In Proc. 2nd National Workshop on Control Architectures of Robots (CAR'07), Paris, France, may 2007.

A formal approach to designing autonomous systems: from Intelligent Transport Systems to Autonomous Robots

Fabrice KORDON[1] and Laure PETRUCCI[2]

[1] Université Pierre & Marie Curie, Laboratoire d'Informatique de Paris 6/MoVe
4, place Jussieu, F-75252 Paris CEDEX 05, France
fabrice.kordon@lip6.fr
[2] LIPN, CNRS UMR 7030, Université Paris 13
99, avenue Jean-Baptiste Clément, F-93430 Villetaneuse, France
laure.petrucci@lipn.univ-paris13.fr

Abstract. Emerging transport systems involve more and more fully automatic parts that communicate together in order to optimise traffic and security. Such systems are highly distributed, mobile and require physical constraints to be taken into account. The communicating entities may be included in vehicles or the infrastructure ; they must comply with real time and real space constraints ; they should also have some autonomous behaviour in case of e.g. network failure. The systems designed should be proved reliable before being put in operation. We propose to use formal specification and verification techniques for designing these models, prior to any costly hardware implementation.

1 Introduction

Emerging transport systems involve more and more fully automatic parts that communicate together in order to optimise traffic and security. Such systems are highly distributed, mobile and require physical constraints to be taken into account. The communicating entities may be embedded in vehicles or included in the infrastructure.

In such systems, distribution leads to a huge complexity and a strong need to deduce possible (good and bad) behaviours on the global system, from those known of its actors. Moreover, at least part of these systems is embedded with intrinsic real-time and real-space constraints. They are due to the critical, highly reactive environment where both timing and positionning are critical issues.

For such systems, we know that classical development methods are not adequate since the coverage of possible executions is too low [GL97]. This observation leads to investigate the use of formal methods. However, these still lack user-friendly languages and tools that can enable their use by non-specialists. Hence, even though major actors in companies or institutions dealing with critical applications acknowledge the necessity of using formal methods, they also agree on the fact they should be able to scale up: today, only parts of systems are formally analysed.

Up to now, two main types of formal methods are available: *algebraic* approaches and *model checking*. Algebraic approaches such as B [Abr96] allow for describing a system with axioms and then proving a property on the specification as a theorem to be demonstrated from these axioms. These methods are very interesting since the proof is parameterised. However, theorem provers that are required to elaborate the proof are difficult to use and still require highly skilled and experienced engineers.

In contrast, model checking [CGP00,BBF$^+$01] is the exhaustive investigation of a system state space and can be automated very easily. This technique is theoretically limited by the combinatorial explosion and can mainly address finite systems. However, recent symbolic techniques[3] scale up to more complex systems.

Thus, since formal verification techniques are getting more mature, our capability to build even more complex systems also grows quickly. To catch up with problems complexity and get significant results with formal analysis, we must cope with the complexity at every stage of the process: from the specification phase to the verification itself. The methodology to be applied makes a *pragmatic* use of formal methods, i.e. assumptions simplifying the system under study should be made, which are usually domain specific. Hence, variations of the traditional (unified) process development approaches are necessary.

This paper proposes to tackle the design methodology and techniques that can be applied in order to handle very large systems throughout the modelling and verification processes. Such techniques are here concerned with Intelligent Transport Systems (ITS), i.e. mechanisms which provide driving assistance to a vehicle. This application domain is particularly representative of tomorrow distributed systems including real-time and real-space features, for which traditional programming approaches cannot guarantee the required security, and must thus be adapted.

The paper is structured as follow. Section 2 presents ITS concerns and the related verification problems. Then, section 3 describes the formal notations that we shall use to model such systems. A design methodology is sketched in section 4. Section 5 addresses verification using appropriate model checkers. Finaly, section 6 shows the parallel between Intelligent Transport Systems and autonomous robots.

2 Intelligent Transport Systems

Intelligent Transport Systems (ITS) are highly critical since a failure can lead to dramatic consequences such as fatal accidents. They also involve a significant number of partners that must thus cooperate in an efficient and secure manner. The agents of such a system are road operators, infrastructure, vehicles and their drivers. Some of these might be equipped with active embedded software while the others travel in the usual fashion. Their reaction is then unpredictable and it is essential to obtain relevant and often updated information from captors in order to take them into account.

Development of ITS is a challenge supported by research programs in Europe, USA and Japan [Bis05].

In this section, some ITS issues are first illustrated through a simple example. Then, major problems encountered during the formal specification are discussed.

[3] The word *symbolic* is associated with two different techniques. The first one is based on state space encoding and was introduced in [BCM92]. The second one relies on set-based representations of states having similar structures and was introduced in [CDFH91].

2.1 ITS Example: Safe Insertion in a Motorway

A typical example of an ITS problem is the so-called *black-spot*, which is a dangerous section in the motorway. It is basically a freeway entrance in which *safe insertion* should be guaranteed. Figure 1 represents a motorway with two lanes: L_1 (right lane) and L_2 (left lane). An entrance to the motorway, L_0, is connected to L_1. Vehicles already on the motorway use both lanes L_1 and L_2. Vehicles are supposed to carry an identity which is a number. The notation $V_{i,j}$ indicates that vehicle j is circulating on lane number i. We aim at studying a cooperative insertion of vehicles arriving in the entrance lane L_0.

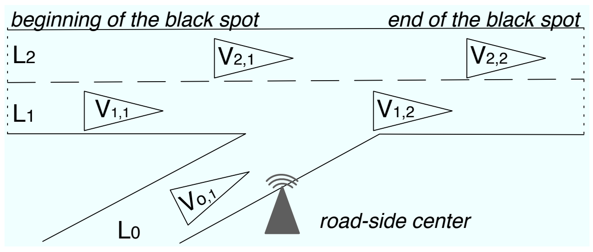


Fig. 1. Safe Insertion in a motorway.

The vehicles on the insertion lane L_0 must enter the motorway without violating the following properties:

1. the distance between two vehicles in the same lane must be greater than a minimum safe distance to let drivers react to sudden and maybe unexpected events;
2. $V_{0,j}$ vehicles must eventually get into the motorway;
3. $V_{i,j}$ vehicles should not have to stop.

We propose the following strategy to ensure a safe motorway entrance:

(a) The motorway has a *road-side center* (RSC) enabling communication with vehicles and which can compute commands related to safety or flow control.
(b) Vehicles receive their positions using satellite localisation technology [Blo05] which may also be combined with ground installations and digitised maps). They periodically their position to the infrastructure. Subsequently, the infrastructure maintains a dynamic map of all vehicles in its communication range.
(c) The infrastructure, vehicles behaviour and interactions operate the following interaction cycle:
    (i) vehicles get their position;
    (ii) they send this information to the infrastructure;
    (iii) when the infrastructure has received all vehicles positions, it issues commands w.r.t. a predefined strategy.

In order to simplify the problem, we assume that all vehicles are equipped with communication devices and drivers follow the instructions issued by the road-side center. Vehicles without the embedded equipment are considered and modelled differently.

2.2 Modelling Issues for ITS

For such critical and reactive systems, both *quantitative* and *qualitative* properties should be ensured. *Quantitative* properties express performance requirements while *qualitative* properties allow for checking whether a faulty behaviour may occur.

Modelling and verifying such systems require:

- managing dynamic actors such as cars that enter and leave the black-spot;
- modelling of physical aspects;
- preserving a fair progression of the system so that actors perform actions at a similar pace.

A qualitative analysis is a first step in the development process: it will ensure a global correct functionning. Then, the specification can be refined so as to include timing or hybrid features, reflecting the actual real-time and real-space behaviour. Quantitative analysis will determine whether the envisionned strategies satisfy physical constraints. In this paper, we will focus on the qualitative aspects.

2.3 Specification Issues

The first issue consists in *selecting an appropriate specification formalism* [HKP06]. Of course, the formalism should be able to capture the relevant aspects of the problem under study, and allow for the necessary verifications. Moreover, a *design methodology* will prove useful, to carry out verifications step by step on more amenable models.

The choice of a specification formalism and the design methodology are of utmost importance for verification to be as successful as possible.

A very popular candidate for specification is UML [COTM05]. Even though, it is useful for structuring a system and having a better view of the interactions between components, UML is not suitable for formal analysis of the system behaviour. Normalisation efforts tends to counter this problem by giving a more precise semantics, but the connection between diagrams is still too loose and leads to various interpretations.

Algebraic techniques such as B can be useful for the verification of behavioural components as Siemens proved in the METEOR project [B]. However, it was also known as a difficult technique to automate compared to model checking based approaches. Thus, this latter type of techniques seems better suited to provide more automated tools.

Many tools allow for model checking. They address different kinds of models. In a first approach, we will focus on the verification of behavioural properties of the system, i.e. qualitative analysis. Thus, we will be able to check that the chosen strategies are relevant, indepently of real-time or real-space constraints. Therefore, a simple model is selected, which is powerful enough for our modelling purposes and provides up-to-date efficient analysis techniques, namely *Symmetric Nets*[4]. The CPN-AMI tool [cpn] constitutes a complete spefication and verification environment for symmetric nets.

[4] *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 [HKPT06].

In a further step of our modelling and verification process, the models will be enhanced so as to capture the real-time and real-space aspects, i.e. perform quantitative analysis. Model checking tools can handle time (e.g. UPPAAL [upp], based on timed automata; or TINA [tin] based on Timed Petri Nets) or hybrid constructs (e.g. HYTECH [hyt]). It will then be necessary to design a methodology which guarantees a compatibility between the models designed for qualitative analysis and those derived for quantitative analysis purposes.

The remainder of this paper will on behavioural properties. Therefore, section 3 presents the symmetric nets formalism.

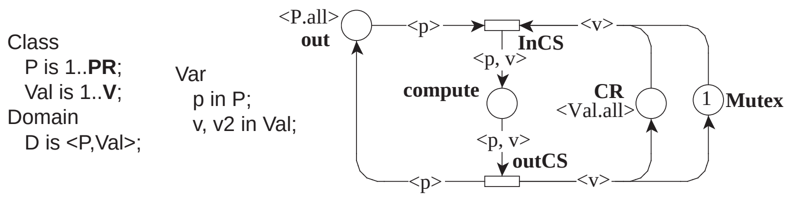
3 Symmetric Nets

This section provides an informal presentation of Symmetric Nets as well as associated analysis techniques. Formal definitions can be found in [CDFH91,GV03].

3.1 Formalism Basic Features

Symmetric Nets are Petri nets enhanced with high-level features: tokens can carry data. Therefore, a data type is associated with each place, indicating the data type for tokens sitting in that place. In contrast with Coloured Petri Nets [Jen92], only simple data and manipulation functions are permitted, allowing for powerful analysis techniques. Finite enumerated types, intervals, tuples are allowed and the basic functions are predecessor, successor, selector (in a tuple) and “broadcast”. The latter function allows for generating one copy of each possible value in the data type. This is very convenient for e.g. modelling network protocols where a station sends messages to all other stations on the network.

Let us now illustrate Symmetric Nets (SN) by means of a small example. The Petri net in figure 2 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 parameters for the system.


```
Class
  P is 1..PR;
  Val is 1..V;
Domain
  D is <P,Val>;

Var
  p in P;
  v, v2 in Val;
```

Fig. 2. Example of a Symmetric Net.

The class of threads is represented by places **out** (typed after *P*) and **compute**. Place **compute** (typed after $D = P \times Val$) corresponds to some computation on the basis of the value provided by **CR**. At this stage, each thread holds a value that is given back when the calculus is finished. Place **Mutex** handles mutual exclusion between threads. Place **out** initially holds one token for each value in P (the marking is then noted $< P.all >$) and place **CR** holds one value for each value in Val (marking $< Val.All >$). Place **Mutex** only contains one token with no value (like a black token in place/transition nets).

Transitions represents evolution of the system. A transition is fired when all precondition places hold a sufficient marking. For example, Transition **inCS** can be fired if there is one token in **out**, one token in **CR** and one token in **Mutex**. When it fires, variables p and v are bound to the values of tokens from place **out** and **CR** respectively (place **Mutex** has no type, hence its tokens do not carry any data). When this transition is fired, a token carrying the value of pair $< p, v >$ is created in the postcondition place **compute**.

3.2 From Symmetric Nets to Place/Transition Nets

A symmetric net can easily be *unfolded* into an equivalent place/transition net.

A SN-place is transformed into a set of PTN-places, one per possible value. The place/transition net in figure 3 is the unfolding of the symmetric net in figure 2 with $P = [1..2]$ and $Val = [1..2]$.

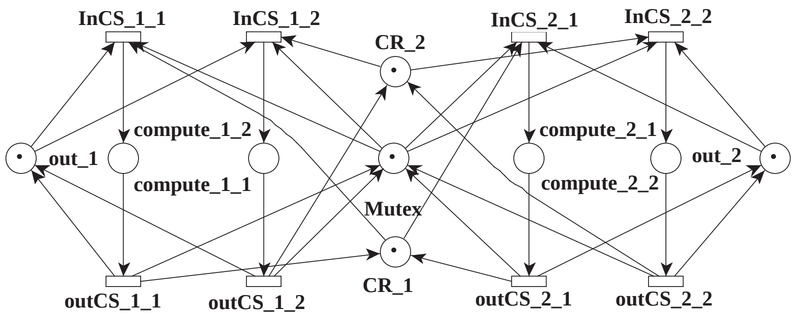


Fig. 3. Unfolded P/T Net from figure 2.

Even large models can be handled, using decision diagram based techniques [KLPA06].

Then, structural properties of the model can be computed from the unfolded net. They are formulas that can be computed without exploring the full state space [GV03], and hold independently of the initial marking (in fact, it often intervenes in a constant value only).

3.3 Symbolic Reachability Graph

One of the main analysis techniques is based on the *state space* (also called *reachability graph*) exploration. It represents all concrete states and possible evolutions of the system. Figure 4 presents the reachability graph for the Petri net of figure 2 with constants

PR and **V** equal to 2. This state space has 5 states (the initial state is represented by a double circle). When increasing the number of possible values, the size of the state space will grow following the cardinality of the cartesian product $P \times Val$.

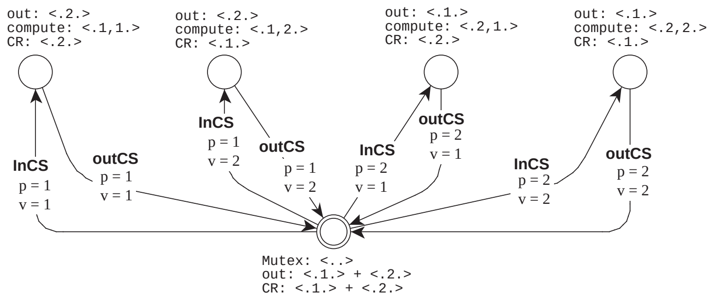


Fig. 4. Reachability Graph of the Net in Figure 2.

Similar to symmetric nets which are a compact representation of a system, the *symbolic reachability graph* is a condensed reprsentation of the states in the system, particularly adapted to the analysis of symmetric nets. A state in the symbolic reachability graph does not represent a concrete state but a set of concrete states that have a similar structure. The symbolic reachability graph of our example is depicted in figure 5. It is composed of only two nodes and does not grow when the types *P* and *Val* allow for more values.

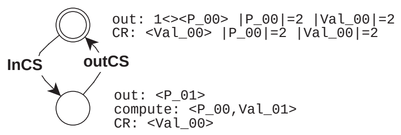

out: 1<><P_00> |P_00|=2 |Val_00|=2
CR: <Val_00> |P_00|=2 |Val_00|=2
InCS
outCS
out: <P_01>
compute: <P_00,Val_01>
CR: <Val_00>

Fig. 5. Symbolic Reachability Graph of the Net in Figure 2.

The definition of states in figure 5 must be read as follow. In the initial state, all possible values in type *P* are stored in place **out** and all possible values in type *Val* are stored in place **CR**. In the other state (when transition **InCS** fires), all possible values of type *P* but one are in place **out** and all possible values of type *Val* but one are in place **CR**. Place **compute** then contains one token composed with one value of type *P* (the one that is not in place **out**) and one value of type *Val* (the one that is not in place **CR**). Thus, this symbolic state represents all possible permutations of the pair of tokens extracted from places **out** and **CR** when firing transition **InCS**.

This symbolic technique is thus based on computed symmetries in the net [TMDM03]. It is successful when representing very large state spaces: there is an exponential gain w.r.t. the construction of concrete states [HTMK$^+$04]. This set-based representation is very efficient, especially when systems are symmetric, which is the case in numerous distributed and embedded systems.

Such a technique is well-suited for analysing Intelligent Transport Systems since they present intrinsic symmetries as similar algorithms are supposed to be executed in each car.

4 Modelling Methodology for ITSs

Considering that the specification formalism for Intelligent Transport Systems is Symmetric nets, we now aim at a suitable dedicated design method. Therefore, we will consider the techniques that are optimal for this kind of systems.

4.1 Model Components and Abstraction Level

The whole ITS system to be modelled involves several components that interact via several mechanisms. The communication can either be asynchronous (and is then modelled by *place fusion*) or synchronous (corresponding to *transition fusion*).

It is thus necessary to enhance the symmetric nets formalism with both of these communication mechanisms. They provide further advantages. In particular, several designs of a same component can be tested without changing the models of the other components, provided that all these designs communicate in the same manner with the rest of the system. But they can operate different strategies or configurations.

This approach leads to a hierarchy of modules, which can have as many levels as necessary. Analysing the system then starts by “flattening” the whole model into a single symmetric net. It is also possible to take advantage of modular analysis techniques such as [LP04].

Since the systems under study are complex, we also consider using refinement of modules [LL01]. This approach allows for first designing an abstract model, analysing it, and then include additional details in a consistent manner (so as to keep the analysis results as much as possible). This process is repeated until the desired abstraction level is obtained. A similar modelling approach will lead to include the real-time and real-space features once the global functionning of the system is proved correct.

Finally, the specific problems identified in section 2.2 must be addressed.

4.2 Managing Dynamic Actors

The vehicles involved in the black-spot are dynamic: they can enter and leave the motorway zone nder study. A natural way of modelling this aspect would be to create new vehicles getting in and discarding vehicles getting out.

However, such an approach is not suitable for several reasons. First, that would mean associating a new number with each new vehicle. The chosen formalism of symmetric nets permits only finite types, thus having a arbitrary high numbering of vehicles is not feasible. Moreover, let us assume that there is a maximum numbering of vehicles. The states space of the system will be uselessly large when taking into account many different vehicle numbers.

An approach to managing this dynamic aspect is then to consider that there can be a limited number of vehicles on the black-spot. This is a reasonnable assumption since the cars and motorway lengths are a physical limit. Moreover, the identity of vehicles has no consequence on the functionning of the system. They must still be distinguished to consider e.g. different positionning or driving strategies. Hence, when a vehicle gets out of the system, its number can be reused by a new vehicle.

This technique also brings forward an interesting feature. The corresponding system is not expected to deadlock. Thus, a deadlock may correspond either to a property violation or to some mistake in the model itself.

4.3 Modelling Complex Functions

As mentionned in section 3.1, symmetric nets only offer a limited set of mathematical functions to the system designer. This is required for keeping the mathematical structure that enables the computation of symmetries in the specification, necessary for using the symbolic reachability graph [TMDM03].

To cope with the modelling of complex functions (for example, computation of braking distance according to the current speed of a vehicle), they can be discretised and represented in a dedicated place of the Petri net. This approach is similar to sampling and can be applied to arbitrarily complex functions, deterministic or not. However, the discretisation of a function becomes a modeling hypothesis and must be validated separately (to evaluate the accuracy of the sampling).

The main drawback of this technique is a loss in precision compared to continuous systems that require appropriate hybrid techniques [CEF05]. If such a discretisation enables the use of more user-friendly techniques, they must be checked. For example, if we consider distances in our black-spot example, we must ensure that uncertainty remains in a safe range. This means that our metrics must be compliant with the precision to ensure, for example, that if $V_{1,1}$ follows $V_{1,2}$, the minimum distance guarantees that no intersection between the associated volumes is possible.

Using the discretisation technique may be a preliminary to putting in operation more complex formalisms such as timed or hybrid ones which will allow for quantitative analysis of continouous models.

4.4 Fair Execution among System Components

The different actors in an ITS behave in parallel. Their actions should evolve at a similar pace. This aspect is not guaranteed by Petri net behaviour unless appropriate mechanisms are set. To avoid the progress of one component while the other components are stopped, several techniques can be used:

– the addition of a timeline, as in Timed Petri nets [Jen92] changes the firing conditions so that time can advance only if there is no enabled transition at the current time anymore;

– the state space construction could include a branching option that would discard unsuitable sequences.

In the black-spot example, at each time slot, all vehicles should make a move and the infrastructure take decisions, thus following the execution cycle described in section 2.1.

5 Towards Analysis of Intelligent Transport Systems

As mentionned in section 4.2, ITSs include many components, namely the vehicles, which have a similar behaviour. The vehicles identifiers are only used to track them within the system. Therefore, this kind of system is highly symmetric: the diffrent vehicles play the same role. Thus, the use of symmetric nets and the associated analysis technique, i.e. the symbolic reachability graph (see section 3.3), is relevant.

Nevertheless, analysis remains quite difficult since currently implemented model checkers are not sufficient. The ones that implements a concrete state space cannot handle more than a few 10^8 states.

GreatSPN [gsp], a model checker implementing the symbolic reachability graph was successfully used to analyse a middleware core having approximately 10^{18} concrete states [HTMK$^+$04], but it seems inadequate for the complexity of ITS systems when discretisation is realistic and requires types with many values (in [BHKF06], only small configurations could be analysed). Model checkers supporting a symbolic encoding of the state space such as SVM [smv] present the same drawbacks.

A close examination of model checkers behaviour shows that current techniques cannot scale up for these systems yet.

However, model checkers use new techniques that are promising for analysing ITSs:

– symbolic/symbolic techniques;
– distributed model checkers running on clusters of machines;
– handling stable markings;
– hierarchical encoding and modular techniques.

In the following subsections, we will sketch these.

5.1 Symbolic/Symbolic Techniques

The symbolic reachability graph, as described in section 3.3, allows for mastering the complexity of large state spaces, similar to the encoding of states using decision diagrams [BCM92] (also called *symbolic techniques*). The term *symbolic* can thus have sevral meanings. It can relate to:

– grouping of similar states represented by a single abstract one;
– adequate encoding of the states to have a better use of computer memory.

To illustrate the state encoding techniques, let us consider that a state in the system is represented as a boolean vector defining the values of a set of variables. An action in the system usually changes only part of the system state. Hence, we can consider a differential encoding of states. It is not necessary to encode once more the value of the unchanged variables. Binary Decision Diagrams (BDDs) promote sharinf of common parts in the system. The main drawback of this technique is that its efficiency is strongly related to the ordering of variables.

BDDs are dedicated to systems using booleans, but many decision diagrams based techniques were introduced so as to capture more elaborate models. Among them, Data Decision Diagrams (DDDs) [CEPA$^+$02] encode discrete values instead of binary ones. It is a basis to support symbolic/symbolic techniques [TMIP04] that combine *symbolic encoding* and *symbolic reachability graph*. Experiments show that this technique is promising for the storage of very large state spaces.

5.2 Distributed Model Checking

The main problem of model checking is memory consumption. However, with diagram decision based techniques, another problem arises. The principle of these techniques is to trade memory against CPU. As a typical example, when a new symbolic state is computed, it has to be compared with the existing ones. This requires all states to be canonised in order to have a common and comparable representation.

So, distributing a model checker on a cluster of machines has advantages [KP04]:

– states are generated in parallel using a hash function which distributes states on machines;
– it takes advantage of the CPU and memory available in the whole system.

Initial results are promising and this is a currently active research area. SPIN model checker has already been experimented in a parallel setting [LS99,BFLW05]. A distributed version of GreatSPN has been recently implemented, which provides a supra-linear acceleration factor for many examples [HKTM07]. However, even though the distributed generation of the state space has been implemented, analysing properties on those is still to be developped further.

5.3 Management of Stable Marking

The discretisation technique presented in section 4.3 generates places with a large marking which remains constant. Most model checkers do not handle such cases, and the stable marking is represented once per generated state, leading to a huge and useless memory consumption.

Model checkers using a symbolic encoding of states, such places should be detected since their marking is highly shared by all states in the system. An *a priori* analysis of the specification can easily detect such configuration and provide hints for a more appropriate encoding technique.

5.4 Hierarchical Encoding and Modular Techniques

Symbolic encoding of a state space (concrete or symbolic) relies on the sharing of state patterns in the state space of a system. Recent work investigates a hierarchical representation that could increase the sharing of such patterns on a larger scale. For that purpose, new representations, such as Set Decision Diagrams (SDD) [CTM05] are being investigated. In favourable cases (i.e. when the system exhibits very regular symmetries), the results are impressive. E.g. it is possible, using a recursive folding of the dining philosophers problem [phi], to store the state space for to 2^{10000} philosophers within 512 Mbytes of memory, as reported in [TM04]. The intrinsic high symmetry in ITSs should allow for similar results.

An additional and complementary investigation axis is modular analysis [LP04]. Since the methodology applied for specifying large systems includes design by component, this kind of analysis techniques is suitable. The idea consists in representing the state space using not a single graph, but several: one per component, representing the component local behaviour and local states, plus a synchronisation graph which explicits the synchronisations between the different components and the global states to achieve those.

Combining modular and symbolic techniques should permit to handle very large systems such as ITSs.

6 Towards Autonomous Robots

Autonomous robots are meant to evolve within an environment which may have unexpected behaviour. This can be due to e.g. unknown terrain to explore, other agents (e.g. robots handled by another system, animals), . . . ‘Moreover, the human control on such robots is rather minor. In order for autonomous robots to take into account the characteristics of the environment, it is necessary for them to get information via sensors. This allows for having an abstract vision of the current situation. Thus, as for ITSs, modelling autonomous robots requires both real-time and real-space concerns to be taken care of.

Thus, the approach developped here for ITSs can be used to tackle autonomous robots verification as well. The analogy could be relating the robots themselves to the vehicles and the interacting humans to the infrastructure. The map of the environment for ITSs is rather simple and it may be more complex for robots. However, there is often an intended route which can initially be modelled using simple data and symmetric nets and can later be further refined when taking into account the space and time aspects.

7 Conclusion

Emerging transport systems involve more and more fully automatic parts that communicate together in order to optimise traffic and security. Such systems are highly distributed, mobile and require physical constraints to be taken into account. The communicating entities may be included in vehicles or the infrastructure ; they must comply with real time and real space constraints ; they should also have some autonomous behaviour in case of e.g. network failure. The systems designed should be proved reliable before being put in operation.

In this paper, we have shown how the design methodologies and current analysis techniques can handle such very large systems. The different approaches have their own advantages and are complementary. Therefore, our efforts will focus on their combination. An obvious necessity emerging from preliminary analysis is to consider design and analysis issues in parallel so as to capture and handle the relevant problems in a consistent and efficient manner. In particular, domain specific features have to be taken into account at a very early stage.

References

[Abr96] J-R. Abrial. *The B book - Assigning Programs to meanings*. Cambridge University Press, 1996.

[B] *Atelier B*.

[BBF$^+$01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BCM92] J.R. Burch, E.M. Clarke, and K.L. McMillan. Symbolic model checking: 10^{20} states and beyond. *Information and Computation (Special issue for best papers from LICS90)*, 98(2):153–181, 1992.

[BFLW05] J. Barnat, V. Forejt, M. Leucker, and Michael Weber. DivSPIN - a SPIN compatible distributed model checker. In M. Leucker and J. van de Pol, editors, *4th International Workshop on Parallel and Distributed Methods in verifiCation (PDMC'05)*, Lisbon, Portuga, 2005.

[BHKF06] F. Bonnefoi, L. Hillah, F. Kordon, and G. Frémont. An approach to model variations of a scenario: Application to Intelligent Transport Systems. In *Workshop on Modelling of Objects, Components, and Agents (MOCA'06)*, Turku, Finland, June 2006.

[Bis05] R. Bishop. Intelligent Vehicle R&D: a review and contrast of programs worldwide and emerging trends. *Annals of Telecommunications - Intelligent Transportation Systems*, 60(3-4):228–263, March-April 2005.

[Blo05] J-M. Blosseville. Driving assistance systems and road safety: State-of-the-art and outlook. *Annals of Telecommunications - Intelligent Transportation Systems*, 60(3-4):281–298, March-April 2005.

[CDFH91] Giovanni Chiola, Claude Dutheillet, Giuliana Franceschinis, and Serge Haddad. On well-formed coloured nets and their symbolic reachability graph. In Kurt Jensen and Grzegorz Rozenberg, editors, *Procedings of the 11th International Conference on Application and Theory of Petri Nets (ICATPN'90). Reprinted in High-Level Petri Nets, Theory and Application.* Springer, 1991.

[CEF05] P. Christofides and N. El-Farra. *Control Nonlinear And Hybrid Process Systems: Designs for Uncertainty, Constraints And Time-delays*. Springer, 2005.

[CEPA$^+$02] J.-M. Couvreur, E. Encrenaz, E. Paviot-Adet, D. Poitrenaud, and P.-A. Wacrenier. Data decision diagrams for Petri net analysis. In *Proc. of ICATPN'2002*, volume 2360 of *Lecture Notes in Computer Science*, pages 101–120. Springer, June 2002.

[CGP00] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[COTM05] B. Charroux, A. Osmani, and Y. Thierry-Mieg. *UML 2*. Pearson Education, 2005.

[cpn] *The CPN-AMI Home page*. `http://www.lip6.fr/cpn-ami`.

[CTM05] J-M. Couvreur and Y. Thierry-Mieg. Hierarchical decision diagrams to exploit model structure. In *25th International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'05)*, Lecture Notes in Computer Science. Springer, October 2005.

[GL97] J. Gogen and Luqi. Formal methods: Promises and problems. *IEEE Software*, 14(1):75–85, 1997.

[gsp] *GreatSPN V2.0*. `http://www.di.unito.it/~greatspn/index.html`.

[GV03] C. Girault and R. Valk. *Petri Nets for Systems Engineering*. Springer Verlag - ISBN: 3-540-41217-4, 2003.

[HKP06] S. Haddad, F. Kordon, and L. Petrucci. *Méthodes formelles pour les systèmes répartis et coopératifs*. Hermès, November 2006.

[HKPT06] 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.

[HKTM07] A. Hamez, F. Kordon, and Y. Thierry-Mieg. libDMC: a Library to Operate Efficient Distributed Model checking. Technical report, Master's thesis, LIP6, Université P. & M. Curie, 2007.

[HTMK$^+$04] 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.

[hyt] HYTECH *homepage*. `http://www-cad.eecs.berkeley.edu/~tah/HyTech/`.

[Jen92] K. Jensen. *Coloured Petri Nets: Basic concepts, analysis methods and practical use. Volume 1: basic concepts*. Monographs in Theoretical Computer Science. Springer, 1992.

[KLPA06] 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)*, pages 339–355, Paris, France, September 2006. IFIP.

[KP04] L. Kristensen and L. Petrucci. An approach to distributed state space exploration for coloured Petri nets. In *Proc. 25th Int. Conf. Application and Theory of Petri Nets (ICATPN'2004), Bologna, Italy, June 2004*, volume 3099 of *Proc. 25th Int. Conf. Application and Theory of Petri Nets (ICATPN'2004), Bologna, Italy, June 2004*, pages 474–483. Springer, June 2004.

[LL01] C. Lakos and G. Lewis. Incremental state space construction of coloured Petri nets. In *Proc. 22nd Int. Conf. Application and Theory of Petri Nets (ICATPN'01), Newcastle, UK, June 2001*, volume 2075 of *Lecture Notes in Computer Science*, pages 263–282. Springer, 2001.

[LP04] C. Lakos and L. Petrucci. Modular analysis of systems composed of semiautonomous subsystems. In *Proc. 4th Int. Conf. on Application of Concurrency to System Design (ACSD'04), Hamilton, Canada, June 2004*, pages 185–194. IEEE Comp. Soc. Press, June 2004.

[LS99] F. Lerda and R. Sisto. Distributed-memory model checking with SPIN. In *Proc. of the 5th International SPIN Workshop*, volume 1680 of *Lecture Notes in Computer Science*. Springer, 1999.

[phi] *Dining philosophers problem*. `http://en.wikipedia.org/wiki/Dining_philosophers_problem`.

[smv] *The SMV System*. `http://www.cs.cmu.edu/~modelcheck/smv.html`.

[tin] *TINA, TIme Petri Net Analyzer*. `http://www.laas.fr/tina`.

[TM04] Y. Thierry-Mieg. *Techniques for the model checking of high-level specifications*. PhD thesis, Université P. & M. Curie, 2004.

[TMDM03] Y. Thierry-Mieg, C. Dutheillet, and I. Mounier. Automatic symmetry detection in well-formed nets. In *Proc. of ICATPN 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 82–101. Springer, June 2003.

[TMIP04] Y. Thierry-Mieg, J-M. Ilié, and D. Poitrenaud. A symbolic symbolic state space representation. In *24th International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'04)*, volume 3235 of *Lecture Notes in Computer Science*, pages 276–291. Springer, July 2004.

[upp] *UPPAAL home page*. `http://www.uppaal.com/`.