstraction of irrelevant details. To this end, we used a statistical inference process, more precisely distribution fitting, which consists of learning the best probability distribution that characterizes the performance data. Finally, we combine the high-level functional behavior with the probabilistic performance characterization to obtain a stochastic performance model using a model calibration step. Calibration consists of injecting the learned

probability distributions within the functional model by introducing probabilistic variables and time progress in the case of timing performance for instance.

All the above steps compose the *ASTROLABE* integrated approach we proposed for performance modeling and analysis at system-level. In this approach, performance modeling relies on the steps above and the obtained performance model is analyzed with respect to performance requirements using the stochastic abstraction and the statistical model checking techniques. We also depicted a tool-flow covering most of the steps of the *ASTROLABE* approach, e.g., DistFit tool for distribution fitting and BIP2MCAPI code generator for the STHORM many-core platform and the MCAPI runtime. The approach and its associated tool-flow was illustrated on a fragment of a real-life case study for image recognition. The initial results of this study were encouraging and showed that good performance models can be obtained quickly and easily from high-level specifications. Moreover, analysis using statistical model checking provided quick and accurate results. Analysis is for now done only for timing aspects and we are planning to extend it for further performance dimensions as stated in the perspective section hereafter.

Perspectives

In the above contributions, several amelioration and extensions are sought to enable covering additional aspects and hence ensure more generality for the proposed approach for system-level performance modeling and analysis.

The $\mathcal{S}$BIP Extension

The $\mathcal{S}$BIP formalism can be extended at different levels. First, with respect to time granularity. As presented in Chapter 3, the proposed semantics only covers discrete-time, i.e., MDPs and LMCs. It would be interesting to extend this semantics to cover in addition continuous time. Such extension would help to build models that capture timing aspects more naturally. For instance, when calibrating functional models with probability distributions characterizing computation or communication time, one can think of adding stochastic clocks that evolve continuously with respect to the learned probability distributions, instead of adding probabilistic variables and modeling time evolution via discrete transitions. To do so, we may rely on the real-time semantics of the BIP formalism [1] where time is a first-class concept, that is, a full-fledged part of the semantics. In contrast, to the BIP semantics, in real-time BIP, time is not explicitly simulated using discrete transitions, e.g., *tick*, but is implicitly handled by an engine that implements the real-time semantics.

Considering stochastic clocks is not trivial, especially in a component-based formalism because of the induced complexity of parallel composition of components. Various works have proposed models combining time and stochastic behavior such as Stochastic Petri nets [4], stochastic process algebra [112, 114], stochastic automata [71, 72], stochastic timed automata [70, 36] used for instance in MoDest, and priced timed automata used in Uppaal [74]. These models often have CTMCs or more general GSMPs [98, 7] semantics.

A second amelioration in $\mathcal{S}$BIP may cover the probabilistic schedulers, which are currently limited to uniform distributions. As stated in Chapter 3, this is the default choice we made for simplicity of implementation. Allowing additional probability distributions for schedulers, implies to extend the $\mathcal{S}$BIP modeling language. This can be done for instance using weights added on interactions or on the ports of the components. Given a set of enabled interactions at some point, a normalization is then performed in order to compute the underlying probabilities and to accordingly select the interaction to fire. We may also extend the proposed formalism with a reward semantics. This will be useful when considering other performance aspects such as energy and temperature. We believe that such extension is straightforward if we use $\mathcal{S}$BIP variables and associate them with reward functions. This is for instance the case in Uppaal [74] and Prism [139].

Stochastic Abstraction

In the present work, we used the AAlergia algorithm to instantiate our approach for automatically building sound abstractions. We recall that the approach is limited to bounded LTL properties when using algorithms only able to learn deterministic models such as AAlergia, and when the used data contains non-determinism. This represents a hindrance towards checking unbounded properties using probabilistic model checking for instance, especially that the proposed projection potentially introduces non-determinism in the data. We may use other algorithms able to learn non-deterministic models, such as[1] [194, 193, 154, 180] to remedy to this issue. However, we recall that when learning non-deterministic models, it is only possible to compute upper and lower probability bounds for various schedulers required to resolve non-determinism.

With respect to the projection part, there are also possibilities of improvement. The current projection definition enables building abstraction for classes of properties since it considers the support of a given property as the set of its explicitly appearing symbols. We may for example take into account the semantics of LTL operators to define coarser projections and obtain consequently more compact abstractions, albeit this has the limitation

1. Additional work are presented in the related work section in Chapter 4.

of recalculating a new abstraction for each single property.

Remark that the models obtained by using our abstraction technique are LMCs or MDPs. These may be easily transformed into $\mathcal{S}$BIP models as shown in Section 3.4. However, the proposed transformation captures the given model as a single $\mathcal{S}$BIP component, that is, we loose compositionality. It would be interesting to learn $\mathcal{S}$BIP models in a compositional fashion, that is, models having different interacting components. Such approach would require in addition to learn connections or a glue between the obtained components, which is a challenging task. Some work has been done in this direction within the BIP framework [25, 27, 26, 39], although still in a preliminary state. This approach may be directly integrated within a learning algorithm, which is quite difficult, or performed post learning, that is, once an LMC or MDP model is obtained.

Statistical Inference

The statistical inference procedure described in Chapter 6 is another learning technique that aims at inferring probabilistic models from input data. In the present work, we used distribution fitting as to learn probability distributions. A more general approach is model fitting that fits an arbitrary model to the data. In the case of distribution fitting, the target model can be only a probability distribution and obey to some assumptions such as data independence as discussed in Chapter 6. A well known family of model fitting techniques is regression analysis [8, 96, 175, 3]. Techniques as such ARMA and ARIMA can be used to characterize the behavior of a certain random variable as a linear or non-linear function of a single or several other variables.

A learned lesson from the case study we performed in this work is that statistical inference techniques require human expertise and intervention. Various activities in such process rely on human interpretation and are thus difficult to automate, such as the exploratory analysis phase in the proposed distribution fitting approach. A possible fully automated procedure may rely on machine learning. In the related work section in Chapter 4, we discussed several algorithms that are able to learn probabilistic and timed models for instance. These may be appropriate to characterize in more detail timing aspects of embedded systems. Consequently, they may be used in advanced phases of the design to learn more detailed description. Such algorithms produce Markov models and not probability distributions. Thus, model calibration will consist of transforming the learned model to an SBIP component and to inject it in the functional model, which requires to add extra connectors for synchronization.

The *ASTROLABE* Approach

At the level of the proposed *ASTROLABE* approach, we suggest additional future directions. The proposed approach provides quantitative analysis results with respect to performance requirements. However, such feed-backs do not explicitly guide the designer to improve or modify specific parts of the input models or mapping. In design space exploration, the idea is to keep, at each design phase, configurations ensuring functional correctness and good trade-offs between the explored performance aspects. Coming-up with different design alternatives is a separate matter which is more related to human expertise.

Design trade-offs may be obtained by exploring the space of quantitative feed-backs that concern the performance dimensions of interest, e.g., timing, memory utilization, energy consumption. Exploration can be performed manually [92], or using exhaustive [215] or randomized [49, 9] search for instance. Additional possibilities exist such as the reduction to a single objective [57] or problem-dependent approaches [195] (see [136, 31] for a detailed classifications and discussion). These exploration techniques rely on different search and optimization algorithms often multi-objective since they concern various aspects of the systems, which are generally contradictory and of different nature. A well known technique that enables comparison of such heterogeneous aspects is Pareto Dominance, which allows for a partial ordering. Example of approaches used for design space exploration rely on genetic algorithms [48], multi-objective optimization [174, 122], dependency [97] and Pareto sensitivity [90] analysis techniques.

We recall that in our work, we rely on statistical model checking to quantitatively evaluate design alternatives with respect to performance requirements. Such evaluation is based on verifying formal properties expressed in LTL for example. We can formalize properties that cover the different performance dimensions as a conjunction of several aspects. Consequently, one obtains a single probability measure, that quantify the different performance dimensions, per configuration. Now for different configurations it is easier to pick the highest probability.

Note that in the current state of the work we only cover one performance dimension, that is, timing. The reason for this limitation is the distribution fitting approach and the strong assumption of data independence. If we extend our statistical inference technique as discussed in the previous section, it is possible to handle other dimensions such as energy and temperature, which often expose dependencies.

Technical Improvements

In addition to the above improvement, we may think of technical enhancement for the tool-flow we implemented. For instance, the BIP$^{\mathcal{SMC}}$

tool may benefit from a graphical interface instead of command-line. Several optimization may be also operated, such as the properties monitoring algorithm, in addition to the possible extension of the property specification language to cover nested operators, which requires systematic procedure for generating more complex monitors. It is also worth continuing the exploration of the image processing case study with respect to trade-offs between time and memory utilization and to consider the full S1 layer to challenge the statistical model checking technique we are using for analysis. This may require using the abstraction technique for instance as the obtained model might be too big. The instrumentation of the generated code to get performance data can be automated through the use of special annotations. The latter could be introduced at the level of the functional BIP model and deduced from the performance properties under consideration. Later, when performing code generation the introduced annotations may be transformed to specific functions calls, given the target runtime and platform support. We can also automate the calibration process, at least for timing information, following the rules defined in Chapter 5.

List of Figures

1.1 The Y-chart Scheme for mapping application to architecture. 6
1.2 Schematic view of exploring the design space. 7
1.3 A generic process of design space exploration. 8
1.4 Building separate purely functional and performance models. 15
1.5 Illustration of the model checking verification method. 19
1.6 Overview of the *ASTROLABE* method. 22

2.1 An LMC model for the Craps Gambling Game. 29
2.2 An MDP example. 30
2.3 Markov chain induced by scheduler $\mathscr{S}$ on the MDP of Figure 2.2. 32

3.1 An example of a BIP atomic component. 43
3.2 A BIP component of a processing unit of a video decoding system. 45
3.3 BIP example: Sender-Buffer-Receiver system. 47
3.4 Priority restriction example. 48
3.5 Behavior of the *System* component in Figure 3.3. 49
3.6 Example of a stochastic atomic component $\mathcal{B}^{\mathfrak{s}}$ and its behavior. 51
3.7 Example of a stochastic atomic component $\mathcal{B}^{\mathfrak{s}}$ and associated MDP. 54
3.8 A Stochastic Multimedia Stream Source model. 54
3.9 Illustration of the stochastic semantics of composition in $\mathcal{S}$BIP. 56
3.10 $\mathcal{S}$BIP component of an MPEG2 Stream Source. 58
3.11 LMC induced by a uniform scheduler $\mathscr{S}$ on the MDP Figure 3.7c. 59
3.12 An abstract view of a multimedia SoC. 60
3.13 Evolution of the play-out buffer fill-level over time. 61
3.14 Frequency distributions of I,P, and B frames. 62
3.15 $\mathcal{S}$BIP model of the multimedia SoC. 62
3.16 A generic $\mathcal{S}$BIP model of a buffer component. 63
3.17 $\mathcal{S}$BIP model of the Player component. 63

3.18 $\mathcal{S}$BIP model of the *Observer* component. 65
3.19 Play-out buffer fill-level and probability of ϕ for `cact.m2v`. . . 66
3.20 Play-out buffer fill-level and probability of ϕ for `mobile.m2v`. 67
3.21 Play-out buffer fill-level and probability of ϕ for `tennis.m2v`. 67
3.22 Transformation rules of LMCs to $\mathcal{S}$BIP model. 69
3.23 An example of transformation of an LMC to an $\mathcal{S}$BIP model. 70
3.24 Behavior of the obtained $\mathcal{S}$BIP component from an LMC. . . 72
3.25 Transformation rules of MDPs to $\mathcal{S}$BIP model. 74
3.26 An example of transformation of an MDP to an $\mathcal{S}$BIP model. 75
3.27 Behavior of the obtained $\mathcal{S}$BIP component from an MDP. . . 76

4.1 Main steps of the AAlergia algorithm. 82
4.2 Illustration of our approach for learning abstract models. . . . 83
4.3 Learned abstract models of the Craps Gambling Game. 86
4.4 Abstract Herman's Self Stabilizing Protocol models. 89
4.5 Statistical Model Checking Time of φ^{10} using PESTIM. . . . 91
4.6 φ^L verification results using PESTIM for $M = \{7, 11\}$. 91
4.7 φ^L verification results using PESTIM for $M = \{19, 21\}$. . . . 92
4.8 Abstract Herman's Self Stabilizing Protocol model with $M =$ 7 for ϕ. 93

5.1 The *ASTROLABE* approach. 102
5.2 Memory access and arbitration mechanism. 106
5.3 Impact of arbitration mechanism on execution time. 107
5.4 Calibration of functional models with computation time. . . . 109
5.5 Calibration of functional model with communication time. . . 110
5.6 Example of communication through a fifo buffer. 110
5.7 Calibration of the functional model in Figure 5.6. 111
5.8 Bad calibration of the functional model in Figure 5.6. 112

6.1 Histogram of 500 Normally distributed observations. 118
6.2 Box-Whisker Plot of 500 Normally distributed observations. . 118
6.3 Lag Plot of 500 Normally distributed observations. 119
6.4 Fitted Normal distributions obtained using the estimation methods. 122
6.5 Q-Q plot of the different fits. 122
6.6 Comparison of the CDFs of the obtained fits. 123

7.1 Illustration of the *ASTROLABE* tool-flow. 128
7.2 DistFit Tool: the different steps, inputs, and outputs. 131
7.3 BIP$^{\mathcal{SMC}}$ tool architecture: diagram of the main modules. . . 132
7.4 Example of a DTMC model and its specification in $\mathcal{S}$BIP. . . 134
7.5 The *ASTROLABE* tool-flow extended with several tools. . . . 137
7.6 A KPN model example consisting of a split/join schema. . . . 137
7.7 The BIP model of the *Worker* example Figure 7.6. 138

8.1 HMAX models algorithm overview. 142
8.2 Example of input and output images of the HMAX S1 layer. . 143
8.3 MCAPI domains on STHORM. 144
8.4 An abstract view of the S1 layer of HMAX models algorithm. 145
8.5 The simplified model of the S1 layer of HMAX models algorithm. 147
8.6 A functional BIP model of the *Gabor* process. 147
8.7 A BIP model of a FIFO buffer. 148
8.8 Box and Runs plots of the *Gabor* execution time. 149
8.9 Box and Runs plots of the *Gabor* execution time without outliers. 150
8.10 Lag plot of the *Gabor* execution time. 150
8.11 Histogram and CDF of the *Gabor* execution time observations. 151
8.12 Cullen and Frey graph for the *Gabor* execution time. 152
8.13 Graphical comparison between the Normal and the Gamma fits. 153
8.14 Exploratory Analysis for *Splitter* reading time. 154
8.15 Normal Fit Summary for *Splitter* reading time. 155
8.16 The calibrated model of the *Gabor* process. 156
8.17 Probabilities of ϕ_2 for $P \in \{0, 2\}$. 158

List of Tables

3.1 Summary of the stochastic extension of the BIP formalism. . 60

4.1 Verification results of φ_1 on Craps: original vs. abstract models 86
4.2 Summary of the possible settings and the corresponding results. 88
4.3 Abstraction and verification results of φ^{10} and ψ^{30} using PESTIM. 90
4.4 SMC of ψ_M^L on the original and the learned Herman's models. 93

6.1 Results of the goodness-of-fit tests of the obtained fits. 125
6.2 BIC and AIC scores for the obtained fits. 125

7.1 MCAPI endpoints memory attributes. 129

8.1 GOF evaluation results for the Normal and Gamma fits. . . . 152
8.2 $P(\phi_1)$ and number of used SMC traces when varying Δ ($P = 0$).157
8.3 $P(\phi_2)$ and number of used SMC traces when varying Ψ ($P = 0$).157
8.4 $P(\phi_2)$ and number of used SMC traces when varying Ψ ($P = 2$).157

Bibliography

[1] Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Rigorous implementation of real-time systems–from theory to application. *Mathematical Structures in Computer Science*, 23(04):882–914, 2013.

[2] Gul Agha, José Meseguer, and Koushik Sen. Pmaude: Rewrite-based specification language for probabilistic object systems. *Electronic Notes in Theoretical Computer Science*, 153(2):213–239, 2006.

[3] Sumit Ahuja, Deepak A Mathaikutty, Avinash Lakshminarayana, and Sandeep K Shukla. Scope: Statistical regression based power models for co-processors power estimation. *Journal of Low Power Electronics*, 5(4):407–415, 2009.

[4] Marco Ajmone Marsan, Gianni Conte, and Gianfranco Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS)*, 2(2):93–122, 1984.

[5] Husain Aljazzar, Manuel Fischer, Lars Grunske, Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. Safety analysis of an airbag system using probabilistic fmea and probabilistic counterexamples. In *Quantitative Evaluation of Systems, 2009. QEST'09. Sixth International Conference on the*, pages 299–308. IEEE, 2009.

[6] Musab AlTurki and José Meseguer. Pvesta: A parallel statistical model checking and quantitative analysis tool. In *CALCO*, pages 386–392, 2011.

[7] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for probabilistic real-time systems. In *Automata, Languages and Programming*, pages 115–126. Springer, 1991.

[8] Theodore Wilbur Anderson. *The new statistical analysis of data*. Springer, 1996.

[9] Urs Anliker, Jan Beutel, Matthias Dyer, Rolf Enzler, Paul Lukowicz, Lothar Thiele, and Gerhard Troster. A systematic approach to the

design of distributed wearable systems. *Computers, IEEE Transactions on*, 53(8):1017–1033, 2004.

[10] Lacramioara Aştefănoaei, Souha Ben Rayana, Saddek Bensalem, Marius Bozga, and Jacques Combaz. Compositional invariant generation for timed systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 263–278. Springer, 2014.

[11] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time markov chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):162–170, 2000.

[12] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Computer Aided Verification*, pages 155–165. Springer, 1995.

[13] Christel Baier, Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and M Grosser. Almost-sure model checking of infinite paths in one-clock timed automata. In *Logic in Computer Science, 2008. LICS'08. 23rd Annual IEEE Symposium on*, pages 217–226. IEEE, 2008.

[14] Christel Baier, Edmund M Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan. *Symbolic model checking for probabilistic processes*. Springer, 1997.

[15] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[16] Amol Bakshi, Viktor K Prasanna, and Akos Ledeczi. Milan: A model based integrated simulation framework for design of embedded systems. *ACM Sigplan Notices*, 36(8):82–93, 2001.

[17] Felice Balarin. *Hardware-software co-design of embedded systems: the POLIS approach*. Springer, 1997.

[18] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, 2003.

[19] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, and Nihal Pekergin. Cosmos: a statistical model checker for the hybrid automata stochastic logic. In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pages 143–144. IEEE, 2011.

[20] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, and Nihal Pekergin. Hasl: an expressive language for statistical verification of stochastic models. In *Proceedings of the 5th International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 306–315. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[21] Benoît Barbot, Serge Haddad, and Claudine Picaronny. Coupling and importance sampling for statistical model checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2012.

[22] Ananda Basu, Saddek Bensalem, Marius Bozga, Paraskevas Bourgos, Mayur Maheshwari, and Joseph Sifakis. Component assemblies in the context of manycore. In *FMCO*, pages 314–333, 2011.

[23] Ananda Basu, Saddek Bensalem, Marius Bozga, Paraskevas Bourgos, Mayur Maheshwari, and Joseph Sifakis. Component assemblies in the context of manycore. In *Formal Methods for Components and Objects*, pages 314–333. Springer, 2013.

[24] Ananda Basu, Saddek Bensalem, Marius Bozga, Paraskevas Bourgos, and Joseph Sifakis. Rigorous system design: the bip approach. In *Mathematical and Engineering Methods in Computer Science*, pages 1–19. Springer, 2012.

[25] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Caillaud, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. In *Formal Techniques for Distributed Systems*, pages 32–46. Springer, 2010.

[26] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, and Axel Legay. Statistical abstraction and model-checking of large heterogeneous systems. *STTT*, 14(1):53–72, 2012.

[27] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, Axel Legay, and Emmanuel Sifakis. Verification of an afdx infrastructure using simulations and probabilities. In *RV*, pages 330–344, 2010.

[28] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in bip. In *SEFM*, pages 3–12, 2006.

[29] Ananda Basu, Matthieu Gallien, Charles Lesire, Thanh-Hung Nguyen, Saddek Bensalem, Félix Ingrand, and Joseph Sifakis. Incremental component-based construction and verification of a robotic system. In *ECAI*, volume 178, pages 631–635, 2008.

[30] Gerd Behrmann, Alexandre David, and Kim G Larsen. A tutorial on uppaal. In *Formal methods for the design of real-time systems*, pages 200–236. Springer, 2004.

[31] Meena Belwal and TSB Sudarshan. A survey on design space exploration for heterogeneous multi-core. In *Embedded Systems (ICES), 2014 International Conference on*, pages 80–85. IEEE, 2014.

[32] Saddek Bensalem, Marius Bozga, Benoît Delahaye, Cyrille Jégourel, Axel Legay, and Ayoub Nouri. Statistical model checking qos properties of systems with sbip. In *ISoLA (1)*, pages 327–341, 2012.

[33] Saddek Bensalem, Marius Bozga, Thanh-Hung Nguyen, and Joseph Sifakis. D-finder: A tool for compositional deadlock detection and verification. In *CAV*, pages 614–619, 2009.

[34] Saddek Bensalem, Andreas Griesmayer, Axel Legay, Thanh-Hung Nguyen, Joseph Sifakis, and Rongjie Yan. D-finder 2: Towards efficient correctness of incremental design. In *NASA Formal Methods*, pages 453–458. Springer, 2011.

[35] Saddek Bensalem, Axel Legay, Ayoub Nouri, and Doron Peled. Synthesizing distributed scheduling implementation for probabilistic component-based systems. In *MEMOCODE*, pages 87–96, 2013.

[36] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Groesser, and Marcin Jurdzinski. Stochastic timed automata. *arXiv preprint arXiv:1410.2128*, 2014.

[37] Simon Bliudze and Joseph Sifakis. The algebra of connectors - structuring interaction in bip. *IEEE Trans. Computers*, 57(10):1315–1330, 2008.

[38] Simon Bliudze and Joseph Sifakis. A notion of glue expressiveness for component-based systems. In *CONCUR 2008-Concurrency Theory*, pages 508–522. Springer, 2008.

[39] Simon Bliudze and Joseph Sifakis. Synthesizing glue operators from glue constraints for the construction of component-based systems. In *Software Composition*, pages 51–67. Springer, 2011.

[40] Alex Bobrek, Joshua J Pieper, Jeffrey E Nelson, JoAnn M Paul, and Donald E Thomas. Modeling shared resource contention using a hybrid simulation/analytical approach. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 2, pages 1144–1149. IEEE, 2004.

[41] Jonathan Bogdoll, Luis María Ferrer Fioriti, Arnd Hartmanns, and Holger Hermanns. Partial order methods for statistical model checking and simulation. In *Formal Techniques for Distributed Systems*, pages 59–74. Springer, 2011.

[42] Borzoo Bonakdarpour, Marius Bozga, Mohamad Jaber, Jean Quilbeuf, and Joseph Sifakis. A framework for automated distributed implementation of component-based models. *Distributed Computing*, 25(5):383–409, 2012.

[43] P Bourgos, A Basu, S Bensalem, K Huang, and J Sifakis. Integrating architectural constraints in application software by source-to-source transformation in bip. Technical report, Technical Report TR-2011-1, Verimag Research Report, 2010.

[44] Paraskevas Bourgos. *Rigorous Design Flow for Programming Manycore Platforms*. PhD thesis, Univesité de Grenoble, 2013.

[45] Paraskevas Bourgos, Ananda Basu, Marius Bozga, Saddek Bensalem, Joseph Sifakis, and Kai Huang. Rigorous system level modeling and analysis of mixed hw/sw systems. In *MEMOCODE*, pages 11–20, 2011.

[46] Patricia Bouyer, Kim G Larsen, and Nicolas Markey. Model-checking one-clock priced timed automata. In *Foundations of Software Science and Computational Structures*, pages 108–122. Springer, 2007.

[47] George EP Box and David A Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970.

[48] Carlo Brandolese, William Fornaciari, Luigi Pomante, Fabio Salice, and Donatella Sciuto. Affinity-driven system design exploration for heterogeneous multiprocessor soc. *Computers, IEEE Transactions on*, 55(5):508–519, 2006.

[49] Davide Bruni, Alessandro Bogliolo, and Luca Benini. Statistical design space exploration for application-specific unit synthesis. In *Design Automation Conference, 2001. Proceedings*, pages 641–646. IEEE, 2001.

[50] Peter Bulychev, Alexandre David, Kim G Larsen, Axel Legay, and Marius Mikučionis. Computing nash equilibrium in wireless ad hoc networks: A simulation-based approach. *arXiv preprint arXiv:1202.4506*, 2012.

[51] Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Marius Mikucionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. Uppaal-smc: Statistical model checking for priced timed automata. In *QAPL*, pages 1–16, 2012.

[52] Doron Bustan, Sasha Rubin, and Moshe Y Vardi. Verifying ω-regular properties of markov chains. In *Computer Aided Verification*, pages 189–201. Springer, 2004.

[53] Lukai Cai and Daniel Gajski. Transaction level modeling: an overview. In *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 19–24. ACM, 2003.

[54] Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *ICGI*, pages 139–152, 1994.

[55] Samarjit Chakraborty, Simon Künzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, volume 3, page 10190. Citeseer, 2003.

[56] Souymodip Chakraborty, Joost-Pieter Katoen, Falak Sher, and Martin Strelec. Modelling and statistical model checking of a microgrid. *International Journal on Software Tools for Technology Transfer*, pages 1–18, 2014.

[57] Chantana Chantrapornchai, Edwin H-M Sha, and Xiaobo Sharon Hu. Efficient design exploration based on module utility selection. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(1):19–29, 2000.

[58] Stavros Tripakis Christopher Brooks, Edward A. Lee. Ptolemy tutorial: Exploring models of computation with ptolemy ii. *A tutorial presented in ESWeek*, 2010.

[59] Frank Ciesinski and Marcus Größer. On probabilistic computation tree logic. In *Validation of Stochastic Systems*, pages 147–188. Springer, 2004.

[60] Edmund M Clarke, Alexandre Donzé, and Axel Legay. Statistical model checking of mixed analog circuits. Formal Verification of Analog Circuits, Workshop of CAV, 2008.

[61] Edmund M Clarke, Alexandre Donzé, and Axel Legay. On simulation-based probabilistic model checking of mixed-analog circuits. *Formal Methods in System Design*, 36(2):97–113, 2010.

[62] Edmund M Clarke and E Allen Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic*. Springer, 1982.

[63] Edmund M Clarke, James R Faeder, Christopher J Langmead, Leonard A Harris, Sumit Kumar Jha, and Axel Legay. Statistical model checking in biolab: Applications to the automated analysis of t-cell receptor signaling pathway. In *Computational Methods in Systems Biology*, pages 231–250. Springer, 2008.

[64] Edmund M Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *Automated Technology for Verification and Analysis*, pages 1–12. Springer, 2011.

[65] Jamieson M Cobleigh, Dimitra Giannakopoulou, and Corina S Păsăreanu. Learning assumptions for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2003.

[66] Joseph E Coffland and Andy D Pimentel. A software framework for efficient system-level performance evaluation of embedded systems. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 666–671. ACM, 2003.

[67] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM (JACM)*, 42(4):857–907, 1995.

[68] Glen Cowan. *Statistical data analysis*. Oxford University Press, 1998.

[69] Andreas E Dalsgaard, Mads Chr Olesen, Martin Toft, René Rydhof Hansen, and Kim Guldstrand Larsen. Metamoc: Modular execution

time analysis using model checking. In *OASIcs-OpenAccess Series in Informatics*, volume 15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[70] Pedro R D'Argenio, Holger Hermanns, Joost-Pieter Katoen, and Ric Klaren. Modest:a modelling and description language for stochastic timed systems. In *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*, pages 87–104. Springer, 2001.

[71] Pedro R D'Argenio, Joost-Pieter Katoen, and H Brinksma. A stochastic automata model and its algebraic approach. 1997.

[72] Pedro Ruben D'Argenio. *Algebras and automata for timed and stochastic systems*. Universiteit Twente, 1999.

[73] Alexandre David, DeHui Du, Kim G Larsen, Marius Mikučionis, and Arne Skou. An evaluation framework for energy aware buildings using statistical model checking. *Science China Information Sciences*, 55(12):2694–2707, 2012.

[74] 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 *Formal Modeling and Analysis of Timed Systems*, pages 80–96. Springer, 2011.

[75] Benoît Dupont de Dinechin, Renaud Ayrignac, P-E Beaucamps, Patrice Couvert, Benoit Ganne, Pierre Guironnet de Massas, François Jacquet, Samuel Jones, Nicolas Morey Chaisemartin, Frédéric Riss, et al. A clustered manycore processor architecture for embedded and accelerated applications. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–6. IEEE, 2013.

[76] Colin De la Higuera. *Grammatical inference*, volume 96. Cambridge University Press Cambridge, 2010.

[77] Colin de la Higuera and José Oncina. Identification with probability one of stochastic deterministic linear languages. In *ALT*, pages 247–258, 2003.

[78] Colin de la Higuera, José Oncina, and Enrique Vidal. Identification of dfa: data-dependent vs data-independent algorithms. In *ICGI*, pages 313–325, 1996.

[79] André de Matos Pedro, Paul Andrew Crocker, and Simão Melo de Sousa. Learning stochastic timed automata from sample executions. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 508–523. Springer, 2012.

[80] Marie Laure Delignette-Muller, Regis Pouillot, Jean-Baptiste Denis, and Christophe Dutang. *Fitdistrplus: Help to fit of a parametric distribution to non-censored or censored data*, 2010.

[81] François Denis, Yann Esposito, and Amaury Habrard. Learning rational stochastic languages. In *COLT*, pages 274–288, 2006.

[82] David M Diez, Christopher D Barr, and Mine Cetinkaya-Rundel. *Open-Intro statistics*. CreateSpace independent publishing platform, 2012.

[83] Laurent Doyen, Thomas A Henzinger, and Jean-François Raskin. Equivalence of labeled markov chains. *International journal of foundations of computer science*, 19(03):549–563, 2008.

[84] Cindy Eisner and Dana Fisman. *A practical introduction to PSL*. Springer, 2007.

[85] Diana El Rabih and Nihal Pekergin. Statistical model checking using perfect simulation. In *Automated Technology for Verification and Analysis*, pages 120–134. Springer, 2009.

[86] Huixing Fang, Jianqi Shi, Huibiao Zhu, Jian Guo, Kim Guldstrand Larsen, and Alexandre David. Formal verification and simulation for platform screen doors and collision avoidance in subway control systems. *International Journal on Software Tools for Technology Transfer*, pages 1–23, 2014.

[87] Peter H Feiler and David P Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012.

[88] Peter H Feiler, Bruce A Lewis, and Steve Vestal. The sae architecture analysis & design language (aadl) a standard for engineering performance critical systems. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 1206–1211. IEEE, 2006.

[89] Bernd Finkbeiner and Henny Sipma. Checking finite traces using alternating automata. *Formal Methods in System Design*, 24(2):101–127, 2004.

[90] William Fornaciari, Donatella Sciuto, Cristina Silvano, and Vittorio Zaccaria. A sensitivity-based design space exploration methodology for embedded systems. *Design Automation for Embedded Systems*, 7(1-2):7–33, 2002.

[91] Masahiro Fujita, Patrick C. McGeer, and JC-Y Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal methods in system design*, 10(2-3):149–169, 1997.

[92] Daniel D Gajski, Frank Vahid, Sanjiv Narayan, and Jie Gong. System-level exploration with specsyn. In *Proceedings of the 35th annual Design Automation Conference*, pages 812–817. ACM, 1998.

[93] Paul Gastin and Denis Oddoux. Fast ltl to büchi automata translation. In *Computer Aided Verification*, pages 53–65. Springer, 2001.

[94] Dimitra Giannakopoulou and Klaus Havelund. Automata-based verification of temporal properties on running programs. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 412–416. IEEE, 2001.

[95] KAHN Gilles. The semantics of a simple language for parallel programming. In *In Information Processing 74: Proceedings of the IFIP Congress*, volume 74, pages 471–475, 1974.

[96] Paolo Giusto, Grant Martin, and Ed Harcourt. Reliable estimation of execution time of embedded software. In *Proceedings of the conference on Design, automation and test in Europe*, pages 580–589. IEEE Press, 2001.

[97] Tony Givargis, Frank Vahid, and Jörg Henkel. System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 10(4):416–422, 2002.

[98] Peter W Glynn. On the role of generalized semi-markov processes in simulation output analysis. In *Proceedings of the 15th conference on Winter simulation-Volume 1*, pages 39–44. IEEE Press, 1983.

[99] E Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[100] M González Harbour, JJ Gutiérrez García, JC Palencia Gutiérrez, and JM Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 125–134. IEEE, 2001.

[101] Radu Grosu, Doron Peled, CR Ramakrishnan, Scott A Smolka, Scott D Stoller, and Junxing Yang. Using statistical model checking for measuring systems. 2014.

[102] Peter Habermehl and Tomás Vojnar. Regular model checking using inference of regular languages. *Electr. Notes Theor. Comput. Sci.*, 138(3):21–36, 2005.

[103] Amaury Habrard, François Denis, and Yann Esposito. Using pseudo-stochastic rational languages in probabilistic grammatical inference. In *Grammatical Inference: Algorithms and Applications*, pages 112–124. Springer, 2006.

[104] Wolfgang Haid, Matthias Keller, Kai Huang, Iuliana Bacivarov, and Lothar Thiele. Generation and calibration of compositional performance analysis models for multi-processor systems. In *Systems, Architectures, Modeling, and Simulation, 2009. SAMOS'09. International Symposium on*, pages 92–99. IEEE, 2009.

[105] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[106] John Heath, Marta Kwiatkowska, Gethin Norman, David Parker, and Oksana Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257, 2008.

[107] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis–the symta/s approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

[108] David Henriques, Joao G Martins, Paolo Zuliani, André Platzer, and Edmund M Clarke. Statistical model checking for markov decision processes. In *Quantitative Evaluation of Systems (QEST), 2012 Ninth International Conference on*, pages 84–93. IEEE, 2012.

[109] Thomas A Henzinger and Joseph Sifakis. The embedded systems design challenge. In *FM 2006: Formal Methods*, pages 1–15. Springer, 2006.

[110] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *VMCAI*, pages 73–84, 2004.

[111] Ted Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2):63–67, 1990.

[112] Holger Hermanns, Ulrich Herzog, and Joost-Pieter Katoen. Process algebra for performance evaluation. *Theoretical computer science*, 274(1):43–87, 2002.

[113] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic cegar. In *Computer Aided Verification*, pages 162–175. Springer, 2008.

[114] Jane Hillston. *A compositional approach to performance modelling*, volume 12. Cambridge University Press, 2005.

[115] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[116] Wallace Hui, Yulia R Gel, and Joseph L Gastwirth. lawstat: an r package for law, public policy and biostatistics. *J Stat Softw*, 28:1–26, 2008.

[117] David N Jansen, Joost-Pieter Katoen, Marcel Oldenkamp, Mariëlle Stoelinga, and Ivan Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *Hardware and Software: Verification and Testing*, pages 69–85. Springer, 2008.

[118] Cyrille Jegourel, Axel Legay, and Sean Sedwards. Cross-entropy optimisation of importance sampling parameters for statistical model checking. In *Computer Aided Verification*, pages 327–342. Springer, 2012.

[119] Cyrille Jégourel, Axel Legay, and Sean Sedwards. A platform for high performance statistical model checking - plasma. In *TACAS*, pages 498–503, 2012.

[120] Cyrille Jegourel, Axel Legay, and Sean Sedwards. Importance splitting for statistical model checking rare properties. In *Computer Aided Verification*, pages 576–591. Springer, 2013.

[121] Sumit K Jha, Edmund M Clarke, Christopher J Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A bayesian approach to model checking biological systems. In *Computational Methods in Systems Biology*, pages 218–234. Springer, 2009.

[122] Zai Jian Jia, Tomás Bautista, Antonio Núñez, Andy D Pimentel, and Mark Thompson. A system-level infrastructure for multidimensional mp-soc design space co-exploration. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(1s):27, 2013.

[123] J Katoen and Ivan S Zapreev. Simulation-based ctmc model checking: An empirical evaluation. In *Quantitative Evaluation of Systems, 2009. QEST'09. Sixth International Conference on the*, pages 31–40. IEEE, 2009.

[124] Joost-Peter Katoen and Doron Peled. Taming confusion for modeling and implementing probabilistic concurrent systems. In *Programming Languages and Systems*, pages 411–430. Springer, 2013.

[125] Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf. Three-valued abstraction for probabilistic systems. *The Journal of Logic and Algebraic Programming*, 81(4):356–389, 2012.

[126] 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.

[127] Kurt Keutzer, A Richard Newton, Jan M Rabaey, and Alberto Sangiovanni-Vincentelli. System-level design: orthogonalization of concerns and platform-based design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(12):1523–1543, 2000.

[128] Bart Kienhuis, Ed F Deprettere, Pieter Van Der Wolf, and Kees Vissers. A methodology to design programmable embedded systems. In *Embedded processor design challenges*, pages 18–37. Springer, 2002.

[129] Shinji Kikuchi and Yasuhide Matsumoto. Performance modeling of concurrent live migration operations in cloud computing systems using prism probabilistic model checker. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 49–56. IEEE, 2011.

[130] Stuart A Klugman, Harry H Panjer, and Gordon E Willmot. *Loss models: from data to decisions*, volume 715. John Wiley & Sons, 2012.

[131] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[132] Heiko Koziolek, Bastian Schlich, and Carlos Bilich. A large-scale industrial case study on architecture-based software reliability analysis. In *Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on*, pages 279–288. IEEE, 2010.

[133] Marwan Krunz and Satish K Tripathi. On the characterization of vbr mpeg streams. In *ACM SIGMETRICS Performance Evaluation Review*, volume 25, pages 192–202. ACM, 1997.

[134] Nirman Kumar, Koushik Sen, José Meseguer, and Gul Agha. A rewriting based model for probabilistic distributed object systems. In *FMOODS*, pages 32–46, 2003.

[135] S Kunzli, Francesco Poletti, Luca Benini, and Lothar Thiele. Combining simulation and formal methods for system-level performance analysis. In *Design, Automation and Test in Europe, 2006. DATE'06. Proceedings*, volume 1, pages 1–6. IEEE, 2006.

[136] Simon Kunzli. *Efficient design space exploration for embedded systems*. PhD thesis, Swiss Federal Institute of Technology Zurich, Switzerland, 2006.

[137] Marta Kwiatkowska, Gethin Norman, and David Parker. Symmetry reduction for probabilistic model checking. In *Computer Aided Verification*, pages 234–248. Springer, 2006.

[138] Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101–150, 2002.

[139] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

[140] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, 1994.

[141] Sophie Laplante, Richard Lassaigne, Frédéric Magniez, Sylvain Peyronnet, and Michel de Rougemont. Probabilistic abstraction for model checking: An approach based on property testing. *ACM Trans. Comput. Log.*, 8(4), 2007.

[142] Richard Lassaigne and Sylvain Peyronnet. Approximate verification of probabilistic systems. In *Process Algebra and Probabilistic Methods: Performance Modeling and Verification*, pages 213–214. Springer, 2002.

[143] Jean-Yves Le Boudec. *Performance evaluation of computer and communication systems*. EPFL Press, 2010.

[144] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *RV*, pages 122–135, 2010.

[145] Alexios Lekidis, Pareskivas Bourgos, Djoko-Djoko Simplis, Marius Bozga, and Saddek Bensalem. Building distributed sensor network applications using bip. In *In IEEE Sensors Applications Symposium*, 2015.

[146] Alexios Lekidis, Marius Bozga, and Saddek Bensalem. Model-based validation of canopen systems. In *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on*, pages 1–10. IEEE, 2014.

[147] Alexios Lekidis, Marius Bozga, Didier Mauuary, and Saddek Bensalem. A model-based design flow for can-based systems. In *14th International CAN Conference, Eurosites République, Paris*, volume 4, 2013.

[148] Martin Leucker. Learning meets verification. In *FMCO*, pages 127–151, 2006.

[149] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. *The glory of the past*. Springer, 1985.

[150] Paul Lieverse, Pieter Van Der Wolf, Kees Vissers, and Ed Deprettere. A methodology for architecture exploration of heterogeneous signal processing systems. *Journal of VLSI signal processing systems for signal, image and video technology*, 29(3):197–207, 2001.

[151] Greta M Ljung and George EP Box. On a measure of lack of fit in time series models. *Biometrika*, 65(2):297–303, 1978.

[152] Mirko Loghi, Federico Angiolini, Davide Bertozzi, Luca Benini, and Roberto Zafalon. Analyzing on-chip communication in a mpsoc environment. In *Proceedings of the conference on Design, automation and test in Europe-Volume 2*, page 20752. IEEE Computer Society, 2004.

[153] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. Learning probabilistic automata for model checking. In *QEST*, pages 111–120, 2011.

[154] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D Nielsen, Kim G Larsen, and Brian Nielsen. Learning markov decision processes for model checking. *arXiv preprint arXiv:1212.3873*, 2012.

[155] Diego Melpignano, Luca Benini, Eric Flamand, Bruno Jego, Thierry Lepley, Germain Haugou, Fabien Clermidy, and Denis Dutoit. Platform 2012, a many-core computing accelerator for embedded socs: performance evaluation of visual analytics applications. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1137–1142. ACM, 2012.

[156] Marius Mikučionis, Kim Guldstrand Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen,

and Poul Hougaard. Schedulability analysis using uppaal: Herschel-planck case study. In *Leveraging Applications of Formal Methods, Verification, and Validation*, pages 175–190. Springer, 2010.

[157] Natasa Miskov-Zivanov, Paolo Zuliani, Edmund M Clarke, and James R Faeder. Studies of biological networks with statistical model checking: application to immune system cells. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics*, page 728. ACM, 2013.

[158] Sumit Mohanty and Viktor K Prasanna. Rapid system-level performance evaluation and optimization for application mapping onto soc architectures. In *ASIC/SOC Conference, 2002. 15th Annual IEEE International*, pages 160–167. IEEE, 2002.

[159] Imed Moussa, Thierry Grellier, and Giang Nguyen. Exploring sw performance using soc transaction-level modeling. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 120–125. IEEE, 2003.

[160] Jim Mutch and David G Lowe. Object class recognition and localization using sparse features with limited receptive fields. *International Journal of Computer Vision*, 80(1):45–57, 2008.

[161] Gethin Norman, David Parker, Marta Kwiatkowska, Sandeep Shukla, and Rajesh Gupta. Using probabilistic model checking for dynamic power management. *Formal Aspects of Computing*, 17(2):160–176, 2005.

[162] Ayoub Nouri, Saddek Bensalem, Marius Bozga, Benoit Delahaye, Cyrille Jégourel, and Axel Legay. Statistical model checking qos properties of systems with sbip. *International Journal on Software Tools for Technology Transfer*, pages 1–15, 2014.

[163] Ayoub Nouri, Marius Bozga, Anca Molnos, Axel Legay, and Saddek Bensalem. Building faithful high-level models and performance evaluation of manycore embedded systems. In *Formal Methods and Models for Codesign (MEMOCODE), 2014 Twelfth ACM/IEEE International Conference on*, pages 209–218. IEEE, 2014.

[164] Ayoub Nouri, Anca Molnos, Julien Mottin, Marius Bozga, Saddek Bensalem, Arnaud Tonda, and Francois Pacull. A model-based approach for rapid prototyping of parallel applications on manycore. Technical report, 2014.

[165] Ayoub Nouri, Balaji Raman, Marius Bozga, Axel Legay, and Saddek Bensalem. Faster statistical model checking by means of abstraction and learning. In *Runtime Verification*, pages 340–355. Springer, 2014.

[166] Doron Peled, Moshe Y. Vardi, and Mihalis Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2002.

[167] Jorge M Pena and Arlindo L Oliveira. A new algorithm for exact reduction of incompletely specified finite state machines. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(11):1619–1632, 1999.

[168] Andy D Pimentel. The artemis workbench for system-level performance evaluation of embedded systems. *International Journal of Embedded Systems*, 3(3):181–196, 2008.

[169] Andy D Pimentel, Cagkan Erbas, and Simon Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *Computers, IEEE Transactions on*, 55(2):99–112, 2006.

[170] Andy D Pimentel, Mark Thompson, Simon Polstra, and Cagkan Erbas. Calibration of abstract performance models for system-level design space exploration. *Journal of Signal Processing Systems*, 50(2):99–114, 2008.

[171] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[172] Amir Pnueli, Jessie Xu, and Lenore Zuck. Liveness with (0, 1,)-counter abstraction. In *Computer Aided Verification*, pages 107–122. Springer, 2002.

[173] Amir Pnueli and Lenore D Zuck. Probabilistic verification. *Information and computation*, 103(1):1–29, 1993.

[174] Luigi Pomante. System-level design space exploration for dedicated heterogeneous multi-processor systems. In *Application-Specific Systems, Architectures and Processors (ASAP), 2011 IEEE International Conference on*, pages 79–86. IEEE, 2011.

[175] Petro Poplavko. Multiprocessor performance analysis: Ensuring guaranteed throughput of dynamic multimedia streaming applications. 2009.

[176] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on Programming*, pages 337–351. Springer, 1982.

[177] Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[178] Balaji Raman, Ayoub Nouri, Deepak Gangadharan, Marius Bozga, Ananda Basu, Mayur Maheshwari, Axel Legay, Saddek Bensalem, and Samarjit Chakraborty. Stochastic modeling and performance analysis of multimedia socs. In *ICSAMOS*, pages 145–154, 2013.

[179] Wolfgang Reisig. *Petri nets: an introduction*. Springer-Verlag New York, Inc., 1985.

[180] Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of the eighth annual conference on Computational learning theory*, pages 31–40. ACM, 1995.

[181] Gerardo Rubino, Bruno Tuffin, et al. *Rare event simulation using Monte Carlo methods*. Wiley Online Library, 2009.

[182] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.

[183] Jan JMM Rutten, Marta Kwiatkowska, Gethin Norman, and David Parker. *Mathematical techniques for analyzing concurrent and probabilistic systems*. American Mathematical Soc., 2004.

[184] Alberto Sangiovanni-Vincentelli and Grant Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, 18(6):23–33, 2001.

[185] Roberto Segala. A compositional trace-based semantics for probabilistic automata. In *CONCUR'95: Concurrency Theory*, pages 234–248. Springer, 1995.

[186] Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[187] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Learning continuous time markov chains from sample executions. In *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, pages 146–155. IEEE, 2004.

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

[189] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *Computer Aided Verification*, pages 266–280. Springer, 2005.

[190] Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *QEST*, pages 251–252, 2005.

[191] Vitaly Shmatikov. Probabilistic analysis of an anonymity system. *Journal of Computer Security*, 12(3):355–377, 2004.

[192] Joseph Sifakis. Rigorous system design. *Foundations and Trends® in Electronic Design Automation*, 6(EPFL-ARTICLE-185999):293–362, 2012.

[193] Andreas Stolcke. *Bayesian learning of probabilistic language models*. PhD thesis, University of California, Berkeley, 1994.

[194] Andreas Stolcke and Stephen Omohundro. Hidden markov model induction by bayesian model merging. *Advances in neural information processing systems*, pages 11–11, 1993.

[195] Radoslaw Szymanek, Francky Catthoor, and Krzysztof Kuchcinski. Time-energy design space exploration for multi-layer memory architectures. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 1, pages 318–323. IEEE, 2004.

[196] R Core Team et al. R: A language and environment for statistical computing. 2012.

[197] Bart D Theelen. *Performance modelling for system-level design*. PhD thesis, Eindhoven University of Technology, 2004.

[198] Lothar Thiele, Iuliana Bacivarov, Wolfgang Haid, and Kai Huang. Mapping applications to tiled multiprocessor embedded systems. In *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on*, pages 29–40. IEEE, 2007.

[199] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 4, pages 101–104. IEEE, 2000.

[200] Moshe Y Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 327–338. IEEE, 1985.

[201] Moshe Y Vardi. Alternating automata and program verification. In *Computer Science Today*, pages 471–485. Springer, 1995.

[202] Sicco Verwer, Rémi Eyraud, and Colin de la Higuera. Results of the pautomac probabilistic automaton learning competition. In *ICGI*, pages 243–248, 2012.

[203] Sicco E Verwer, Mathijs M de Weerdt, Cees Witteveen, Louis Wehenkel, Pierre Geurts, and Raphael Maree. Efficiently learning timed models from observations. In *Benelearn*, pages 75–76, 2008.

[204] David Vose. *Risk analysis: a quantitative guide*. John Wiley & Sons, 2008.

[205] Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[206] Ying-Chih Wang, Anvesh Komuravelli, Paolo Zuliani, and Edmund M Clarke. Analog circuit verification by statistical model checking. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 1–6. IEEE Press, 2011.

[207] Duminda Wijesekera and Jaideep Srivastava. Quality of service (qos) metrics for continuous media. *Multimedia Tools and Applications*, 3(2):127–166, 1996.

[208] Pierre Wolper. Constructing automata from temporal logic formulas: A tutorial. In *Lectures on Formal Methods and PerformanceAnalysis*, pages 261–277. Springer, 2001.

[209] Hakan L Younes. *Verification and planning for stochastic processes with asynchronous events*. PhD thesis, 2005.

[210] Håkan L. S. Younes. Ymer: A statistical model checker. In *CAV*, pages 429–433, 2005.

[211] Håkan LS Younes. Error control for probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation*, pages 142–156. Springer, 2006.

[212] Håkan LS Younes, Edmund M Clarke, and Paolo Zuliani. Statistical verification of probabilistic properties with unbounded until. In *Formal Methods: Foundations and Applications*, pages 144–160. Springer, 2011.

[213] Håkan LS Younes and Reid G Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification*, pages 223–235. Springer, 2002.

[214] Ivan S Zapreev. Model checking markov chains: Techniques and tools. 2008.

[215] Qingfeng Zhuge, Zili Shao, Bin Xiao, and EH-M Sha. Design space minimization with timing and code size optimization for embedded dsp. In *Hardware/Software Codesign and System Synthesis, 2003. First IEEE/ACM/IFIP International Conference on*, pages 144–149. IEEE, 2003.

[216] Paolo Zuliani. Statistical model checking for biological applications. *arXiv preprint arXiv:1405.2705*, 2014.

[217] Paolo Zuliani, André Platzer, and Edmund M Clarke. Bayesian statistical model checking with application to stateflow/simulink verification. *Formal Methods in System Design*, 43(2):338–367, 2013.