Design, modeling and analysis of ITS using UML and Petri Nets

Fabien Bonnefoi and Lom Messan Hillah and Fabrice Kordon and Xavier Renault

***Abstract*— This paper is about the application of formal methods to model and analyze complex systems in the context of Intelligent Transport Systems (ITS). It suggests a specification methodology based on a set of UML diagrams to generate a complete analyzable formal model. The methodology integrates the requirements of incremental and modular development for complex systems. The analysis made on the formal model is carried out through qualitative criteria, verified by model checking tools. The proposed guidelines are illustrated by a case study which considers cars in traffic situations, exchanging information about their states to reach consistency among their driving decisions.**

I. INTRODUCTION

Transportation networks are still subject to congestion and safety problems. There is a need to improve quality, efficiency and safety on transport systems. In that context, application of formal methods on the design, modeling and analysis of Intelligent Transport Systems (ITS) can provide an appropriate framework for those improvements.

ITS development is aiming at full cooperation between vehicles and infrastructures so as to take advantage of the detection and control capabilities of the vehicle ad-hoc network and the centralized infrastructure support. There is a high number of involved entities and various concurrent strategies and decisions are implemented at different levels. Therefore ITS are very complex and often imply safety-critical applications.

Such safety and complexity require a particular attention for ITS design that ”traditional” development processes sometimes are not fully adapted for. At the specification and design stages, non-formal approaches such as text documents and nonstandard diagrams are often used. Thus, in such a way, analysis and verification of the specifications is nearly impossible. We believe that formal methods could be used to master some key consistency and safety requirements. Also, the dynamics of ITS evolution requires a modular and flexible design.

In that context, Model Driven Development (MDD) [1] appears to be a good direction. The main interest of that approach is the focus on the construction of models as primary artifacts to describe the system. Furthermore, when models are expressed using formal description techniques, it is possible to verify and prove properties: fairness of shared resources access, fault tolerance or checking given events' occurrence. For example, a designer may want to check the occurrence of a given undesired situation, related to a *safety property*, according to some hypotheses on the system. Therefore, formal techniques are well suited to evaluate implementation choices on complex systems [2], [3].

However, the use of formal methods in this approach requires formal specifications. In particular, behavioral and temporal[1] information must be provided.

The objective of this paper is to propose a design and specification methodology that relies on both an industrial standard such as UML [4] and a formal description language such as Petri Nets, enhancing the relationship between the two formalisms.

The paper is structured as follows. Section II presents the issues raised by a model driven design and specification approach. Then, section III explains our methodology that is applied in section IV to a simple case study for illustrative purposes. Section V concludes and discusses perspectives.

II. RAISED PROBLEMS

The complexity of ITS systems is related to the number of actors in the system as well as its intrinsic parallelism and dynamics. The objective of our methodology is to lead designers to a stage where they can verify qualitative properties on their specifications. This implies the use of formal methods and thus, needs a precise specification (especially on behavioral aspects) to be transformed into a formal language. The transformation of an UML specification into a formal language for verification purpose raises some problems.

This section presents encountered problems that came out from the reflection around the design of the proposed approach.

Early requirements: *Choosing the appropriate modeling notation*

The primary purpose of this approach is to produce a formal analyzable ITS model from its specification. The combined use of UML and formal methods is not new. Many approaches have been proposed and have shown the feasibility and benefits of using both modeling approaches, in particular for performance evaluation [5], [6], [7]. We additionally provide a way to ease this relation between UML specification and the formal model, in a modular and incremental approach.

UML offers a semi-formal flexible and extendable notation for architectural and behavioral descriptions of almost any

F. Bonnefoi is with DSO R&D, Cofiroute, 6 - 10 rue Troyon, 92310 Sèvres, France. fabien.bonnefoi@cofiroute.fr

L.M. Hillah, F. Kordon and X. Renault are in the Université Pierre et Marie Curie (Paris6), with the Modeling and Verification team in the Department of Network and Distributed Systems at LIP6, CNRS UMR 7606, Paris, France. {lom-messan.hillah, fabrice.kordon, xavier.renault}@lip6.fr

[1] refers to causality, contrarily to *timed* that refers to time management.

kind of industrial system. Its use has been assessed on large European research projects like SAFESPOT [8]. The wide range of diagram types coupled with extensibility features in addition to its maturity highlights UML as the most appropriate entry point for our approach [9].

Formal analysis on concurrent and distributed systems is however best performed with a more formal language than UML, from which such capability is not yet achieved. Petri Nets do offer a formal mathematically founded framework for systems analysis.

Moreover, Petri Nets are extensively tooled to perform automatic verification on various key aspects of the considered systems such as qualitative or stochastic evaluation, time performance and so forth. In particular, Symmetric Nets, which help capture *similar behaviors* in distributed systems, are well-suited to automatic symmetry detection. This valuable feature enables the reduction of the state explosion problem induced by the size of complex systems [10], [11].

Problem 1, *modeling diagrams:* We need to ensure that the specifications will be adapted to perform formal verification. To formally and unambiguously describe main concerns (physical environment, control, vehicles data, etc.), appropriate UML diagrams must be chosen and their use guided. Without these restrictions UML diagrams can be used in many ways to design similar specifications.

Therefore our methodology should provide guidelines which are suitable for different "ITS case studies".

The proposed methodology, detailed in section III, describes how to specify a selected set of UML diagrams. In the next section, we raise the need of defining a model template suitable for the specification of ITS.

Problem 2, *structuring the specification:* There are heterogeneous elements to consider in ITS: computerized actors such as cars or controllers in a motorway infrastructure have to deal with physical variables such as braking distances, speed and weight. Also, these numerous actors are communicating and interacting in parallel.

To have a hierarchical and structured specification using a relevant subset of UML diagrams, we need to design an appropriate template. Also, it is important to address a class of problems instead of one single problem. So, the template must enable variation of architectures and components parameters.

The template presented in section III describes the system's architecture and its environment components.

Problem 3, *specifying behaviors:* UML State Machines are a potential candidate to express internal behaviors of components. However, there is no formal tool able to cope with state diagrams. Under some conditions, it is possible to transform a state chart into a formal notation like Petri Nets for analysis purpose as described in [12]. Activity diagrams have also been studied and manually translated into Stochastic Petri Net models[13]. Such approaches are interesting but generated Petri Nets for relatively simple components are quite large and non-optimized to capture symmetries. Thus, analysis of a complete system faces combinatorial explosion.

Petri Nets can also alternatively be used to directly specify the internal behavior of a system. This formalism is quite similar to UML State Machines, in addition to be mathematically founded. However, they are less familiar to designers.

Problem 4, *analyzing the system:* Analysis of a system can be carried out using two principal means: simulation and formal methods.

On one hand, simulation can be highly parameterized and thanks to this flexibility may not often require a lot of CPU and memory usage. However, it has the major drawback to be partial, thus undesired or unexpected behaviors can be missed. It can also be run endlessly, without no real observation of relevant behavior patterns from which formal analysis can be drawn.

On the other hand, formal methods which are mathematically founded, reach exhaustivity. There are two main methods [14]. *Theorem proving* is used by algebraic approaches to prove properties on systems, possibly infinite, described by the means of axioms. This method is difficult to use because it is less tooled, seldom fully automatic and require highly skilled and experienced engineers. Consequently, it takes a lot of time to build a proof. In contrast, *model checking* which is the exhaustive exploration of a system's state space, is fully automated but faces the combinatorial explosion problem and can mainly address finite systems. As a consequence, it generally requires a lot of CPU and memory usage.

III. METHODOLOGY

We propose some extra steps to the classical "V" software life cycle in order to help designers to specify and formally analyze ITS case studies. From the specification of minimal required subset of UML diagrams a formal model can be reached, on which safety properties can be verified. In particular, this methodology in the specific field of ITS aims at proposing solutions to the problems presented in section II.

Model Driven Development advocates the use of models as primary artifacts for systems development. Consequently, from raised abstractions in the design, refinement and model transformations are performed throughout the development cycle [1]. The methodology described in this paper takes place at early design stages. From the perspective of implementing the proposed framework into a tools suite for software engineers, we are investigating the use of model transformation techniques to build the target formal model from annotated UML models which are component, interface and state machines diagrams. As for any other MDD approach, it raises issues in models traceability, refinement and transformation.

Here are the solutions to the main problems introduced in section II. Extra steps in the software development process are outlined in Fig. 1.

Solution to problem 1: The high-level specifications are made using UML component and class diagrams. Component diagrams are used to describe the system's architecture as illustrated in Fig. 3. Interfaces are defined using UML

class diagrams, emphasizing on the types of exchanged data through offered methods (point 1 in Fig. 1).

Solution to problem 2: We provide a template that can be adapted to different ITS applications. This template is elaborated from the investigation of case studies of the SAFESPOT and TrafficView projects [15], [16].

The modular architecture are defined, involving components and their interconnections through interfaces. For example, the Vehicle subsystem is composed of several components. This architecture is realized using UML component model. Fig. 2 and 3 give an overview of the design template (point 2 in Fig. 1).

Solution to problem 3: The behavioral description of each component could be specified using candidate notations such as Petri Nets, UML state machines or activity diagrams. This description must be compatible with the interfaces definition since at the next step the final model is assembled based on the architectural and behavioral descriptions. As formal verification is a major objective of this paper, we have chosen Symmetric Nets to specify behaviors. This enables detection of symmetries on similar behaviors, reducing the combinatorial explosion during the model checking phase (point 3 in Fig. 1).

Solution to problem 4: Then the generation of formal specification of either a given component or the global system must be performed. This operation relies on automatic rules except for behavioral aspect that require inputs from engineers (point 4 in Fig. 1).

To express behavior of the system, a language like Petri Nets proposes several analysis techniques [3].

The main one is model checking that consists in the exhaustive exploration of the state space of the system. This is a way to verify if an undesired configuration occurs in the system (e.g., two vehicles cannot collide). It is also a way the verify causal relationship between two states (e.g., a decision can be computed only if a message occurs).

Other techniques allow to compute structural properties without having to elaborate the full state space. For example, they can be used to verify that mutual exclusion access to some resources is enforced.

From this point of view, Symmetric Nets are interesting because structural information can be computed and used to activate dedicated compression techniques. Consequently, it is possible to generate and handle very large state spaces [17].

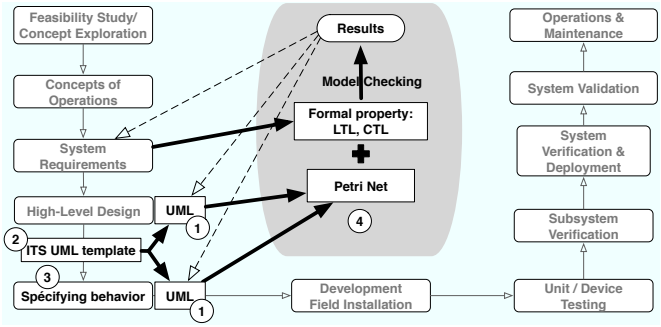

Fig. 1. New steps in the classical V-software life-cycle

Different activities are defined in our methodology (see Fig. 1). They cover the modeling of architectural and behavioral aspects, the generation of a formal specification and its analysis.

A. Modular architecture of the system

This architecture is based on the template. It uses *component diagrams* which represent the system's overall description. This description involves subsystems' components and their dependencies through interfaces.

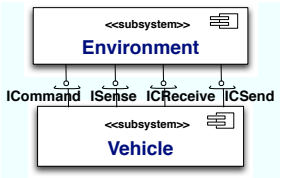

Fig. 2. Subsystems in the ITS UML template.

A part of the template is illustrated by the component diagrams in Fig. 2 and Fig. 3. It comes from the consideration that most ITS problems could be studied based on an architecture which is made up of the following subsystems and their components. From the experience of several projects (SAFESPOT, TrafficView), we deduce the following configuration:

- *Vehicles* subsystem including a strategy component that realizes the safety applications of the ITS system, sensors and communication components.
- Their *Environment* which implements the physics models and the communication canal.

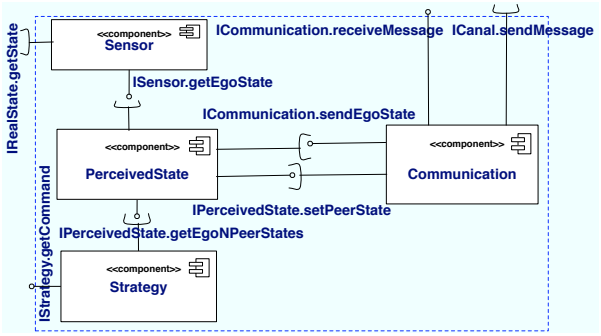

Fig. 3. Components in the vehicle subsystem.

Not represented in Fig. 2 but included in the complete formal model is the *Global Clock* which represents an abstraction of the elapsing time symbolized by execution cycles of the system, in order to assure fairness for all vehicles execution.

B. Definition of components interfaces

The second activity is the design of the interfaces. At this step, interfaces are defined in a *class diagram*, with input and output data handled by the offered methods. Their

names are directly related to the ones described in the component diagram. There is no restriction on the level of detail provided by the class diagram excepted that no behavioral properties should appear on it.

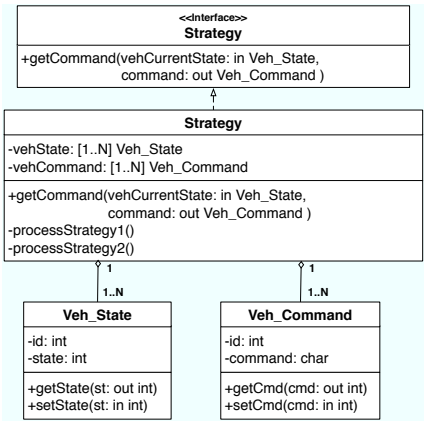

Fig. 4. An example of the Strategy interface and its realization.

Fig. 4 is an example of the class diagram of a strategy component.

C. Definition of components behavior

Components behavior in UML may either be defined with UML activity diagrams, state machines or directly using Symmetric Petri Nets.

For the first incremental development of this methodology, we mainly focused on the selection and coherence of the modeling notations of the specification chain. Model transformation from UML behavioral diagrams to Symmetric nets, which is another study of its own, will occur in a further step. Therefore, the behavioral model of each component is directly specified using Symmetric Nets.

As an example, component *VehicleMain* behavior is shown in Fig. 6 in section IV.

D. Assembling of the formal model

Some approaches have been proposed to transform UML models into performance models such as Layered Queuing Networks (LQN) or Stochastic Petri Nets [18], [13], [12]. The proposed transformations rely on specifying the *relations and mappings* between the source and target metamodels and implementing them with transformation rules. However, these approaches neither do deal with Symmetric Nets nor do they propose a design methodology suitable for ITS.

The methodology proposed in this paper allows for the generation the formal model of a single component or for the entire system. Here is the list of applied rules to transform our UML diagrams into a Symmetric Net model:

- Interfaces in the class diagram are exclusively bound to transitions in the behavioral model of each component (specified by or transformed into a Symmetric Net model), to enable the assembly step.
- Input and output variables of the class diagram are then bound to input and output variables in valuations of arcs falling into or emerging from the transitions.
- Then behavioral descriptions are manually translated or directly expressed in Symmetric Net formalism, and transitions of public methods used in interfaces are fused with the ones described above.

The complete formal model is automatically assembled using Petri Net tools developed internally. Particularly we used a very useful scripting language to handle large Petri Nets specifications: PetriScript [19].

The environment of the formal model of a single component is represented by Petri Net places which handle its input and output variables.

E. Analysis

The analysis is performed at two levels: individually on the components and on the complete model. Experiments have shown that individual components desired behavior must as a minimum be checked before assembling them. This is useful for debugging purposes since components behavior can be changed to test different strategies.

Requirements of the system are expressed as properties on its model. Those properties are expressed as formula. Causal properties are expressed using LTL or CTL queries [20]. Then, automatic verification is processed using dedicated model checkers which are automatically handled by specialized Petri Net based tools [21]. In Fig. 1, dotted arrows represent feedbacks resulting from this model checking phase.

In the following sections, an experiment is reported about some properties and future work is sketched.

IV. EXPERIMENTATION AND RESULTS

We are studying different case studies related to ITS. The proposed case study in this section is used to illustrate the application of the modeling methodology. The objective is to assess the effectiveness and the applicability of formal methods for ITS.

A. Case study

Let us consider groups of cooperative vehicles, which collaborate in a fully decentralized approach. They exchange information about the traffic state. Then, each driver is able to make a decision according to the traffic context (avoiding traffic jam or collision spot, reducing speed, changing lane, etc.).

In this case study, the road space is considered as a shared resource and is divided into cells. Spatial cells are moving at the same speed as the average speed of vehicles. A single vehicle occupies an entire cell, but some cells may not be occupied and are considered as free cells if they offer a sufficient space for a vehicle to move in.

A vehicle wanting to move into a free cell must notify adjacent vehicles to this cell. These adjacent vehicles are potential candidates to obtain the cell, so they must reach a consensus to enable a vehicle to move into that cell. Vehicles also share the communication medium and their communication range is limited to a given number of cells.

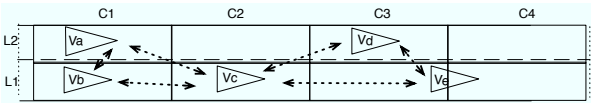

Fig. 5. System of cooperative vehicles exchanging information

The overview of the system is depicted by Fig. 5. In this configuration, vehicle *Va* is in and moving with cell *L2C1*. It communicates with vehicles *Vb* and *Vc* which are respectively in cells *L1C1* and *L1C2*. It is out of the communication range with vehicle *Vd* which is in cell *L2C3*. It may want to move into cell *L2C2*, therefore it should obtain the agreement of *Vb* and *Vc* before moving into that cell.

This kind of system may be used to implement and test different traffic control strategies [16].

We make some assumptions on the environment and the behavior of vehicles, in order to specify the system.

- Vehicles communicate with each other using WiFi devices or any wireless communication technology. The communication medium can be constrained by the number of simultaneous open connections.
- The communication infrastructure is based on message passing: information to be exchanged is aggregated in messages and sent to other peers.
- In a first approach, the information is about "states" of vehicles. Here, vehicles states are abstract, but may be refined in a further study to correctly represent their speed, radial acceleration, etc.
- Each vehicle perceives its own state from local sensors and GPS devices. That information is stored in a local database.
- Each vehicle perceives its neighbors' states from wireless communication devices. That information is also stored in the local database.
- The representation of the environment in a vehicle, stored in the local database, is not as accurate as the real environment: stored information is updated, so at an i instant, local variables and real states are not always synchronized (due to the update period).

B. Basic specification

From the formulated ground hypotheses, needed components must be defined. In the methodology presented in section III all components are defined in the proposed notation which is UML and Petri Nets based.

The physics model in the environment subsystem manages vehicles' real states which are perceived by their sensors. It also implements the function which updates these states based on commands issued by vehicles. Its activity is under the supervision of a local scheduler. A global clock is designed in the final formal model. The purpose of this clock is to ensure fair execution steps for all vehicles and their interactions with the environment. It thus represents an abstraction of the elapsing time which is symbolized by a cycle. Hence, all vehicles must accomplish a common set of actions in a cycle before moving to the next.

A detailed view of vehicles subsystem is presented in Fig. 3. The following components are involved:

- The *Sensor*, which perceives its states from the environment.
- The *Perceived State*, which stores that information (as a database) necessary to compute a command.
- The *Strategy*, which is responsible for issuing a command to the environment in order to update each vehicle's state, based on its perceived state and its neighbors' ones.
- The *Communication* component handles messages passing through the environment's channel between neighbors.

C. Assembling the Petri Net model

If the minimal set of UML specifications is provided, it is possible to obtain the Petri Net model of a single component or the model of the whole system. The complete Petri Net model of the system is assembled from the architectural and behavioral descriptions of its components.

The model of Fig. 6 shows the formal model of vehicles subsystem general scheduling. It handles the common set of actions vehicles must perform during a cycle. These actions are related to services offered by components in vehicles subsystem.

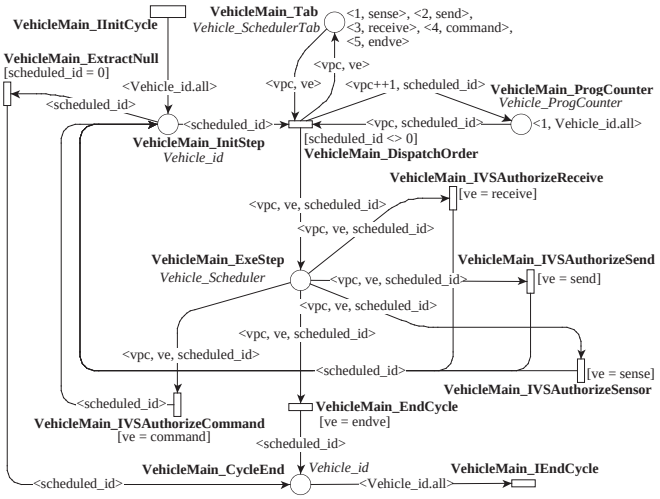

Fig. 6. Petri net model: general scheduling of vehicles subsystem.

D. Analysis

To generate the model for a single component, the interface, class and behavioral diagrams are sufficient. It is then possible to verify some properties on the isolated component by means of the translation of "requirements" into temporal logic languages such as LTL. Consequently we have been able to determine for example that the strategy component will not provide a given type of recommendation to the driver if the states of peer vehicles have not been received.

In addition to the verification of the correct types of exchanged variables, the formal analysis of the system enabled us to evaluate and optimize chosen algorithms.

At this stage, we mainly focused on defining the methodology and tested few algorithms. Some of them satisfied our minimum requirements about the number of cycles to reach a consensus. However, further analysis concerning the variation of the maximum number of simultaneous open connections and the communication range are necessary to provide relevant results.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a design and specification methodology dedicated to the elaboration of solutions to ITS distributed control issues. We want to address four main problems identified in section II:

1) modeling diagrams,
2) structuring the specification,
3) specifying behaviors, and
4) analyzing the system.

To be compatible with current approaches in software design, the methodology relies on UML and stick to the "classical" V-software life cycle. We also proposed to start the modeling from a specification template containing components that can be adapted to a given problem. To do so, ITS designers should follow guidelines to integrate the elements that are relevant for a given ITS case study. The specification template also preserves by construction a fair execution of the system. Discretization of the physical aspects of the system can be integrated to stick to realistic conditions.

Once the specification template customized for a given case study, designers must specify the behavior for the components in the system. Standard UML behavioral diagrams can be used since links to some formal notations are provided. Alternatively, a formal specification can also directly be used as shown in our example (see section IV).

Our objective is to guide designers to achieve ITS specifications at a level suitable for formal analysis and verification. This is also a way to enhance the use of achieved results from one modeling formalism to the other. The next step of our work consists in working at the verification level. In particular, the handling of ITS' dynamics requires optimized model checking techniques. Some of these techniques were already experimented in [22] but without high-level methodology.

We obtained expected results on the verification of qualitative properties of the system and its components. It is also worth following up with quantitative verification using additional specification in UML set of diagrams and other specific PN tools.

Another point of interest is the full automation of translations from UML models into Symmetric Nets models in the framework of a development tool. To do so, we plan to use Eclipse Modeling Framework [23] as a basis for such an implementation.

REFERENCES

[1] B. Hailpern and P. Tarr, "Model-driven development: The good, the bad and the ugly," *IBM Systems Journal*, vol. 45, no. 3, p. 451, 2006.

[2] J. Gogen and Luqi, "Formal methods: Promises and problems," *IEEE Software*, vol. 14, no. 1, pp. 75–85, 1997.

[3] C. Girault and R. Valk, *Petri Nets for Systems Engineering: A Guide to Modeling, Verification and Applications*, 2003.

[4] OMG, *Unified Modeling Language: Superstructure - Version 2.0 formal/05-07-04*, OMG, March 2006. [Online]. Available: http://www.uml.org/

[5] C. Snook and M. Butler, "UML-B: Formal Modeling and Design Aided by UML," in *ACM Transaction on Sofware Engineering and Methodology*, vol. 15, no. 1, January 2006, pp. 92–122.

[6] J. Campos and J. Merseguer, "On the integration of UML and Petri nets in software development." in *27th ICATPN - Petri Nets and other models of concurrency*, S. Donatelli and P. Thiagarajan, Eds., vol. 4024. Springer-Verlag Berlin Heidelberg, June 2006, pp. 19–36.

[7] B. Bordbar, L. Giacomini, and D. Holding, "UML and Petri nets for design and analysis of distributed systems," in *IEEE Conference on Control Applications*, 2000, pp. 610–615.

[8] F. Bonnefoi, F. Bellotti, and T. Scendzielorz, "From user needs to application, the SAFESPOT approach based on roads data analysis," in *6th European Congress and Exhibition on Intelligent Transport Systems and Services*, Aalborg, Denmark, June 2007.

[9] J. Trujillo, "A Report on the First International Workshop on best practices of UML (BP-UML'05)," in *SIGMOD Record*, vol. 35, no. 3, September 2006.

[10] Y. Thierry-Mieg, C. Dutheillet, and I. Mounier, "Automatic symmetry detection in well-formed nets." in *24th International Conference on Applications and Theory of Petri Nets 2003*, ser. LNCS, W. M. P. van der Aalst and E. Best, Eds., vol. 2679. Springer Verlag, 2003, pp. 82–101.

[11] G. Chiola, C. Dutheillet, G. Franceschini, and S. Haddad, "On Well-Formed Coloured Nets and their Symbolic Reachability Graph," *High-Level Petri Nets. Theory and Application, LNCS*, 1991.

[12] S. Bernardi, S. Donatelli, and J. Merseguer, "From UML sequence diagrams and statecharts to analysable petri net models," in *WOSP '02: Proceedings of the 3rd international workshop on Software and performance*. New York, NY, USA: ACM Press, 2002, pp. 35–45.

[13] J. P. Lopez-Grao, J. Merseguer, and J. Campos, "From UML activity diagrams to Stochastic Petri nets: application to software performance engineering," in *WOSP '04: Proceedings of the 4th international workshop on Software and performance*. New York, NY, USA: ACM Press, 2004, pp. 25–36.

[14] A. Valmari, "The State Explosion Problem," in *Lectures on Petri Nets I: Basic Models*, ser. Lecture Notes in Computer Science, no. 1491. Springer-Verlag, 1998, pp. 429–528.

[15] "Safespot project," 2007. [Online]. Available: http://www.safespot-eu.org/pages/page.php

[16] 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 Semiannual Vehicular Technology Conference*, I. C. Press, Ed., 2004.

[17] F. Kordon, "Mastering Complexity in Formal Analysis of Complex Systems: Some Issues and Strategies Applied to Intelligent Transport Systems," in *International Symposium on Object-oriented Real-time distributed Computing (ISORC'07)*. Santorini, Greece: IEEE Computer Society, May 2007, p. to be published.

[18] A. D'Ambrogio, "A model transformation framework for the automated building of performance models from UML models," in *WOSP '05: Proceedings of the 5th international workshop on Software and performance*. New York, NY, USA: ACM Press, 2005, pp. 75–86.

[19] A. Hamez and X. Renault, *PetriScript Reference Manual*, LIP6, www-src.lip6.fr/logiciels/mars/CPNAMI/MANUAL_SERV.

[20] C. Dutheillet, I. Vernier-Mounier, J.-M. Ilié, and D. Poitrenaud, *State-space-based methods and model checking*, Petri nets and system engineering (Claude Girault and Rudiger Valk Eds), first ed. Springer Verlag, 2003, ch. 14, pp. 201–276.

[21] LIP6-MoVe, *The CPN-AMI Home page.*, http://www.lip6.fr/cpn-ami.

[22] 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.

[23] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. Grose, *Eclipse Modeling Framework*, ser. The Eclipse Series, E. Gamma, L. Nackman, and J. Wiegand, Eds. Addison-Wesley Professional, August 2003.