Computational Issues in Intelligent Control: Discrete-Event and Hybrid Systems

Technical Report of the ISIS Group
at the University of Notre Dame
ISIS-99-005
May, 1999

Xenofon D. Koutsoukos and Panos J. Antsaklis
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556

Interdisciplinary Studies of Intelligent Systems

Computational Issues in Intelligent Control: Discrete-Event and Hybrid Systems

Xenofon D. Koutsoukos and Panos J. Antsaklis
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556
e-mail: xkoutsou,antsaklis.1@nd.edu

Abstract

Intelligent control methodologies are being developed to address the control needs of complex systems that exhibit complicated dynamical behaviors. The design, simulation, and verification of intelligent control systems is highly nontrivial and typically involves significant amount of computations. In this paper, we identify and discuss several computational issues that are central in intelligent control. In particular, we discuss how the design, simulation, and verification of discrete-event and hybrid systems, which are central in intelligent control, requires the development of computationally efficient algorithms and approaches. Petri net models are used to describe discrete event and hybrid systems. Computational issues of various problems and algorithms concerning the analysis and synthesis of such systems are discussed. In view of hybrid systems, we also review basic computational issues for hybrid automata. Finally, we present a parallel computing architecture for intelligent control systems and we illustrate its advantages by considering parallel discrete event simulations.

Contents

1 Introduction **3**

2 Intelligent control **5**

2.1 General Concepts 5

2.2 Models for Intelligent Controllers 7

2.2.1 Discrete Event System Models 7

2.2.2 Hybrid System Models 8

3 Elements of Computational Complexity Theory **9**

3.1 Algorithms and Turing Machines 10

3.1.1 Basic Notions 10

3.1.2 Decidable and undecidable problems 11

3.1.3 Computational Complexity 11

3.1.4 The Classes $\mathcal{P}$ and $\mathcal{NP}$ 12

3.1.5 Subroutines and Numerical Considerations 13

3.2 Computational Complexity in Intelligent Control 13

4 DES in Intelligent Control using Petri Nets **14**

4.1 Petri Nets: Basic Notions 15

4.2 Decidability Issues in Petri Nets 16

4.3 Checking properties using integer programming 18

4.4 State space search using unfolding 19

4.5 Supervisory control theory of Petri nets 21

5 Hybrid Systems in Intelligent Control **22**

5.1 Computational Issues in Hybrid Automata 23

5.2 Programmable Timed Petri Nets 27

6 A Parallel Computing Architecture for Intelligent Control **30**

6.1 Parallel Discrete Event Simulation 32

7 Conclusions **34**

1 Introduction

The quest for machines that cause physical systems to exhibit higher autonomy has been the driving force in the development of control systems over the centuries. For systems with high degrees of autonomy, intelligent control methodologies appear to be appropriate. An intelligent control system should be able to operate appropriately and with a high degree of autonomy under significant uncertainty which is caused by the fact that its components, control goals, plant models and control laws are not always completely defined, either because they were not known at the design time or because they changed unexpectedly. Intelligent and autonomous control fundamentals are discussed for example in [7, 25, 3, 4].

In order to control complex systems, one has to deal effectively with the computational complexity issue. This has been in the periphery of the interests of the researchers in conventional control, but it is clear that computational complexity is a central issue whenever one attempts to control complex systems. The physical processes of interest in intelligent control are usually more general and complex than the processes that appear in conventional control. They often exhibit complicated phenomena as nonlinear behaviors and switching mechanisms. In addition, the goals of intelligent control problems are more ambitious [3]. Apart from the usual problems of conventional control, concepts such as liveness and deadlock developed in operations research and computer science arise in intelligent control. To develop tools that facilitate the use of intelligent control systems it is essential to capture the phenomena of interest accurately and in tractable mathematical form. A good mathematical description must be detailed enough to accurately describe the phenomena of interest and at the same time simple enough to be amenable to analysis and especially for design procedures.

The study of the computational issues in intelligent control is very helpful in the evaluation of the progress of the research towards building systems with higher degrees of autonomy. It is also useful in identifying specific algorithms and methodologies that appear to be computational intractable and reconsider their mathematical modeling. By modeling at different levels of abstraction, computationally tractable solutions for complex intelligent control problems can be identified. On the practical side, the utilization of the available computer resources can be improved by considering the computational complexity of the involved procedures.

In this work, we concentrate on computational issues in intelligent control that arise when discrete event and hybrid models are used to describe mathematically the processes of interest. The reader, of course, should be aware of the computational results on conventional control algorithms (see for example [64]) which have appeared in the literature in recent years. These results are not studied in this paper. The use of discrete event and hybrid models in intelligent control systems has been discussed at length for example in [81, 8, 62, 5, 60]. Here, we focus on the computational issues of specific approaches that have been proposed for the analysis, synthesis and simulation of such systems. It should be noted that the treatment of the subject it is not complete by far. The importance of studying the computational issues in discrete event and hybrid systems is being recognized by the research community and a number of relevant articles have appeared in the literature. For example, computational issues of supervisor control theory for discrete event systems [69] have been addressed in [59, 80, 57, 71]. Complexity results for hybrid systems can be found in [11, 74, 31, 9]. In this chapter, we study computational issues in recent approaches to discrete event and hybrid system analysis and design that were by our group using Petri nets.

We also present computational issues of related analysis and synthesis problems that have been appeared in the literature. A quantitative theory of intelligent control based on formal models such as discrete event or hybrid system models may result in algorithms of high complexity. Often, there are applications for which the same algorithms can be applied efficiently. There are also cases where the designer may compromise for a “suboptimal” solution which can be computed in an efficient manner.

The modeling tool that we select to study the computational issues in intelligent control here is that of Petri nets. Petri nets are a powerful modeling paradigm for a variety of systems. Their basic characteristic is that they provide an excellent tool for capturing concurrency and conflict within a system. They have an appealing graphical and mathematical representation and they have been used extensively to model information processing systems, manufacturing systems, communication systems, and chemical processes among others. Petri nets have been used extensively as a tool for modeling, analysis and synthesis for discrete event systems [53, 15]. In this paper, ordinary Petri nets are used in the design of supervisor in discrete event systems [51] and a class of timed Petri nets, named programmable timed Petri nets is used for studying hybrid systems [39]. Petri nets can be viewed as a generalization of finite automata. Petri nets are used instead of finite automata because of the following reasons. The first is the expressiveness of Petri nets. Petri net languages include the regular languages described by finite automata and further, they can model switching policies that describe conflict, concurrency, synchronization, and buffer sizes. Another reason is that recent results in the supervisory control of discrete-event systems using ordinary Petri nets [51] have made possible to design supervisors in an efficient and transparent manner. In general, a Petri net representation for a concurrent process will be more compact (fewer vertices) than its associated automaton representation and with the use of partial order semantics it is now possible to search the Petri net’s state space in a efficient manner [47]. The compactness of Petri nets may also lead to algorithms of high complexity. Fortunately, theoretical results concerning Petri net modeling power and limitations exist in the literature. as Petri nets have been used in a wide range of applications. For example, in industrial process control Petri nets have been used to implement real-time controllers, and to serve as a replacement for programmable logic controllers [17].

The aim of this chapter is to investigate the importance and suitability of Petri net based models for intelligent control by studying computational issues that arise in such context. Our framework for intelligent control is discussed in Section 2, where a hierarchical functional architecture that can facilitate the study of fundamental issues of a quantitative theory of autonomous intelligent control is used. Note that such architecture also offers advantages with respect to computational issues. In Subsection 2.2, the need for discrete event and hybrid models in intelligent control systems are discussed as well together with the levels of abstraction in the hierarchical architecture where such models frequently appear.

In Section 3 elements of complexity theory are discussed. Subsection 3.1 contains some basic notions from complexity theory which are necessary for the study of computational issues in intelligent control. The importance of computational complexity in intelligent control and and the different contexts (namely verification, design, simulation) where it can be studied are discussed in Subsection 3.2.

Petri nets are discussed in Section 4. Some basic notions are first introduced in Subsection 4.1. Then, in Subsection 4.2 computational aspects of Petri nets are discussed including decidability issues for various analysis problems. An integer programming technique for checking properties of interest is discussed in Subsection 4.3 and an approach based on partial order semantics (unfolding) for searching the state space is discussed in 4.4. Synthesis results and supervisor control of Petri nets based on place invariants are discussed in Subsection 4.5.

The computational aspects of hybrid models are discussed in Section 5. First in Subsection 5.1, computational issues in hybrid automata are discussed at length. Hybrid automata provide a general modeling formalism for the formal specification and algorithmic analysis of hybrid systems [1] and they are widely used in both the computer science and the control communities. The computational issues of some important synthesis approaches proposed in the literature are also discussed. Programmable timed Petri nets are presented in Subsection 5.2, with emphasis on the computational complexity of algorithms for the analysis and supervision of hybrid systems.

The computational aspects of simulating intelligent control systems are discussed in Section 6. In particular, a parallel computing architecture for intelligent control is presented. The discussion describes ongoing research for development of parallel computing tools for large, computationally demanding, irregular applications where the computational load may change during run-time. Our motivation was a parallel run-time system intended for symmetric multiprocessors (SMPs), which has implemented on an IBM RISC 6000/SP machine. This parallel architecture is an application-driven scheme for applications that require great computational tasks like intelligent control systems. We discuss the suitability of this parallel computing scheme for simulation of intelligent control systems, and in Subsection 6.1, we illustrate its advantages by considering parallel discrete event simulations.

2 Intelligent control

2.1 General Concepts

Intelligent control describes the discipline where the control methods developed attempt to emulate important characteristics of human intelligence. These characteristics include adaptation and learning, planning under large uncertainty and coping with large amounts of data. Today, the area of intelligent control tends to encompass everything that is not characterized as conventional control. Intelligent control is interdisciplinary as it combines and extends theories and methods from areas such as control, computer science and operations research. It uses theories from mathematics and seeks inspiration and ideas from biological systems. Intelligent control methodologies are being applied to robotics and automation, communications, manufacturing, traffic control, to mention but a few application areas. Neural networks, fuzzy control, genetic algorithms, planning systems, expert systems, hybrid systems are all areas where related work is taking place. The areas of computer science and in particular artificial intelligence provide knowledge representation ideas, methodologies and tools such as semantic networks, frames, reasoning techniques and computer languages such as Lisp and Prolog. Concepts and algorithms developed in the areas of adaptive control and machine learning help intelligent controllers to adapt and learn. Advances in sensors, actuators, computation technology and communication networks help provide the necessary for implementation intelligent control hardware.

Why is intelligent control needed? The fact is that there are problems of control today, that

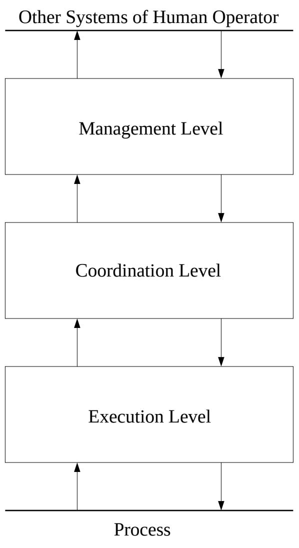

Figure 1: Intelligent Autonomous Controller Functional Architecture

cannot be formulated and studied in the conventional differential/difference equation mathematical framework using “conventional (or traditional) control” methodologies which were developed in the past decades to control dynamical systems [3]. To address these complex problems in a systematic way, a number of methods have been developed in recent years that are collectively known as “intelligent control” methodologies. Intelligent control uses conventional control methods to solve “lower level” control problems and conventional control is included in the area of intelligent control. Intelligent control attempts to build upon and enhance the conventional control methodologies to solve new challenging control problems.

In order to control complex systems, one has to deal effectively with the computational complexity issue. This has been in the periphery of the interests of the researchers in conventional control, but it is clear that computational complexity is a central issue whenever one attempts to control complex systems. Computational complexity issues are usually addressed by using hierarchies to describe the operation of complex systems. A hierarchical functional architecture of a controller that is used to attain high degrees of autonomy has been proposed in [8] (for intelligent control architectures see also [72], the contributions in [7] and the references therein). This hierarchical architecture, which is shown in Fig. 1, has three levels: the execution level, the coordination level, and the management and organization level. The architecture exhibits certain characteristics, which have been shown in the literature to be necessary and desirable in autonomous intelligent systems. Such a hierarchical architecture can facilitate the study of fundamental issues of a quantitative theory of autonomous intelligent control. The representation of a complex system using formal models at different levels of this hierarchy enables the researcher to use standard control theoretic analysis (for example conventional control or supervisor control theory of discrete-event systems). More importantly, in view of the content of this paper, it enables the study of the computational complexity of important problems in intelligent control.

We now briefly outline some characteristics of the architecture. There is a successive delegation of duties from the higher to lower levels; consequently the number of distinct tasks increases as we go down the hierarchy. Higher levels are concerned with slower aspects of the system's behavior and with its larger portions, or broader aspects. There is then a smaller contextual horizon at lower levels, i.e. the control decisions are made by considering less information. Also notice that higher levels are concerned with longer time horizons than lower levels. Due to the fact that there is the need for high level decision making abilities at the higher levels in the hierarchy, the proposition has been put forth that there is increasing intelligence as one moves from the lower to the higher levels. This is reflected in the use of fewer conventional numeric-algorithmic methods at higher levels as well as the use of more symbolic-decision making methods. This is the "principle of increasing intelligence with decreasing precision" of Saridis (see also [73] and the references therein). The decreasing precision is reflected by a decrease in time scale density, decrease in bandwidth or system rate, and a decrease in the decision (control action) rate. These properties have been studied for a class of hierarchical systems in [61]. All these characteristics lead to a decrease in granularity of models used, or equivalently, to an increase in model abstractness.

2.2 Models for Intelligent Controllers

In highly autonomous control systems, the plant is sometimes so complex that it is either impossible or inappropriate to be described by conventional mathematical system models consisting of only differential or difference equations. Even though it might be possible to accurately describe some systems with highly complex nonlinear differential equations, such descriptions may be inappropriate if it makes subsequent analysis too difficult or too computationally complex to be useful. The complexity of the plant model needed in design depends on both the complexity of the physical system and on how demanding the design specifications are. There is a tradeoff between model complexity and our ability to perform analysis on the system via the model. Frequently, if the control performance specifications are not too demanding, a more abstract, higher level model can be utilized, which will make subsequent analysis simpler. This model intentionally ignores some of the system characteristics, specifically those that need not be considered in attempting to meet the particular performance specifications. For example, a simple temperature controller could ignore almost all dynamics of the house or the office and consider only a temperature threshold model of the system to switch the furnace off or on (see also the discussion on hybrid systems later in this article).

2.2.1 Discrete Event System Models

Discrete Event System (DES) models that use finite automata or Petri nets, queuing network models, Markov chains, etc. are quite useful for modeling the higher level decision making processes in an intelligent autonomous controller. The choice of whether to use such models will, of course, depend on what properties of the autonomous system are to be studied. More specifically, DES models are appropriate for general expert control systems, planning systems, abstract learning control and often the higher "management and coordination levels" in the hierarchical architecture for intelligent autonomous systems. DES analysis and controller synthesis techniques (for example [69]) have been developed with remarkable success. Other important topics for intelligent control include approaches to controllability, reachability, stability, and performance analysis. Applications of DES theoretic techniques have been reported for the modeling and analysis of AI planning systems and the stability analysis of expert control systems (see for example [62, 63]). Discrete event systems are of course important in their own right and they have been studied using many approaches. They are also very useful in connection to hybrid systems. Recently, an efficient methodology for supervisory controller design for DES was developed using Petri nets [51, 84, 50]. The approach uses the concept of place invariants of the net to design control supervisors that enforce linear constraints on the marking and firing vectors of the net. This approach is discussed later in this chapter with emphasis on its computational efficiency and simplicity. Potential applications of the approach in intelligent control include real-time control reconfiguration and planning different control tasks, for example in manufacturing and hybrid systems.

In general, when considering the application of DES theoretic techniques to intelligent control systems, it is important to study their computational aspects. Problems like reachability, liveness, and deadlock detection arise in many intelligent control applications. Studying the computational issues of DES approaches can be very important in automated verification, controller synthesis, on-line reconfiguration, and task planning among others. Several models have been proposed in the literature to describe the dynamics of DES. An important observation is that higher expressiveness of the model typically results in algorithms of higher complexity. Petri nets are a trade off between expressiveness and complexity and are suitable for describing concurrent processes that appear frequently in intelligent systems. Petri nets are studied at length in this paper with respect to their computational properties.

2.2.2 Hybrid System Models

Hybrid systems are dynamical systems the behavior of interest of which is determined by interacting continuous and discrete dynamics (see for example [6]). These systems typically contain variables or signals that take values from a continuous set (e.g. the set of real numbers) and also variables that take values from a discrete, typically finite set (e.g. the set of symbols $\{a, b, c\}$). These continuous or discrete-valued variables or signals depend on independent variables such as time, which may also be continuous or discrete; some of the variables may also be discrete event driven in an asynchronous manner.

There are several reasons for using hybrid models to represent the dynamic behavior of interest. Reducing complexity was and still is an important reason for dealing with hybrid systems; this is accomplished by incorporating models of dynamic processes having different levels of abstraction. For example a thermostat typically sees a very simple, but adequate for the task in hand, model of the complex heat flow dynamics. For another example, in order to avoid dealing directly with a set of nonlinear equations one may choose to work with sets of simpler equations (e.g. linear), and switch among these simpler models. The advent of digital machines has made hybrid systems very common. Whenever a digital device interacts with the continuous world, the behavior involves hybrid phenomena that need to be analyzed and understood.

Hybrid control systems typically arise from computer aided control of continuous processes in industrial processes, manufacturing and communication networks for example. They also arise from the hierarchical organization of complex control systems. There, hierarchical organization helps manage complexity and higher levels in the hierarchy require less detailed models (discrete abstractions) of the functioning of the lower levels (continuous dynamics), necessitating the interaction of discrete and continuous components. The study of hybrid control systems is essential in designing sequential supervisory controllers for continuous systems, and it is central in designing intelligent control systems with a high degree of autonomy. Hybrid system analysis and controller synthesis techniques could provide an approach for design and verification of intelligent control systems that exhibit a truly autonomous operation.

Hybrid control systems appear in the intelligent autonomous control system framework whenever one considers the execution level together with control functions performed in the higher coordination and management levels. Examples include expert systems supervising and tuning conventional controller parameters, planning systems setting the setting points of local control regulators, and sequential controllers deciding which one of a number of conventional controllers is to be used to control a system, to mention but a few.

The analysis, design, simulation, and verification of hybrid systems requires the development of computationally efficient algorithms and approaches. Several models have been proposed in the literature for the development of analysis and controller synthesis techniques. Timed automata and hybrid automata have been used by several researchers for modeling, verification and controller synthesis techniques of hybrid systems. Although the initial results concerning the complexity of approaches based on timed and hybrid automata were negative, recent efforts have proposed systematic techniques that are applicable to a large class of problems. Because of the importance of hybrid automata based methods, we outline basic computational issues of hybrid automata based approaches later in this contribution.

Recently, a class of timed Petri nets named *programmable timed Petri nets* [41] has been used to model hybrid control systems. The main characteristic of the proposed modeling formalism is the introduction of a clock structure which consists of generalized local timers that evolve according to continuous-time vector dynamical equations. They can be seen as an extension of the approach taken in [2, 1] that provide a simple, but powerful way to annotate the Petri net graph with generalized timing constraints expressed by propositional logic formulae. It is expected that the more powerful expressiveness of Petri nets will result in analysis and controller synthesis approaches of higher complexity than those based on hybrid automata. However, there are complex systems which include concurrency and/or conflict, buffer sizes that can be modeled more compactly using Petri nets than finite automata. There are also control specifications, for example mutual exclusion constraints, that can be studied more efficiently in a Petri net framework. Moreover, there is the need to investigate the applicability of recent results in Petri nets in a hybrid framework. Stability and supervisory control design of hybrid systems modeled by programmable timed Petri nets have been studied in [39]. In Subsection 4.5, we briefly outline this approach and focus on its computational advantages.

3 Elements of Computational Complexity Theory

This section contains some basic notions of complexity theory which are necessary for the study of computational issues in intelligent control. The discussion here is kept rather informal, and for precise results the reader is referred to texts in complexity theory (see for example [34, 58]).

3.1 Algorithms and Turing Machines

3.1.1 Basic Notions

First, some definitions are in order [34]. An *alphabet* is a finite set of symbols. A *string* over an alphabet Σ is a finite length sequence of symbols from Σ. We denote the set of all strings over a fixed alphabet Σ by Σ^*. A *language* L over an alphabet Σ is a set of strings of symbols over Σ.

In the following, the word *problem* is used to define a general question to be answered which may have several parameters whose values are to be determined. A problem is usually defined by describing its parameters and specifying the properties an (optimal) solution is required to satisfy. An *instance* of a problem is a list of values, one value for each parameter of the problem. In order to give a precise definition of a problem Π, we consider a fixed alphabet Σ (e.g. $\Sigma = \{0, 1, \Box\}$) and an encoding scheme that translates any instance of the problem to a string of symbols over Σ. Therefore, a problem can be defined mathematically as a subset Π of $\Sigma^* \times \Sigma^*$. Each string $\sigma \in \Sigma^*$ than encodes all the known parameters of the problem is called *input* of Π. A string $\tau \in \Sigma^*$ is called an *output* or *solution* of Π if $(\sigma, \tau) \in \Pi$. A *decision problem* is a problem with yes or no answer. A decision problem can be defined mathematically as a subset of Σ^*, or equivalently as a language over Σ.

In order to solve problems we develop procedures that utilize computing resources. The formal descriptions of these procedures are called *algorithms.* An algorithm is identified with some computer model and therefore, the study of algorithms requires the definition of a computer model.

The model that is used more often to represent a real-world computer is the Turing machine. A multi-tape Turing machine consists of a finite control with k tape heads and k tapes. On a single step, depending on the state of the finite control and the symbol scanned by each of the tape head, the machine can: (i) change state, (ii) print a new symbol on each of the cells scanned by its tape heads, (iii) move each of its tape heads, independently, one cell to the left or right, or keep it stationary.

Definition A *k-tape Turing machine* is denoted by the seven-tuple

$$M = (Q, \Gamma, \Sigma, \delta, b, q_0, q_f)$$

where

1. Q is the finite set of *states.*
2. Γ is the *tape alphabet.*
3. Σ is the *input alphabet*, $\Sigma \subseteq \Gamma$.
4. $b \in \Gamma \setminus \Sigma$ is the *blank* symbol.
5. q_0 is the *initial state.*
6. q_f is the *final (accepting) state.*
7. $\delta : Q \times \Gamma^k \to Q \times (\Gamma \times \{L, R, S\})^k$ is the *transition function.* δ is a map that for some $(k+1)$-tuples consisting of k tape symbols and a state, it gives a new state, k tape symbols and the direction for each of the tape heads.

A Turing machine is thought of as implementing a computation as follows. Initially, the first tape holds a string of input symbols, one symbol per cell. All other tapes are completely blank. The machine is said to be in internal state q_0. The computation proceeds in discrete steps. At the beginning of each step, the machine is in exactly one of the states in Q and is scanning a new symbol on each of the tapes, say $(a_1, a_2, \ldots, a_k)$, $a_i \in T$, $i = 1, \ldots, k$. Assume that $\delta(q, a_1, a_2, \ldots, a_k) = (q', (a'_1, d_1), (a'_2, d_2) \ldots, (a'_k, d_k))$, then the machine first replaces the symbol a_i to a'_i on the current cell of the i-tape head $(i = 1, \ldots, k)$, it moves the i-tape head in the direction d_i $(i = 1, \ldots, k)$, and finally it changes its internal state to q'.

The language accepted by M, denoted by $L(M)$, is the set of words in Σ^* that cause M to enter a final state. Given a Turing machine M recognizing a language L, it is assumed that M halts whenever the input is accepted. For not accepted words, it is possible that M will never halt. A language that is accepted by a Turing machine is said to be *recursively enumerable* (r.e.). Another important class of languages are the *recursive* languages which are defined as those accepted by at least one Turing machine that halts on all inputs.

An algorithm can be considered formally as a Turing machine M. The description of the parameters of the problem constitutes the input string of the Turing machine (after the application of an encoding scheme). The algorithm *solves* the problem for each input string and initial state of the machine q_0, if after a finite number of moves of the tape head, it stops in state q_f, while in tape 1 it writes a string which is a solution of the problem. Consider now a decision problem Π and encoding instances of the problem by strings of symbols over Σ. For these problems it is assumed that the answer is yes if the machine M halts and no otherwise. Therefore, the question whether there exists an algorithm for solving a decision problem can be transformed to whether or not a particular language is recursive.

3.1.2 Decidable and undecidable problems

The discussion now is focused on the existence of algorithms for decision problems. While it may seem restrictive to consider only decision problems, in fact this is not the case since many general problems can be transformed to decision problems that are provably as difficult as the general problem.

A problem whose language is recursive is said to be *decidable*. Otherwise, the problem is *undecidable*. That is, a problem is undecidable if there is no algorithm that takes as input an instance of the problem and determines whether the answer to that instance is “yes” or “no”. *Semi-decidable* procedures are often proposed to deal with undecidable problems. These algorithms produce the correct answer if they terminate, but their termination is not guaranteed.

3.1.3 Computational Complexity

It follows from the previous discussion that there are problems that are unsolvable on a Turing machine (and by Church’s thesis on any computer). The following discussion is focused on decidable problems. In particular, we classify decidable problems based on the amount of time, space, or other resource needed to solve a problem (recognize the corresponding language) on a universal computer model, such as a Turing machine.

Consider a multi-tape Turing machine M. If for a given string σ of length n, M makes at most $T(n)$ number of steps before halting, then M is said to be of *time complexity* $T(n)$ with *time complexity function* $T(n) : \mathbb{N} \to \mathbb{N}$. The language $L(M)$ accepted by M is also said to be of *time complexity* $T(n)$.

Similarly, if for every input string of length n, M scans at most $S(n)$ cells on any tape, then M is said to be of *space complexity* $S(n)$ with *space complexity function* $S(n) : \mathbb{N} \to \mathbb{N}$. The language $L(M)$ is also said to be of *space complexity* $S(n)$.

3.1.4 The Classes $\mathcal{P}$ and $\mathcal{NP}$

An algorithm is said to be of *polynomial time (space) complexity* if its time (space) complexity function $f(n)$ satisfies $f(n) \leq p(n)$ for some polynomial p. The class of all decision problems for which a polynomial time algorithm exists is called the class $\mathcal{P}$. Intuitively, $\mathcal{P}$ is the class of problems that can be solved efficiently. The class of decision problems that can be solved by a deterministic Turing machine by using a polynomial amount of working space is denoted by PSPACE. Similarly, EXPSPACE is used to denote the class of problems that need an exponential amount of space.

There is a number of important problems that do not appear to be in $\mathcal{P}$ but have *nondeterministic polynomial algorithms*. This class of problems is denoted by $\mathcal{NP}$. To define formally the class $\mathcal{NP}$, the *nondeterministic Turing machine* is introduced.

A *k-tape nondeterministic Turing machine consists* of a finite control with k tape heads and k tapes. The difference is that the transition function δ is a mapping from $Q \times \Gamma^k$ to subsets of $Q \times (\Gamma \times \{L, R, S\})^k$. For a given state and a list of k tape symbols, δ returns a finite number of choices for the next step. Each choice consists of the new state, k new tape symbols and k directions of the tape heads. The nondeterministic Turing machine may choose any of these steps, but is not permitted to perform a step where the next state is selected from one choice and the new tape symbols and/or directions from another. The nondeterministic TM accepts its input if any sequence of choices of steps leads to an accepting state. Note that the addition of the nondeterminism to the Turing machine does not allow the device to accept any new language.

A problem belongs to the class $\mathcal{NP}$ if there exists a nondeterministic polynomial algorithm (which is identified by a nondeterministic Turing machine). Of course, any attempt to execute a nondeterministic algorithm in a deterministic device will need much more time since all possible choices have to be performed. Given a language L in the class $\mathcal{NP}$ then L is accepted by a deterministic Turing machine of time complexity $k^{p(n)}$, for some constant k and polynomial p. However, despite enormous research effort there is no language L in $\mathcal{NP}$ that has been proved that is not in $\mathcal{P}$.

Finally, we close the discussion of basic notions in complexity theory with the class of $\mathcal{NP}$-complete problems. A problem Π (or language) is $\mathcal{NP}$-complete if $\Pi \in \mathcal{NP}$ and every other problem in $\mathcal{NP}$ can be polynomially reduced to Π. What is interesting about this class is that if a polynomial algorithm exists for any of these problems, then all problems in $\mathcal{NP}$ will be polynomial time solvable. $\mathcal{NP}$-completeness characterizes problems that are "hard" in a well-defined sense and more likely they are not in $\mathcal{P}$. Note that the notion of completeness is more general and can be applied to any class $\mathcal{C}$ of problems. We say that a problem Π is $\mathcal{C}$-complete if $\Pi \in \mathcal{C}$ and every Π' in $\mathcal{C}$ can be polynomially reduced to Π.

3.1.5 Subroutines and Numerical Considerations

It is very convenient both in practice and in theoretical studies to use subroutines to define elementary processes within an algorithm. The subroutines found in programming languages can be simulated in Turing machines. The general idea is to write a Turing machine as a subroutine which has an initial and a return state. To design a Turing machine which calls a subroutine, a new set of states for the subroutine is made and a move from the return state of the subroutine is specified. The call is implemented by a move in the initial state of the subroutine and the return by the move from the return state.

Subroutines can be used to describe arithmetic operations in a Turing machine. It is assumed that the numbers involved are represented in binary notation (using some encoding scheme). Each arithmetic operation takes a number of steps (moves of the tape head) to be completed. To study the complexity of an algorithm is more natural to consider the *elementary operations.* In the case of a numerical algorithms these are the elementary arithmetic operations (addition, subtraction, multiplication, division, comparison) rather than the moves of a tape head of a fictitious Turing machine. In this case, we analyze the complexity of an algorithms in an *arithmetic model.* Therefore, the time complexity of an algorithm will be different in the arithmetic model than that in a Turing machine. An elementary arithmetic operation takes a number of steps (moves of the tape head) which is bounded by a polynomial in the encoding lengths of the arguments. These operations can be implemented by subroutines and executed in a Turing machine in polynomial time. If the number of digits of the numbers in the intermediate steps occurring during the run of the algorithm is bounded by a polynomial in the encoding length, then a polynomial number of arithmetic operations can be executed in polynomial time in a Turing machine. An algorithm, is *strongly polynomial time* if it is polynomial space algorithm and performs a number of elementary arithmetic operations which is bounded by a polynomial in the number of input numbers. Of course, it is important to study the encoding scheme and their precision, but this is out of the scope of this introductory discussion.

3.2 Computational Complexity in Intelligent Control

In this contribution, the computational aspects of intelligent control methodologies are considered. In particular, computational issues in the analysis, controller synthesis and simulation of discrete event and hybrid systems are studied. As it is discussed earlier, discrete event and hybrid models are used in the higher levels of the hierarchical architecture of an intelligent autonomous system. The choice of whether to use such models will, of course, depend on what properties of the autonomous system need to be studied. However, the use of these models requires computational efficient approaches for analysis and controller synthesis. Therefore, it is important to review basic computational issues for formal discrete event and hybrid models. For complex processes, as expected these formal models give rise to algorithms of high complexity.

Simulation is the cornerstone of modern design engineering. The main objective of simulations is in general the extraction of useful information about the system to facilitate the design of decision making algorithms to control the system for it to exhibit desirable behavior. Simulation of complex systems has to be efficient since it is normally required to run repeatedly. Efficient simulation is a trade off between speed and accuracy. For intelligent control systems, it is important to simulate at the right level of abstraction. The importance of simulation of discrete event and hybrid systems in intelligent control applications is clear. Consider for example, the simulation of manufacturing systems. We must be able to simulate different system components, such as different number and type of machines and support equipment (transporters, conveyors, buffers, queues). Usually, the basic issue in such systems is the performance evaluation which can be affected by scheduling, control strategies, quality control and so on. Therefore, there is the need for simulation at different levels of abstraction. To address the computational requirements of intelligent control applications we present in Section 6 a high performance parallel architecture.

There are many cases when simulation cannot guarantee the proper operation of a complex system. Especially, in safe critical systems (power plants, traffic management systems and so forth) the desirable behavior must be verified. Verification is concerned with proving properties of interest independently of input choices. Formal verification is a set of techniques that facilitate proving that some specified properties are true for a design. These techniques require a formal model of the behavior of the system, as well as of the properties we wish to verify. Therefore, when intelligent control methodologies are used for systems that must exhibit properties that are required to be verified, formal methods have to be used. This is another case where the use of discrete event and hybrid systems in intelligent control is necessary. If the verification procedure determines that the system description satisfies the critical properties, then we can proceed with the actual design stage. Otherwise, error trace can be used to discover the source of the error and modify the design accordingly. The complexity of processes studied in intelligent control makes the use of the proposed hierarchical architecture very useful in the verification phase. The use of discrete event and hybrid models facilitate the use of verification techniques at different levels of abstraction. Computational issues of formal verification techniques for discrete event and hybrid models are studied at length in this contribution.

The purpose of simulation and verification techniques is to facilitate the design of decision making algorithms so that the system operates properly. During the design phase, which is an iterative process, the system is modified by synthesizing controllers that restrict the behavior of interest. In the case when the behavior of the system is too complex, it is sometimes desirable to consider restrictive controllers that that are not necessarily “optimal”. Usually such controllers can be designed more efficiently and they also lead to simpler implementations. Computational issues concerning controller synthesis for discrete event and hybrid systems are discussed throughout this work.

4 DES in Intelligent Control using Petri Nets

In this section, Petri nets are used to study computational issues that appear in connection with DES in intelligent control. As it was mentioned in the introduction, there are several results on computational issues of DES that use finite automata and the reader is referred to [80, 57, 71] and the references therein for more information. Petri net models have a wide range of applications in intelligent control as task planning and fault diagnosis. They are especially useful in the case of concurrent systems and they can be enhanced to model various dynamical systems. In this section, several computational issues in the analysis and design of systems modeled by Petri nets are studied. Decidability issues for checking basic properties in Petri nets are discussed. The use of integer programming for checking system properties is also presented. In addition, the computational advantages of unfolding algorithms that address the state explosion problem in Petri nets are examined. Finally, a synthesis method for Petri net supervisors is briefly presented with the emphasis on its computational efficiency.

4.1 Petri Nets: Basic Notions

Petri nets are a powerful modeling paradigm for a variety of systems [66, 70, 53]. Their basic characteristic is that they provide an excellent tool for capturing concurrency and conflict within a system. They have an appealing graphical and mathematical representation and they have been used extensively to model information processing systems, manufacturing systems, communication systems, industrial processes and so forth. In the following, some basic notions of Petri nets that are necessary for the following sections are presented.

Definition (Petri net) A *Petri net structure* is defined as a 3-tuple $N = (P, T, F)$ where P is a finite set of *places*, T is a finite set of *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is the *incidence relation* representing a set of directed arcs connecting places to transitions and vice versa.

The *preset* and *postset* of a place p are defined by $\bullet p = \{t \mid (t, p)\} \in F$ and $p\bullet = \{t \mid (p, t) \in F\}$. The *preset* and *postset* of a transition t are respectively $\bullet t = \{p| \ (p, t) \in F\}$ and $t\bullet = \{p| \ (t, p) \in F\}$.

The *marking* of a Petri net is a mapping $\mu : P \to \mathbb{N}$ from the set of places onto the nonnegative integers which assigns to each place p a number of tokens $\mu(p)$. The dynamics of ordinary Petri nets are characterized by the evolution of the marking vector which is referred to as the *state* of the net. A *net system* $\langle N, \mu_0 \rangle$ is a net $N = (P, T, F)$ with initial marking μ_0.

The marking can be represented by an m-dimensional column vector $\mu = (\mu_1, \ldots, \mu_m)$, where $m = |P|$ is the number of places. The vector μ gives for each place p_i, the number of tokens in that place, $\mu_i = \mu(p_i)$. The marking can be identified also with the multiset containing $\mu(p_i)$ copies of p_i for every $p_i \in P$. A multiset is a collection of elements over some domain that, unlike a set, allows multiple occurrences of the elements. To avoid confusion, the marking μ is interpreted as a mapping when it is appeared with an argument and as a vector of nonnegative integers otherwise. Multiset relations are frequently used. For example, the notation $\mu \subset \mu'$ is interpreted as a multiset inclusion relation and it is true if and only if $\mu(p_i) < \mu'(p_i)$ for all $p_i \in P$.

The transition t is *enabled* when each one of its input places is marked with at least one token, $\mu(p) > 0$ for all $p \in \bullet t$. An enabled transition may fire. If $\mu(p)$ and $\mu'(p)$ denote the marking of place p before and after the firing of enabled transition t, then

$$\mu'(p) = \begin{cases} \mu(p) + 1 & \text{if } p \in t\bullet \setminus \bullet t \\ \mu(p) - 1 & \text{if } p \in \bullet t \setminus t\bullet \\ \mu(p) & \text{otherwise} \end{cases} \tag{1}$$

In words, firing an enabled transition t causes one token to be removed from each place $p \in \bullet t$, and one token to be added to each $p \in t\bullet$. The firing of the transition t which is enabled at marking μ and results in the new marking μ' is denoted as $\mu \, [\, t \,\rangle \, \mu'$.

A *firing sequence* from a marking μ_0 is a sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ such that $\mu_0 \, [\, t_1 \,\rangle \, \mu_1 \, [\, t_2 \,\rangle \, \mu_2 \, \ldots \, [\, t_n \,\rangle \, \mu_n$. A marking μ is *reachable* in the net system $\langle N, \mu_0 \rangle$ if there exists a firing sequence such that $\mu_0 [\sigma \rangle \mu$. The set of reachable markings from μ_0 in the Petri net N is denoted by $R(N, \mu_0)$.

State space description of Petri nets The dynamic behavior of concurrent systems modeled by Petri nets can be described also by matrix equations. These equations are similar to the difference equation that are used to describe linear discrete-time systems. However, their use is more difficult because their solutions are restricted to be nonnegative integers that represent the number of times a certain transition fires.

Let $\mathbb{N}$ be the set of nonnegative integers and let $m = |P|$ and $n = |T|$ denote the number of places and transitions respectively. The incidence relation can be represented using two matrices. The arcs connecting transitions to places are described by the matrix $D^+ \in \mathbb{N}^{m \times n}$ and the arcs connecting places to transitions are described by the matrix $D^- \in \mathbb{N}^{m \times n}$. Then the Petri net *incidence matrix* is defined as $D = D^+ - D^-$. Recall that the marking is represented with the m dimensional integer vector μ and describes the distribution of tokens throughout the net. Let μ_k denote the marking of the Petri net after the kth execution. Using the incidence matrix, the next marking μ_{k+1} is determined by

$$\mu_{k+1} = \mu_k + Dq \tag{2}$$

where q is the n-dimensional *firing vector*. Each entry of the vector q represents the number of times the corresponding transition has fired during the k^{th} execution of the net. Equation (2) is called the *state equation* of a Petri net. A given firing vector represents a valid possible firing if all of the transitions for which it contains nonzero entries are enabled. The validity of a firing vector q can be determined by checking the *enabling condition* $\mu \geq D^- q$. In the remaining of the section, both the graphical and algebraic representations of Petri nets are used to discuss the computational complexity of central analysis and synthesis problems.

4.2 Decidability Issues in Petri Nets

In spite of the rather large expressive power of Petri nets, most of the interesting properties for verification purposes are decidable, however they tend to have large complexities. In the following, we review some basic decidability results for Petri nets. For more details see [19] and references therein.

Boundness A net system is *bounded* if there exists finite $k \in \mathbb{N}$ such that $\mu(p) \leq k$ for every place p and reachable marking $\mu \in R(N, \mu_0)$. The set of reachable markings for a bounded Petri net is finite. If the net is used to model systems with buffers or registers then the verification of the boundness property is essential to guarantee that there will be no overflows in the system. The boundness problem for Petri nets is decidable. Checking boundness requires at least space $2^{c\sqrt{n}}$ where c is a constant and n is the size of the Petri net that reflects the number of places, transitions and their interconnections. In the case when the bound k is constant and $k \geq 4$ then the problem is PSPACE-complete. A net N is *structurally bounded* if it is bounded for all possible markings. It has been shown that a net is structurally bounded if and only if the system of linear inequalities $XD \leq 0$ has a solution [49]. Therefore, the structural boundness can be decided in polynomial time using linear programming.

Reachability The reachability problem is one of the fundamental problems for Petri net analysis. Given a marking μ of the net system $\langle N, \mu_0 \rangle$, the reachability problem is the problem of deciding if $\mu \in R(N, \mu_0)$. The reachability problem is decidable. A lower bound for its complexity is that it needs at least exponential space and exponential time. An extension of Petri nets named *extended Petri nets* have been defined to increase the expressive power of ordinary Petri nets. These nets contain *inhibitor arcs* from places to transition. If the place p is connected with the transition t via an inhibitor arc, then t can fire only if $\mu(p) = 0$ (zero detector). It is interesting that the reachability problem of Petri nets with one inhibitor arc is decidable while with at least two is undecidable.

Liveness The notion of liveness is fundamental for the detection and avoidance of deadlocks. A transition t is *live* in a marking μ_0 if for each $\mu \in R(N, \mu_0)$ there exists a firing sequence σ such that μ enables t. A net system is live with respect to the initial marking if every transition is live. The liveness problem is recursively equivalent to the reachability problem, and thus decidable. Relevant to the liveness notion is deadlock-freedom. A Petri net is deadlock-free with respect to μ_0 if every reachable marking $\mu \in R(N, \mu_0)$ enables a transition. The problem of deadlock-freedom can be reduced in polynomial time to the reachability problem.

Persistence Persistence is a useful property in the verification of parallel computing protocols and asynchronous circuits [37]. It is related to conflict-freedom and is also central to identifying and allocating shared resources in manufacturing systems. A Petri net is persistent if for any marking in $R(N, \mu_0)$ an enabled transition can be disabled only by its own firing. If a Petri net is persistent, then for any two enabled transitions, the firing of the one transition will not disable the other. The problem to decide if a given Petri net is persistent is decidable. In [37] persistence of Petri nets is efficiently analyzed using unfoldings (see discussion later in the section).

Equality problem for Petri net reachability sets Consider two net systems $\langle N_1, \mu_1 \rangle$ and $\langle N_2, \mu_2 \rangle$, then the problem of checking if $R(N_1, \mu_1) = R(N_2, \mu_2)$ is undecidable. Deciding if $R(N_1, \mu_1) \subseteq R(N_2, \mu_2)$ is also undecidable. The proof of these statements is based on the *Hilbert's tenth problem* [66]. It can be shown that the language inclusion problem is also undecidable for Petri nets. Assume that the system and the desired specifications have been modeled by the Petri nets N_1 and N_2 respectively. The above undecidability results prohibit the automated verification for proving that the specifications represented by the net N_2 are satisfied by the system N_1. However, there are subclasses of Petri nets for which these problems are decidable and algorithms for automated verification have been developed. An interesting special case is for bounded Petri nets where the set of reachable markings is finite.

Reachability tree The simplest way to investigate the reachability problem of Petri nets is to expand its *reachability tree*. The reachability tree represents an exhaustive enumeration of all the reachable markings. Starting with the initial marking μ_0 all the enabled transitions are fired. This leads to a set of new possible markings. Taking each of those as a new root, the reachability tree can be constructed recursively. If the Petri net is bounded and therefore it has a finite reachability set then this procedure will terminate. In the case of an unbounded net, it is possible that the reachability tree could grow indefinitely. However, by using a special symbol ω as pseudo-infinity to represent number of tokens that can be made arbitrarily large, it can be proved [66] that the reachability tree is finite. The analysis of Petri nets using the reachability tree has its limitations. For example, it cannot, in general, be used to solve the reachability or the liveness problems because the presence of the pseudo-infinity problem leads to a loss of information. In addition, the size of the reachability tree can grow exponentially with respect to the size of the original Petri net, thus the use of the reachability tree for analysis of Petri nets is computationally inefficient. Alternative methods for avoiding the state explosion problem have been proposed; see the discussion on unfolding later in this section.

4.3 Checking properties using integer programming

As it was discussed earlier, the dynamic behavior of a Petri net can be described by a matrix equation known as *state* or *marking equation*. The use of the marking equation makes possible the application of linear algebraic techniques for the analysis and verification of Petri nets. In particular, we are interested in how integer programming can be used to check properties of interest. For more details the reader is referred to [48].

The marking equation is derived using the initial marking and the incidence matrix of the net and it can be seen as a set of linear constraints $\mathcal{L}$ that every reachable marking must satisfy. It is important to notice that the set of reachable markings is a subset of the solutions of the linear constraints $\mathcal{L}$. Assume we want to check a property of interest P and let $\mathcal{L}_P$ be a set of linear constraints that specify the markings that do not satisfy P. Then if the system $\mathcal{L} \cup \mathcal{L}_P$, which can be solved using integer programming, does not have a solution every reachable marking satisfies the property P. The disadvantage of this method is that the solution of $\mathcal{L} \cup \mathcal{L}_P$ may or may not correspond to a reachable marking.

If $\mu \in R(N, \mu_0)$ then the following problem has at least one solution with respect to the n-dimensional vector x, which corresponds to the firing sequence σ such that $\mu_0 \, [\, \sigma \,\rangle \, \mu$.

$$\begin{aligned}
&\text{Variables: } x \text{ integer} \\
&\mu = \mu_0 + Dx \\
&x \geq 0
\end{aligned}$$

It is often desirable to check a property P which corresponds to linear (or equivalently convex) constraints on the marking of the Petri net. These properties are general enough and usually correspond to generalized mutual exclusion constraints. Such a property can be described by the set of linear inequalities $L\mu \leq b$, where L, b are of appropriate dimensions and consist of integers. If the following integer programming problem does not have any solution with respect to the vectors x and μ, then every reachable marking satisfies the property $L\mu > b$.

$$\begin{aligned}
&\text{Variables: } x, \mu \text{ integer} \\
&\mu = \mu_0 + Dx \\
&L\mu \leq b \\
&x, \mu \geq 0
\end{aligned}$$

Integer programming and mixed integer programming can be used to check other properties of Petri nets as deadlock-freedom (see [48]). An additional disadvantage is the $\mathcal{NP}$-completeness of the integer programming problem. There are many applications in the area of intelligent control (for example manufacturing systems or communication protocols) that is desirable for the discrete state to satisfy convex constraints. For large scale systems, to check if such a property holds can be computationally expensive. Another approach is to modify the system to guarantee that such constraints will be satisfied. In Subsection 4.5, a method for designing a supervisor to enforce linear constraints on the marking is discussed. The method is very simple and computationally efficient. The Petri net is changed by adding appropriate monitor places that are determined by a single matrix multiplication.

4.4 State space search using unfolding

Net unfoldings is a well known partial order semantics of Petri nets [56, 18] and provide a method of searching the state space without considering all the interleavings of concurrent events. An unfolding technique to avoid the state explosion problem in the verification of systems modeled by Petri nets has been proposed by McMillan [47]. Specifically, an algorithm to construct a finite prefix of the unfolding which contains full information about the reachable states is introduced and then the algorithm is used for deadlock detection. The unfolding technique has been enhanced and applied to other verification problems, see for example [20, 37]. The advantage of the unfolding technique over an exhaustive state space search is that it takes into consideration the captured conflict in the net and narrows the interleavings of concurrent transitions. This section presents briefly the basic notions and the computational advantages of Petri net unfoldings.

The first result of the unfolding algorithm is an acyclic net called occurrence net. Briefly, an occurrence net is a Petri net without backward conflict (two transitions outputting in the same place), and without cycles. Formally, an *occurrence net* is a net $N' = (P', T', F')$ such that (i) for every $p \in P'$, $| \bullet p'| \leq 1$, (ii) F' is acyclic, i.e. the (irreflexive) transitive closure of F' is a partial order, (iii) N' is finitely preceded, i.e. for every $x' \in P' \cup T'$, the set of elements $y' \in P' \cup T'$ such that (y', x') belongs to the transitive closure of F is finite, and (iv) no transition $t' \in T'$ is in self conflict. In the following, $Min(N')$ denotes the set of minimal elements of $P' \cup T'$ with respect to the transitive closure of F'.

Let $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$ be two nets. A *homomorphism* from N_1 to N_2 is a mapping $h : P_1 \cup T_1 \to P_2 \cup T_2$ such that: (i) $h(P_1) \subseteq P_2$ and $h(T_1) \subseteq T_2$ and (ii) for every $t \in T_1$, the restriction of h to $\bullet t$ is a bijection between $\bullet t$ (in N_1) and $\bullet h(t)$ (in N_2), and similarly for $t\bullet$ and $h(t)\bullet$.

A *branching process* of a net system $\langle N, \mu_0 \rangle$ is a pair $\beta = (N', h)$ where $N' = (P', T', F')$ is an occurrence net and h is a homomorphism from N to N' such that (i) the restriction of h to $Min(N')$ is a bijection between $Min(N')$ and μ_0 (μ_0 is interpreted as a multiset), and (ii) for every $t_1, t_2 \in T'$, if $\bullet t_1 = \bullet t_2$ and $h(t_1) = h(t_2)$ then $t_1 = t_2$. Two branching processes $\beta_1 = (N_1, h_1)$ and $\beta_2 = (N_2, h_2)$ of a net system are *isomorphic* if there is a bijective homomorphism h from N_1 to N_2 such that $h_2 \circ h = h_1$. Furthermore, (N_1, h_1) contains (N_2, h_2) if $N_2 \subseteq N_1$ and the restriction of h_1 to nodes in N_2 is identical to h_1. An *unfolding* is the maximal branching process up to isomorphism associated with a net system.

Consider the unfolding $\beta = (N', h)$ of the net system $\langle N, \mu_0 \rangle$. The homorphism h can be seen as a label function that preserves the environment of the transitions. More specifically, the following remarks are concluded from the previous definitions. Since N' is an occurrence net, the unfolding is finitely preceded and it contains neither forward conflict ($| \bullet p'| \leq 1$) nor self-conflict. Since β is a branching process, it has no redundancy (for every $t_1, t_2 \in T'$, if $\bullet t_1 = \bullet t_2$ and $h(t_1) = h(t_2)$

then $t_1 = t_2$). Since h is a homomorphism, the labels of the preset and postset of any transition in the unfolding match the preset and postset of the corresponding transition in the original net (for every $t \in T_1$, the restriction of h to $\bullet t$ is a bijection between $\bullet t$ (in N_1) and $\bullet h(t)$ (in N_2), and similarly for $t\bullet$ and $h(t)\bullet$). Also, the labels of the places in the unfolding with no predecessors match the initial marking of the original net system (the restriction of h to $Min(N')$ is a bijection between $Min(N')$ and μ_0).

In general, the unfolding of a net system is infinite in size. It is possible, however, to construct finite prefixes of a maximal branching process which enumerate the reachable markings in a computationally efficient manner. An important theoretical notion regarding occurrence nets is that of a *configuration*. A configuration is a set of events representing a possibly partially ordered run of the net. In an unfolding, each transition corresponds to a transition of the original net (via the mapping h). We can associate each configuration of the unfolding with a state (marking) of the original net by simply identifying those places whose tokens are produced but not consumed by the transitions in the configuration. Then, it can be shown that every marking represented in a branching process is reachable, and that every reachable marking is represented in the unfolding of the net system. The *local configuration* associated with any transition consists of that transition and all of its predecessors in the dependency order. This is the set of transitions which necessarily are contained in any configuration containing the given transition.

Consider the problem of building a fragment of the unfolding which is large enough to represent all reachable markings of the original net. The process starts with a set of places corresponding to the initial marking of the original net. The unfolding is grown by finding a set of places which correspond to the inputs (preset) of a transition in the original net, then adding a new instance of that transition to the unfolding, as well as a new set of places corresponding to its outputs (postset). If the new transition has no conflicts in its local configuration (more precisely, if it has a local configuration) it is kept, otherwise it is disregarded. This is because the existence of a conflict means that the new transition can occur in no configuration of the unfolding. The unfolding of a net system is always complete. A *complete* prefix contains as much information as the unfolding. Since a bounded net system has only finitely many reachable markings, its unfolding contains at least one complete finite prefix.

The key to termination of the unfolding is to identify a set of transitions of the unfolding to act as *cut-off points*. This set must have the following property: any configuration containing a cut-off point must be equivalent (in terms of final state) to some configuration containing no cut-off points. From this, it follows that any successor of a cut-off point can be safely omitted from the unfolding without sacrificing any reachable markings of the original net. A sufficient condition for a transition to be a cut-off point is the following: the final state of its local configuration is the same as that of some other transition whose local configuration is smaller (see [20] for details).

Unfoldings of Petri nets provide a method for avoiding the state-space explosion in analysis problems. The main advantage is the reduced size of the unfolding in comparison with the reachability tree. A simple and elegant algorithm for the construction of the unfolding of a Petri net is presented in [47]. The size of the produced unfolding can be exponential in the size of the Petri net. However, this is only the worst case, there are interesting applications when the size of the unfolding is linear in the size of the Petri net. In general, the size of the unfolding will be smaller than the size of the corresponding reachability tree and will depend on the degree of parallelism of the Petri net. In [47] it is also shown that the problem of existence of a marking that will result in deadlock in an occurrence net is $\mathcal{NP}$-complete. This is in agreement with the worst case analysis for the size of the unfolding of a Petri net. However, there are cases when the exponential complexity is avoided (for example the dining philosophers problem).

An improvement of McMillan's algorithms has been presented in [20] where it is shown that a minimal complete prefix can be constructed with size polynomial in the size of the Petri net. A technique that results also in a compression of the size of the unfolding is presented in [37]. Unfoldings can be used for the analysis of Petri nets to study all problems that are related to the reachability problem. Such problems include liveness, deadlock avoidance, boundness, and persistence. In view of the time complexity of the resulting algorithms, it can be easily shown that it will be polynomial in the size of the unfolding. Therefore, the size of the unfolding is the important factor in the analysis of Petri net following this approach.

4.5 Supervisory control theory of Petri nets

The methods discussed above are concerned with the analysis and verification of systems modeled by Petri nets and in general, do not lead to computationally efficient procedures. Another approach, motivated by the supervisory control theory in [69], aims at the modification of the original Petri net model (open loop plant) so that the resulting Petri net (closed loop) will satisfy the desirable properties.

If we assume that the specifications are expressed as a set of legal markings for the system, then the aim of the control is to restrict the behavior of the net so that only legal markings can be reached. In the following, we will briefly discuss approaches that rely on the linear algebraic representation of the Petri net model of the plant.

Li and Wonham [42, 43] consider the synthesis of maximally permissive feedback control policies when the legal markings are specified by a system of linear predicates. They showed that under certain assumptions the supervisory state feedback control problem can be reduced to solving a linear integer programming problem (in the presence of uncontrollable transitions). The attraction of the general linear integer programming approach to Petri nets is that the synthesis of supervisory control policies is reduced to the solution of a standard optimization problem, eliminating the need to compute the reachability graph of the Petri net.

Linear constraints on the marking vector can also be enforced by *monitor* or *controller* places. These places represent control places that are connected to existing transitions of the Petri net model. A methodology for DES control based on Petri net place invariants has been developed in [85, 51, 50, 52]. A *place invariant* of a Petri net is defined as every integer vector x which satisfies $x^T\mu = x^T\mu_0$ where μ_0 is the initial marking and μ any reachable subsequent marking. Place invariants characterize sets of places whose the weighted sum of tokens remains constant at all reachable markings and is determined only by the initial marking. Consider linear constraints of the form $L\mu_p \leq b$ on the marking vector μ_p of the plant net. This inequality can be transformed to the equality $L\mu_p + \mu_c = b$ by introducing an external Petri net controller whose places are represented by the "slack variables" μ_c. The incidence matrix of the controller is then computed by the equation $D_c = -LD_p$ and its initial marking is $\mu_{c_0} = b - L\mu_{p_0}$. The controller introduces place invariants in the closed loop system, that enforce the linear constraint $L\mu_p \leq b$. For more details on the place invariant method for controllable transitions see [85].

The significance of invariant based supervision techniques to Petri net controller design is that the control net can be computed very efficiently, thus the method shows promise for controlling large, complex systems, or for recomputing the control law online due to some plant failure. An invariant based supervisor is computed very efficiently by a single matrix multiplication, and its size grows polynomially with the number of specifications. In the case when all the transitions of the plant net are controllable and observable then the invariant based control method is shown to be maximally permissive [85].

A more challenging case arises by the presence of uncontrollable and unobservable transitions. Li and Wonham [42, 43, 44] show that optimal, or maximally permissive, control actions which account for uncontrollable transitions can be found by analytically solving an integer linear programming problem. If is not possible to solve the integer programming problem symbolically, then it is necessary for the controller to numerically solve integer programs at every iteration of the evolution of the discrete event system. This can be computationally very expensive for large problems. The approach presented in [50] for supervising a plant with uncontrollable/unobservable transitions is to actually modify the constraints themselves so that the new constraints account for the difficult structures in the plant. If it is possible to obtain an analytic solution for the transformed constraints, then the controller logic itself will be very simple. Two techniques are presented in [50] for generating transformations of linear constraints that will facilitate the controller synthesis in the presence of uncontrollable and/or unobservable transitions. The first technique involves the solution of an integer linear problem and the other the triangularization of an integer matrix through constrained row operations. Although the derived supervisors are not always maximally permissive, a more restricted control policy can be easily computed and implemented with monitor places. These suboptimal controllers may be sufficient for many tasks, depending on the application. The suitability of this technique has also be examined for deadlock avoidance and liveness. These methods that apply to nets where uncontrollable and/or unobservable transitions may be present, involve finding the invariants or siphons of a Petri net and computationally reduce to finding elements of the kernel of an integer matrix for which established algorithms exist (for more details see [51]).

5 Hybrid Systems in Intelligent Control

Hybrid control systems typically arise from the interaction of discrete planning algorithms and continuous processes, and, as such, they provide the basic framework and methodology for the analysis and synthesis of autonomous and intelligent systems. Whenever a computer program interacts with a physical process, hybrid system methodologies are necessary to guarantee the desirable operation of the system. Hybrid automata have been proposed as a model for hybrid systems and they have been studied extensively for the verification of computer programs that involve continuous variables. Because of the importance of hybrid automata in the study of hybrid systems, we review in Subsection 5.1 basic computational issues concerning the analysis of such systems. Some recent results for controller synthesis are also outlined with comments on their computational complexity. Although, most of the problems are computationally very difficult and even undecidable, for many interesting applications efficient algorithms can be developed. In Subsection 5.2, programmable timed Petri nets are presented as a model for hybrid systems and some analysis and controller synthesis algorithms are described with emphasis on their computational advantages.

5.1 Computational Issues in Hybrid Automata

Hybrid automata provide a general modeling formalism for the formal specification and algorithmic analysis of hybrid systems [1]. They are used to model dynamical systems that consist of both discrete and analog components which arise when computer programs interact with an analog environment in real time. In the following we review some computational issues in the analysis and verification of hybrid systems modeled by hybrid automata.

A hybrid automaton is a finite state machine equipped with a set of real-valued variables. More specifically, a hybrid automaton consists of a finite set $X = \{x_1, \ldots, x_n\}$ of real-valued variables and a labeled directed graph (V, E). A vertex $v \in V$ is called a *control mode* or *location* and is equipped with the following labeling functions: A *flow condition* or *activity* described by a differential equation in the variables in X and an *invariant condition* $inv(v) \in \mathbb{R}^n$. An edge $e \in E$ is called *control switch* or *transition* and is labeled with guarded assignment in the variables in X. A transition is enabled when the associated guard is true and its execution modifies the values of the variables according to the assignment. Another labeling function assigns to each transition an event from a finite set Σ.

A *state* $\sigma = (v, x)$ of the hybrid automaton consists of a control location $v \in V$ and a valuation $x \in \mathbb{R}^n$ of the variables in X. The state can change either by a discrete and instantaneous transition or by a time delay. A discrete transition changes both the control location and the real valued variables while a time delay changes only the values of the variables in X according to the flow condition. A *run* of a hybrid automaton H is a finite or infinite sequence

$$\rho: \ \sigma_0 \rightarrow^{t_0}_{f_0} \sigma_1 \rightarrow^{t_1}_{f_1} \sigma_2 \rightarrow^{t_2}_{f_2} \ldots$$

where $\sigma_i = (v_i, x_i)$ are the state of H and f_i is the flow condition for the vertex v_i such that (i) $f_i(0) = x_i$, (ii) $f_i(t) \in inv(v_i)$ for all $t \in \mathbb{R} : 0 \leq t \leq t_i$, and (iii) σ_{i+1} is a transition successor of $\sigma'_i = (v_i, f_i(t_i))$ and σ'_i is a time successor of σ_i.

An important notion for the realizability of the hybrid automaton is the divergence of time. A hybrid automaton is said to be *nonzeno* if it cannot prevent time for diverging. If a hybrid automaton is nonzeno only finitely many transitions can be executed in every bounded time interval.

Example [1] The hybrid automaton of Fig. 2 models a thermostat controlling the temperature of a room by turning on and off a heater. The system has two control modes *off* and *on*. When the heater is off the temperature of the room (denoted by the real valued variable x) is governed by the differential equation $\dot{x} = -Kx$ (flow condition). When the heater is on (control mode *on*) the temperature of the system evolves according to the flow condition $\dot{x} = K(h - x)$, where h is a constant. The location invariants and the transition relation are specified by logical formulas and by guarded commands in the variables in X respectively. These labeling functions detect when the temperature crosses the thresholds m and M and trigger an appropriate control switching.

Complex systems can be modeled by using the parallel composition of simple hybrid automata. The basic rule for the parallel composition is that two interacting hybrid automata synchronize the execution of transitions labeled with common events (for more details see [1]).

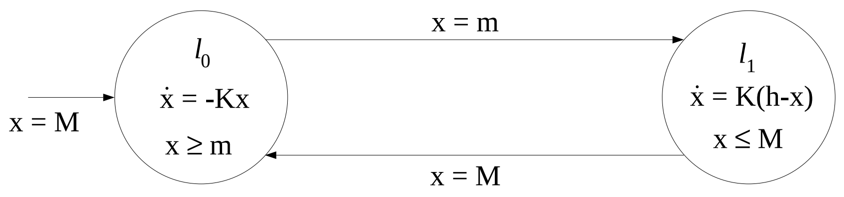

Figure 2: Hybrid automaton describing a thermostat

Linear hybrid automata The modeling formalism of hybrid automata is particularly useful in the case when the flow conditions, the invariants and the transition relations are described by linear expressions in the variables in X. However, for a lot of significant results that have been reported in the literature, hybrid systems are modeled by more general hybrid automata [46, 68, 29].

A hybrid automaton is *linear* if its flow conditions, invariants and transition relations can be defined by linear expressions over the set X of variables.Note, the special interpretation of the term linear in this context. More specifically, for the control modes the flow condition is defined by a differential equation of the form $\dot{x} = k$, where k is a constant, one for each variable in X and the invariant $inv(v)$ is defined by a linear predicate (which corresponds to a convex polyhedron) in X. Also, for each transition the set of guarded assignments consists of linear formulas in X, one for each variable. Note that the run of a linear hybrid automaton can be described by a piecewise linear function whose values at the points of first order discontinuity are finite sequences of discrete changes. An interesting special case of a linear hybrid automaton is a *timed automaton* [2]. In a timed automaton each continuous variable increases uniformly with time (with slope 1) and can be considered as a *clock.* A discrete transition either resets the clock of leaves it unchanged.

Another interesting case of a linear hybrid automaton is a rectangular automaton [28]. A hybrid automaton is rectangular if the flow conditions are independent of the control modes, and the variables are pairwise independent. In a rectangular automaton, the flow condition has the form $\dot{x} = [a, b]$ for each variable $x \in X$. The invariant condition and the transition relation are described by linear predicates that also correspond to n-dimensional rectangles. Rectangular automata are interesting because they characterize an exact boundary between the decidability and undecidability of verification problems of hybrid automata.

Decision Problems The main decision problems concerning the analysis and verification of hybrid systems is the emptiness problem, the language inclusion problem and the reachability problem. In the following we discuss some of the decidability results reported in the literature for hybrid automata [28, 31] and timed automata [2].

The emptiness problem is concerned with the existence of a divergent run and is a fundamental task for the verification of liveness requirements in hybrid automata. The emptiness problem for rectangular hybrid automata is PSPACE-complete. Checking the emptiness of timed automata is also PSPACE-complete (see [31] and [2] for complete proofs). On the negative side, the emptiness problem is undecidable for linear hybrid automata. This follows from stronger undecidability results reported in [1] for restricted classes of linear hybrid automata.

The reachability problem is formulated as follows. Let σ and σ' be two states in the infinite state space S of a hybrid automaton H. Then, σ' is reachable from σ if there exists a run of H that starts is σ and ends in σ'. The reachability problem is central to the verification of hybrid systems.

In particular, the verification of invariance properties is equivalent to the reachability problem. For example, a set $R \subset S$ is invariant if no state in $S \setminus R$ can be reached from an initial state of H. From the undecidability of the emptiness problem it follows that the reachability problem is also undecidable for linear hybrid automata. For decidability and undecidability results for particular classes of hybrid automata see [1, 32, 31].

Timed Automata The language inclusion problem for timed automata is very important in the automatic verification of finite state real-time systems. Given two timed automata A_1 and A_2 over an alphabet Σ, the problem of checking if $L(A_1) \subseteq L(A_2)$ is undecidable. However, if A_2 is deterministic, the previous problem is PSPACE-complete [2]. (In a deterministic timed automaton all the edges which stem from the same state have mutually exclusive clock constraints). The language inclusion problem for linear hybrid automata is more general and it is studied by introducing two labeled transition systems. The *time transition system* abstracts continuous flows retaining only information for the source and the destination locations and the duration of the flow. The *time-abstract transition system* abstracts also the duration of the flows. The timed inclusion problem compares the runs of a hybrid automaton with timed specifications, and the time-abstract inclusion problem compares the runs of a hybrid automaton with a time-abstract specification. For details see [28].

Verification of linear hybrid automata While the reachability problem is undecidable even for very restricted classes of hybrid automata, two semi-decision procedures, forward and backward analysis, have been proposed in [1] for the verification of safety specifications of linear hybrid automata. A *data region* R_v is a finite union of convex polyhedra in $\mathbb{R}^n$. A *region* $R = (v, R_v)$ consists of a location $v \in V$ and a data region R_v and is a set of states of the linear hybrid automaton. Given a region R, the precondition of R, denoted by $pre(R)$, is the set of all states σ such that R can be reached from σ. The postcondition of R, denoted by $post(R)$ is the set of all the reachable states from R. For linear hybrid automata both $pre(R)$ and $post(R)$ are regions, i.e. the corresponding data region is a finite union of convex polyhedra. Given a linear hybrid automaton H, an initial region R and a target region T, the reachability problem is concerned with the existence of a run of H that drives a state from R to a state in T. Two approaches for solving the reachability problem have been proposed. The first one computes the target region $post^*(R)$ of all states that can be reached from the initial state R and checks if $post^*(R) \cap T = \emptyset$ (forward reachability analysis). The second approach computes the region $pre^*(T)$ of the states that can be reached from the initial region R and checks if $pre^*(T) \cap R = \emptyset$ (backward reachability analysis). Since the reachability problem for linear hybrid automata is undecidable, these procedures may not terminate (semi-decision procedures). They terminate with a positive answer if T is reachable from R and a negative answer if no new states can be added and T is not reachable from R. The crucial step in these approaches is the computation of the precondition or postcondition of a region.

Controller synthesis approaches based on hybrid automata The undecidability of the reachability problem is a fundamental obstacle in the analysis and controller synthesis for linear hybrid automata. Nevertheless, considerable research effort has been focused on developing systematic procedures for synthesizing controllers for large classes of problems.

Tittus and Egardt [78] study control design for a class of hybrid systems with continuous dynamics described by pure integrators. Although this class of hybrid systems is rather limited, these models are very important for control of batch processes. Note that even in the case the continuous dynamics of the physical system are more complicated, it is efficient to use low-level continuous controllers to impose linear ramp-like setpoints. By using traditional feedback control in the execution level of the hierarchical architecture (Fig. 1), the dynamics of the low-level closed loops is abstracted by integrators in the coordination level. More specifically, the continuous dynamics in [78] are described by differential equations of the form $\dot{x}(t) = k_v$ where k_v is a constant vector associated with the control mode v of the hybrid automaton. The control specifications are represented by data regions $R_v = \{x \in \mathbb{R}^n : A_v x + b_v \leq 0\}$ and by a set Q_f of *forbidden control modes* or *forbidden control switches.* Controllability of hybrid integrator systems is defined with respect to a pair of regions of the hybrid state space. A hybrid system is controllable with respect to (R_1, R_2) if there exists an acceptable trajectory that drives the state (v, x) from R_1 to R_2. An acceptable trajectory is a trajectory of the hybrid system that satisfies the control specifications. For example, no forbidden control mode $v \in Q_f$ is visited and for every legal control mode v the continuous state x lies in R_v. Based on the definition of controllability, a semi-decidable algorithm is described that uses backward reachability analysis. The algorithm that analyzes these integrator hybrid systems with respect to controllability and, as a by-product, generates a set of correct control laws that switch the system between a predefined number of control modes. The semi-decidability of the algorithm is due to the undecidability of the reachability problem of linear hybrid automata. Note that an algorithm for backward reachability which can be applied in more general cases has been presented in [75].

Discrete time control for rectangular hybrid automata has been studied in [30], where it is shown that rectangular automata form a maximal class of systems for which the sampling-controller synthesis problem can be solved algorithmically. A realistic assumption for controller synthesis is that while the plant evolves in continuous time, the controller samples the state of the system in discrete time. The methodology for controller synthesis for hybrid automata can be seen as an extension of supervisor control theory [69]. Let Q be the set of states $(v, x), v \in V, x \in \mathbb{R}^n$ of the hybrid automaton H. A controller C is defined as a mapping $f_C : Q \to \Sigma$ from the set of states to the set of controllable events. The coupling of the hybrid automaton H with the controller C is defined as a infinite-state transition system. Given a region R of unsafe states, the basic control problem is to determine whether there exists a controller C such that the region R is unreachable in the closed loop system (H, C). In [30] this problem is called the *safety control decision problem.* In the case when the answer to this problem is affirmative, the problem of constructing a controller is referred as *safety controller synthesis problem.* It is proven in [30] that the safety control decision problem can be solved in PSPACE and the safety controller synthesis problem can be solved in exponential time. The safety control problem can be solved by iterating a predecessor operator on regions. In particular, the operator used is called the *uncontrollable-predecessor operator* $UPre(R) : 2^Q \to 2^Q$ and represents the set of states that no controller can keep out of R for even one transition.

A semi-decision procedure for synthesizing controllers for a larger class of linear hybrid automata has been presented in [83]. In this work, the continuous dynamics are governed by differential inclusions of the form $A\dot{x} \geq b$ where A and b are a constant matrix and vector respectively. The control problem is formulated as a safety requirement represented as a linear region R. A controller C is *legal* if all states that can be reached from the initial states of the hybrid automaton H lie in the safe region R. The supervisor control problem is concerned with the existence and the construction of a legal controller. It is shown in [83] that the controller synthesis problem for this class of linear hybrid automata with linear safety requirements is semi-decidable. The control problem in this case is also solved by iterating an appropriate predecessor operator. Controller synthesis procedures are presented under either full or partial observability and sufficient conditions for the nonzenoness of the synthesized controller are given. The efficiency of the method depends heavily on the efficiency of the algorithm implementing the predecessor operator.

A methodology for synthesizing controllers for nonlinear hybrid automata has been presented in [79]. Motivated by problems in aircraft conflict resolution, the authors developed a synthesis procedure based on game theoretic methodologies. The continuous dynamics are described by nonlinear differential equations (that satisfy appropriate conditions for the existence and uniqueness of solutions). The regions of the hybrid state space consist of arbitrary invariant conditions for the control modes and regions of the form $G = \{x \in \mathbb{R}^n : l(x) < 0\}$ where $l : \mathbb{R}^n \to \mathbb{R}$ is a differentiable function. The control specifications are expressed as acceptance conditions on the system's state. The controller synthesis problem is formulated as a dynamic game between the controller and the environment. The goal is to construct the largest set of states for which the control can guarantee that the acceptance condition is met despite the action of the disturbance. The problem is solved by iterating two appropriate predecessor operators. Consider a region K of the hybrid state space. The *controllable predecessor* of K contains all states in K for which the controllable actions can force the state to remain in K for at least one discrete step. The *uncontrollable predecessor* contains all states in K^c (the complement of K) and all states from which the uncontrollable actions may be able to force the state outside K. The computation of the predecessor operators is carried out using an appropriate Hamilton-Jacobi-Bellman equation. The computational efficiency of the synthesis procedure depends on the ability to solve efficiently this equation.

In summary, recent research efforts towards controller synthesis results have shown that there are classes of hybrid systems for which computationally tractable procedures can be applied. Although, many important problems related to hybrid automata are intrinsically difficult, there are efficient algorithms for large classes of systems. Many practical applications can be modeled accurately enough by simple hybrid models. Again, the choice of such models depend on their suitability for studying specific problems.

5.2 Programmable Timed Petri Nets

In this section, a class of timed Petri nets named *programmable timed Petri nets* [41] is used to model hybrid control systems. The main characteristic of the proposed modeling formalism is the introduction of a clock structure which consists of generalized local timers that evolve according to continuous-time vector dynamical equations. They can be seen as an extension of the approach taken in [1] that provide a simple, but powerful way to annotate the Petri net graph with generalized timing constraints expressed by propositional logic formulae. In contrast to previous efforts to include continuous processes in the Petri net modeling framework (for example [40, 24, 22, 16]), the proposed model still consists of discrete places and transitions, and it preserves the simple structure of ordinary Petri nets. The information for the continuous dynamics of a hybrid system is embedded in the logical propositions that label the different elements of the Petri net graph. In view of result on hybrid automata, corresponding problems of PTPNs will be of the same or higher complexity. The introduction of hybrid Petri nets does not aim at solving problems similar to those presented at Subsection 5.1. The motivation is to develop a framework to use supervisor control design similar to the one presented in Subsection 4.5. Supervisor control of Petri nets based on place invariants is a special case of the general supervisor control theory for which controllers can be synthesized very efficiently. With respect to continuous dynamics, the basic idea is to follow a natural invariant's approach as presented in [75]. In contrast to the hybrid automata based approaches presented above, these considerations limit the potential problems to cases where the continuous and discrete specifications are uncoupled.

Formally, a *programmable timed Petri net*, (PTPN) is denoted by the ordered tuple

$$(\mathcal{N}, \mathcal{X}, \ell_P, \ell_T, \ell_I, \ell_O)$$

where

- $\mathcal{N} = (P, T, I, O)$ is an ordinary Petri net where P, T, I, and O denote the set of places, transitions, input arcs (from places to transitions) and output arcs respectively.
- $\mathcal{X}$ is a set of N *local clocks* which can be seen as a collection of continuous-time dynamical systems. The ith clock, $\mathcal{X}_i$ is described by $\dot{x}_i = f(x_i)$ where $x_i \in \Re^n$ is the continuous state (local time) and $f : \Re^n \to \Re^n$ is Lipschitz continuous automorphism over $\Re^n$ characterizing the local clock's rate $\dot{x}_i$.
- $\ell_P : P \to \mathcal{P}$, $\ell_T : T \to \mathcal{P}$, $\ell_I : I \to \mathcal{P}$, and $\ell_O : O \to \mathcal{P}$ are functions that label the places, transitions, input arcs, and output arcs (respectively) of the Petri net $\mathcal{N}$. $\mathcal{P}$ is the set of the logical formulas which are constructed by applying propositional connectives between *rate constraints* ($\dot{x}_i = f(x_i)$), *time constraints* ($h(x_i) < 0$ and/or $h(x_i) = 0,\ h : \Re^n \to \Re$) and *reset equations* ($x_i(\tau) = \bar{x}_0$).

For more details in PTPN modeling of hybrid systems see [39, 41].

A PTPN can be used to model a hybrid dynamical system in the following manner. The network, $\mathcal{N}$, is used to represent the logical dependencies between mode switches. The timers, $\mathcal{X}$, of the PTPN are the dynamical equations associated with the continuous-time dynamics of the system. The labels ℓ_P, ℓ_T, ℓ_I, and ℓ_O are chosen to represent conditions on the continuous state for mode switches as well as describing the various switching behavior within the network.

Analysis of hybrid systems modeled by PTPNs PTPNs have been used in [39] for studying the uniform ultimate boundness of hybrid systems consisting of multiple linear time invariant plants and switching mechanisms that uses a logical rule described by a Petri net. Sufficient conditions for the stability of LTI switched systems can be found in [65, 10, 35]. Computational methods based on solving linear matrix inequalities (LMIs) for checking the sufficient conditions for switched system stability are provided in [36, 67]. The sufficient conditions that are used to compute candidate Lyapunov functionals can be very conservative, unless the structure of the switching law is explicitly account for. Petri net models of the switching logics can be used to extract useful information to formulate the appropriate LMIs. A sufficient condition [27] for the Lyapunov stability and ultimate bounded behavior of a switched LTI system is that a set of feasible LMIs associated only with the fundamental cycles of the system's reachability graph exist. The fundamental cycles can be found using computationally efficient techniques based on the unfoldings of the PTPN. This approach addresses the problem of identifying potential system faults that violate the specifications without having to resort to exhaustive simulation. The computational complexity of the approach depends on the complexity of the unfolding algorithms for Petri nets and on algorithms for solving LMIs.

Supervision of hybrid systems In a hybrid control architecture, the supervisor control algorithms must guarantee the proper and safe operation of the system for a *large number* of different plans. Moreover, the algorithms should exhibit some capability to *react* to the perceived situation in order to handle unexpected events and uncertain plant behavior. Supervisor control algorithms based on invariant properties of the discrete and continuous dynamics have been proposed in [39, 38]. These algorithms are realized using state feedback control (discrete or continuous) and therefore the control action depends on the state of the system.

A methodology for DES control based on Petri net place invariants has been briefly discussed in Subsection 4.5. A feedback controller based on place invariants is implemented by adding control places and arcs to existing transitions in the Petri net structure. Although the method was developed for ordinary Petri nets, the introduction of time delays associated with each transition will not affect the controlled behavior of the Petri net with respect to the discrete specifications. With respect to continuous dynamics, the basic idea is to follow a natural invariant's approach. The natural invariants of the system are used to partition the state space into regions. The switching policy for the hybrid system is then derived by determining the region where the continuous state lies. The basic property of these regions is that their boundaries satisfy certain conditions that preclude the state trajectories from crossing them. The resulting conditions can be embedded very efficiently in the PTPN model of the hybrid system by changing the label functions.

The underlying Petri net structure, which generates the switching policy offers two important computational advantages. First, it makes possible to efficiently design the supervisor that satisfy specifications that frequently appear in complex systems such as generalized mutual exclusion constraints. Second, it reduces considerably the search for common flow regions, since only desirable switching strategies generated by the controlled Petri net have to be examined. The set of all invariant hypersurfaces can be found by solving analytically a differential equation (the characteristic equation for the vector field of the system [75]). The task of determining suitable invariant hypersurfaces is very difficult in general. For special cases (e.g. integrator systems), the differential equation can be solved analytically. Otherwise, a computerized procedure for identifying the common flow regions using backtracking from the target region can be used. The computational complexity of the algorithm is of the order q^n, where q is the number of quantization levels and n is the number of the continuous states. There are also interesting cases where it is sufficient for the control objective to approximate the invariant hypersurfaces using Lyapunov functionals [39]. This approach is more efficient and can be applied to a larger class of systems; furthermore, the design based on Lyapunov functions exhibits desirable robustness properties. However, by assuming that the common flow regions are bounded by manifolds defined by Lyapunov functionals, we impose restrictive conditions on the dynamics of the continuous subsystems. In most of the cases, these conditions are quite restrictive but they provide a systematic way to compute common flow regions.

Programmable timed Petri nets provide a very powerful modeling formalism for hybrid systems. It is shown that certain problems in the analysis and synthesis of hybrid systems can be addressed using PTPNs and efficient algorithms are developed. Current research effort aims at identifying additional problems in hybrid systems where the use of Petri net will offer computational advantages.

6 A Parallel Computing Architecture for Intelligent Control

As it was discussed in Subsection 3.2, an important requirement for the evaluation and control of intelligent systems is the availability of efficient simulations tools. A hierarchical functional architecture was used throughout this chapter to describe a number of computational issues that arise in intelligent control. Such architecture requires the availability of simulation tools at different levels of abstraction and the means to transfer efficiently useful information between the different levels. These issues have been studied for example in [87, 12]. Integration of heterogeneous mathematical models and algorithms is necessary because of the complexity of the physical processes involved and the generality of the control objectives. The simulation of intelligent control systems may require highly diverse discrete event system simulators, optimization algorithms, on-line control reconfiguration algorithms, and task planning among others. Furthermore, because of the size of the systems of interest, simulation of intelligent control systems often requires great computational resources. It is therefore natural to consider the parallel execution of such simulations. The objective of simulations in general, is to extract useful information about the system to facilitate decision making algorithms to control the system so that it exhibits desirable behavior. The aim of parallelizing techniques is to discover a set of modules that are as independent as possible in order to minimize the communication costs among the components. The purpose of parallel simulations is to reduce the execution time of a simulation by distributing the modules of a system model among a number of simulation agents running in parallel. These agents organize the simulation of the whole model by the interchange of messages among each other.

Recent advances in parallel computing have ignited considerable research effort towards exploiting the parallelization of heterogeneous simulations (see for example [88, 33]). We present now a parallel computing architecture appropriate for modeling parts of intelligent control systems. Our purpose is to take a step towards the development of an application-driven parallel computing scheme for intelligent control applications. Motivated by a parallel run-time system for the efficient implementation of adaptive applications on distributed memory machines [14], our goal is to explore recent advances in parallel computing for intelligent control applications. The architecture of the overall run-time system and its layers are depicted in Fig. 3. It must be noted that this section describes current research for the use of high performance computing for large, irregular applications. Intelligent control applications have been identified as a challenging area where there is the need for the development of high performance computing tools. We describe now the basic characteristics of this architecture.

The first layer, named Data Movement and Control Substrate (DMCS) [14], consists of the following three modules: (i) a *threads* module, (ii) a *communications* module, and (iii) a *control* module. The threads module provides machine dependent code for creating, running, and stopping threads. It also provides an easy interface for writing and porting thread packages. The communication module is implemented on top of a generic active message implementation on the IBM RISC/6000 SP [82, 13]. The fundamental idea in active messages is that every message is sent along with a reference to a handler which is invoked on receipt of the message. DMCS provides the notion of a *global pointer* through which remote data can be accessed. A global pointer consists of a processor *id* and a pointer to the local address space. The integration of the communication module and threads takes place in the control module. The control sub-package provides support for *remote service request* and *load balancing*. A remote service request consists of a remote

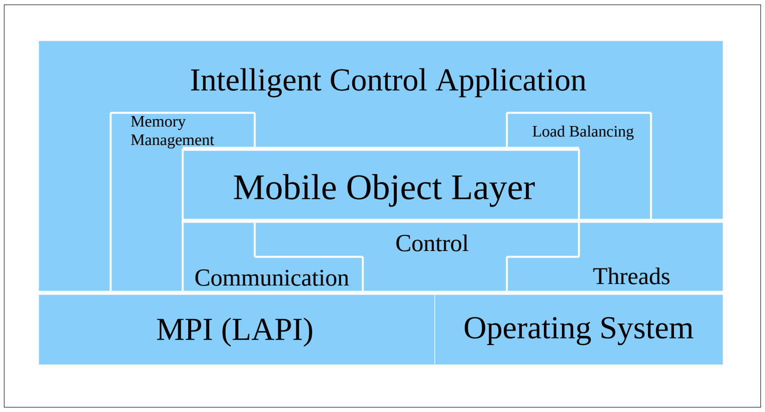

Figure 3: Parallel run-time system: architecture

context (processor), a function to be executed at the remote context and the arguments of the function. In addition, a type argument is also passed, indicating the type of the remote service request (threaded, non-threaded) and its priority (lazy, urgent). DMCS implements a simple parameterized load balancing primitive. The load on a processor is defined to be simply the number of threads on that processor. DMCS also provides a primitive that enables a processor to start a new thread on the least loaded processor within a certain window size which can be customized.

The second layer, named Mobile Object Layer (MOL) [26], provides the tools to build distributed data structures consisting of mobile objects linked with mobile pointers. For example, a directed graph might be built using one mobile object for each node. Each node holds a list of mobile pointers to other nodes. The data structure of the mobile object can be moved from processor to processor and all the mobile pointers remain valid. MOL uses a decentralized directory and updates the local directories of each processor using a *lazy* protocol to reduce the overhead of broadcasting updates. The specific implementation is built on top of the DMCS layer to handle messages sent to objects. MOL provides the mechanisms to support mobile objects and mobile pointers, but it does not specify the policies that govern the use of mobile objects. It is the responsibility of the application to decide the migration policy. MOL supports both threaded and non-threaded models of execution.

The parallel run-time system described above is an application driven scheme which is general enough and uncoupled from the specific application. Its main advantages are that it hides the bookkeeping of data structures and messages from the application developer and it provides efficient tools for remote service request and load balancing. The parallel run-time system provides to the application developer a very simple but powerful interface for building application programs or libraries.

The program complexity of intelligent control applications increases due to the computation and communication requirements that are dynamic, data-dependent, and irregular. Simulations of intelligent control applications usually consist of several algorithms (for example continuous or discrete event simulations, task planning, feedback control, optimization algorithms and so on) that are implemented using different models (discrete-event, continuous, hybrid). The formal models of the physical processes involved can be represented as mobile objects holding several data structures.

Then the algorithms can be viewed as methods that can be invoked upon the receipt of an active message by the object. The parallel architecture discussed above offers the maintenance of complex and distributed data structures and a sophisticated run-time system for low latency communication and load balancing. At the same time it hides the details from the application developer to allow fast and efficient programming. The run-time system maintains automatically the validity of global pointers as data migrates from one processor to another and implements a correct and efficient message forwarding and communication mechanism between the migrating objects. In intelligent control applications of large scale systems the workload is known only at run-time. There are many heuristic algorithms for the dynamic load balancing problem and can be incorporated easily using the proposed architecture (migration policy). In summary, the parallel computing architecture outlined above provides the primitives for utilizing powerful symmetric multiprocessors (SMPs) to solve large problems and to speed-up computations.

As an architecture of how such computing architecture can be used in intelligent control, consider a hybrid system described by a large programmable timed Petri net. The PTPN can be viewed as a data structure consisting of mobile objects (nodes) linked with mobile pointers. Each mobile object holds substructures representing the rate constraints (differential equations), generalized time constraints, and reset equations and methods that can be applied to these substructures describing ODE solvers, feedback control algorithms, or algorithms for solving optimization problems. Assume that we want to initiate the simulation of the continuous dynamics according to the label $\ell_P(p)$, that associates with the place p of the PTPN a differential equation. The place of the PTPN is considered as a mobile object which holds appropriate representations of its label functions. A message can be sent to the object using a mobile pointer. When the message reaches the object, a user-specified handler is invoked. The remote invocation call, (which can be caused for example by a change of the operation point of the system), can initiate the simulation of the continuous dynamics using appropriate ODE solvers, a different local feedback control algorithm, or the contribution of the local mode to a global optimization problem.

We believe that such a framework can be very useful to the design of intelligent control applications. The designer can focus on the application-specific problems and not on the implementation of the parallel computing protocols. On the other hand, the architecture is sufficiently open to allow the efficient use of existing codes for a variety of problems. The main characteristic is basically the use of the global pointer which can be incorporated easily to existing application programs. The reader is referred to [14, 26] for issues concerning the portability of the implementation as well as other systems using similar architecture.

6.1 Parallel Discrete Event Simulation

We have investigated the advantages of this approach for parallel discrete event simulation (PDES). It should be clear that in our view of intelligent control, PDES is an essential part in the design of intelligent control applications. Our intention is not to study new techniques for PDES, but rather to show that results that have appeared in the literature can be incorporated in the application development using the proposed parallel architecture. Discrete event simulations are very useful for the evaluation of an intelligent control system at a level of abstraction where discrete event system models or event-based control of hybrid systems are used [76, 77]. Discrete event system representations in intelligent control have been also used in [86]. A discrete event simulation model assumes that the system being simulated changes state only at discrete points in simulated time. When we choose to model a real world system using discrete event simulation, we give up the ability to capture a degree of detail that can only be described as smooth continuous change. In return, we get simplicity that allows us to capture important features of interest that are too complex to capture with continuous simulations.

Discrete event simulations have been studied in [23, 54, 21, 45] and typically require significant computational effort. A *discrete event simulation*, discretizes the observation of the simulated system at event occurrence instants. When executed sequentially, a discrete event simulation repeatedly processes the occurrence of events in simulated time, often called *virtual time*, by maintaining a time ordered *event list*, holding time-stamped events scheduled to occur in the future, and using a (global) *clock* indicating the current time and *state variables* defining the current state of the system. A *simulation engine* drives the simulation by continuously taking the first event out of the event list (i.e. the one with the lowest time-stamp), simulating the effect of the event by changing the state variables and scheduling new events in the event list. This is performed until some predefined end-time is reached, or until there are no further events to occur. The objective of parallel discrete event simulations is to accelerate the execution of simulations using P processors. The parallelism in discrete event simulations can be exploited at different levels. At the *function level*, the execution time of the simulation is reduced due to the distribution of the subroutines, constituting a simulation experiment, to the available processors. At the *component level* the simulation model is decomposed into sub-models to reflect the inherent model parallelism. Model parallelism exploitation at the next lower level, the *event level*, aims at a distribution of single events among processors for concurrent execution. The event list can be a *centralized* data structure maintained by a master processor. A higher degree of parallelism can be exploited in strategies that allow the concurrent simulation of events with different time stamps. In this scheme, each node maintains its own *decentralized* event list. Schemes following this idea require protocols for local synchronization, which in turn may cause increase communication costs.

The main idea for all simulation strategies at the event level is to partition the discrete event model into a set of communicating *logical processes*, (LPs). The objective is to exploit the parallelism inherent among the model components with the concurrent execution of the logical processes. A parallel discrete event simulation can be viewed as a collection of communicating and synchronizing simulations of submodels.

Using the proposed parallel architecture, most of the difficulties in parallel discrete event simulation can be addressed very efficiently. Consider the case when timed Petri net are used for discrete event simulation. The performance of the simulation depends on how the partition of the overall system into LPs captures the inherited parallelism of the involved processes. To achieve high performance, automated PDES must measure workload at run-time, and dynamic remap it when needed, for example dynamic remapping algorithms have been proposed in [55] to address load imbalancing. It was discussed above that the nodes of the Petri net can be viewed as mobile objects connected using mobile pointers. The initial partition of the Petri net model results in a distribution of mobile objects to different processors. Several dynamic remapping algorithms can be implemented with migration policies of the mobile objects so that the designer will not have to keep track of the location of the objects. The additional communication overhead due to the remote service requests has been measured in certain applications and it is only 7 -10 %.

7 Conclusions

In considering intelligent control of complex systems, it is necessary to address the computational complexity issue. In this work, computational aspects of intelligent control methodologies are discussed at length. In particular, computational issues in the analysis, controller synthesis, and simulation of discrete event and hybrid systems were studied. Emphasis is put on computational issues in recent approaches to discrete event and hybrid system design that were developed by our group using Petri nets.

Acknowledgments The authors would like to thank Prof. Nikos Chrisochoides of the Computer Science and Engineering Department at the University of Notre Dame for his valuable assistance concerning the parallel computing architecture. The partial financial support of the National Science Foundation (ECS95-31485) and the Army Research Office (DAAG55-98-1-0199) is gratefully acknowledged.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P-H. Ho, X. Nicollin, A. Oliveiro, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical and Computer Science*, 138:3–34, 1995.

[2] R. Alur and D. Dill. The theory of timed automata. *Theoretical and Computer Science*, 126:183–235, 1994.

[3] P.J. Antsaklis. Defining intelligent control. *IEEE Control Systems*, pages 4–5 & 58–66, June 1994. Report of the Task Force on Intelligent Control, P.J Antsaklis, Chair.

[4] P.J. Antsaklis. Intelligent control. In *Encyclopedia of Electrical and Electronics Engineering.* John Wiley & Sons, Inc., 1997.

[5] P.J. Antsaklis, M. Lemmon, and J.A. Stiver. Learning to be autonomous: Intelligent supervisory control. In M.M. Gupta and N.K. Sinha, editors, *Intelligent Control Systems: Theory and Applications*, pages 28–62. IEEE Press, 1996.

[6] P.J. Antsaklis and A. Nerode. Hybrid control systems: An introductory discussion to the special issue. In P.J. Antsaklis and A. Nerode, editors, *IEEE Trans. Automatic Control, Special Issue on Hybrid Control Systems*, volume 43, pages 457–460, April 1998.

[7] P.J. Antsaklis and K.M. Passino, editors. *An Introduction to Intelligent and Autonomous Control.* Kluwer Academic Publishers, 1993.

[8] P.J. Antsaklis and K.M. Passino. Introduction to intelligent control systems with high degrees of autonomy. In P.J. Antsaklis and K.M. Passino, editors, *An Introduction to Intelligent and Autonomous Control*, pages 1–26. Kluwer, 1993.

[9] V.D. Blondel and J.N. Tsitsiklis. Complexity of elementary hybrid systems. In *Proceedings of the European Control Conference 97*, Brussels, Belgium, July 1997.

[10] M. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):475–482, 1998.

[11] M.S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical and Computer Science*, 138:67–100, 1995.

[12] F.E. Cellier. Integrated continuous-system modeling and simulation environments. In D. Linkens, editor, *CAD for Control Systems*, pages 1–29. Marcel Dekker, 1993.

[13] C. Chang, G. Gzajkowski, C. Hawblitzell, and T. von Eicken. Low-latency communication on the IBM risc system/6000 SP. In *Proceedings of Supercomputing'96*, 1996.

[14] N. Chrisochoides, I. Kodukula, and K. Pingali. Data movement and control substrate for parallel scientific computing. volume 1199 of *Lecture Notes in Computer Science*, pages 256–268. Springer-Verlag, 1997.

[15] R. David and H. Alla. Petri nets for modeling of dynamic systems: A survey. *Automatica*, 30(2):175–202, 1998.

[16] I. Demongodin and N.T. Koussoulas. Differential Petri nets: Representing continuous systems in a discrete-event world. *Transactions on Automatic Control*, 43(4):573–579, 1998.

[17] A.A. Desrochers and P.Y. Al-Jaar. *Applications of Petri Nets in Manufacturing Systems.* IEEE Press, 1995.

[18] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28:575–591, 1991.

[19] J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Journal of Information Processing and Cybernetics*, 30(3):143–160, 1994.

[20] J. Esparza, S. Römer, and W. Volger. An improvement of McMillan's unfolding algorithm. In T. Margaria and B. Steffen, editors, *Proceeding of TACAS'96*, volume 1055 of *LNCS*, pages 87–106. Springer, 1996.

[21] A. Ferscha. Parallel and distributed simulation of discrete event systems. In *Handbook of Parallel and Distributed Computing.* McGraw-Hill, 1995.

[22] J-M. Flaus and H. Alla. Structural analysis of hybrid systems modelled by hybrid flow nets. In *Proceedings of the European Control Conference 97*, Brussels, Belgium, July 1997.

[23] R.M. Fuzimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[24] A. Giua and E. Usai. High-level hybrid Petri nets: A definition. In *Proceeding of the 35th Conference on Decision and Control*, Kobe, Japan, December 1996.

[25] M.M. Gupta and N.K. Sinha, editors. *Intelligent Control Systems: Theory and Applications.* IEEE Press, 1996.

[26] C. Hawblitzel and N. Chrisochoides. Mobile object layer: A data migration framework for active messages communication paradigm. Technical Report TR-98-7, Department of Computer Science and Engineering, University of Notre Dame, 1998.

[27] K.X. He and M.D. Lemmon. Lyapunov stability of ontinuous valued systems under the supervision of discrete event transition systems. In T.A. Henzinger and S. Sastry, editors, *HSCC 98: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1386, pages 175–189. Springer-Verlag, 1998.

[28] T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.

[29] T.A. Henzinger, P-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *Transactions on Automatic Control*, 43(4):540–554, 1998.

[30] T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. In P.Degano et al., editor, *ICALP 97: Automata, Languages, and Programming*, volume 1256 of *LNCS*, pages 582–593. Springer, 1997.

[31] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, to appear.

[32] T.A. Henzinger and V. Rusu. Reachability verification for hybrid automata. In T.A. Henzinger and S. Sastry, editors, *HSCC 98: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1386, pages 190–204. Springer-Verlag, 1998.

[33] P.T. Homer and R.D. Schlichting. A software platform for constructing scientific applications from heterogeneous resources. *Journal of Parallel and Distributed Computing*, 21(3):301–315, 1994.

[34] J. E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[35] L. Hou, A.N.Michel, and H.Ye. Stability analysis of switched systems. In *Proceedings of the 35th Conference on Decision and Control*, Kobe, Japan, December 1996.

[36] M. Johansson and A. Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):555–559, 1998.

[37] A. Kondratyev, M. Kishinevsky, A. Taubin, and S. Ten. A structural approach for the analysis of petri nets by reduced unfoldings. In *Applications and Theory of Petri Nets 1996*, volume 1091 of *LNCS*, pages 346–365. Springer, 1996.

[38] X.D. Koutsoukos and P.J. Antsaklis. Hybrid control systems using timed Petri nets: Supervisory control design based on invariant properties. In P. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, and S. Shastry, editors, *Hybrid Systems V*, volume 1567 of *Lecture Notes in Computer Science*, pages 142–162. Springer, 1999.

[39] X.D. Koutsoukos, K.X. He, M.D. Lemmon, and P.J. Antsaklis. Timed Petri nets in hybrid systems: Stability and supervisory control. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 8(2):137–173, 1998.

[40] J. Le Bail, H. Alla, and R. David. Hybrid Petri nets. In *Proceedings 1st European Control Conference*, Grenoble, France, July 1991.

[41] M.D. Lemmon, K.X. He, and C.J. Bett. Modeling hybrid control systems using programmable Petri nets. In *3rd International Conference ADMP'98, Automation of Mixed Processes: Dynamic Hybrid Systems*, pages 177–184, Reims, France, March 1998.

[42] Y. Li and W.M. Wonham. Control of vector discrete-event systems I - the base model. *IEEE Transactions on Automatic Control*, 38(8):1214–1227, 1993.

[43] Y. Li and W.M. Wonham. Control of vector discrete-event systems II - controller synthesis. *IEEE Transactions on Automatic Control*, 39(3):512–531, 1994.

[44] Y. Li and W.M. Wonham. Concurrent vector discrete-event systems. *IEEE Transactions on Automatic Control*, 40(4):628–638, 1995.

[45] Y-B. Lin and P.A. Fischwick. Asynchronous parallel discrete event simulation. *IEEE Transactions on Systems, Man and Cybernetics*, 26(4):249–286, 1996.

[46] J. Lygeros, D.N. Godbole, and S. Sastry. Multiagent hybrid system design using game theory and optimal control. In *Proceedings of the 35th IEEE Conference on Decision and Control*, pages 1190–1195, Kobe, Japan, December 1996.

[47] K.L. McMillan. A technique of a state space search based on unfoldings. *Formal Methods in System Design*, 6(1):45–65, 1995.

[48] S. Melzer and J. Esparza. Checking system properties via integer programming. In H.R. Nielson, editor, *Proceeding of ESOP'96*, volume 1058 of *LNCS*, pages 250–265. Springer, 1996.

[49] G. Memmi and G. Roucairol. Linear algebra in net theory. In *Net Theory and Applications*, volume 84 of *LNCS*, pages 213–223. Springer-Verlag, 1980.

[50] J.O. Moody. *Petri Net Supervisors for Discrete Event Systems*. PhD thesis, Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN, 1997.

[51] J.O. Moody and P.J. Antsaklis. *Supervisory Control of Discrete Event Systems using Petri Nets*. Kluwer Academic Publishers, 1998.

[52] J.O. Moody and P.J. Antsaklis. Petri net supervisors for DES with uncontrollable and unobservable transitions. *IEEE Transactions on Automatic Control*, 1999. To appear.

[53] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77(4):541–580, 1989.

[54] D.M. Nicol and R.M¿ Fuzimoto. Parallel simulation today. *Annals of Operations Research*, 53:249–286, 1994.

[55] D.M. Nicol and S Roy. Parallel simulation of timed Petri nets. In *Proceedings of the 1991 Winter Simulation Conference*, pages 574–583, Phoenix, Arizona, 1991.

[56] M. Nielsen, G. Plotkin, and G. Winskel. Event structures and domains. *Theoretical Computer Science*, 13(1):85–108, 1980.

[57] C.M. Özveren, A.S. Willsky, and P.J. Antsaklis. Stability and stabilizability of discrete event dynamic systems. *Journal of the ACM*, 38(3):730–752, 1991.

[58] C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[59] C.H. Papadimitriou and J.N. Tsitsiklis. Intractable problems in control theory. *SIAM J. of Control and Optimization*, 24(4):639–654, 1986.

[60] K.M. Passino. Toward bridging the perceived gap between conventional and intelligent control. In M.M. Gupta and N.K. Sinha, editors, *Intelligent Control Systems: Theory and Applications*, pages 3–27. IEEE Press, 1996.

[61] K.M. Passino and P.J. Antsaklis. Event rates and aggregation in hierarchical discrete event systems. *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 1(3):271–287, 1992.

[62] K.M. Passino and P.J. Antsaklis. Modeling and analysis of artificially planning systems. In P.J. Antsaklis and K.M. Passino, editors, *An Introduction to Intelligent and Autonomous Control*, pages 191–214. Kluwer, 1993.

[63] K.M. Passino and A.D. Lunardhi. Qualitative analysis of expert control systems. In M.M. Gupta and N.K. Sinha, editors, *Intelligent Control Systems: Theory and Applications*, pages 404–442. IEEE Press, 1996.

[64] R.V. Patel, A.J. Laub, and P.M. Van Dooren, editors. *Numerical Linear Algebra Techniques for Systems and Control.* IEEE Press, 1994. Selected Reprint Series.

[65] P. Peleties and R. DeCarlo. Asymptotic stability of m-switched systems using Lyapunov-like functions. In *Proceedings of the American Control Conference*, pages 1679–1684, 1991.

[66] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[67] S. Pettersson and B. Lennartson. Stability and robustness of hybrid systems. In *Proceedings of the 35th Conference on Decision and Control*, pages 1202–1207, Kobe, Japan, December 1996.

[68] A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Shastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 359–369. Springer, 1995.

[69] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–89, January 1989.

[70] W. Reisig. *Petri Nets.* Springer-Verlag, 1985.

[71] K. Rudie and J.C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Transactions on Automatic Control*, 40(7):1313–1319, 1995.

[72] G.N. Saridis. Architecture for intelligent controls. In M.M. Gupta and N.K. Sinha, editors, *Intelligent Control Systems: Theory and Applications*, pages 127–148. IEEE Press, 1996.

[73] G.N. Saridis and K.P. Valavanis. Analytical design of intelligent machines. *Automatica*, 24(2):123–133, 1988.

[74] E.D. Sontag. Interconnected automata and linear systems: A theoretical framework in discrete-time. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III, Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 436–448. Springer, 1996.

[75] J.A. Stiver, P.J. Antsaklis, and M.D. Lemmon. Interface and controller design for hybrid control systems. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 462–492. Springer, 1995.

[76] J.A. Stiver, P.J. Antsaklis, and M.D. Lemmon. An invariant based approach to the design of hybrid control systems. In *IFAC 13th Triennial World Congress*, volume J, pages 467–472, San Francisco, CA, 1996.

[77] J.A. Stiver, P.J. Antsaklis, and M.D. Lemmon. A logical DES approach to the design of hybrid control systems. *Mathl. Comput. Modelling*, 23(11/12):55–76, 1996.

[78] M. Tittus and B. Egardt. Control design for integrator hybrid system. *IEEE Transactions on Automatic Control*, 43(4):491–500, 1998.

[79] C. Tomlin, J. Lygeros, and S. Sastry. Synthesizing controllers for nonlinear hybrid systems. In T.A. Henzinger and S. Sastry, editors, *HSCC 98: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1386, pages 360–373. Springer-Verlag, 1998.

[80] J.N. Tsitsiklis. On the control of discrete-event dynamical systems. *Mathematics of Control, Signals and Systems*, 2(2):95–107, 1989.

[81] K.P. Valavanis and G.N. Saridis, editors. *Intelligent Robotic Systems: Theory, Design and Applications.* Kluwer Academic Publishers, 1992.

[82] T. von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser. Active messages: A mechanism for integrated communication and computation. In *Proceedings of the 19th International Symposium on Computer Architecture.* ACM Press, 1992.

[83] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *Proceedings of the IEEE Conference on Decision and Control*, pages 4607–4612, San Diego, CA, December 1997.

[84] K. Yamalidou and J.C. Kantor. Modeling and optimal control of discrete-event chemical processes using Petri nets. *Computers in Chemical Engineering*, 15(7):503–519, 1991.

[85] K. Yamalidou, J. Moody, M. Lemmon, and P. Antsaklis. Feedback control of Petri nets based on place invariants. *Automatica*, 32(1):15–28, 1996.

[86] B.P. Zeigler. DEVS representation of dynamical systems: Event-based intelligent control. *Proceedings of IEEE*, 77(1):72–80, 1989.

[87] B.P. Zeigler, S.D. Chi, and F.E. Cellier. Model-based architecture for high autonomy systems. In S.G. Tzafestas, editor, *Engineering Systems with Intelligence - Concepts, Tools and Applications*, pages 3–22. Kluwer Academic Publishers, 1991.

[88] B.P. Zeigler and G. Zhang. Mapping hierarchical discete event models to multiprocessor systems: Concepts, algorithms, and simulation. *Journal of Parallel and Distributed Computing*, 9:271–281, 1990.