A discretization method from coloured to symmetric nets: application to an industrial example

Fabien **Bonnefoi**
DSO/DSETI,
Cofiroute,
6 - 10 rue Troyon,
92310 Sèvres, France
Fabien.Bonnefoi@cofiroute.com

Christine **Choppy**
LIPN, CNRS UMR 7030,
Université Paris XIII,
99 av. J-B Clément,
93430 Villetaneuse, France
Christine.Choppy@lipn.univ-paris13.fr

Fabrice **Kordon**
LIP6 - CNRS UMR 7606,
Université P. & M. Curie,
4 Place Jussieu,
75252 Paris Cedex 05, France
Fabrice.Kordon@lip6.fr

1 Introduction

Future supervision systems tend to be distributed and at least partially embedded. Distribution brings a huge complexity and then, a strong need to deduce possible (good and bad) behaviours on the global system, from the known behaviour of its actors. This is crucial since mission critical or life critical missions are more and more supervised by such systems. Intelligent Transport Systems (ITS) are a typical example: more and more functions tend to be integrated in vehicles and road infrastructure.

Moreover, in many cases (like ITS), physical constraints are part of the system description. Analysis techniques based on discrete models must integrate such constraints : we then speak of *hybrid* systems.

So, 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 (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 [15] might be a solution to model and analyze hybrid systems but no tool is available to test neither safety nor temporal logic properties [11].

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) [25, 26] to Symmetric Petri Nets (SN) [9, 7].

CPN allow an easy modelling of the system to be analyzed. SN are of interest for their analysis because of the symbolic reachability graph that is efficient to represent the state space of large systems. Moreover, since SN only 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 may generate “precision errors” that could turn a given verified property into a wrong one. In that case, the property to be verified might have to be transformed to take into consideration such precision errors.

Section 2 briefly recalls the notions of CPN, SN and abstraction/refinement, type issues. Our methodology which involves modelling, discretization and verification is presented in Section 3, and we show in Section 4 how we model our Emergency Braking application. The various issues regarding discretization on our case study are detailed in Section 5, and issues on net analysis are presented in Section 6. Some open issues are discussed in Section 7 before a conclusion (Section 8).

2 Building Blocks

This section presents the building blocks from the state of the art used to set up our transformation methodology.

2.1 Coloured Petri Nets

Coloured Petri nets [25, 26] 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 [26]. It 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 [33], object oriented languages [5]), and an extension of the Standard ML language was chosen for CPN Tools [13]. 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 Section 2.2). Here, we want to allow a specification language that fits as much as possible what is needed to describe the problem under study, and then 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 $Var[e] \subseteq V$.

Definition 2.1. *A non-hierarchical coloured Petri net CPN [26] is a tuple* $CPN = (P, T, A, \Sigma, V, N, C, G, E, I)$ *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* $Type[v] \in \Sigma$ *for all variables* $v \in V$.
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 free 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 Section 3, the first step of our methodology is to produce a CPN model for the application under study. The next step is a transformation motivated by the discretization of continuous functions to obtain a symmetric net.

2.2 Symmetric Nets

Symmetric nets[1] were introduced in [9] and [7], with the goal of exploiting symmetries in distributed systems to provide a more compact representation of the state space.

[1] *Symmetric nets* were formerly known as *Well-Formed nets*, a subclass of *High-level Petri nets*. The new name was chosen in the context of the ISO standardisation of Petri nets [21].

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 [22].

Basically, types must be finite enumerations and can only be combined by means of cartesian products. Allowed functions in arc valuation are: Id, 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 2.1.

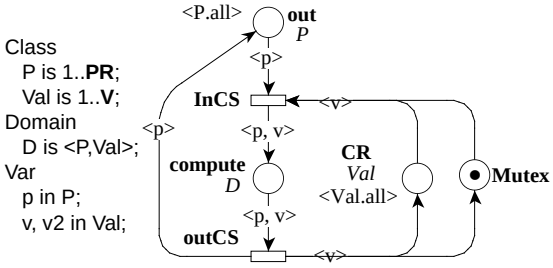

Figure 1: Example of Symmetric Net

The Symmetric net in Figure 1 represents a class of threads (identified by an identity in 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 [23]). Place **out** initially holds one token for each value in *P* (the marking is then denoted $< P.all >$) and place **CR** holds one value for each value in *Val*.

Verification of properties can be achieved either by a structural analysis, on the symbolic reachability graph (model checking), or on the unfolded associated Place/Transition (PT) net (model checking as well as 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 [28, 29], the type refinement, the node refinement and the subnet refinement. The idea for these refinements to be correct is that behaviours should be preserved, and to any behaviour of a refined net it should be possible to match a behaviour of the abstract net.

We have here another motivation that is raised by the use of tools to check the behaviour and properties of the model, and that 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 a complex system. We first give an overview of the approach and then detail its main steps and the involved techniques.

3.1 Overview of the Methodology

Figure 2 sketches our methodology. It takes as input a set of requirements structured following the FRAME method [20]. 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.

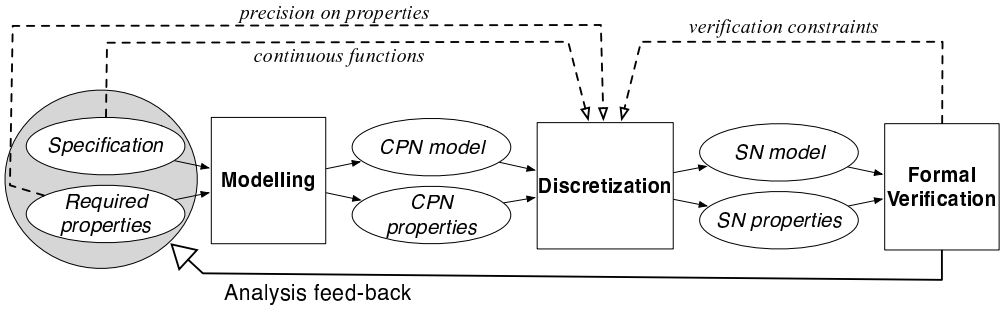

Figure 2: Overview of our methodology

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 one 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 analyzed 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 [3]. 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.

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 infrastructure) have to deal with physical variables such as braking distances, speed and weigth. In [3] we presented a methodology to model large and complex ITS starting from a specification mainly based on a subset of UML diagrams.

This methodology [3] is also based on the definition and use of an ITS template. To have a hierarchical and structured specification using a relevant subset of UML diagrams, we proposed an ITS template that allows variations of architectures and component variables. The architectures are defined, involving components and their interconnections through interfaces. This enables us to change and update components of the architecture and to generate the Petri Net model easily. This template was elaborated from the investigation of case studies of the SAFESPOT and TrafficView projects [4, 14].

The system high level architecture is specified using UML component diagrams. Interfaces between components are specified with class diagrams. This first step of the methodology is used to identify the different components of the system and their counterparts in Petri nets. It is also used to define how they should be assembled to compose a complete model. Then, the behaviour of each component can be specified either with UML activity diagrams, UML state machines or Petri nets. This methodology relies on the use of Petri scripts to assemble the complete model but also for modelling complex components.

This methodology is well suited to have a fast, efficient, modular and incremental approach in modelling large systems. But only a subpart of the "required properties" of the system could be checked. Especially, it

was not possible to verify properties related to quantitative variables as they are usually abstracted in the Petri nets.

The work presented in this paper 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 Section 4. The "interfaces" of the Petri net model, presented in Section 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 modeled 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

- Continuous feature discretization.
- Error propagation computing
- Type transformation and modelling of complex functions in Symmetric Nets.

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 that are used in this paper to avoid ambiguity:

Definition 3.1. *A **region** is a n-dimentional polygon (i.e. a polytope) made by adjacent points of an n-dimentional discretized function.*

Definition 3.2. *A **mesh** is a set of regions used to represent a n-dimentional 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 [17].

- **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) [17] produce a mesh over the entire n-dimentional 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** 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:

$$\Delta = \frac{x_{max} - x_{min}}{k} \tag{3.1}$$

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 below [6]:

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 [30, 6]. The most current one is to determine the separate contribution due to errors on input variables and to combine the individual contributions in quadrature.

$$\Delta_{f(x,y,..)} = \sqrt{\Delta_{fx}^2 + \Delta_{fy}^2 + ...} \tag{3.2}$$

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 error on x (i.e. Δ_x) with the partial derivative of $f(x,y,..)$:

$$\Delta_{fx} = \frac{\partial f(x,y,..)}{\partial x}\Delta_x \tag{3.3}$$

- The computational method computes the variation directly by a finite difference:

$$\Delta_{fx} = \mid f(x+\Delta_x,y,..) - f(x,y,..) \mid \tag{3.4}$$

The use of individual contribution 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 $2 * \Delta_x$ 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:

$$f(x) = f(x_{disc}) \pm \Delta_{f(x)} \tag{3.5}$$

$$\Delta_{f(x)} = f(x \pm \Delta_x) - f(x) \tag{3.6}$$

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

$$\Delta_{f(x)} \in [Min(f(x \pm \Delta_x) - f(x)), Max(f(x \pm \Delta_x) - f(x))] \tag{3.7}$$

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:

$$\Delta_f \in [Min(f(x \pm \Delta_x, y \pm \Delta_y, ..) - f(x,y,..)), Max(f(x \pm \Delta_x, y \pm \Delta_y, ..) - f(x,y,..))] \tag{3.8}$$

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

Type transformation Once the best discretization actions are decided upon as regards our goals, the CPN specification may be transformed. The resulting net is 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 graph 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 Figure 3, 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.

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.

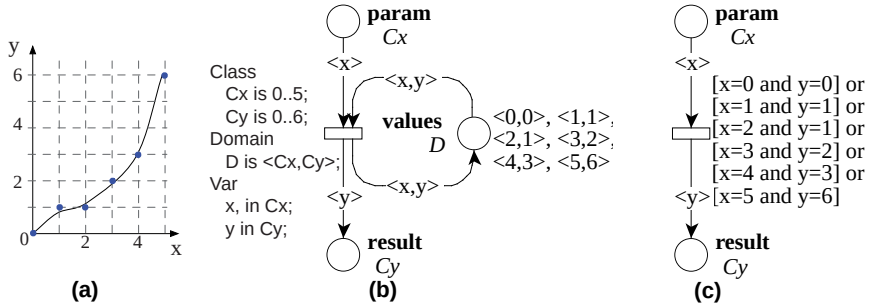

Figure 3: Example of complex function discretization by means of a place or a transition guard

Figure 3 represents an example of function discretization. The left side (a) of Figure 3 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 constants.

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 potential 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 [10]. Thus, the choice of a discretization schema must be evaluated, for example to ensure that uncertainty remains in a safe range.

3.4 Verification

We use CPN-AMI [31] to perform verification. So far, our models can be analyzed 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 [27] is used to derive the corresponding P/T net to compute structural properties.
- *Model checking*, we designed 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 [22, 3].

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 Figure 2, 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 [4]. 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 [34, 19, 24]), will extend in space and time the driver's awareness of the surrounding environment.

The SAFESPOT applications [2] 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 Figure 4, information measured by sensors is provided to the "Data Processing / Fusion" module or transmitted through the network to the "Data Fusion 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.

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

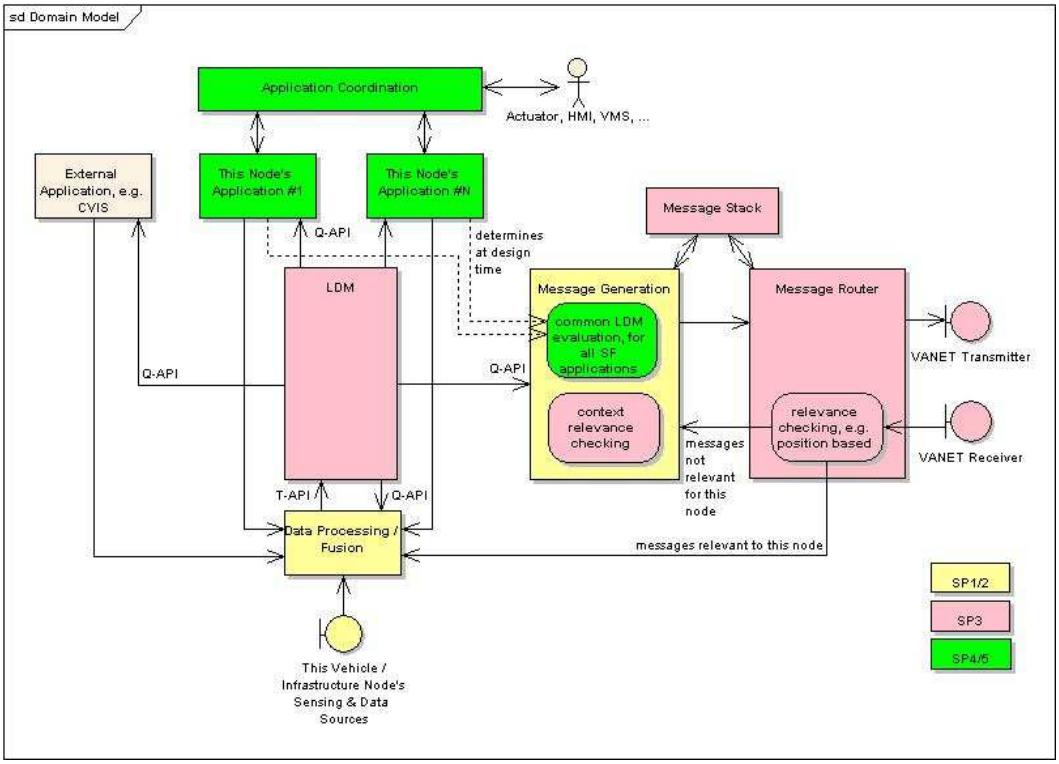

Figure 4: SAFESPOT High Level Architecture

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

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 of the application can be verified with Petri nets by making an abstraction of the data flow where "continuous" variables are involved. This is where we face a huge combinatorial explosion and have to enhance the Petri net formalism and modelling methodology to enable the modelisation and verification of this kind of systems.

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 Figure 5. 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).

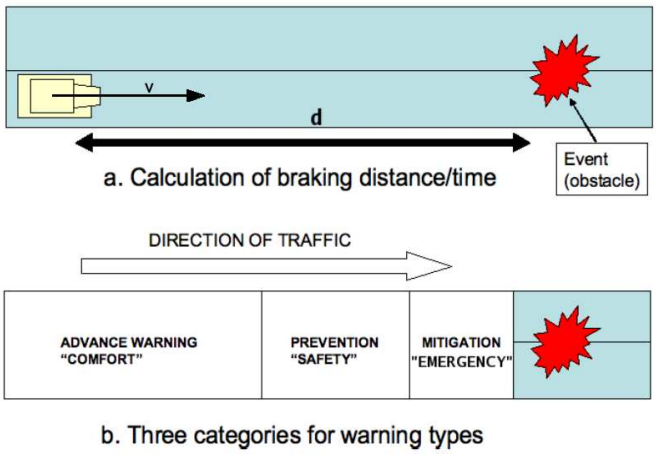

Figure 5: Emergency braking safety strategy

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:

$$f(v,b) = \frac{v^2}{2b} \tag{4.1}$$

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 sate" to a "Safety state", and from a "Safety state" to an "Emergency state". Those thresholds are based on the time left to 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 follow:

$$EB\_Safety = \frac{v^2}{2b} + v * 3 - d \tag{4.2}$$

$$EB\_Emergency = \frac{v^2}{2b} + v * 1 - d \tag{4.3}$$

The resulting algorithm of the strategy function can be represented with this pseudocode:

```
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 exemple, 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 to be satisfied by the system. An analysis of the H&IW required properties shows that over the 47 main requirements, 18 involve continuous space and/or time constraints (i.e. 38%). 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**: 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.
- **Property 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.

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 coloured Petri net of Figure 6 exhibits this generic behaviour (i.e. the template mentioned in Section 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 from data.

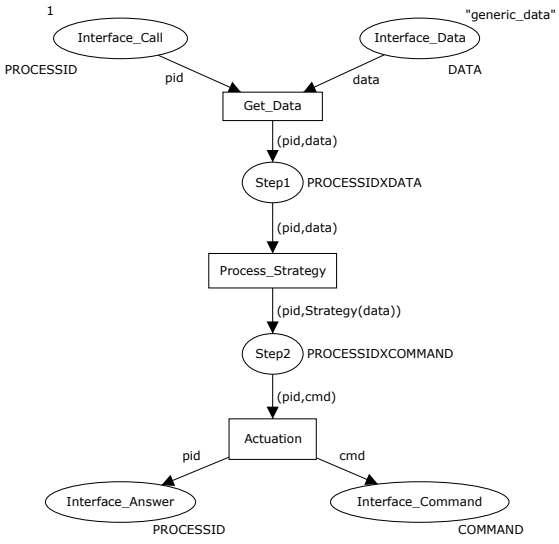

Figure 6: Template Coloured Petri net for the H&IW applications

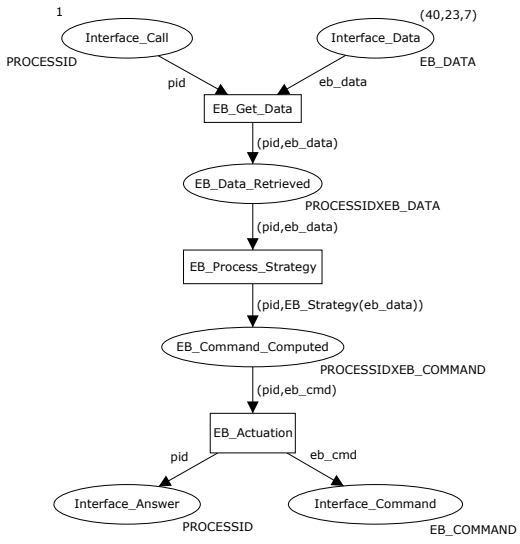

Figure 7: Coloured Petri net instantiated for the Emergency Braking application

In Figure 7 this generic schema is instanciated for the Emergency Braking Application (so, `generic_data` and `generic_command` become `EB_DATA` and `EB_COMMAND`). Data for this application are `Distance`, `Velocity`, and `Braking_Factor`, thus:

`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 Section 3.

First, CPN types must be transformed into discrete types. Using the equal width interval binning discretization method (presented in Section 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 3.2 and 3.1) 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.

Section 3.3 presents two solutions to model complex functions in Symmetric Nets. We select solution b in Figure 3 because it is more efficiently represented in the Symbolic Reachability Graph. Therefore we add place "EB_Strategy_Table" in the Symmetric Petri net (Figure 8). Thus, cardinalities of the domains for places "Interface_Data" and "EB_Strategy_Table" (respectively named "EB_Data" and "EBStgyTable" in Figure 8) are computed using the formula: $Card(EBData) = Card(EBStgyTable) = Card(D \times V \times B)$. This means that these cardinalities are equal to the number of regions of the discretized function.

This method could provide very large markings (that is with a large number of tuples) in the resulting Symmetric Net. However, the use of a appropriate state space representation (by means of decision diagrams like in [12]) does not impact the size of the generated state space since the large marking is just represented once (the marking of places encoding complex functions is stable).

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. Finally, depending on the kind of expected analysis, it is also possible to compute and use the equivalence classes. Those aspects are discussed later on this paper in sections 7.1 and 6.2.2.

5.2 Computation of the error propagation in Symmetric Nets

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

$$\Delta_{Eb\_Safety} = Eb\_Safety(v \pm \Delta_v, b \pm \Delta_b, d \pm \Delta_d) - Eb\_Safety(v, b, d) \tag{5.1}$$

$$\Delta_{Eb\_Safety} = \left(\frac{(v \pm \Delta_v)^2}{2(b \pm \Delta_b)} + 3(v \pm \Delta_v) - (d \pm \Delta_d)\right) - \left(\frac{v^2}{2b} + 3v - d\right) \tag{5.2}$$

$$\Delta_{Eb\_Safety} = \frac{(v \pm \Delta_v)^2}{2(b \pm \Delta_b)} - \frac{v^2}{2b} \pm 3\Delta_v \pm \Delta_d \tag{5.3}$$

For example, let us consider a classic private vehicle driving at $v = 14m/s$ (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:

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

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

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 determinant on the computation of error bounds. Exploiting those precisions, to validate the Symmetric Net model and its properties, requires to consider carefully those values.

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

Table 1: Error bounds for different discretization parameters

5.3 Validating the discretization in Symmetric Nets

Discretization of variables and function in the Symmetric Net model in Figure 8 introduces imprecision. Depending on properties that need to be verified, this imprecision must be considered. For example, properties presented section 4.3 can be verified using CTL (Computation Tree Logic) [18] formulae. With a discretization factor of $k = 100$ values on input variables, property 1 can be verified with an accuracy smaller than $\pm 7,3m$ on a relative distance, for a velocity of $14m/s$ on a dry road.

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 value of input variables must be chosen (i.e. an higher speed, a lower braking factor or a closer obstacle) in the CTL formula. In our case, the simplest way is to choose a lower value of obstacle position that cover the discretization error.

In some cases, it is possible to compute the discretization of input variables depending on the required precision on the function. This solution is discussed in section 7.2.

5.4 Transformation to obtain the Symmetric net

The SN in Figure 8 is derived from the CPN in Figure 7. Our purpose is to obtain a manageable state space for model checking, and, as presented in Section 3.3 and in Figure 3, this leads us to discretize some types and also to adopt some modelling for complex functions.

Thus, the different fields of `EB_DATA` in Figure 7 are now discretized. For example, type Distance is discretized into an enumeration: `_0`, `_50`, `_100`, etc.

`EB_DATA` is associated to *EBData* in the SN of Figure 8 that is a list of 3-uples (`Distance`, `Velocity`, `Braking_Factor`).

Now, as explained in Section 3.3 and shown in Figure 3 (b), the approach for modelling the function `EB_Strategy` is to add a place with a marking that is a conversion table for the discretized function. Thus, the `EB_Strategy` function in Figure 7 is associated in Figure 8 to a table *EBStgyTable* that represents the discretized function (`Emergency`, `Safety` or `Comfort`). This result is retrieved by means of place `EB_Strategy_Table` connected to transition `EB_Process_Strategy`.

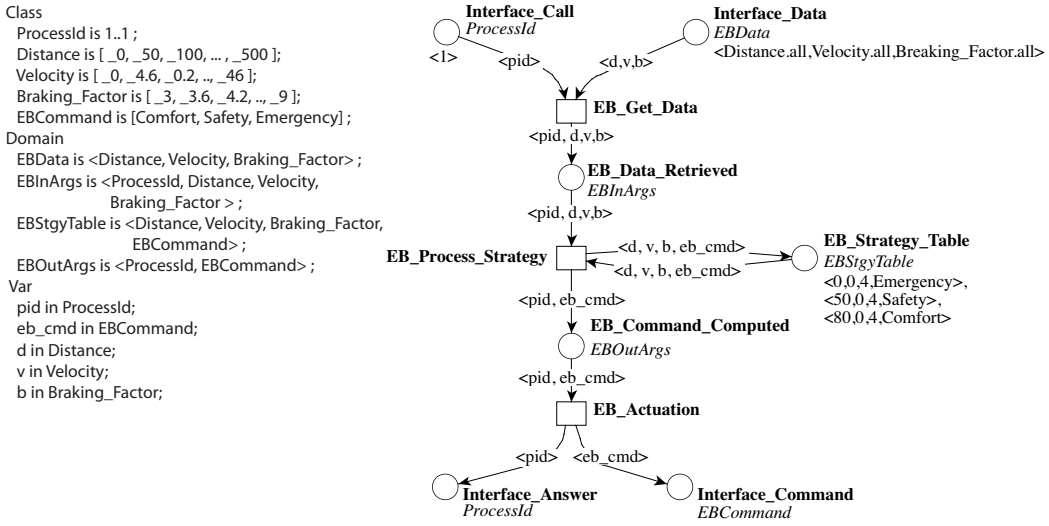

Figure 8: Symmetric Petri net for Emergency Braking module

Of course, the models presented in this paper are only sub-parts of a system. They can be independently verified but the purpose is to integrate them in a more complete representation of the system. This integration may introduce new discretization constraints and verification formulae must be rewritten. Those aspects are discussed in section 7 of this paper.

6 Net analysis

The use of a discretization method with symmetric nets generates complex models with large markings. It is important to know what are the consequences on the net analysis and model checking tools. In this section we present an overview of the analysis results obtained on the model.

Objectives The objectives of this analysis was first to analyse the properties of the net and sources of combinatorial explosion. Another interesting aspect of this analysis was to find the limitation of the tools used, which are not a priori suited to this type of net, and find some optimisation methods.

Experimental method As the complexity of the models presented in this paper is mainly dependent on the discretisation made on the three input variables, we focussed on the impact of this discretization. The symetric net model of Figure 8 was adapted to the experiment by connecting place "Interface_Answer" to place "Interface_Call" with two arcs and a transition with arcs expressions assigning variable $< pid >$ to the arcs. This allows the net to loop until the marking in place "Interface_Data" is empty. The marking of place "Interface_Data" was initialized with all values of domain "EBData" as presented Figure 8. Then scripts were used to initialize the class declaration and the marking of place "EB_Strategy_Table" depending on the chosen discretization level. Figure 9 gives an overview of the subset of properties that were tested.

Technical aspects The analysis of the Petri Net is a complex operation that requires different transformations of the model like unfolding or reduction. Different tools were used to make various analysis. First the CPN models were designed with CPN-Tools [13], then the symmetric models were designed using Coloane and

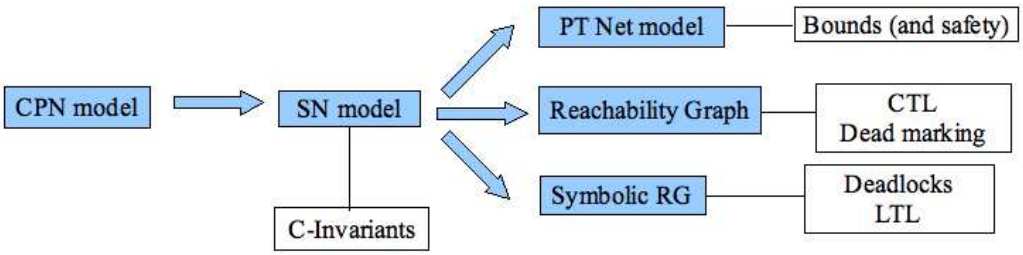

Figure 9: Overview of the analyses

Petriscript [31].To make the analysis of the model we chose the CPN-AMI [31] environment that provides a unified access to different tools like: a Petri net unfolding tool, PROD [32] or GreatSPN [8].

6.1 Structural analysis

The first analyses made on the net are structural analyses. They do not require the construction of the reachability graph and then, do not require to apply firing rules. Therefore they are less complex than behavioural analyses.

6.1.1 Symmetric net analysis

We made the computation of Coloured-Invariant on the Symmetric net with different discretisations. The only invariant detected is the marking of place "EB_Strategy_Table", as expected. The results show that the discretization does not have a significant impact on the memory used for the computation. But due to the size of the marking of place "EB_Strategy_Table", which is composed of all associations between variables and commands, the tool is not able to show the invariant for large markings even if it claims to have made the computation.

6.1.2 Unfolding the net

The computation of the unfolded net requires an increasing amount of memory depending on the discretization of input variables. It also gives an increasing unfolded net. In fact, the size of the domain "EBData" has a cubic growth. An analysis of the symmetric net shows that the size of the unfolded net in terms of places (np) follows the law: $np = 5 * (k)^3 + 8$,where k is the discretization level. The use of the CPN-AMI unfolder confirms that the unfolding of the symmetric net did follow this law. It also appears that the memory used to compute the unfolded net grows even more quickly than the size of the unfolded net. This explains why we faced a combinatorial explosion in the computation of the unfolded net which has bounded the coverage of our experiment.

6.1.3 Bound computation

We were able to test the bounds and safety of the net. The net is effectively bounded but the complexity of the computation, in terms of memory and time used, is the same as the one of the unfolding operation.

6.2 Behavioural analysis

The behavioural analysis is based on the use of Great-SPN and Prod to produce the reachability graphs.

6.2.1 Computation of the reachability graph

The generation of rechability graphs seems, according to the few tests that we made, to have about the same complexity in terms of memory and time as that of the unfolding of the net. The size seems also to follow

a cubic growth. Using the symbolic rechability graph of GreatSPN is a little bit more efficient. We did not make enough LTL and CTL queries to provide conclusions but the detection of deadlocks is not generating an additionnal combinatorial explosion.

6.2.2 Semantic equivalence Classes in the Model

If we consider the computation of *safety properties* (also called reachability properties) in the reachability graph, we can deduce that numerous states correspond to similar execution path in the original program or specification.

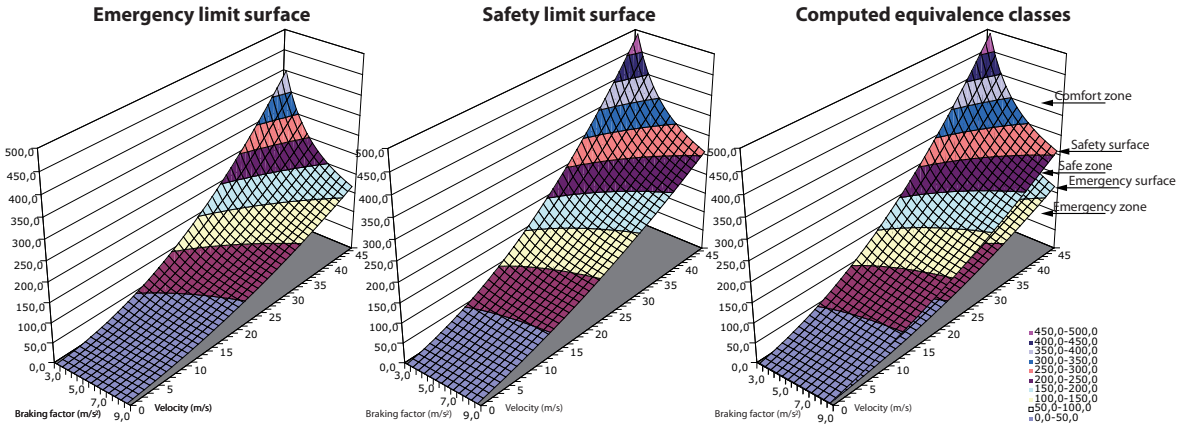

Figure 10: Building behavioural equivalence classes from the surfaces generated by the resolution of safety and emergency equations.

Thus, some accessibility properties could be preserved through equivalence classes. It is then of interest to exploit these, for example, when the computed subnet is integrated into a larger specification.

In our example, the equivalence classes are based on the net behaviour and must be computed from the physics equations that define the limits between the different situations of the system: “Comfort”, “Safety”, “Emergency”. To determine the surface that delimits equivalence classes in the state space, we compute solutions equations 4.2 and 4.3. We then get the two surfaces displayed on the left part of Figure 10.

From these surfaces, we can deduce five equivalence zones in the reachability graph as shown on the right part of Figure 10:

- the one above safety limit surface,
- the safety limit surface itself,
- the one between safety and emergency surfaces,
- the emergency surface itself,
- the one below the emergency surface.

For reachability properties, it is possible to provide a coherent discretization that reaches at least all these equivalence classes by randomly selecting any point in each zone and adding the corresponding value of b (from B axis), d (from D axis) and v (from V axis). Then, for each discretized colour B, D and V, we can take the coordinates of five points randomly chosen in these five zones. This approach is similar to the one proposed in [16] that was dealing with one colour domain only and used guards from the Petri nets to compute equivalence classes.

6.3 Conclusion on the analysis

The conclusions on the analysis of the nets are balanced. We were able to check properties but not for large models. The limitation comes from either the limitation of the tools, that are often not able to manage large markings, or from the complexity of the property to be analysed. We think anyway, that exploiting behavioural symmetries will solve some of those limitations.

7 Discussion and Open Issues

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 Other discretization parameters

The methodology presented in this paper is based on the use of a discretization algorithm to discretize continuous variables. In section 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 5.3) with respect to the variable v is:

$$\frac{\partial \Delta_{Eb\_Safety}}{\partial v} = \frac{v \pm \Delta_v}{b \pm \Delta_b} - \frac{v}{b} \tag{7.1}$$

This allows to find optimized discretization parameters considering the respective influence of each involved variable. This is done by considering different parameters for each discretized variable depending on its influence on the error propagation.

Discretization parameters	$v = 14m/s, b = 8m/s^{-2}, d = 500m$	$v = 46m/s, b = 4m/s^{-2}, d = 100m$
$k_v = 114, k_b = 73, k_d = 120$	$\Delta_{Eb\_Safety} \in [-6.167m, 6.203m]$	$\Delta_{Eb\_Safety} \in [-12.09m, 12.40m]$
$card(EBData) < 10^6$	$\Delta_{Eb\_Emergency} \in [-5.367m, 5.403m]$	$\Delta_{Eb\_Emergency} \in [-11.29m, 11.60m]$

Table 2: Discretization with optimized criteria

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

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

7.2 Tunning the discretisation

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 fonction (4.1) presented section 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 section 3.3 we get:

$$\pm\Delta_f = \frac{(v \pm \Delta_v)^2}{2 * (b \pm \Delta_b)} - \frac{v^2}{2 * b} \tag{7.2}$$

We then obtain[2]:

$$\Delta_b = -\frac{2*b^2*\Delta_f - 2*b*\Delta_v*v - b*\Delta_v^2}{2*b*\Delta_f + v^2} \tag{7.3}$$

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:

$$Card(V) = \frac{v_{max} - v_{min}}{2*\Delta_v} \tag{7.4}$$

$$Card(B) = \frac{b_{max} - b_{min}}{2*\Delta_b} \tag{7.5}$$

Now, consider that we want the same cardinalities for V and B colour sets ($k_b = k_v$). We obtain[3]:

$$\Delta_v = \frac{(v_{max} - v_{min})\Delta_b}{b_{max} - b_{min}} \tag{7.6}$$

Using the value of Δ_v of equation (7.6) in equation (7.3), 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.

8 Conclusion

In this paper, we proposed a way to integrate continuous aspects of complex specifications into a discretized Petri Net model. Our approach was studied in the context of Intelligent Transport Systems and, more precisely, management of emergency braking when an obstacle is identified on the road. An application to this case study is provided.

This discretization method relies on the use of equations modelling the problem. Such equations come from the physical models that interact with the system. We attach these equations to a CPN template and then proceed to its transformation in order to be able to have an analyzable model (i.e. that remains finite).

The equations modeling the problem are used to:

- Provide a discretized abstraction
- To evaluate the quality of this abstraction with regards to the proof of properties on the resulting model.

This is a key point in modeling and evaluating a system by means of formal specification. It is crucial for engineers to evaluate the quality of the proven properties and, if necessary when assumptions are done (here, they come from the discretization), to evaluate their impact on the system's properties. Typically, imprecision raised by discretization may have to be corrected by either applying a more precise discretization or adding constants in formulas expressing properties to be checked.

In our paper, discretization is applied on symmetric Nets deduced from CPN since our tools rely on symmetric nets. Of course, it is also valid on the CPN models.

In our methodology, different discretization algorithms can be applied. We used in this paper a simple algorithm as a first approach but other ones based on non-uniform discretization intervals are promising alternatives. This will introduce new constraints in formal verification and in error propagation computation but it is a interesting field for future works.

Also, managing more than one module is of interest. In the context of a SAFESPOT application, several modules run in parallels and may introduce more continuous types and variables. Future work will then have to evaluate how a larger number of variable (and constraints) could be managed. In particular, experimenting,

[2]We intentionally removed the $\pm$ operator to increase readability.
[3]Note that it is possible to choose another factor between $Card(V)$ and $Card(B)$ as explained section 7.1

propagation of discretization contraints between different modules need a particular attention.

Acknowledgements: We would like to thank the anonymous referees fro their careful reading and helpful comments.

References

[1] J-R. Abrial. *The B book - Assigning Programs to meanings*. Cambridge Univ. Press, 1996.

[2] 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*, Beijing, China, October 2007.

[3] 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, Seattle, USA, September 2007. IEEE Press.

[4] R. Brignolo. Co-operative road safety - the SAFESPOT integrated project. In *APSN - APROSYS Conference*. Advanced Passive Safety Network, May 2006.

[5] Didier Buchs and Nicolas Guelfi. A formal specification framework for object-oriented distributed systems. *IEEE Trans. Software Eng.*, 26(7):635–652, 2000.

[6] R. Brown C. Covault and D. Driscoll. *Uncertainties and Error Propagation - Appendix V of Physics Lab Manual*. Case Western Reserve University, 2005.

[7] 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.

[8] 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.

[9] Giovanni Chiola, Claude Dutheillet, Giuliana Franceschinis, and Serge Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Trans. Computers*, 42(11):1343–1360, 1993.

[10] P. Christofides and N. El-Farra. *Control Nonlinear And Hybrid Process Systems: Designs for Uncertainty, Constraints And Time-delays*. SPringer Verlag, 2005.

[11] 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`.

[12] J-M. Couvreur and Y. Thierry-Mieg. Hierarchical Decision Diagrams to Exploit Model Structure. *Formal Techniques for Networked and Distributed Systems - FORTE 2005*, pages 443–457, 2005.

[13] The CPN Tools Homepage, 2007. http://www.daimi.au.dk/CPNtools.

[14] S. Dashtinezhad, T. Nadeem, B. Dorohonceanu, C. Borcea, P. Kang, and L. Iftode. TrafficView: A Driver Assistant Device for Traffic Monitoring based on Car-to-Car Communication. In IEEE Computer Press, editor, *IEEE Semiannual Vehicular Technology Conference*, 2004.

[15] René David and Hassane Alla. On Hybrid Petri Nets. *Discrete Event Dynamic Systems: Theory and Applications*, 11(1-2):9–40, 2001.

[16] M. Doche, I. Vernier-Mounier, and F. Kordon. A modular approach to the specification and validation of an electrical flight control system. In *Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*, pages 590–610. Springer-Verlag, 2001.

[17] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *International Conference on Machine Learning*, pages 194–202, 1995.

[18] E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.*, 30(1):1–24, 1985.

[19] IEEE 802.11 Working Group for WLAN Standards. *IEEE 802.11 tm Wireless Local Arae Networks*. IEEE, 2008.

[20] Frame Forum. The FRAME forum home page, `http://www.frame-online.net`.

[21] L. Hillah, F. Kordon, L. Petrucci, and N. Trèves. PN standardisation : a survey. In *International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06)*, pages 307–322, Paris, France, September 2006. IFIP.

[22] J. Hugues, Y. Thierry-Mieg, F. Kordon, L. Pautet, S. Baarir, and T. Vergnaud. On the Formal Verification of Middleware Behavioral Properties. In *9th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'04)*, pages 139–157. Elsevier, September 2004.

[23] 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.

[24] D.Luca J.Daniel. 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.

[25] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 1, vol. 2 et vol. 3*. Springer-Verlag, London, UK, 1995.

[26] Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets, Modelling and Validation of Concurrent Systems*. Monograph to be published by Springer Verlag, 2008.

[27] F. Kordon, A. Linard, and E. Paviot-Adet. Optimized Colored Nets Unfolding. In *International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06)*, volume 4229 of *LNCS*, pages 339–355, Paris, France, September 2006. Springer Verlag.

[28] Charles Lakos and Glenn Lewis. Incremental state space construction of coloured Petri nets. In *Proc. 22nd Int. Conf. Application and Theory of Petri Nets (ICATPN'01)*, volume 2075 of *Lecture Notes in Computer Science*, pages 263–282. Springer, 2001.

[29] Glenn Lewis. *Incremental specification and analysis in the context of coloured Petri nets*. PhD thesis, University of Hobart,Tasmania, 2002.

[30] Vern Lindberg. *Uncertainties and Error Propagation - Part I of a manual on Uncertainties, Graphing, and the Vernier Caliper*. Rochester Institute of Technology, 2000.

[31] LIP6/MoVe. The CPN-AMI home page, `http://www.lip6.fr/cpn-ami/`.

[32] K. Varpaaniemi, J. Halme, K. Hiekkanen, and T. Pyssysalo. Prod reference manual. Technical report, Helsinki University of Technology, 1995.

[33] Jacques 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, June 1986*, pages 293–308, London, UK, 1987. Springer-Verlag.

[34] ISO TC204 WG-16. *CALM architecture*. ISO, 2007.