A Discretization Method from Coloured to Symmetric Nets: Application to an Industrial Example

Fabien Bonnefoi[1], Christine Choppy[2], and Fabrice Kordon[3]

[1] DSO/DSETI, Cofiroute, 92310 Sèvres, France
Fabien.Bonnefoi@cofiroute.fr,
[2] LIPN - CNRS UMR 7030, Université Paris XIII, 93430 Villetaneuse, France
Christine.Choppy@lipn.univ-paris13.fr
[3] LIP6 - CNRS UMR 7606, Université P. & M. Curie, 75252 Paris, France
Fabrice.Kordon@lip6.fr

Abstract. Intelligent Transport Systems (ITS) tend to be more distributed and embedded. In many cases, continuous physical parameters are part of the systems description. Analysis techniques based on discrete models must integrate such constraints. In this paper, we propose a methodological way to handle such hybrid systems with model checking on Petri Nets and algebraic methods. Our methodology is based on transformations from Coloured Petri Nets (CPN) for their expressiveness to Symmetric Petri Nets (SN) to take advantage of their efficient verification techniques. Our methodology also addresses the impact of discretization on the precision of verification. In scientific computing, the discretization process introduces "error intervals" that could turn a given verified property into a false positive one. In that case, the property to be verified might have to be transformed to cope with such precision errors.

Key words: Discretization, Symmetric Petri Nets, Coloured Petri Nets, Intelligent Transport Systems, Hybrid Systems.

1 Introduction

Future supervision systems tend to be more distributed and embedded. Parallelism brings a huge complexity and then, a strong need to deduce good and bad behaviours on the global system, from the known behaviour of its actors. This is crucial since safety critical missions can be supervised by such systems. Intelligent Transport Systems (ITS) are a typical example: many functions tend to be integrated in vehicles and road infrastructure. Moreover, in many cases physical constraints are part of such systems. Analysis techniques based on discrete models must integrate such constraints: we then speak of *hybrid* systems.

A major trend in formal analysis is to cope with such systems. This raises many issues in terms of analysis complexity. Some techniques are dedicated to

continuous analysis such as algebraic approaches like B [1]. However, such approaches are difficult to set up and most industries prefer push-button tools. Model checking easily offers such push-button tools but does not cope well with continuous systems. Most model checking techniques deal with discrete and finite systems. Thus, management of hybrid systems is not easy or leads to potentially infinite systems that are difficult to verify. For example, management of continuous time requires much care, even to only have decidable models. Hybrid Petri Nets [19] might be a solution to model and analyze hybrid systems but no tool is available to test neither safety nor temporal logic properties [40].

In this paper, we propose a methodology to handle hybrid systems with model checking on Petri Nets and algebraic methods. Our methodology is based on transformations from Coloured Petri Nets (CPN) [30] to Symmetric Petri Nets[4] (SN) [11, 12]. CPN expressiveness allows an easy modelling of the system to be analyzed. SN are of interest for their analysis because of the symbolic state space that is efficient to represent the state space of large systems. Since SN offer a limited set of operations on colours, transformation from CPN requires much care from the designer as regards the types to be discretized.

Our methodology also addresses an important question: what is the impact of discretization on the precision of verification? As in scientific computing, the discretization process introduces "precision errors" that could turn a given verified property into a false positive one. In that case, the property to be verified might have to be transformed to take into consideration such precision errors.

Sect. 2 briefly recalls the notions of CPN and SN, as well as abstraction/refinement, type issues. Our methodology which involves modelling, discretization and verification is presented in Sect. 3, and we show in Sect. 4 how we model our Emergency Braking application. The various discretization concepts on our case study are detailed in Sect. 5, and our experiments on net analyses are presented in Sect. 6. Some perspectives on discretization are also discussed in Sect. 7 before a conclusion.

2 Building Blocks

This section presents the building blocks from the state of the art used in the discretization method.

2.1 Coloured Petri Nets

Coloured Petri nets [30] are high level Petri nets where tokens in a place carry data (or colours) of a given type. Since several tokens may carry the same value, the concept of multiset (or bag) is used to describe the marking of places.

In this paper, we assume the reader is familiar with the concept of multisets. We thus recall briefly the formal definition of coloured Petri nets as in [30]. It

[4] *Symmetric nets* were formerly known as *Well-Formed nets*, a subclass of *High-level Petri nets*. The new name appeared during the ISO standardisation of Petri nets [27].

should be noted however that the types considered for the place tokens may be basic types (e.g. boolean, integers, reals, strings, enumerated types) or structured types – also called compound colour sets – (e.g. lists, product, union, etc.). In both cases, the type definition includes the appropriate (or usual) functions.

Different languages were proposed to support the type definition for coloured Petri nets (e.g. algebraic specification languages as first introduced in [43], object oriented languages [8]), and an extension of the Standard ML language was chosen for CPN Tools [17]. As always, there may be a tradeoff between the expressivity of a specification language, and efficiency when tools are used to compute executions, state graphs, etc. If expressivity is favored, it could be desirable to allow any appropriate type and function, while when tools should be used to check the behaviour and the properties of the system studied, the allowed types and functions are restricted (as the language allowed for CPN Tools or as in Symmetric Nets presented in Sect. 2.2). Here, to start with we want to allow a specification language that fits as much as possible what is needed to describe the problem under study, and then to show how the specification is transformed so as to allow computations and checks by tools.

In the following, we refer to $EXPR$ as the set of expressions provided by the net inscription language (net inscriptions are arcs expressions, guards, colour sets and initial markings), and to $EXPR_V$ as the set of expressions $e \in EXPR$ such that $Var[e] \subseteq V$ ($Var[e]$ denotes the set of variables occurring in e).

Definition 21 *A non-hierarchical coloured Petri net CPN [30] is a tuple $CPN = (P, T, A, \Sigma, V, C, G, E, I)$ such that:*

1. *P is a finite set of places.*
2. *T is a finite set of transitions such that $P \cap T = \emptyset$.*
3. *$A \subseteq P \times T \cup T \times P$ is a set of directed arcs.*
4. *Σ is a finite set of non empty colour sets (types).*
5. *V is a finite set of typed variables such that $\forall v \in V, Type[v] \in \Sigma$.*
6. *$C : P \rightarrow \Sigma$ is a colour set function assigning a colour set (or a type) to each place.*
7. *$G : T \rightarrow EXPR_V$ is a guard function assigning a guard to each transition such that $Type(G(t)) = Bool$, and $Var[G(t)] \subseteq V$, where $Var[G(t)]$ is the set of variables of $G(t)$.*
8. *$E : A \rightarrow EXPR_V$ is an arc expression function assigning an arc expression to each arc such that $Type(E(a)) = C(p)_{MS}$, where p is the place connected to the arc a.*
9. *$I : A \rightarrow EXPR_V$ is an initialisation function assigning an initial marking to each place such that $Type(I(p)) = C(p)_{MS}$.*

As explained in Sect. 3, the first step of our methodology is to produce a CPN model for the application. The next step is a transformation motivated by the discretization of continuous functions to obtain a symmetric net.

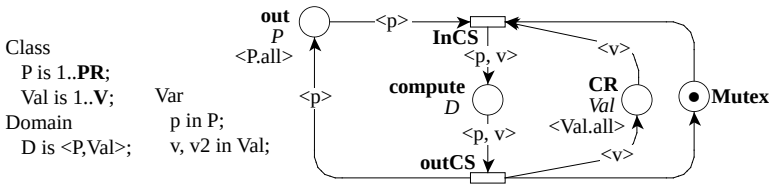

Fig. 1. Example of Symmetric Net

2.2 Symmetric Nets

Symmetric nets were introduced in [11, 12], with the goal of exploiting symmetries in distributed systems to get a more compact representation of the state space.

The concept of symmetric nets is similar to the coloured Petri net one. However, the allowed types for the places as well as allowed colour functions are more restricted. These restrictions allow us to compute symmetries and obtain very compact representations of the state space, enabling the analysis of complex systems as in [28].

Basically, types must be finite enumerations and can only be combined by means of cartesian products. Allowed functions in arc expressions are: identity, successor, predecessor and broadcast (that generates one copy of any value in the type). These constraints affect points 4, 6, 7, 8, 9 in Definition 21.

The Symmetric net in Fig. 1 models a class of threads (identified by type P) accessing a critical resource **CR**. Threads can get a value within the type *Val* from **CR**. Constants **PR** and **V** are integer parameters for the system. The class of threads is represented by places **out** and **compute**.

Place **compute** corresponds to some computation on the basis of the value provided by **CR**. At this stage, each thread holds a value that is replaced when the computation is finished. Place **Mutex** handles mutual exclusion between threads and contains token with no data ("black tokens" in the sense of the Petri Net standard [29]). Place **out** initially holds one token for each value in P (this is denoted <P.all>) and place **CR** holds one value for each value in type *Val*.

The main interest of SN is the possibility to generate a *symbolic state space*. A symbolic state (in the symbolic state space) represents a set of concrete states

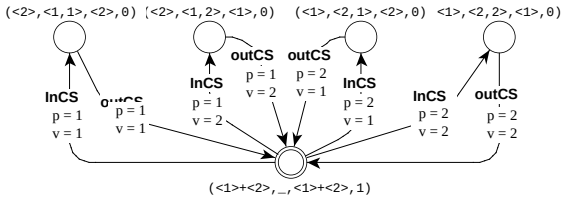

Fig. 2. State space of the model in Fig. 1

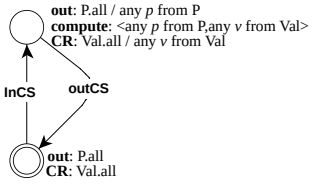

Fig. 3. Symbolic state space

with a similar structure. This is well illustrated in Fig. 3 where the upper symbolic state represents the four upper states in Fig. 2.

In Fig. 3, the bottom state corresponds to the initial marking where **out** contains one token per value in type P and **CR** one token per value in type Val. The top state represents a set of states with permutations on binding variables p and v. There, place **compute** holds one token only while places **out** and **CR** hold one token per value in the place type minus the value used to build the token in place **compute**.

Let us note that in some case (like here), the symbolic[5] state space does not change with types P and Val or initial markings, which yields a very compact representation of the system behaviour. The symbolic state space may even be exponentially smaller than the explicit state space.

Verification of properties can be achieved either by a structural analysis, on the symbolic state space (model checking), or on the unfolded associated Place/Transition (P/T) net (essentially to compute structural properties).

2.3 Transformation, Abstraction and Refinement

Abstraction and refinement are part of the use of formal specifications. While abstraction is crucial to concentrate on essential aspects of the problem to be solved (or the system to be built), and to reason about them, more elaborate details need to be further introduced in the refinement steps. A similar evolution is taking place when a general pattern or template is established to describe the common structure of a family of problems, and when this template is instantiated to describe a single given problem.

Three kinds of refinement for coloured Petri nets are introduced in [34, 35], the type refinement, the node refinement and the subnet refinement. These refinements are correct if behaviours are preserved and if, to any behaviour of a refined net it is possible to match a behaviour of the abstract net.

Another motivation is raised by the use of tools to check the behaviour and properties of the model, since it may involve the discretization of some domains so as to reduce the number of possible values to consider in the state space. It thus involves a simplification of some domains that may be considered as an abstraction.

3 Methodology for Discretization

This section presents our methodology to model and analyse an hybrid system. We give an overview and then detail its main steps and the involved techniques.

[5] The word *symbolic* has several meanings in model checking. Here, it refers to the symbolic state space, which is a set-based representation of the state space. It is also used in *symbolic model checking* to indicate the encoding of explicit states by means of decision diagrams (such as BDD). This term is also used later in the paper.

3.1 Overview of the Methodology

Fig. 4 sketches our methodology. It takes as input a set of requirements. It is thus divided in two parts:

- the *specification* describes the system (we only consider in this work the behavioural aspects),
- the *required properties* establish a set of assertions to be verified by the system.

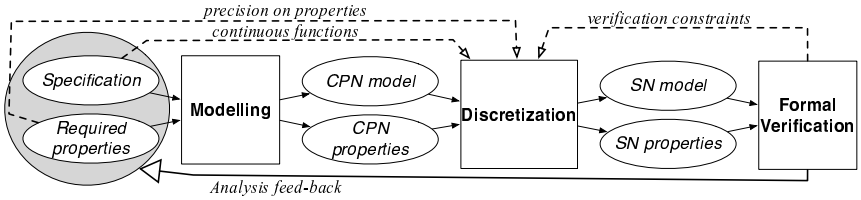

Fig. 4. Overview of our methodology

Once the specification written using "classical" techniques, the system is modelled using high-level Petri Nets (CPN) that allow one to insert complex colour functions such as ones involving real numbers. These functions come from the specifications of the system (in Intelligent Transport Systems, numerous behaviours are described by means of equations describing physical models). These functions are inserted in arc labels into the CPN-model produced by the **Modelling** step. Required properties are also set in terms of CPN.

However, the CPN system cannot be analysed in practice since the system is too complex (due to the data and functions involved). So, the **Discretization** step is dedicated to the generation of an associated system expressed using Symmetric Nets. Symmetric Nets are well suited to specify such systems that are intrinsically symmetric [5]. Operations such as structural analysis or model checking can be achieved for much larger systems. Formal analysis of the system is performed at the **Formal Verification** step.

Let us note that a similar transformation is achieved in [3] instead from coloured Petri nets to counter systems. The goal in this work is similar, being able to analyze a CPN specification, but not in the domain of hybrid systems.

The following sections present the three main steps of our methodology and especially focus on the **Discretization** step that is the most delicate one as well as the main contribution of this paper.

3.2 Modelling

There are heterogeneous elements to consider in Intelligent Transport Systems (ITS): computerized actors (such as cars or controllers in a motorway infrastruc-

ture) have to deal with physical variables such as braking distances, speed and weight.

If those continuous variables were not modelled, only a subpart of the "required properties" of the system could be checked. Especially, it is not possible to verify properties related to quantitative variables.

Our work aims at providing a more precise representation of the system in the Petri net models by representing those quantitative variables. To design the CPN model we used a template adapted to the case study presented in Sect. 4. The "interfaces" of the Petri net model, presented in Sect. 5, were already identified. The main task was to identify control and data flows that are involved in this subpart of the system, and that must be modelled to allow formal verification. Also, operations made on those flows were identified.

Then, the different selected variables of the system were represented using equivalent types in CPN. For example, continuous variables of the system were modelled with the real type of CPN formalism. The functions of the system that manipulate the continuous variables were represented using arc expressions.

3.3 Discretization

The discretization step takes CPN with their properties as inputs, and produces SN with their properties as outputs. To achieve this goal, a discretization of the real data and functions involved is performed. As a result, the types involved in the CPN are abstracted, and the real functions are represented by a place providing tuples of appropriate result values.

We propose different steps to manage the discretization of continuous functions in Symmetric Nets:
- Step 1: Continuous feature discretization
- Step 2: Error propagation computing
- Step 3: Type transformation and modelling of complex functions in Symmetric Nets.

Step 1 - Continuous feature discretization Discretization is the process of transforming continuous models and equations into discrete counterparts. Depending on the domain to which this process is applied we use also the words "digitizing", "digitization", "sampling", "quantization" or "encoding". Techniques for discretization differ according to application domains and objectives.

Let us introduce the following definitions to avoid ambiguity in this paper:

Definition 31 *A **region** is a n-dimensional polygon (i.e. a polytope) made by adjacent points of an n-dimensional discretized function.*

Definition 32 *A **mesh** is a set of regions used to represent a n-dimensional discretized function for modeling or analysis.*

There exist many discretization methods that can be classified between global or local, supervised or unsupervised, and static or dynamic methods [20].

– **Local methods** produce partitions that are applied to localized regions of the instance space. Those methods usually use decision trees to produce the partitions (i.e. the classification).
– **Global methods** (like binning) [20] produce a mesh over the entire n-dimensional continuous instance space, where each feature is partitioned into regions. The mesh contain $\prod_{i=1}^{n} k_i$ regions, where k_i is the number of partitions of the ith feature.

In our study we consider the **equal width interval binning method** [20] as a first approach to discretize the continuous features. Equal width interval binning is a global unsupervised method that involves dividing the range of observed values for the variable into k equally sized intervals, where k is a parameter provided by the user. If a variable x is bounded by x_{min} and x_{max}, the interval width is:

$$\Delta = \frac{x_{max} - x_{min}}{k} \tag{1}$$

Step 2 - Error propagation computing To model a continuous function in Symmetric Nets it is necessary to convert it into an equivalent discrete function. This operation introduces inaccuracy (or error) which must be taken into account during the formal verification of the model. This inaccuracy can be taken into account in the Symmetric Net properties in order to keep them in accordance with the original system required properties. The other solution is to change the original required properties taking into account the introduced inaccuracy.

The issues are well expressed in [10]: *"in science, the terms uncertainties or errors do not refer to mistakes or blunders. Rather, they refer to those uncertainties that are inherent in all measurements and can never be completely eliminated.(...) A large part of a scientist's effort is devoted to understanding these uncertainties (error analysis) so that appropriate conclusions can be drawn from variable observations. A common complaint of students is that the error analysis is more tedious than the calculation of the numbers they are trying to measure. This is generally true. However, measurements can be quite meaningless without knowledge of their associated errors."*

There are different methods to compute the error propagation in a function [36, 10]. The most current one is to determine the separate contribution due to errors on input variables and to combine the individual contributions in a quadrature.

$$\Delta_{f(x,y,..)} = \sqrt{\Delta_{fx}^2 + \Delta_{fy}^2 + ...} \tag{2}$$

Then, different methods to compute the contribution of input variables to the error in the function are possible, like the "derivative method" or the "computational method".

– The derivative method evaluates the contribution of a variable x to the error on a function f as the product of the error on x (i.e. Δ_x) with the partial derivative of $f(x, y, ..)$:

$$\Delta_{fx} = \frac{\partial f(x, y, ..)}{\partial x} \Delta_x \tag{3}$$

– The computational method computes the variation by a finite difference:

$$\Delta_{fx} = \mid f(x + \Delta_x, y, ..) - f(x, y, ..) \mid \tag{4}$$

The use of individual contributions in a quadrature relies on the assumption that the variables are independent and that they have a Gaussian distribution for their mean values. This method is interesting as it gives a good evaluation of the error. But we do not have a probabilistic approach, and we do not have a Gaussian distribution of the "measured" values.

In this paper, we prefer to compute the maximum error bounds on f due to the errors on variables as it gives an exact evaluation of the error propagation. Let $f(x)$ be a continuous function, x be the continuous variable, and x_{disc} the discrete value of x. If we choose a discretization step of $2 \cdot \Delta_x$ we can say that for each x_{disc} image of x by the discretization process, $x \in [x_{disc} - \Delta_x, x_{disc} + \Delta_x]$ (which is usually simplified by the expression $x = x_{disc} \pm \Delta_x$). We can compute the error $\Delta_{f(x)}$ introduced by the discretization:

$$f(x) = f(x_{disc}) \pm \Delta_{f(x)} \qquad \Delta_{f(x)} = f(x \pm \Delta_x) - f(x) \tag{5}$$

We can also say that the error on $f(x)$ is inside the interval :

$$\Delta_{f(x)} \in [Min(f(x \pm \Delta_x) - f(x)), Max(f(x \pm \Delta_x) - f(x))] \tag{6}$$

This method can also be applied with functions of multiple variables. In this case, for a function f of n variables $f(x \pm \Delta_x, y \pm \Delta_y, ..)$ has 2^n solutions. The maximum error bounds on f are:

$$\Delta_f \in [Min(f(x \pm \Delta_x, y \pm \Delta_y, ..) - f(x, y, ..)), Max(f(x \pm \Delta_x, y \pm \Delta_y, ..) - f(x, y, ..))] \tag{7}$$

An example of this method applied to an emergency braking function is presented in Sect. 5.2.

Step 3a - Type transformation Once the best discretization actions are decided with regards to our goals, the CPN model may be transformed into a symmetric net.

Let us first note that some types do not need to be transformed because they are simple enough (e.g. enumerated types) and do not affect the state space complexity.

When the types are more complex, two kinds of transformation are involved in this process, that concern the value set (also called carrier set), and the complex functions. The value set transformation results from the discretization of all infinite domains into an enumerated domain.

A node refinement is applied to transitions that involve a complex function on an output arc expression. As explained below and in Fig. 5, there are two possibilities to handle this. In our method, such functions are represented by tuples of discrete values (values of the function arguments and of the result) that are stored in a **values** place. The **values** place is both input and output of the refined transition, thus for any input data provided by the original input arc(s), the **values** place yields the appropriate tuple with the function result.

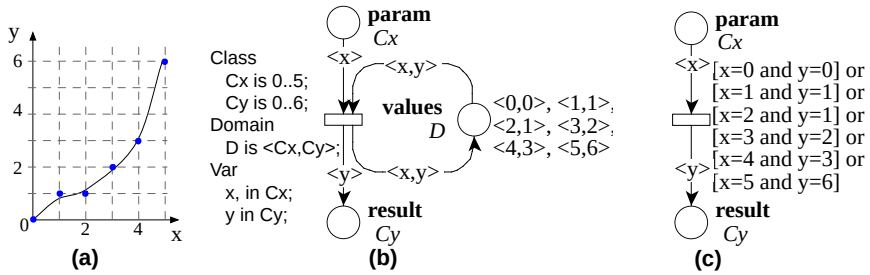

Fig. 5. Example of function discretization using a place or a transition guard

Step 3b - Modelling of complex functions in Symmetric Nets To cope with the modelling of complex functions in Symmetric Nets (for example, the computation of braking distance according to the current speed of a vehicle), we must discretize and represent them either in a specific place or as a guard of a transition. When a place is used, it can be held in an SN-module ; it then represents the function and can be stored in a dedicated library.

Fig. 5 represents an example of function discretization. The left side (a) shows a function that is discretized, and the right side shows the corresponding Petri net models : in model (b), the function is discretized by means of a place, in model (c), it is discretized by mean of a transition guard. In both cases, correct associations between x and y are the only ones to be selected when the transition fires. Note that in model (b) **values** markings remain constant.

This technique can be generalized to any function $x = f(x_1, x_2, ..., x_n)$, regardless of its complexity. Non deterministic functions can also be specified in the same way (for example, to model errors in the system). Let us note that:

- the discretization of any function becomes a modelling hypothesis and must be validated separately (to evaluate the impact of imprecision due to discretization),
- given a function, it is easy to automatically generate the list of values to be stored in the initial marking of the place representing the function, or to be put in the guard of the corresponding transition.

The only drawback of this technique is a loss in precision compared to continuous systems that require appropriate hybrid techniques [14]. Thus, the choice of a discretization schema must be evaluated, for example to ensure that uncertainty remains in a safe range.

It is also possible to model functions by using inequations in the guard of Fig. 5(c). However, comparison between free variables break the model symmetries if there are any. This is why, in model of Fig. 5(c) the guard only use comparisons between a free variable and a constant.

3.4 Verification

Our models are analysed using:

– *Structural techniques* (invariant computation, structural bounds, etc) on P/T nets. Since our nets are coloured, an unfolding tool able to cope with large systems [33] is used to derive the corresponding P/T net to compute structural properties.
– *Model checking*, there exist efficient model checking techniques that are dedicated to this kind of systems and make intensive use of symmetries as well as of decision diagrams. Such techniques revealed to be very efficient for this kind of systems by exploiting their regularity [28, 5].

However, due to the complexity of such systems, discretization is a very important point. If symmetric net coloured classes are too large (i.e. the discretization interval is too small), we face a combinatorial explosion (for both model checking or structural analysis by unfolding). On the other hand, if the error introduced by the discretization is too high, the property loses its "precision" and the verification of properties may lose its significance.

This is why in Fig. 4, the discretization step needs *verification constraints* as inputs from the verification step. A compromise between combinatorial explosion and precision in the model must be found.

4 Modelling the Emergency Braking Problem

The case study presented in this paper is a subpart of an application from the "Intelligent Road Transport System" domain. It is inspired from the European project SAFESPOT [7]. This application is called "Hazard and Incident Warning" (H&IW), and its objective is to warn the driver when an obstacle is located on the road. Different levels of warning are considered, depending on the criticality of the situation. This section presents the "Emergency Braking module" of the application and how it can be specified using the CPN formalism.

4.1 Presentation of the Case Study

SAFESPOT is an Integrated Project funded by the European Commission, under the strategic objective "Safety Cooperative Systems for Road Transport". The Goal of SAFESPOT is to understand how "intelligent" vehicles and "intelligent" roads can cooperate to produce a breakthrough in road safety. By combining data from vehicle-side and road-side sensors, the SAFESPOT project will allow to extend the time in which an accident is foreseen. The transmission of warnings and advices to approaching vehicles (by means of vehicle-to-vehicle and vehicle-to-infrastructure communications [39, 38, 18]), will extend in space and time the driver's awareness of the surrounding environment.

Functional Architecture The SAFESPOT applications [4] rely on a complex functional architecture. If the sensors and warning devices differ between SAFESPOT vehicles and SAFESPOT infrastructure, the functional architecture is designed to be almost the same for these two main entities of the system providing a peer-to-peer network architecture. It enables real-time exchange of

vehicles' status and of all detected events or environmental conditions from the road. This is necessary to take advantage of the cooperative approach and thus enable the design of effective safety applications.

As presented in Fig. 6, information measured by sensors is provided to the "Data Processing / Fusion" module or transmitted through the network to the "Data Processing / Fusion" module of other entities. This module analyses and processes arriving data to put them on the "Local Dynamic Map" (LDM) of the system. The "Local Dynamic Map" enables the cooperative applications of the system to retrieve relevant variables and parameters depending on their purpose. The applications are then able to trigger relevant warnings to be transmitted to appropriate entities and displayed via an onboard Human Machine Interface (HMI) or road side Variable Message Signs (VMS). In SAFESPOT, five main infrastructure-based applications were defined: "Speed Alert", "Hazard and Incident Warning", "Road Departure Prevention", "Co-operative Intersection Collision Prevention" and "Safety Margin for Assistance and Emergency Vehicles". These applications are designed to provide the most efficient recommendations to the driver.

Hazard and incident application The aim of the "Hazard and Incident Warning" application is to warn the drivers in case of dangerous events on the road. Selected events are: accident, presence of unexpected obstacles on the road, traffic jam ahead, presence of pedestrians, presence of animals and presence of a vehicle driving in the wrong direction or dangerously overtaking. This application also analyses all environmental conditions that may influence the road friction or decrease the drivers' visibility. Based on the cooperation of vehicles and road side sensors, the "Hazard and Incident Warning" application provides warnings to the drivers and feeds the SAFESPOT road side systems and vehicles with information on new driving situations. This application is essential to provide other applications with the latest relevant road description.

The emergency braking module The emergency braking module is one subsystem in the "Hazard and Incident Warning" distributed application. It com-

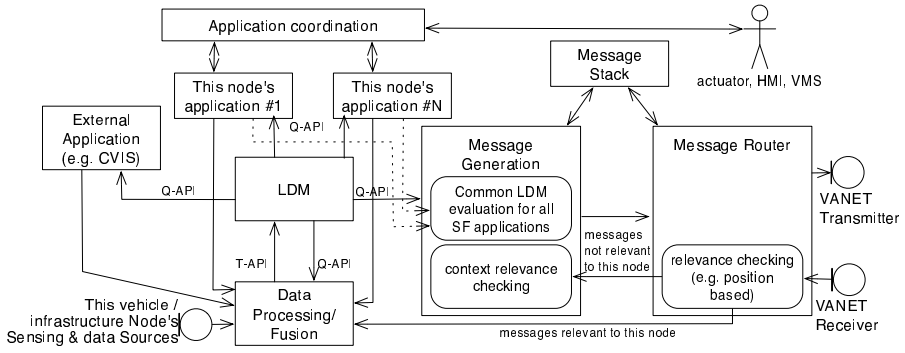

Fig. 6. SAFESPOT High Level Architecture

municates with other subsystems. The behavior of this subsystem is significant in the SAFESPOT system and must be analyzed.

In the case of an obstacle on the road, the emergency braking module receives/retrieves the speed, deceleration capability and the relative distance to a static obstacle for the monitored vehicle. With these data, it will compute a safety command to be transmitted to the driver and to other applications of the system. Those commands represent the computed safety status of a vehicle. The three commands (or warnings) issued by this module are "Comfort" if no action is required from the driver, "Safety" if the driver is supposed to start decelerating, and "Emergency" if the driver must quickly start an emergency braking. This is illustrated in Fig. 7. Note that if a driver in an "Emergency" status does not brake within one second, an automated braking should be triggered by the "Prevent" system (which is another European project).

Petri nets are well suited to describe and analyse this type of application. However, a part of the "Hazard and Incident Warning" application algorithm is based on the analysis of continuous variables like vehicle speed or position of an obstacle. Those data are part of the data flow of the system ; they are also determinant for the control flow of the system.

Many properties can be verified using Petri nets without modelling continuous variables. However, some properties require continuous variables to be modelled, like those presented Sect. 4.3. Then we face a combinatorial explosion and have to enhance the Petri net formalism as well as the modelling methodology to enable the verification of such systems.

4.2 Mathematical Model of the Emergency Braking Module

The *emergency braking module* implements a strategy function to determine the safety status of a given vehicle. This function computes the *braking distance* of a vehicle from its speed and deceleration capabilities.

Let $v \in V$ be the velocity (speed) of a vehicle with $V \subset \mathbb{R}^+$. Let also $b \in B$ be the braking capability of the vehicle with $B \subset \mathbb{R}^+$. The braking distance function is then:

$$f(v, b) = \frac{v^2}{2b} \tag{8}$$

Let then $d \in D$ be the relative distance of the obstacle to the vehicle with $D \subset \mathbb{R}^+$. The main algorithm of the "Emergency braking module" defines two thresholds to determine when a vehicle goes from a "Comfort state" to a "Safety state", and from a "Safety state" to an "Emergency state". Those thresholds are

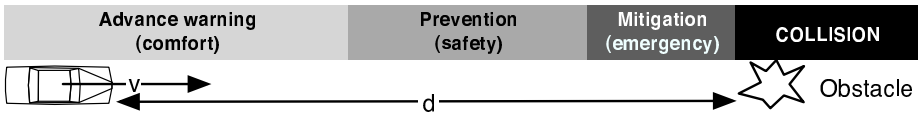

Fig. 7. Emergency braking safety strategy

based on the time left for the driver to react. According to the application specification, if the driver has more than three seconds to react he is in a "Comfort state", then if he has less than three seconds but more than one second he is in the "Safety state", if he has less than one second to react, he is in the "Emergency State". The values of those thresholds are expressed as follows:

$$EB\_Safety = \frac{v^2}{2b} + v * 3 - d \tag{9}$$

$$EB\_Emergency = \frac{v^2}{2b} + v * 1 - d \tag{10}$$

The resulting algorithm of the strategy function can be represented as follows:

```
function Eb_Strategy(d,v,b){
  Eb_Safety = (v^2)/(2b) + v * 3 - d;
  Eb_Emergency = (v^2)/(2b) + v * 1 - d;
  if (Eb_Safety < 0) then  Command = 'Comfort';
  else if (Eb_Emergency < 0) then Command = 'Safety';
     else  Command = 'Emergency'  endif
  return Command;}
```

In SAFESPOT, v values are considered to be in $[0, 46]m/s$, b in $[3, 9]m/s^{-2}$ and d in $[0, 500]m$. If variables are outside those sets, other applications are triggered (this becomes out of the scope of the emergency braking module). For example, speeds above $46m/s$ are managed by the "Speed Alert" application.

4.3 Required Properties

The SAFESPOT and H&IW application specifications are completed with required properties, structured following the FRAME method [22], to be satisfied by the system. An analysis of the H&IW required properties shows that of the 47 main requirements, 18 (i.e. 38%) involve continuous space and/or time constraints. The method presented in this paper focuses on those properties. Here are examples of this kind of properties for the emergency braking module:

- **Property 1**: commands must be appropriately activated. This can be refined as follows.
  - **1.1**: When the braking distance of a vehicle is below its distance from a static obstacle plus one second of driver's reaction time, the H&IW application must trigger an "Emergency" warning.
  - **1.2**: When the braking distance of a vehicle is below its distance from a static obstacle plus three seconds of driver's reaction time, the H&IW application must trigger a "Safety" warning.
  - **1.3**: When multiple obstacles are present on the road, the H&IW application must trigger the associated multiple warnings[6].

[6] When multiple warnings are triggered, they are filtered by the Application Coordinator like shown in Fig. 6.

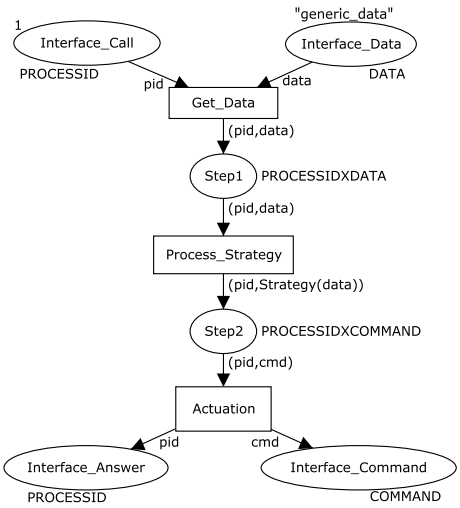

Fig. 8. Template Coloured Petri net for the H&IW applications

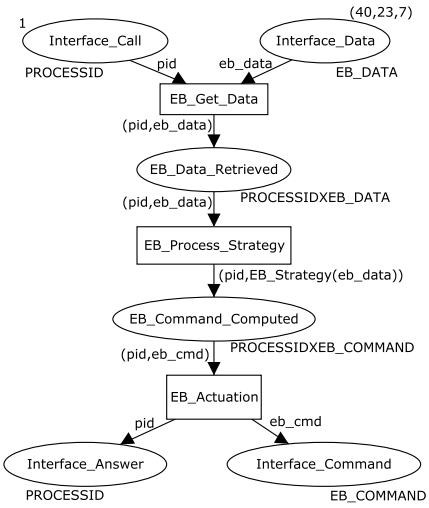

Fig. 9. Coloured Petri net instantiated for the Emergency Braking application

– **Property 2**: commands must be issued progressively stronger
  - When a vehicle is approaching an obstacle, the H&IW application must trigger the different warnings *Confort, Safety, Emergency* in the order corresponding to the danger faced by the vehicle. Therefore, if a driver does not react to the first *Comfort* warning, a *Safety* warning must follow, and then the *Emergency* one.

4.4 The Coloured Petri Net Specification

Several modules in the H&IW application share the same architecture, namely for a given process, data is retrieved from the interface. Then, a command is computed, and sent to appropriate modules in the system. The CPN of Fig. 8 exhibits this generic behaviour (i.e. the template in Sect. 3.2). Transition `Get_Data` has two input arcs from places `Interface_Call` and `Interface_Data`. Place `Interface_Call` is typed with `PROCESSID` which may be an integer subset (here the marking is a token with value 1). Once a process is called and data is retrieved, place `Step1` carries tokens that are couples (pid,data). Transition `Process_Strategy` provides a command resulting from computations on data.

In Fig. 9 this generic schema is instantiated for the Emergency Braking Application (so, `generic_data` and `generic_command` become resp. of type `EB_DATA` and `EB_COMMAND`). Since data are `Distance`, `Velocity`, and `Braking_Factor`:

`EB_DATA = product Distance * Velocity * Braking_Factor`.

Data modelling physical entities are measured with a possible measurement error and are usually represented and computed in $\mathbb{R}^*$ in physics computations. For the CPN specification, we can keep this typing for expressivity sake, while

it is clear that it is not usable in practice (we would use integers for Petri nets tools and float in programming languages).

The `EB_COMMAND` type has three possible values related with the three levels of command or warning, therefore `EB_COMMAND = Comfort | Safety | Emergency`. The appropriate command results from the `EB_Strategy` function computation.

5 Discretization of the Problem

Discretization raises several issues. We propose a way to cope with these issues and apply our solutions to the emergency braking example.

5.1 Implementing Complex Functions in Symmetric Nets

Starting from the CPN model we use the methodology presented in Sect. 3.

First, CPN types must be transformed into discrete types. Using the equal width interval binning discretization method (presented in Sect. 3.3) with a number of k_v, k_b and k_d intervals for each variable we obtain a mesh of $k_v \times k_b \times k_d$ regions (as defined in definitions 32 and 31) in the resulting discretized function. The resulting sets for variables d, v and b are then composed of k ordered elements. For example, with $k = k_v = k_b = k_d = 10$ the resulting discretized type of v is $[0, 4.6, 9.2, ..., 46]$ and the discretized braking function contains 10^3 regions. With $k = 100$, the domain of v is $[0, 0.46, 0.92, ..., 46]$ and the mesh is composed of 10^6 regions.

Sect. 3.3 presents two solutions to model complex functions in Symmetric Nets. We select solution b in Fig. 5 because it is more efficient when computing the symbolic state space. Therefore we add place **EB_Strategy_Table** in the Symmetric Petri net (Fig. 10), and the initial associated marking that is presented in Sect. 5.4.

We chose a simple and generic discretization method that does not take into account the specificity of functions to be discretized. Other discretization methods like those using variable intervals can reduce the number of markings with the same level of accuracy in the resulting discretized function. These aspects are discussed in sections 7.1 and 7.2.

Finally, depending on the analyzed properties, it is also possible to compute and use the equivalence classes. This aspect is discussed in Sect. 6.2.

5.2 Computation of the Error Propagation in Symmetric Nets

As presented in Sect. 3, we compute the precision error introduced by the discretization operation. The resulting error in the computation of the "Threshold"

Discretization par. k / $card(EBData)$	$v = 13m/s,\ b = 8m/s^{-2},$ $d = 500m$	$v = 36m/s,\ b = 4m/s^{-2},$ $d = 100m$
10 / 10^3	$\Delta_{Eb\_Saf} \in [-70.83m, 74.84m]$ $\Delta_{Eb\_Emerg} \in [-61.64m, 65.64m]$	$\Delta_{Eb\_Saf} \in [-118.9m, 144.5m]$ $\Delta_{Eb\_Emerg} \in [-109.7m, 135.3m]$
20 / $8 * 10^3$	$\Delta_{Eb\_Saf} \in [-35.87m, 36.81m]$ $\Delta_{Eb\_Emerg} \in [-31.27m, 32.26m]$	$\Delta_{Eb\_Saf} \in [-61.97m, 68.28m]$ $\Delta_{Eb\_Emerg} \in [-57.37m, 63.68m]$
50 /$12.5 * 10^3$	$\Delta_{Eb\_Saf} \in [-14.45m, 14.61m]$ $\Delta_{Eb\_Emerg} \in [-12.62m, 12.77m]$	$\Delta_{Eb\_Saf} \in [-25.47m, 26.47m]$ $\Delta_{Eb\_Emerg} \in [-23.63m, 24.63m]$
100 /10^6	$\Delta_{Eb\_Saf} \in [-7.25m, 7.29m]$ $\Delta_{Eb\_Emerg} \in [-6.33m, 6.37m]$	$\Delta_{Eb\_Saf} \in [-12.85m, 13.10m]$ $\Delta_{Eb\_Emerg} \in [-11.93m, 12.19m]$

Table 1. Error bounds for different discretization parameters

is:

$$\Delta_{Eb\_Safety} = Eb\_Safety(v \pm \Delta_v, b \pm \Delta_b, d \pm \Delta_d) - Eb\_Safety(v, b, d) \quad (11)$$

$$\Delta_{Eb\_Safety} = (\frac{(v \pm \Delta_v)^2}{2(b \pm \Delta_b)} + 3(v \pm \Delta_v) - (d \pm \Delta_d)) - (\frac{v^2}{2b} + 3v - d) \quad (12)$$

$$\Delta_{Eb\_Safety} = \frac{(v \pm \Delta_v)^2}{2(b \pm \Delta_b)} - \frac{v^2}{2b} \pm 3\Delta_v \pm \Delta_d \quad (13)$$

For example, let us consider (cf. Table 1) a classic private vehicle driving at $v = 13m/s$ (i.e. $50km/h$), on a dry road (i.e. $b = 8m/s^2$), at $d = 500m$ from an obstacle. If we consider $k = 100$ intervals and an error of respectively $\pm 0.45m/s$ for v, $\pm 0.06m/s^2$ for d and $\pm 5m$ for p. Then we obtain:[7]

$\Delta_{Eb\_Safety} \in [-7.25m, +7.29m]$ $\Delta_{Eb\_Emergency} \in [-6.33m, +6.37m]$

For the same vehicle at 100 meters from the obstacle, driving at $v = 36m/s$ (i.e. $130km/h$), on a wet road (i.e. $b = 4m/s^2$), we obtain:

$\Delta_{Eb\_Safety} \in [-12.85m, +13.10m]$ $\Delta_{Eb\_Emergency} \in [-11.93m, +12.19m]$

Those results provide an information on the precision of the Symmetric Net properties. Table 1 gives some error bounds computed from four values for parameter k. As expected, precision of computed thresholds depends on k. However, precision also depends on the values of variables. For example, values of v and b are a determinant for error bound computation. Exploiting those precisions, to validate the Symmetric Net model and its properties, requires to consider carefully those values.

5.3 Validating the Discretization in Symmetric Nets

Discretization of variables and function in the Symmetric Net model in Fig. 10 introduces imprecision. Depending on properties that need to be verified, this imprecision must be considered. For example, properties presented Sect. 4.3 can

[7] In Table 1, Δ_{Eb_Saf} and Δ_{Eb_Emerg} stand for Δ_{Eb_Safety} and $\Delta_{Eb\_Emergency}$ respectively.

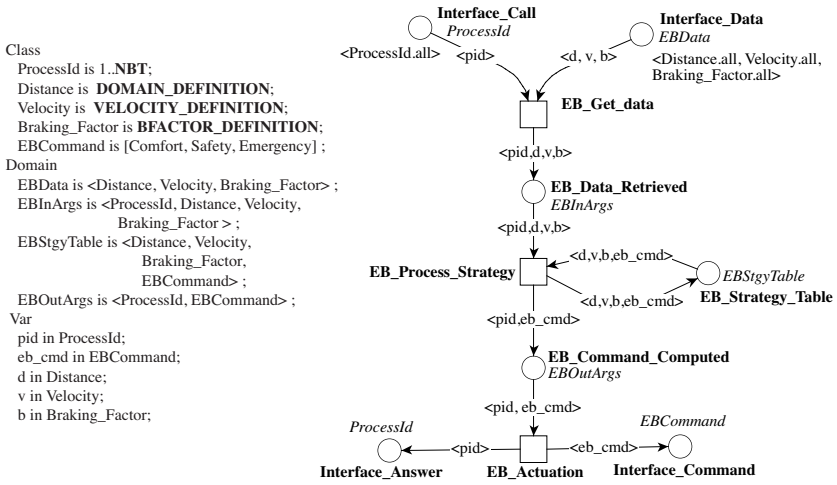

Fig. 10. Symmetric Petri net of the Emergency Braking module (marking of place **EB_Strategy_Table** is not displayed for sake of place)

be verified using CTL (Computation Tree Logic) [21] formulae. With a discretization factor of $k = 100$ values on input variables (cf last line of Table 1), Property 1 (in Sect. 4.3) can be verified with an accuracy smaller than $\pm 7,3m$ on a relative distance, for a velocity of $13m/s$ on a dry road ($b = 8m/s^{-2}$).

If the introduced imprecision is acceptable with regards to the properties to be verified, then the system designer can state that the discretization is valid for those properties. Otherwise, a better accuracy may be required and a new discretization must be done.

It is also possible to integrate the imprecision in the CTL formulae. To do so, more constraining values of input variables must be chosen (i.e. a higher speed, a lower braking factor or a closer obstacle) in the CTL formulae. In our case, the simplest way is to choose a lower value of obstacle position that takes into account the discretization error. For example, the CTL formula:

```
AG((EB_Data_Retrieved == <13,8,500>) => AX(EB_Cmd_Cpt==<Safety>))
```

becomes:

```
AG((EB_Data_Retrieved == <13,8,(500-7.29)>) => AX(EB_Cmd_Cpt==<Safety>)).
```

In some cases, it is possible to compute the discretization of input variables depending on the required precision on the function (see Sect. 7.2).

5.4 Transformation to Obtain the Symmetric Net

This section deals with the transformation of the CPN into a Symmetric Net. First, we present the general principles that are then applied on models dedicated to the verification of the two properties defined in Sect. 4.3.

Computing Discretization Our objective is to get a dedicated Symmetric net from the CPN of Fig. 9. To do so, we must: *i)* discretize continuous types and,

ii) generate the tables corresponding to the functions to be modeled. The result is presented in Fig. 10.

The type `EB_DATA` in Fig. 9 is associated to *EBData* in Fig. 10, that is a cartesian product of three discrete types: `Distance`, `Velocity` and `Braking_Factor`. In Fig. 10, the definition of these types is not represented because it depends on the discretization criteria (e.g. the number of values one can handle).

As presented in Sect. 3.3, the complex function `EB_Strategy` of Fig. 9 is associated to the place **EB_Strategy_Table** in Fig. 10. Its initial marking is a conversion table for the discretized function. The binding between variables d, v, b in inputs arcs of transition **EB_Process_Strategy** enables the selection of the appropriate command in variable eb_cmd.

The marking stored in **EB_Strategy_Table** is generated from the discretized values of domains `Distance`, `Velocity` and `Braking_Factor` by means of the `EB_Strategy` function presented in Sect. 4.2.

The model to verify property 1 (Sect. 4.3) There is no need to change anything to verify property 1 in the model of Fig. 10.

The initial marking of place **Interface_Data** must provide any possible value for inputs since we aim at verifying that all conditions lead to appropriate commands as stated in properties 1.1 and 1.2. This is generated by the tuple of broadcast functions <Distance.all,Velocity.all,Braking_Factor.all>. If we consider several parallel threads to handle several obstacles (see property 1.3 in Sect. 4.3), it is of interest to consider a value greater than one for constant **NBT** in the declaration, thus generating several tokens in place **Interface_Call**. The initial marking of place **Interface_Call** is a set of `ProcessId` (obtained by the broadcast constant <ProcessId.all>). This does not add much complexity in the model and does not invalidate the computation of error propagation. Thus, multiple parallel accesses stated in property 1.3 can be concretely verified.

The model to verify property 2 (Sect. 4.3) This property involves interactions with a functional environment (runtime), and a person (a passive driver). Fig. 11 presents a model on which a passive driver has been added on the right, and part of the functional environment (the runtime is reduced to a feedback loop) on the left. This model allows to verify properties on the module dynamicity, like the fact that, if a driver does not react, the different warning levels (Comfort Safety Emergency) will be issued while the situation becomes more and more dangerous (see Sect. 4.3).

Conclusion This section shows the modelling of some properties to be checked on the emergency braking application. Our models are based on the template proposed in Fig. 9 and we introduce some variations to adapt the template to the verification of properties 1 and 2.

These variations have an impact on the verification complexity. For example, the reachability graph for the second model (with the passive driver) grows linearly with the discretization of the `Distance` colour domain. On the contrary, the complexity of the first model grows much faster.

In these examples, the feedback loops only contain discrete variables (i.e. a command or the PID), but no discretized continuous variables. This avoids the

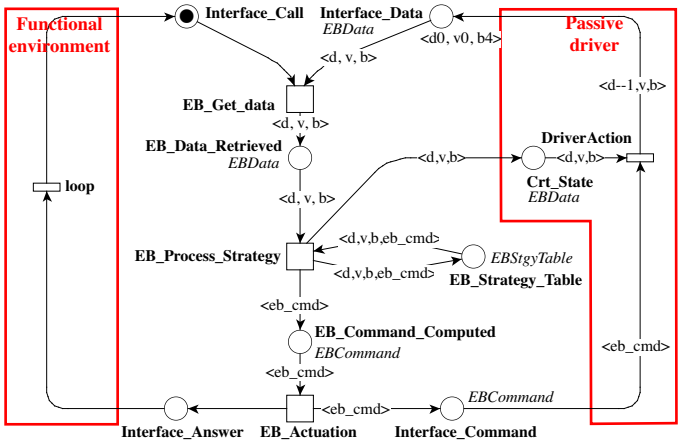

Fig. 11. Model of Fig. 10 enriched with a passive driver (declaration part and marking of place **EB_Strategy_Table** is the one of the model in Fig. 10)

accumulation of errors introduced by the discretization. If a discretized variable is involved in a feedback loop, this entails more constraints on the discretization, and the model should be made in such a way that the errors are not infinitely accumulated.

The next section provides some analysis results. We focus on the first model since it is more challenging with regards to the combinatorial explosion problem.

6 Net Analysis

This section briefly presents the analysis experimentation performed on the symmetric net of Fig. 10 with various configurations.

The use of a discretization method with symmetric nets generates complex models with large markings. It is important to know the consequences on the net analysis and model checking tools.

Objectives The objectives of this analysis are to analyse the net properties as well as to overcome sources of combinatorial explosion.

Experimental method Fig. 12 shows the various techniques that can be used for checking properties. Structural analysis (place invariants, bounds, etc.) can

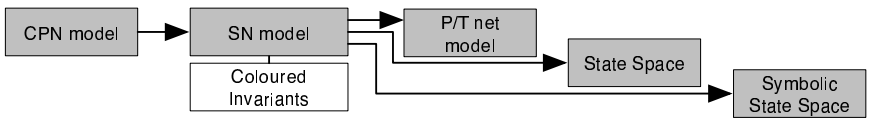

Fig. 12. Overview of the analyses

be performed on the unfolded place/transitions nets as proposed in Sect. 2.2. Coloured invariants can also be computed directly from the coloured net. Model checking can be performed either on the state space or the symbolic state space according to the tool chosen.

Technical aspects The analysis of the Petri Net is a complex operation that requires different transformations of the model like unfolding or reduction. Various tools were used for the analysis. First CPN models were designed with CPN-Tools [17], then symmetric nets were designed using Coloane, an interface for CPN-AMI [37] and PetriScript (a script language to design SN) to generate initial markings. CPN-AMI is used for the analysis of symmetric nets.

6.1 Structural Analysis

We first used structural analysis techniques because they do not require the state space construction and are thus of more reduced complexity.

Symmetric net analysis We computed coloured invariant on the SN, experimenting with various discretizations. The only computed invariant involves place "EB_Strategy_Table", as expected (its marking is stable by definition). Discretization has no significant impact on the memory used for the computation.

Structural Reduction By applying structural reduction on Petri nets [2, 23, 25], it is possible to reduce the net of Fig. 10 into the one of Fig. 13. They are structurally equivalent when we want to check that all situations lead to a planned situation in the system with regards to property 1. The new net avoids most of the useless interleaving when the controller is simultaneously executed by several threads.

The reduced net of Fig. 13 has the same declaration as the one of Fig. 10.

Analysis on unfolded nets Discretization has an impact on the memory required to compute unfolded nets. In fact, the size of the **EB_Data** domain has a cubic growth and thus impacts the resulted P/T net. For example, the number

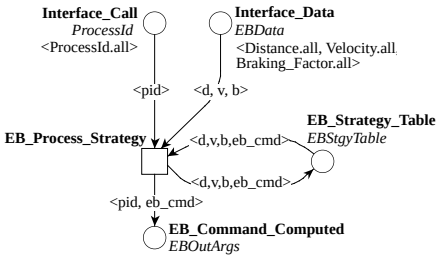

Fig. 13. Reduced net from the one of Fig. 10 (without declaration and marking for **EB_Strategy_Table**)

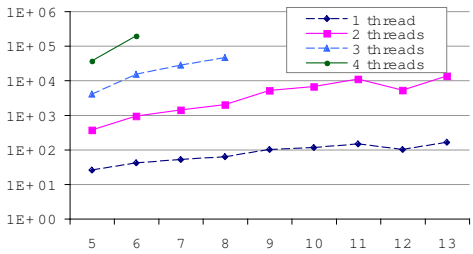

Fig. 14. Evolution of the symbolic state space for the net of Fig. 13 when discretization and involved threads varies

of P/T (np) places grows with $np = 5 * (k)^3 + 8$, where k is the discretization parameter (Sect. 3.3). The CPN-AMI unfolder to P/T nets confirms this formula.

We also proved by unfolding that both nets (Fig. 10 and 13) are bounded. Since the marking has no impact on the validity of structural properties, the analysis is relevant and shows the interest of our methodology.

6.2 Behavioural Analysis

PROD [42] and GreatSPN [13], two model checkers integrated in CPN-AMI [37], were used to complete the behavioural analysis.

State Space Computation According to our tests, the complexity of state space generation is similar to that of unfolding for both memory and time. The symbolic state space generated with GreatSPN shows some (minor) optimization compared to explicit model checking generated by PROD. This is probably due to the lack of symmetries in the marking of place `EB_Strategy_Table`. This should be overcome using two techniques: *i)* the introduction of test arc in SN, *ii)* a dedicated algorithm to enable the model checker to detect places with a stable marking.

The complexity of the binding for transition **EB_Process_Strategy** is:

$$\sum_{i=1..NbP} C^i_{IDM} \tag{14}$$

where $NbP = |ProcessId|$ (the number of values in type *ProcessId*) and $IDM = |M_0(\textbf{Interface\_Data})|$ (the initial number of tokens in place **Interface_Data**). Since $M_0(\textbf{Interface\_Data})$ contains $|Distance| \times |Velocity| \times |Braking\_Factor|$ tokens, this net quickly generates a large state space, requiring dedicated model checking based analysis.

Therefore, we were not surprised when PROD could generate the state space only for a limited number of threads or a very small number of values in `Distance` and `Velocity` types (for our experiments, we fixed `Brakinf_Factor` to 2 values, corresponding to average braking factors for dry and wet roads). This is due to the fact that an explicit representation of state spaces is stored in memory.

Some measures provided by experimentations on GreatSPN are provided in Fig. 14. No value is presented when calculus reached a time-out of 24 hours.

Fig. 14 shows that GreatSPN copes better with the complexity induced by the parallelism when several threads access the Emergency Braking module (we use the automatic detection of symmetries presented in [41]).

Surprisingly, some discretizations could not be computed for more than 2 threads (and with more values, for 2 threads) because of CPU more than memory. This is due to the complexity of the binding of transition **EB_Process_Strategy** that is only symmetric for domain `ProcessId`. As an example, the number of bindings for **EB_Process_Strategy** (computed using formula 14) in the model with 7 values for domains `Distance` and `Velocity` is:

$$C^1_{98} + C^2_{98} + C^3_{98} + C^4_{98} = 3\,769\,227 \tag{15}$$

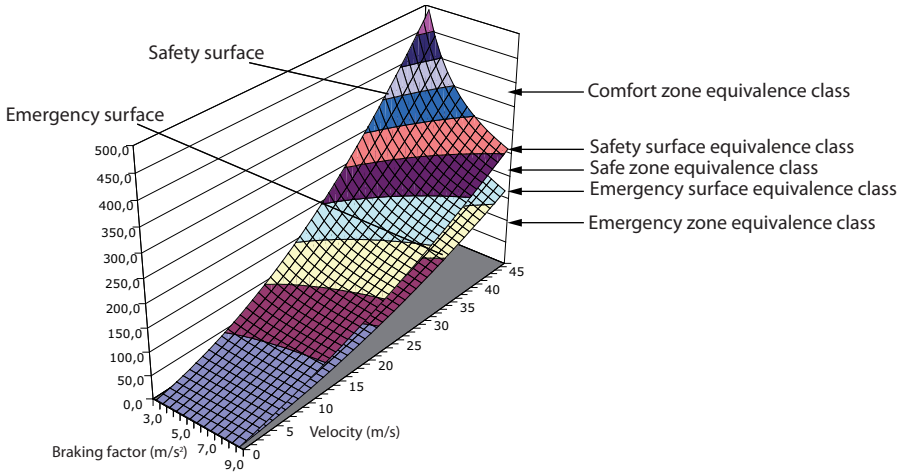

Fig. 15. semantic equivalence classes and surfaces from equations 9 and 10

since places **Interface_Data** and **Interface_Call** initially holds 98 and 4 tokens. The excessive execution time is due to the canonization function that is required by the generation of the symbolic state space. Memory consumption never exceeded 100Mbyte.

6.3 Coping with the Complexity Problem

Our discretization approach allows us to reduce infinite problems to discrete ones. However, verification for a reasonable discretization still raises some problems. This section proposes some hints to cope with such problems.

The first solution we propose consists in a specific modelling technique that can be used for reachability analysis only. The second one resides in the integration of dedicated techniques in model checkers.

Semantic Behavioural Equivalence Classes In SN, the symbolic state space is computed when equivalence classes are provided. They are either defined by the system modeller in colour domains or computed automatically from the SN structure [41]. Here, the structure of our models does not comply with the computation of such equivalence classes.

However, solutions of the equations 9 and 10 define equivalence classes in the system by defining limits between the different situations of the system: “Comfort”, “Safety”, “Emergency”. Fig. 15 shows the five equivalence classes we deduce from the two surfaces computed from the equations of the system.

Thus, we may consider one element per equivalence class in the state space. Projection of these elements on the involved dimensions allows us to reduce colour domains to a very small set of values. In the net of Fig. 10, we may consider only five points in the state space of the system, thus leading to five values in the *distance*, *velocity* and *braking factor* colour domains.

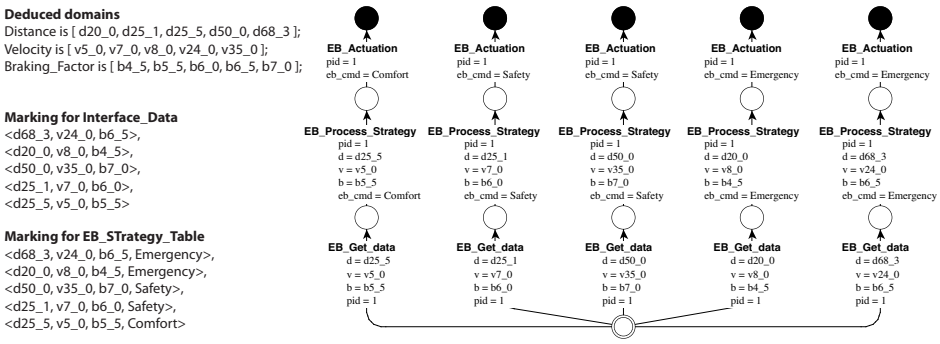

Fig. 16. Reduced state space for the colour domains and place marking defined by means of the semantic equivalent classes

This discretization is relevant for reachability properties only but it remains correct since all the possible equivalence classes in the state space are reachable. Fig. 16 shows the reduced state space for the model of Fig. 10.

We name such equivalence classes *semantic* because, contrary to the classical equivalence classes in SN, they are computed from the definition of the systems and from its structure.

Building Dedicated Model Checkers It is clear that for such systems dedicated model checkers should be designed. Based on our experience, let us identify some useful feature that could cope with the combinatory explosion problem:

1. The use of partial order techniques like in [6]: this is a way to avoid the exploration of redundant paths.
2. Check for places with a stable marking: this is a typically useful optimization when techniques such as decision diagrams like BDD [9] are used. If variables encoding places with a stable marking are place on top of the decision diagram, then the marking is represented only once. For example, PROD stores place **EB_STrategy_Table** and its huge marking was stored for each state in the state space.
3. The use of high-level decision diagrams such as DDD [15], MDD [32] or SDD [16] can also help. These decision diagrams directly encode discrete data (for DDD and MDD) and can handle hierarchy (for SDD). These enable very efficient state space encoding techniques[8] by means of high-level decision diagrams. In particular, this is a way to cope with the previous optimization.
4. The use of recent extension to symmetric Petri Nets where tokens can hold bags themselves [31]. Such an extension allows to fire symbolically transitions with a similar structure than the one of **EB_Process_Strategy** [24].
5. It was shown that GreatSPN could be parallelized in an efficient way [26]. Then, generation of the state space is distributed over a set of machines, thus

[8] The term *symbolic model checking* is used to refer to this diagram-decision based technique. This is different from the symbolic state space.

Discretization par.	$v = 13m/s, b = 8m/s^{-2}$, $d = 500m$	$v = 36m/s, b = 4m/s^{-2}$, $d = 100m$
$k_v = 114, k_b = 73$ $k_d = 120$	$\Delta_{Eb\_Saf} \in [-6.167m, 6.203m]$	$\Delta_{Eb\_Saf} \in [-12.09m, 12.40m]$
$card(EBData) < 10^6$	$\Delta_{Eb\_Emerg} \in [-5.367m, 5.403m]$	$\Delta_{Eb\_Emerg} \in [-11.29m, 11.60m]$

Table 2. Discretization with optimized criteria

allowing the use of more CPU (of interest since canonization in GreatSPN is CPU consuming) and memory.

Unfortunately, these techniques are not yet implemented, or implemented as prototypes only. However, we think they should allow the analysis of discretized systems in the future.

7 Some Perspectives about Discretization

We have described and applied a discretization method to cope with hybrid systems and handle continuous variables in a safe and discrete manner. In this section, we open a discussion on several aspects.

7.1 Optimized Discretization Parameters

The methodology presented in this paper is based on the use of a discretization algorithm to discretize continuous variables. In Sect. 5.2, we used "equal width interval binning" algorithm because it is simple to implement. This algorithm, like many others, relies on discretization parameters that can be optimized for a given set of continuous variables and functions.

However, in the emergency braking module example, we may study the partial derivates of the error on the two thresholds (Δ_{EB_Safety} and $\Delta_{EB\_Emergency}$). We then find that variables v and d are more influent than b. For example, the partial derivate of the error on the *EB_Safety* threshold (equation 13) with respect to the variable v is:

$$\frac{\partial \Delta_{Eb\_Safety}}{\partial v} = \frac{v \pm \Delta_v}{b \pm \Delta_b} - \frac{v}{b} \tag{16}$$

This allows to find optimized discretization parameters considering the respective influence of each involved variable. To do so, parameters for each discretized variable are considered depending on its influence on error propagation.

Table 2 presents the resulting error when discretization parameters are optimized using partial derivatives[9]. It shows that we can reduce the resulting error of about 10% with discretization parameters based on partial derivatives.

[9] Δ_{Eb_Saf} and Δ_{Eb_Emerg} stand for Δ_{Eb_Safety} and $\Delta_{Eb\_Emergency}$ respectively.

The study of the best discretization method and parameters for a given set of continuous variable and function is a complex problem that may give interesting results. It is a promising field for future work on optimization of the methodology presented in this paper.

7.2 Tuning the Discretization

It is of interest to compute the discretization intervals of discretized types (here k_b, k_v and k_d) according to the maximum error tolerated on one type involved in a property where error must be bounded *a priori*.

Let us consider as an example the braking distance function (8) presented Sect. 4.2. It is possible to compute the discretization intervals of variables v and b, based on the accuracy required for the function Δ_f. Let $\pm\Delta_f$ be the tolerated error on f, and $\pm\Delta_v$, $\pm\Delta_b$ be the resulting errors on v and b. Using the error bounds propagation as presented Sect. 3.3 we get:

$$\pm\Delta_f = \frac{(v \pm \Delta_v)^2}{2 * (b \pm \Delta_b)} - \frac{v^2}{2 * b} \tag{17}$$

We then obtain[10]:

$$\Delta_b = -\frac{2 * b^2 * \Delta_f - 2 * b * \Delta_v * v - b * \Delta_v^2}{2 * b * \Delta_f + v^2} \tag{18}$$

and two solutions for Δ_v that are a little bit more complex.

Let v_{min}, v_{max}, b_{min} and b_{max} be the bounds of v and b. The cardinality of V and B sets are:

$$Card(V) = \frac{v_{max} - v_{min}}{2 * \Delta_v} \qquad Card(B) = \frac{b_{max} - b_{min}}{2 * \Delta_b} \tag{19}$$

Now, consider that we want the same cardinalities for V and B colour sets ($k_b = k_v$). We obtain[11]:

$$\Delta_v = \frac{(v_{max} - v_{min})\Delta_b}{b_{max} - b_{min}} \tag{20}$$

Using the value of Δ_v of equation (20) in equation (18), it is now possible to compute Δ_b from the desired Δ_f.

For only two variables, this method is complex as it gives multiple solutions that need to be analyzed to choose the appropriate solutions. However, it provides a way to compute the discretization intervals of input variables depending on the desired output error.

[10] We intentionally removed the $\pm$ operator to increase readability.

[11] Note that it is possible to choose another factor between *Card(V)* and *Card(B)* as explained in Sect. 7.1

8 Conclusion

This paper proposes a way to integrate continuous aspects of complex specifications into a discretized Petri Net model for model checking purpose. Our approach takes place in the context of Intelligent Transport Systems and, more precisely, the management of emergency braking when an obstacle is identified on the road.

Discretization methods rely on the equations describing the problem. In our work, these equations come from the physical model interacting with the system. It is crucial for engineers to evaluate the quality of the proved properties and their impact on the system verification.

To this end, we compute a discretized abstraction from these equations. The abstraction quality is then evaluated with regards to the properties to be checked This is a key point in modelling and evaluating a system by means of formal specification. Typically, imprecision raised by discretization is corrected by either applying a more precise discretization or adding constants in formulas expressing properties to be checked.

These equations are attached to a Coloured Petri net (CPN) template. Discretization is proposed to provide a finite model, even if large. This CPN is then transformed into a symmetric net (SN) to take advantage of the dedicated verification techniques. Analysis is performed on the SN.

Although the analysis performed on our example remains limited by the combinatorial explosion, we can extract the main properties (boundness, reachability properties) when colour domains are of reasonable size. Let us note that without discretization, evaluation of such continuous systems by means of model checking techniques is still an open issue.

In the context of a SAFESPOT application, several modules run in parallel and may introduce more continuous types and variables. In particular, experimenting propagation of discretization constraints between different modules need a particular attention. Future work will then have to evaluate how a larger number of variables and constraints could be managed. We must also investigate to cope with several modules at a time.

In our methodology, several discretization algorithms can be applied. As a first trial, we experimented a simple algorithm but other ones based on non-uniform discretization intervals are promising. These others discretization techniques might introduce new constraints in formal verification and in error propagation computation but it is an interesting field in the future.

Finally, the notion of semantic behavioural equivalences extracted from the equations and injected in the specification by means of initial marking is of interest. It may help in tackling the combinatorial explosion when several modules are considered for reachability analysis. Exploitation of such information from the problem, when possible, seems the most interesting perspective for future work on this methodology.

Acknowledgements We thank the anonymous referees for their detailed and fruitful comments.

References

1. J.-R. Abrial. *The B book - Assigning Programs to meanings*. Cambridge Univ. Press, 1996.
2. G. Berthelot. Transformations and decompositions of nets. In *Advances in Petri Nets*, LNCS 254, pages 359–376. Springer Verlag, 1986.
3. J. Billington, G. E. Gallasch, and L. Petrucci. Verification of the class of stop-and-wait protocols modelled by coloured Petri nets. *Nord. J. Comput.*, 12(3):251–274, 2005.
4. F. Bonnefoi, F. Bellotti, T. Scendzielorz, and F. Visintainer. SAFESPOT Applications for Infrasructure-based Co-operative Road Safety . In *14th World Congress and Exhibition on Intelligent Transport Systems and Services*, 2007.
5. F. Bonnefoi, L. Hillah, F. Kordon, and X. Renault. Design, modeling and analysis of ITS using UML and Petri Nets. In *10th International IEEE Conference on Intelligent Transportation Systems (ITSC'07)*, pages 314–319. IEEE Press, 2007.
6. R. Brgan and D. Poitrenaud. An efficient algorithm for the computation of stubborn sets of well formed Petri nets. In G. D. Michelis and M. Diaz, editors, *Application and Theory of Petri Nets*, LNCS 935, pages 121–140. Springer, 1995.
7. R. Brignolo. Co-operative road safety - the SAFESPOT integrated project. In *APSN - APROSYS Conference*. Advanced Passive Safety Network, May 2006.
8. D. Buchs and N. Guelfi. A formal specification framework for object-oriented distributed systems. *IEEE Trans. Software Eng.*, 26(7):635–652, 2000.
9. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.
10. R. B. C. Covault and D. Driscoll. *Uncertainties and Error Propagation - Appendix V of Physics Lab Manual*. Case Western Reserve University, 2005.
11. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Trans. Computers*, 42(11):1343–1360, 1993.
12. G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability graph for coloured Petri nets. *Theoretical Computer Science*, 176(1–2):39–65, 1997.
13. G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets Performance Evaluation. *special issue on Performance Modeling Tools*, 24((1&2)):47–68, November 1995.
14. P. Christofides and N. El-Farra. *Control Nonlinear and Hybrid Process Systems: Designs for Uncertainty, Constraints and Time-delays*. Springer Verlag, 2005.
15. J.-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P.-A. Wacrenier. Data decision diagrams for Petri net analysis. In *Proc. ICATPN'2002*, LNCS 2360, pages 101–120. Springer Verlag, June 2002.
16. J.-M. Couvreur and Y. Thierry-Mieg. Hierarchical Decision Diagrams to Exploit Model Structure. In *Proc. FORTE'05*, LNCS 3731, pages 443–457. Springer, 2005.
17. The CPN Tools Homepage, 2007. `http://www.daimi.au.dk/CPNtools`.
18. J. Daniel and D. Luca. IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments. In *Proceedings of Vehicular Technology Conference,VTC Spring, IEEE*, pages 2036–2040, May 2008.
19. R. David and H. Alla. On Hybrid Petri Nets. *Discrete Event Dynamic Systems: Theory and Applications*, 11(1-2):9–40, 2001.
20. J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Int. Conf. on Machine Learning, pp 194-202*, 1995.

21. E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.
22. Frame Forum. The FRAME forum home page, http://www.frame-online.net.
23. S. Haddad. A reduction theory for coloured nets. In *Advances in Petri Nets 1989*, LNCS 424, pages 209–235. Springer-Verlag, 1990.
24. S. Haddad, F. Kordon, L. Petrucci, J.-F. Pradat-Peyre, and N. Trèves. Efficient State-Based Analysis by Introducing Bags in Petri Net Color Domains. In *28th American Control Conference (ACC'09)*, St-Louis, USA, June 2009. IEEE.
25. S. Haddad and J.-F. Pradat-Peyre. New efficient Petri nets reductions for parallel programs verification. *Parallel Processing Letters*, 16(1):101–116, Mar. 2006.
26. A. Hamez, F. Kordon, Y. Thierry-Mieg, and F. Legond-Aubry. dmcG: a distributed symbolic model checker based on GreatSPN. In *28th International Conference on Petri Nets and Other Models of Concurrency (ICATPN'07)*, LNCS 4546, pages 495–504. Springer, 2007.
27. L. Hillah, F. Kordon, L. Petrucci, and N. Trèves. PN standardisation : a survey. In *FORTE'06*, LNCS 4229, pages 307–322. Springer Verlag, September 2006.
28. J. Hugues, Y. Thierry-Mieg, F. Kordon, L. Pautet, S. Baarir, and T. Vergnaud. On the Formal Verification of Middleware Behavioral Properties. In *FMICS'04*, pages 139–157. Elsevier, 2004.
29. ISO/IEC-JTC1/SC7/WG19. International Standard ISO/IEC 15909: Software and Systems Engineering - High-level Petri Nets, Part 1: Concepts, Definitions and Graphical Notation, December 2004.
30. K. Jensen and L. M. Kristensen. *Coloured Petri Nets, Modelling and Validation of Concurrent Systems*. Monograph to be published by Springer Verlag, 2008.
31. T. Junttila. *On the symmetry reduction method for Petri Nets and similar formalisms*. PhD thesis, Helsinki University of Technology, Espoo, Finland, 2003.
32. T. Kam, T. Villa, R. Brayton, and A. L. Sangiovanni-Vincentelli. Multi-Valued Decision Diagrams: Theory and Applications. *International Journal on Multiple-Valued Logic*, 4(1-2):9–62, 1998.
33. F. Kordon, A. Linard, and E. Paviot-Adet. Optimized Colored Nets Unfolding. In *FORTE'06*, LNCS 4229, pages 339–355. Springer Verlag, 2006.
34. C. Lakos and G. Lewis. Incremental state space construction of coloured Petri nets. In *ICATPN'01*, LNCS 2075, pages 263–282. Springer, 2001.
35. G. Lewis. *Incremental specification and analysis in the context of coloured Petri nets*. PhD thesis, University of Hobart, Tasmania, 2002.
36. V. Lindberg. *Uncertainties and Error Propagation - Part I of a manual on Uncertainties, Graphing, and the Vernier Caliper*. Rochester Inst. of Technology, 2000.
37. LIP6/MoVe. The CPN-AMI home page, http://www.lip6.fr/cpn-ami/.
38. IEEE 802.11 Working Group for WLAN Standards. *IEEE 802.11 tm Wireless Local Area Networks*. IEEE, 2008.
39. ISO TC204 WG-16. *CALM architecture*. ISO, 2007.
40. Petri Nets Steering Committee. Petri Nets Tool Database: quick and up-to-date overview of existing tools for Petri Nets. `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html`.
41. Y. Thierry-Mieg, C. Dutheillet, and I. Mounier. Automatic Symmetry Detection in Well-Formed Nets. In *ICATPN 2003*, LNCS 2679, pages 82–101. Springer, 2003.
42. K. Varpaaniemi, J. Halme, K. Hiekkanen, and T. Pyssysalo. Prod reference manual. Technical report, Helsinki University of Technology, 1995.
43. J. Vautherin. Parallel systems specifications with coloured Petri nets and algebraic specifications. In *Advances in Petri Nets 1987, covers the 7th European Workshop on Applications and Theory of Petri Nets*, pages 293–308. Springer-Verlag, 1987.