A UML-MARTE Temporal Property Verification Tool based on Model Checking

Ning Ge, Marc Pantel, Xavier Crégut

A UML-MARTE Temporal Property Verification Tool based on Model Checking

Ning Ge, Marc Pantel, and Xavier Crégut

University of Toulouse, IRIT/INPT
{Ning.Ge | Marc.Pantel | Xavier.Cregut}@enseeiht.fr

Keywords: Real-Time System, Temporal Property, UML-MARTE, Time Petri Net, Model Checking

1 Introduction

On today's highly competitive industrial market, the scale and complexity of safety critical real-time system are rapidly increasing due to the growth of functional and non-functional requirements. Consequently, verification of the temporal requirements for real-time system development is becoming more and more difficult and expensive. Therefore, process and methods for the design and implementation of highly safety critical real-time system that allow mastering development cost are still an open problem in both industry and academia.

UML (Unified Modeling Language) [26] was developed to provide a common language for specification, modeling and documentation in the software development process. Today, UML is the industry standard for software modeling. MARTE (Modeling and Analysis of Real-Time and Embedded Systems) [1] provides support for specification, design, verification/validation for real-time embedded systems. We use the term UML-MARTE to indicate the specification language for this work. Its purpose is to propose efficient formal verification tools to ease the use of UML-MARTE when developing large scale real-time systems in the early phases of Model Driven Engineering (MDE). To achieve this goal, we propose a temporal property analysis tool based on Time Petri Nets (TPN) and the TINA toolset for UML-MARTE real-time system designs.

The most significant issue preventing model checking method to be applied widely in industry is the lack of scalability. The classic verification methodology will encounter scalability problem very quickly along with the growth of system size. Complex systems usually have thousands of transitions. Although a huge part of transition's firing sequence has been eliminated by the system's behavior, the possible permutation of all the other ones is still a huge number that will usually lead to combinatorial state explosion. The proposed analysis tool improves the efficiency of verification from the following aspects: mapping UML real-time model to TPN, specifying temporal properties using property patterns, performing efficient model-checking based on observers for the verification of temporal properties, reducing state space before performing model checking. The verification tool also integrates a fault location analysis approach to help end users in locating the origin of fault when a safety property is unsatisfied.

We present the state of the art of temporal property verification in Section 2; Time Petri Net and TINA are presented in Section 3; The temporal property verification tool is detailed in Section 4; A case study is provided in Section 5; Section 6 gives some conclusion.

2 State of the Art of Temporal Property Verification

Verification of a system is the task of determining that the system is built according to its explicit specification. Verification assesses the products against system requirements and ensures that a system will perform as specified.

In practice, temporal property verification in MDE is implemented in two manners: simulation and formal verification. Simulation is relatively inexpensive in terms of execution time. But it only validates and verifies the behavior of a system for one or several particular computation paths. Some existing works have achieved

success in the analysis of real-time systems, such as Cheddar[1], SynDEx[2], UML-MAST[3], and MARTE2MAST[4]. These simulation techniques suffer from the exhaustivity problem, which means it is not possible to simulate all possible behaviors in a design. Contrasting to simulation, formal verification is a systematic process that uses mathematical reasoning to verify that the design intent is preserved in the implementation.

Formal methods, such as model checking, theorem proving, static analysis, etc, allow specifying a system's requirements, designing an implementation, and assessing its consistency, completeness, and correctness in a mathematical exhaustive fashion. Our work uses model checking as formal method to analyze the temporal requirements. Many model checking tools have been developed to assess temporal logic formulae over transition systems/Kripke structure (ktz), such as TINA[5] (for Petri Nets and Time Petri Nets), Roméo [6] (for Time Petri Nets), UPPAAL[7] (for Timed Automata) and SPIN[8] (for Promela).

There also exists some other temporal property verification approaches such as the use of Integer Linear Programming (ILP) by Lauer et al. [19] that used a modeling approach for Integrated Modular Avionic (IMA) based on the tagged signal model [23] and the abstraction of network. The tag system was then transformed into ILP problem. They proposed evaluation method for the end-to-end temporal properties based on ILP, and obtained optimal results.

3 Time Petri Net and TINA Toolset

Time Petri Nets [25] extends Petri Nets with timing constraints on the firing of transitions. We use an example (see Ex. 1) to explain its syntax and semantics.

Example 1 (Time Petri Net (TPN)). This example given in Fig. 1 models the concurrent execution of a process. The transitions in TPN are extended with a time constraint that controls the firing time. P_{init} is the place holding the initial token. When the $fork$ transition is fired, concurrent $task_1$ and $task_2$ start at the same time within respective execution time [11,15] and [19,27] associated to the transitions modeling their end. The time constraints rely on a local clock associated to the transitions that start once the transitions are enabled. When it reaches the $join$ state, the system will exit or restart the whole execution according to the running time.

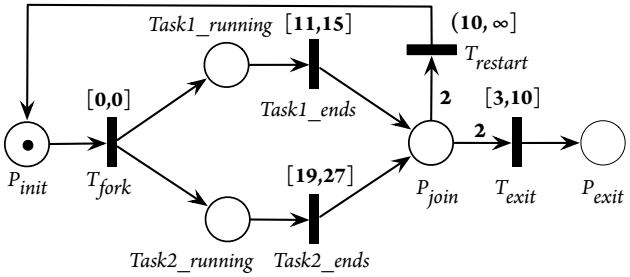

Figure 1. Time Petri Net Example

Time Petri Nets are widely used to formally capture the temporal behavior of concurrent real-time system due to their easy-to-understand graphical notation and semantics. The use of time intervals instead

[1] http://beru.univ-brest.fr/ singhoff/cheddar/index.html
[2] http://www.syndex.org/
[3] http://mast.unican.es/umlmast/
[4] http://mast.unican.es/umlmast/marte2mast/
[5] http://projects.laas.fr/tina/
[6] http://romeo.rts-software.org/
[7] http://www.uppaal.com/
[8] http://spinroot.com/spin/what.html/

of explicit clocks as in timed automaton eases the modeling and analysis. There exist several analysis tools, such as TINA or Roméo. Our work relies on TINA (TIme petri Net Analyzer). TINA is a toolbox for the edition and analysis of Petri Nets, also providing inhibitor and read arcs, Time Petri Nets, with possibly priorities and stopwatches, and an extension of Time Petri Nets with data handling called Time Transition Systems (tts). TINA toolset includes the following tools: **nd** as editor and GUI for Petri Nets, Time Petri Nets and automata; tina for construction of reachability graphs; sift for construction and checking of reachability graph; selt as a State/Event LTL model checker; muse as the μ-calculus model checker; etc.

4 UML Temporal Property Analysis Tool

The main contribution of this work is the design and implementation of a property-driven verification framework dedicated to temporal properties verification for UML-MARTE real-time software designs (see Fig. 2). The verification framework consists of five core elements: semantic mapping from UML-MARTE to TPN; temporal property specification; temporal property verification; state-space reduction and feedback analysis. We details these elements in the follows sections.

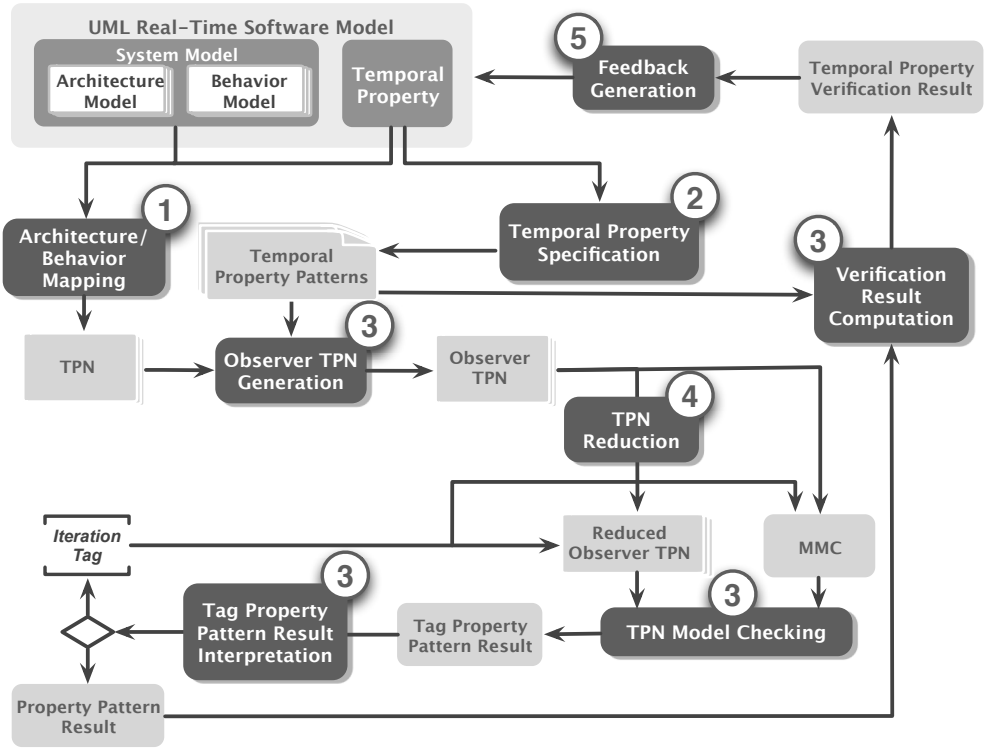

Figure 2. Framework of UML analysis Tool

4.1 Semantic Mapping from UML-MARTE to TPN

UML, by its nature, was intended to be a general purpose software modeling language, and as such, is not simple enough to be efficiently used by non-software experts. For the purpose of this work, we have selected a large enough subset of UML diagrams and diagram elements for modeling real-time software architecture and behavior. We focus on the semantic mapping from UML model to verification model.

From the viewpoint of methodology, this mapping approach is based on the pioneering work [4] by Combemale et al. Aiming the definition of all the steps from the property specification to effective verification,

they introduced a generic approach to define the operational semantics (a semantics of observable events) built upon the properties expressed at the metamodel level. Property-driven means that the formal activities in the development process are based on the purpose of property-verification-ease. From a language point of view, a precise definition of model elements behavior is required for the execution of behavioral models.

We defined operational semantics for the end user model UML-MARTE and map it to the execution model TPN. According to the to-be-assessed temporal property, we have defined the operational and mapping semantics for UML architecture diagram: composite structure diagram and UML behavior diagrams: activity [14] and state machine. For some untimed UML elements not influencing temporal properties, the target TPN semantics can be standardized and homogeneous for all the properties, such as Join, Fork, Merge, Initial nodes in activity diagram, etc. For the timed UML elements, the mapping semantics eliminates the unnecessary information and preserves the minimal set of semantics to reduce the state space in the verification. A universal scheduling algorithm including a preemption option is defined. The scheduling algorithm will guarantee the semantic that is used to decide for the given time T, which resource requester(s) will be allocated by the resource instance(s).

4.2 Temporal Property Specification

The temporal requirements can be semantically decomposed into a set of elementary properties called property patterns. Property patterns, on one hand ease the use of formal methods especially for the non-expert users by providing the recurrent solutions to verification problem; and on the other hand decompose complex properties into a set of simpler ones and thus decrease the cost of verification.

Dwyer et al. firstly proposed qualitative time property patterns for finite-state verification [5, 6]. They also performed a large-scale study in which specifications containing over 500 temporal properties were collected and analyzed. They noticed that over 90% of these could be classified under one of the proposed patterns [6], which encourage other works to extend Dwyer's pattern system to define quantitative time property patterns with additional real-time constraints [18, 17, 2].

From the viewpoint of property verification, we think the quantitative time property patterns based on Dwyer's patterns are not atomic. Most requirement patterns can be checked using a set of atomic patterns. We have defined such a minimal set of atomic property patterns used to compose temporal requirements in purpose of specification and verification. The classic temporal property patterns will be automatically mapped to our atomic patterns using the predefined pattern metamodel and the mapping library.

In the proposed pattern system (see Fig. 3), the temporal requirements will be specified using temporal property patterns (either atomic pattern or composite pattern). The composite patterns can be easily composed by using the binary operators (or, and, imply). Atomic patterns are built using *occurrence modifier*, *basic predicate* and *scope modifier*. Basic predicates are based on state and event modifier while scope modifiers are based on event modifiers. We have applied parts of this approach for a subset of temporal requirements in the work [13, 12].

4.3 Temporal Property Verification

A quantitative time requirement expressed using Dwyer's patterns with real-time extensions can be specified using various formalisms of logic formulae. For example, the time-bounded response property *Exist S responds to P within k* `t.u` *(time unit)* can be specified as $EG(P \Rightarrow AF_{<k}(S))$ in TCTL. Such an expressing method for quantitative properties are interesting for theoretical purpose, but its practical use would be limited due to the capability of model checking tools.

Usually, the above problem is solved by using observers. Observers run in parallel with the model under checking. A certain transitions will be fired and the expected states will be reached if and only if some timed conditions are satisfied. These behaviors will be observed to check the satisfaction of temporal properties.

As we presented in the work [9, 10, 12], a TPN observer is a complementary structure attached to the original system. As shown in Fig. 4, in order to assess a temporal property, a TPN observer is associated with the original system, by linking from the transition T_A in component A and the transition T_B in component B. The observer contains a place P_{tester}, which can assess this temporal property using state-event modal μ-calculus (MMC) formulae in ktz format. According to the temporal requirement, one or more MMC formulae

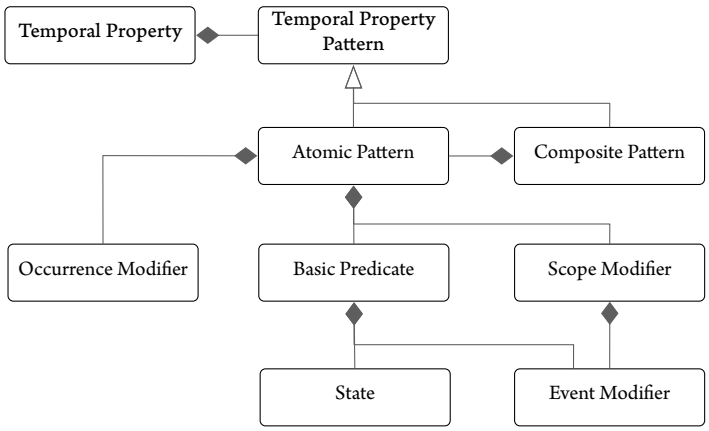

Figure 3. Temporal Property Specification

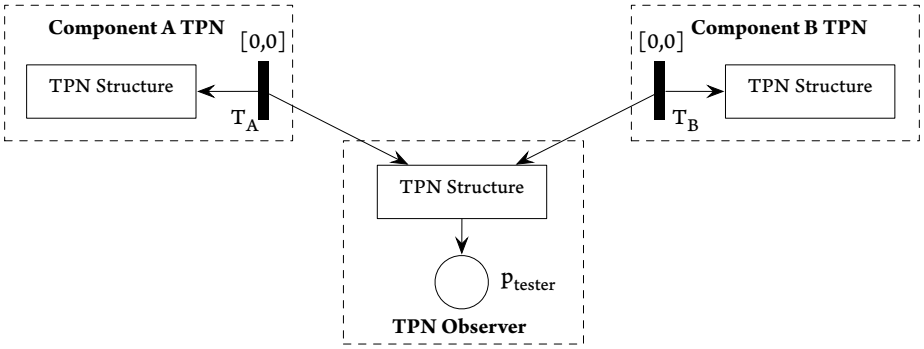

Figure 4. Observer Structure

are generated, such as $\Box(P_{tester} = 1)$ or $\Diamond(P_{tester} = 0)$, etc. The `MMC` formulae will be on-the-fly checked using model checker `muse` in `TINA` toolset.

As model checking consumes a lot of resources, the design and implementation of the observers obligate to follow four principles to ensure that the observers are sound and to minimize the state space: First, the observer should not impact the system's behavior by introducing extra semantics; Second, the introduction of `TPN` observers should bring minimized side effect to the size of the state space of the original system; Third, the `TPN` system model with integrated observers should be able to perform the possible highest abstraction (marking abstraction in our case) when unfolding the reachability graph. This high abstraction reachability graph must preserve all the information related to the expected property; At last, the checking of each property pattern must be independent in terms of unfolding reachability graph to promote parallel computation.

When performing model checking, an observer can give an answer such as YES or NO for the satisfaction of the given property. For quantitative properties, however, users usually expect to know *what is the bound* $[t_{min}, t_{max}]$ *of that property* instead of *whether the property is bounded by* $[t_{min}, t_{max}]$*?* In order to fill this gap of usage, we propose an iterative method (shown as the service 3 in Fig. 2) that will gradually approach the bounding value by integrating the observer into a binary search engine.

4.4 State Space Reduction

A typical system will run with a large amount of transitions and make the above prerequisites hard to maintain. The fundamental reason why this computation-oriented approach is not scalable is because it tries

to preserve all information for the verification afterwards. In this work, we focus on state space reduction approaches related to TPN model dedicated to temporal property verification.

In the above observer-based model checking approach, we have presented the use of on-the-fly and state abstraction approaches. Both of them focus on reducing state space when performing model checking. Another reasonable thinking is to create an equivalence of original TPN in terms of property-related system behavior, but with less transitions. This reduces directly the scale of computation before performing generation of state space. Before being sent to the model checker ,the TPN must be reduced in appropriated ways according to the properties to be assessed.

The reduction is property-driven, which means all structures irrelevant to the verification of a given property will be eliminated. Compared to existing techniques [27], our approach is driven by property, because TPN system has been integrated with property observers. This property-driven characteristic allows removal of much more transitions in TPN model, which directly signifies more reduced state space in model checking.

We propose practical topology-implicit semantic equivalence reduction patterns which are commonly found in TPN model with observer structures for temporal property verification [11]. In this approach, redundant zero-time patterns and sequential encapsulation pattern are proposed. In some cases, it is less trivial to observe and extract a localized topology pattern but still necessary to have the whole TPN reduced. A novel reduction method based on behavioral equivalence is proposed. This method identifies sub-TPN which exhibits the same behavior as the original net. Before reducing the pattern-matched structures, we use bi-simulation function to guarantee the reduced net corresponds exactly to the original behavior.

4.5 Fault Localization in Model Checking

Existing automated fault localization techniques in model checking usually produce a set of suspicious statements without any particular ranking. We designed an automated fault localization approach based on model checking [16, 15] to ease and accelerate debugging by locating and ranking the suspicious elements in model checking when a safety property is unsatisfied. We propose a suspiciousness factor to rank the potentially faulty transitions. We construct error traces in the reachability graph using the violation states. The suspiciousness factor is then computed using the fault contribution of each transition on all the error traces. The fault contribution is computed using the entropy and differential entropy of transition. Based on the mapping semantics from the end user model (in our case UML) to the verification model (in our case TPN), the faulty transitions in the reachability graph can be back-traced from TPN to UML.

5 Case Study

We use an avionic case study investigated by M. Lauer et al. [22, 20, 19, 21], which is a part of a flight management system (FMS) to test our approach. The temporal requirements to be verified is latency temporal properties. We model the case study system in UML-MARTE composite structure and activity diagrams, and then map it to TPN model using the semantic mapping. The temporal property is specified using the property patterns, and then verified using the observer-based model checking. The scalability test shows that our proposal is able to analyze large scale systems by using the TPN reduction.

5.1 Case Study Description

The architecture of the FMS case study is represented by Fig. 5. Seven modules, from Module_1 to Module_7 (named M_1, ..., M_7 afterwards), are used to map the avionic functions. Due to the page limit, the detailed description of the case study can only be consulted in [22, 20, 19, 21].

The functional chain of responding to pilot's sporadic request is in Fig. 6. At any time, the pilot can request some information on a given waypoint. The KU_1 (Keyboard and Cursor Control Unit) controls the physical device used by the pilot to enter his requests. When KU_1 receives a request (req_1), it broadcasts $wpid_1$ and $wpid_2$ to the Flight Managers FM_1 and FM_2 respectively. The *FMs* manage the flight plan, i.e., the trajectory between successive waypoints. When a request occurs, both query the *NDB* (Navigation Database) by sending $query_1$ (resp. $query_2$) to retrieve the static information on the waypoint such as

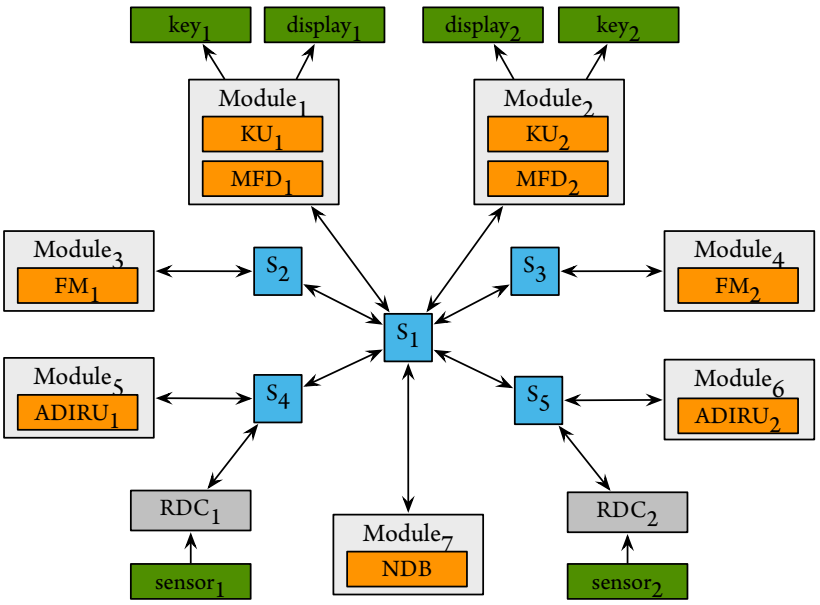

Figure 5. Architecture of Case Study

the latitude and the longitude. The *NDB* separately answers each *FM* by sending a message $answer_1$ (resp. $answer_2$) containing the expected data. Upon reception of this message, each *FM* computes two complementary dynamic data: the distance to the waypoint, and the estimated time of arrival (*ETA*). These data ($wpInfo_1$ and $wpInfo_2$ resp.) are periodically sent to respective *MFDs* (Multi Functional Display) which periodically elaborate the pages to be displayed on the screens.

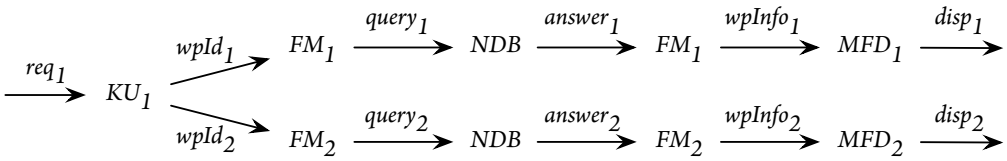

Figure 6. Functional Chain: Sporadic Response to Request

5.2 Latency Temporal Properties

Because of their critical nature, IMA systems must also satisfy strong real-time requirements. In the case study, we focus on the latency temporal requirements. The latency corresponds to the time elapsed between an event at the beginning of a functional chain and the first event depending on it at the end of the chain, i.e. a sporadic input must result in an output before a certain amount of time.

Example 2 (Latency Property Example). The example of latency is represented in Fig. 7. On the functional chain: $\xrightarrow{req_1} KU_1 \xrightarrow{wpId_1} FM_1 \xrightarrow{query_1} NDB \xrightarrow{answer_1} FM_1 \xrightarrow{wpInfo_1} MFD_1 \xrightarrow{disp_1}$, the maximum time between req_1 and the first occurrence of $disp_1$ depending on req_1 must be 700 ms.

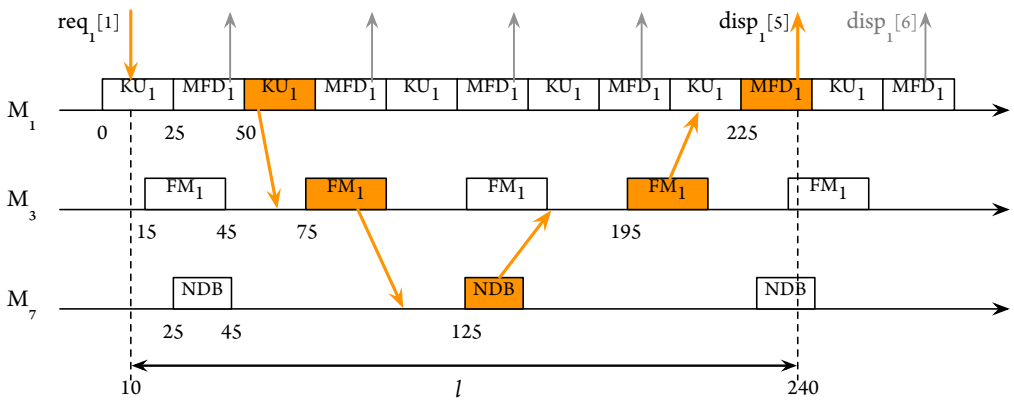

Figure 7. Latency Temporal Property

5.3 Modeling and Semantics

We specify the model of the case study by using `UML-MARTE`. The `IMA` architecture and `AFDX` network are specified using composite structure diagram; the behavior of each module is specified using activity diagram for the functional chain of latency temporal property. An abstraction of `AFDX` network is applied. The abstraction is proposed in [20]. The complexity can be significantly reduced by characterizing the lower (resp. upper) bound of the network traversal time along the virtual link path as a time interval [a, b]. These bounds are determined by the trajectory approach [24] which has been successfully applied to `AFDX` networks in [3].

5.4 Latency Temporal Property Verification

The `UML-MARTE` model is automatically mapped to the verification model `TPN`. Then the observers dedicated to maximum time interval property are added to the associated `TPN` transitions. This observer-integrated model and `MMC` formulae are sent to the `muse` model checker to perform model checking. The computation results are shown in Table 1. For latency temporal property, the `WCT` (resp. `BCT`) is 450.4 (reps. 75.2) `ms`. For freshness temporal property, the `WCT` (resp. `BCT`) is 316429 (resp. 1012) `ms`. The original system for latency property without `TPN` observer has 9378 states and 23250 transitions. `TPN` observers cause side-effect to the size of stat space of original system. The side-effect depends on t_{min} and t_{max} on the tester transition. By applying the reduction techniques, the number of state and transition is largely reduced. Take `WCT` latency property for example, compared to the execution time before reduction (278.313 s), the execution time is reduced to 2.484 s.

Table 1. Latency Temporal Property Verification Result

Property		**Property Value (ms)**	**State/Transition Number**		**Execution Time (s)**	
			Before Reduc.	After Reduc.	Before Reduc.	After Reduc.
Latency	System	`N/A`	9378/23250	`N/A`	`N/A`	`N/A`
	`WCT`	450.4	67105/145024	9/10	278.313	2.484
	`BCT`	75.2	11162/28922	8/9	43.781	3.719

5.5 Scalability Test

The proposed methods target large scale systems. We implement a series of experiments to evaluate whether the cost of verification is acceptable for the large scale systems by extending the case study. We test the

scalability of proposed methods by increasing the length of each functional chain. The latency functional chain is enlarged by increasing the number of *NDB*. Each latency functional chain traverses P *NDB*, i.e. $2P + 3$ functions.

$$L_1 = \xrightarrow{req_1} KU_1 \xrightarrow{wpId_1} FM_1 \xrightarrow{query_1} NDB_1 \xrightarrow{query_2} ... \xrightarrow{query_{P-1}} NDB_{P-1} \xrightarrow{query_P} NDB_P \xrightarrow{query_P} NDB_{P-1} \xrightarrow{answer_{P-1}} ... \xrightarrow{answer_2} NDB_1 \xrightarrow{answer_1} FM_1 \xrightarrow{wpInfo_1} MFD_1 \xrightarrow{disp_1} \quad (1)$$

By increasing P from 1 to 11, we give out the property values and solving time by using proposed verification and reduction approaches in Table 2. The reduction phase consumes much more time than the model checking. Once the reduction is finished, the analysis phase is fast. The binary search method is used to compute the bound value. Fig. 8 shows that the solving time of case study is linear to *NDB* number.

Table 2. Scalability Test of Latency Property

***NDB*/Fun.**	**Prop. Val. (ms)**		**S/T (after R.)**		**Reduction Time (s)**	**Analysis Time (s)**		**Solving Time (s)**	
	WCT	BCT	WCT	BCT		WCT	BCT	WCT	BCT
1/7	75.2	450.4	9/10	8/9	38.049	2.484	1.860	40.533	39,909
2/8	125.2	750.4	9/10	8/9	57.876	2.656	1.883	60.532	59,759
3/9	275.2	1050.4	9/10/	6/5	79.813	2.812	2.079	82.625	81,892
4/10	375.2	1350.4	9/10	6/5	102.500	2.906	2.079	105.406	104,579
5/11	425.2	1650.4	9/10	6/5	124.987	3.015	2.102	128.002	127,089
6/12	575.2	1950.4	9/10	6/5	149.359	2.891	2.196	152.250	151,555
7/13	675.2	2250.4	9/10	6/5	169.607	2.953	2.227	172.560	171,834
8/14	725.2	2550.4	9/10	6/5	193.329	3.031	2.250	196.360	195,579
9/15	875.2	2850.4	9/10	6/5	216.239	3.000	2.211	219.239	218,45
10/16	975.2	3150.4	9/10	6/5	239.953	3.047	2.195	243.000	242,148
11/17	1025.2	3450.4	9/10	6/5	263.049	3.188	2.195	266.237	265,244

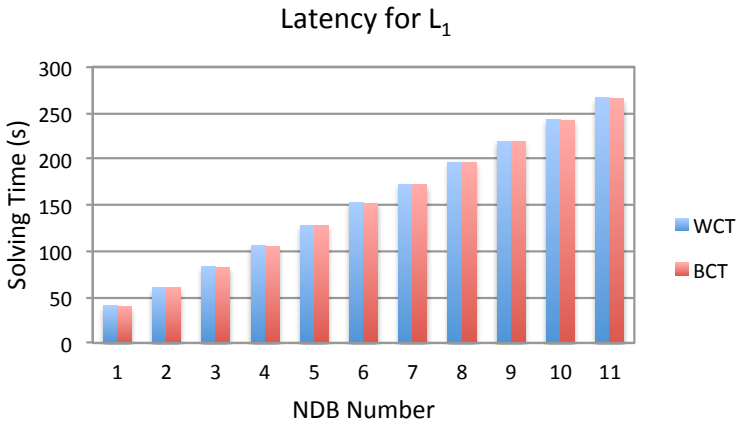

Figure 8. Solving Time of Scalable Latency Property

6 Conclusion

The UML temporal property verification tool is applied during the architecture design phase of MDE. This tool targets the efficient verification, and supports the analysis of temporal requirements in real-time systems. The efficiency is ensured by the approaches integrated in the tool framework, including the property-driven semantic mapping, property specification based on patterns, observer-based model checking, and state space reduction. We test the integrated tool, on a simplified industrial case study: IMA-based airborne system. The test results show the efficiency of the tool.

In the future, we intend to investigate new approaches to diagnosing design errors based on the model checking results. We have got some preliminary results in the works that diagnose faults in the simulation and testing based on Hidden Markov Model [8, 7]. The next step is to investigate how to integrate these approaches in the formal verification of real-time properties and other functional properties.

Acknowledgment

This work was funded by the FUI Projet P and EuroStars HiMoCo projects.

References

1. OMG, UML profile for MARTE: modeling and analysis of real-time embedded systems version 1.0 (Nov 2009)
2. Abid, N., Dal Zilio, S., Le Botlan, D.: Real-time specification patterns and tools. In: Formal Methods for Industrial Critical Systems, pp. 1–15. Springer (2012)
3. Bauer, H., Scharbarg, J.L., Fraboul, C.: Applying and optimizing trajectory approach for performance evaluation of afdx avionics network. In: Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on. pp. 1–8. IEEE (2009)
4. Combemale, B., Crégut, X., Garoche, P.L., Thirioux, X., Vernadat, F.: A property-driven approach to formal verification of process models. In: ICEIS (Selected Papers). Lecture Notes in Business Information Processing, vol. 12, pp. 286–300. Springer (2007)
5. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: Proceedings of the second workshop on Formal methods in software practice. pp. 7–15. ACM (1998)
6. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering. pp. 411–420. ACM (1999)
7. Ge, N., Nakajima, S., Pantel, M.: Efficient online analysis of accidental fault localization for dynamic systems using hidden markov model. In: Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium. p. 16. Society for Computer Simulation International (2013)
8. Ge, N., Nakajima, S., Pantel, M.: Hidden markov model based automated fault localization for integration testing. In: Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on. pp. 184–187. IEEE (2013)
9. Ge, N., Pantel, M.: Time properties verification framework for uml-marte safety critical real-time systems. In: European Conference on Modelling Foundations and Applications. pp. 352–367. Springer (2012)
10. Ge, N., Pantel, M.: Verification of synchronization-related properties for uml-marte rtes models with a set of time constraints dedicated formal semantic (2012)
11. Ge, N., Pantel, M.: Real-time property specific reduction for time petri net. In: International Workshop on Petri Nets and Software Engineering (PNSE@PetriNets). pp. 165–179 (2014)
12. Ge, N., Pantel, M., Berthomieu, B.: A flexible wcet analysis method for safety-critical real-time system using uml-marte model checker (2014)
13. Ge, N., Pantel, M., Crégut, X.: Formal specification and verification of task time constraints for real-time systems. In: Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies, pp. 143–157. Springer (2012)
14. Ge, N., Pantel, M., Crégut, X.: Time properties dedicated transformation from uml-marte activity to time transition system. ACM SIGSOFT Software Engineering Notes 37(4), 1–8 (2012)
15. Ge, N., Pantel, M., Crégut, X.: Automated failure analysis in model checking based on data mining. In: International Conference on Model and Data Engineering. pp. 13–28. Springer (2014)
16. Ge, N., Pantel, M., Crégut, X.: Probabilistic failure analysis in model validation & verification. In: International Conference on Embedded Real Time Software and Systems (ERTS) (2014)
17. Gruhn, V., Laue, R.: Patterns for timed property specifications. Electronic Notes in Theoretical Computer Science 153(2), 117–133 (2006)

18. Konrad, S., Cheng, B.H.: Real-time specification patterns. In: Proceedings of the 27th international conference on Software engineering. pp. 372–381. ACM (2005)
19. Lauer, M., Ermont, J., Boniol, F., Pagetti, C.: Latency and freshness analysis on ima systems. In: Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on. pp. 1–8. IEEE (2011)
20. Lauer, M., Ermont, J., Boniol, F., Pagetti, C.: Worst case temporal consistency in integrated modular avionics systems. In: High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on. pp. 212–219. IEEE (2011)
21. Lauer, M., Ermont, J., Pagetti, C., Boniol, F.: Analyzing end-to-end functional delays on an ima platform. In: Leveraging Applications of Formal Methods, Verification, and Validation, pp. 243–257. Springer (2010)
22. Lauer, M.: Une mthode globale pour la vrification d'exigences temps rel - Application l'Avionique Modulaire Intgre. Ph.D. thesis (juin 2012)
23. Lee, E.A., Sangiovanni-Vincentelli, A.: Comparing models of computation. In: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design. pp. 234–241. IEEE Computer Society (1997)
24. Martin, S., Minet, P.: Worst case end-to-end response times of flows scheduled with fp/fifo. In: Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on. pp. 54–54. IEEE (2006)
25. Merlin, P., Farber, D.: Recoverability of communication protocols–implications of a theoretical study. Communications, IEEE Transactions on 24(9), 1036 – 1043 (1976)
26. OMG: Unified Modeling Language^{TM}, Superstructure (Feb 2009)
27. Sloan, R.H., Buy, U.: Reduction rules for time petri nets. Acta Informatica 33(7), 687–706 (1996)