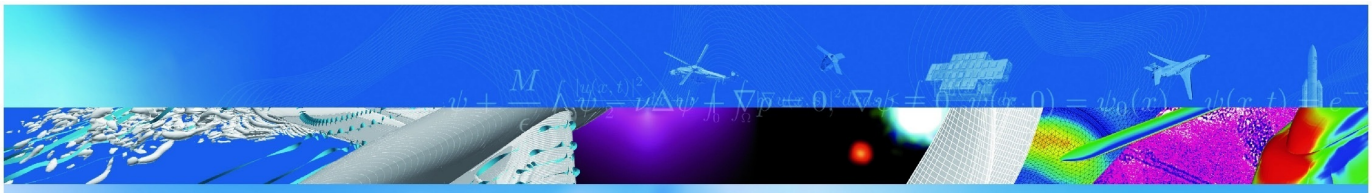
Trace analysis from ERTMS engineering

Antoine Ferlin, Simon Collart-Dutilleul, Philippe Bon, Virginie Wiels



COMMUNICATION A CONGRES

Trace analysis from ERTMS engineering

A. Ferlin, S. Collart-Dutilleul, P. Bon, V. Wiels (ONERA)

COMPRAIL

Madrid, Espagne

19-21 juillet 2016

TP 2016-525

70 2016 ans

ONERA

THE FRENCH AEROSPACE LAB

Trace analysis from ERTMS engineering

Antoine Ferlin, Simon Collart-Dutilleul, Philippe Bon, Virginie Wiels
Univ Nord de France, IFSTTAR/COSYS-ESTAS
20 rue Élisée Reclus, BP 70317
59666 Villeneuve d'Ascq, France
Onéra,
BP74025
2 avenue Edouard Belin
31055 TOULOUSE CEDEX 4, France

Abstract

Interoperability of railway systems is a major concern in the European Union. Therefore, ERTMS (European Railway Train Management System) has been defined. Nevertheless, ERTMS has to be paired with national track rules. The PERFECT project studies the implementation of ERTMS on the French territory. We use an ERTMS 3D Simulator framework which is able to simulate trains, track and communication between the trains and the RBC. The simulator records all events that occur in a sequence of states called trace. Each event is timestamped, so the sequence of events can be temporally analysed. We propose a methodology to analyse the results of a scenario performed on this platform, using temporal properties. One of the strengths of this method is the computation of statistical information on the trace during its analysis, according to a dedicated pattern of property. An automatic analysis of a trace can be used in several domains. In safety, verification of critical temporal property makes sense in ensuring that a program is compliant with its specification. An analysis of an accident or

near accident using temporal properties can provide a safety problem detection and a first level diagnosis. It can also be a interesting tool to evaluate the efficiency of an operator according to several criteria such as tiredness, expertise, repetitiveness of an event.
Keywords: Dynamic Analysis, Runtime Verification, trace, Linear Temporal Logic, statistical information, pattern.

1 Introduction

When a border is crossed by a train, the modification of the embedded signalling system generates an additional cost which is not desirable. Interoperability is vital for the competitiveness of the European railway field, and aims to provide a safe and coutinious railway system. Any rupture of train charge is allowed to ensure the performence of the railway lines. This ability is based on the fact that the train system is compliant with the essential requirements. This paper provides a feedback about the PERFECT [1] project. It aims to contribute to the validation and the implementation of the interoperable railway signalling systems, and in particular ERTMS, the European Rail Trafic Management System.

The signalling system is managed according to national rules which are proper to each European country. Therefore, the evaluation of the safety requirements is difficult. In the PERFECT project, we provide tools and methodologies based on Runtime Verification [2] to evaluate the global consistency of the specification and the operating rules from the safety point of view. Even if this problem is critical for industries, the academic world does not concentrate too much attention on it.

The project is composed of three steps. In the first one, a formal model of a part of the railway rules has been proposed. This model allows the study of the merger of national rules and European specification, in particular in a scenario of a movement authority (MA). The second one provides a methodology using existing formal tools to ensure the complience of the scenario with the specification.

This paper focuses on the last part of the project. It consists in ending the study throughout the use of the ERTMS simulator [3]. After this intruction, some related works are presented in order to position our works according to the specification language and the proposed analysis method. Then, the ERTMS simulator used by the PERFECT project is described. The

[1] `http://perfect.ifsttar.fr/Site`
[2] `http://www.runtime-verification.org/`
[3] `http://www.ersa-france.com`

following section focuses on the proposed methodology. Finally, we conclude and propose some future works.

2 Related works

Runtime Verification works are classifiable in two main categories : On-line and a posteriori methods.

On-line verification consists either in adding assertions to the programme to express the properties to be monitored during the execution, or in executing, in parallel to the programme, an automaton representing the property and taking as inputs data from the programme execution. A lot of work exists for on-line verification, especially for Java programmes and in the aspect-oriented programming community [15, 17, 14, 13, 7].

We work on a proprietary railway platform which records events on a database. Because it is not possible to interface an on-line monitoring system, we are restricted to the a posteriori verification method. In addition, because the platform can be used in a certification context, it is preferable to use an a posteriori method to replay a verification scenario as a justification. Existing works can be ordered using several criteria.

The first one is the choice of the specification language used to express the properties. It depends on the application domain, on the complexity of the properties and on the user-friendliness of the langage. Even if several porposals already exist, we defined a language tuned to the industrial needs inspired by existing works. Indeed, language such as SALT [4] and PSL [1] are complex because they multiply the different temporal operators. Other languages such as LogScope [2] are specialised to a specific context and are not compatible with our requirements. Eagle [3] and its HAWK System extension [5] propose a means to handle finite traces but are not sufficient in term of expressiveness.

The second criterion is the nature of the considered execution traces. In several existing works, complete execution traces can be obtained by listening to all the variables of the programme [16, 3]. In our case, traces are automatically generated by the simulator without any possible customization. The size of the trace depends on the scenario duration and on the number of events that occur. A way to limit the size of the handled trace is to filter the relevant information. Actually, if the property is about a type of message, the limitation of the trace to all messages with that type is a good slicing method.i

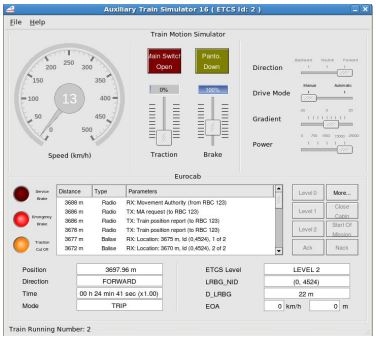
The third issue is the implementation of an efficient verification technique on the traces obtained that can be very large. Existing works are based on

rewriting techniques [13] or specific techniques, such as translation of LTL formula into state machines [12] [6]. We propose to use an existing tool called Ltl2Ba to transform LTL properties into Büchi automaton and propose specific techniques to handle sequence properties using regular expression. Then the Büchi automaton is executed to perform the analysis.

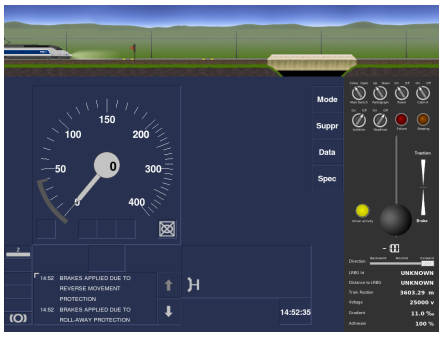
3 The ERTMS simulator

The ERTMS simulator allows the replay of a given scenario. A scenario is an illustration of a real situation. The framework is able to reproduce the railway track including the signalling systems, the Radio Block Centres (RBC), the trains and the communications between the RBC and the trains. In addition, it is possible to command one of the simulated trains as a real driver.

Figure 1b is a screen of the simulated DMI. The driver is able to switch on/off the machine, to define the identifier of the train, the phone number of the RBC, to select the ERTMS level, to receive the movement authority . . . as in a real train. The simulator is able to automatically manage up to 10 trains during a simulation. Each automated train is piloted using a specific interface presented in figure 1a. The Route Map Controller simulator is able to open a train section, define a journey on a map which synthetizes the network. This map is displayed on the screen as in figure 2.



(a) for automated trains



(b) for the driver

Figure 1: The DMI

The strength of this simulator is to provide a real time 3D simulation of what a driver sees in the cabin. This ability of the simulator is useful to consider scenario which may be irreleant or relevant for the human factor point of view. Indeed the different systems (RBC and driver cabin) can be spread over several machines and the dialogues between the driver and the authority can be reproduced as exactly as possible, as shown in figure 3.

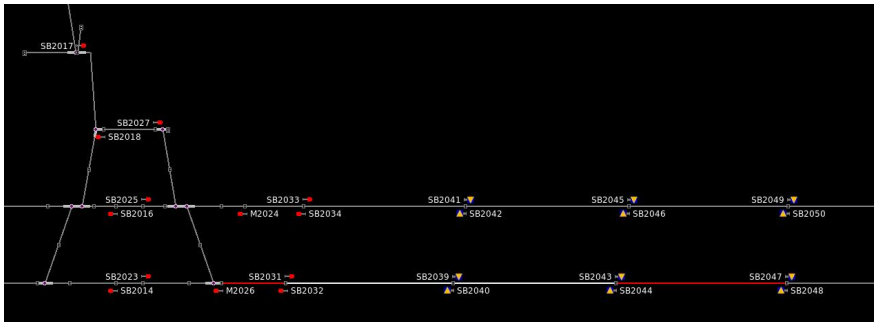


Figure 2: The Route Map Controller

Finally, one of the needs of this project is to simulate the limit of the authority of two RBCs. The solution consists in pairing two simulators across the Network. This system is still under developpment and should be used to treat scenario on a boarder between two European countries.

When a scenario is played, all events are timestamped and recorded in a database as a sequence of states. The goal is to automatically verify a temporal property on this trace. Therefore, we propose a methodology based on Runtime Verification to achieve this goal.

4 A methodology to analyse traces

The methodology used, synthesised in figure 4, is composed of three steps detailed in the next sections: definition of properties on a given scenario, generation of trace during the execution of the scenario and the automated trace analysis. The process coloured in grey is not managed by the proposed methodology. Indeed, the proprietary simulator receives as input the scenario defined for the methodology and gives the execution traces to be verified.

Figure 3: The 3D rendering

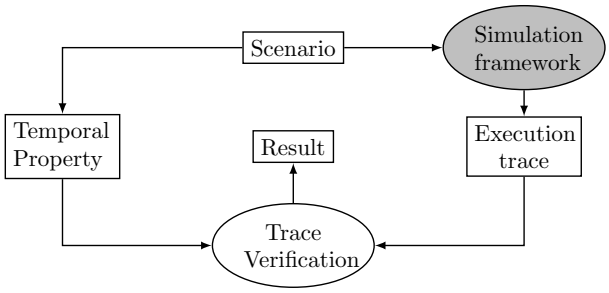


Figure 4: Overview of the methodology

4.1 Definition of the property to be verified

Propositional logic is well adapted to verify a property at a given state of a program or at a given instant. When a property has to be verified over several instants, then a temporal logic has to be used. Because we handle a sequence of states, the natural logic to use to formalise a temporal specification is a linear logic. An extended version of LTL was chosen for different reasons [10]. Indeed, the methodology was initially defined for

verifying avionics softwares. A non-publishable analysis of several avionics softwares led to the gathering of temporal properties to be verified. The approach of this analysis consisted of gathering complex properties, classifying them according to the Dwyer [8] classification, and defining the best logic to express them. During our investigation, Dwyer classification was extended to take into account specific temporal properties. Actually, some properties belonged to frequency class, duration class or a merge with Dwyer classes. Because it is possible to handle numeric variable by defining a boolean operator using numeric comparator, LTL was chosen. In addition, because definition of a sequence properties is a complex task, a regular language was aggregated with LTL. Hence, the syntax of our language is:

$$property ::= \phi | reg$$

The syntax of a LTL property follows:

$$\phi ::= \Box\ \phi | \phi_1 \mathcal{U} \phi_2 | \phi_1 \wedge \phi_2 |\ \neg\ \phi | \bot | \top | p$$

where ϕ, ϕ_i are temporal properties and p is a boolean variable or a comparison of numeric variables, $\Box$ is the *global* operator and $\mathcal{U}$ the *until* operator. When the property is written in regular expression, the syntax will be:

$$reg ::= seq | seq * | seq + | seq(b) \quad (1)$$

$$seq ::= \quad (p) list | list \quad (2)$$

$$list ::= \quad [list\ of\ p] \quad (3)$$

$$| (list\ of\ p) \quad (4)$$

A regular expression is a sequence of p (1). This sequence can be unique, repeated as much as possible or repeated until a condition is reached. A sequence can begin immediately at the beginning of the trace or after an event (2). A sequence can be stuttered (3) or not (4).

4.2 Generation of the trace to be analysed

The next step consists in executing the scenario to be analysed and gathering the information required to verify the temporal properties. The difficulty of this step is to define the data to be collected. Indeed, gathering every state of the execution is not desirable, because it can slow down the simulation and endanger temporal results. The impact of the gathering method has to be limited as much as possible to preserve the realism of the scenario in

comparison with reality. In addition, the gathering of every state during the execution of the scenario can overflow the memory capacity of a computer. Actually, there are several variables in a state and the size of the trace depends on the duration of the scenario. An execution trace can go over more than several billion of states. Therefore, a strategy has to be established to be sure that necessary states are collected. A state is necessary when an operand of the property to be verified is modified. The modification of a variable which does not depend on the property has no impact on the property.

In our case, because the ERTMS simulator is proprietary, we can express properties only on variables which are collected. As a hypothesis, the modification of each variable collected by the simulator is recorded. In previous works, the simulator was parametrisable. In other words, it collected only the required variables in defined programme point. In this case, the solution consists in using static analysis to collect all points of the programme where a variable is modified. The interested reader can find more information about this method in [9] . The impact of the trace recording on the chronology of the events is presumed to be limited for both platforms.

4.3 Automated analysis of the trace

The next step consists in verifying the properties on the generated finite execution trace. The approach is implemented in a prototype called AnTarES. In order to verify a temporal property ϕ on an execution trace σ, we use two main transformations. Firstly, the temporal formula is translated into a Büchi automaton $\mathcal{B}_\phi$. Secondly, because temporal properties have a semantics on infinite traces, the finite trace σ is transformed into an infinite one σ_∞ by looping over the last element. The transformation of the LTL properties into a Büchi automaton is ensured by [11], whereas the transformation of the regular expression is provided in [10]. We focus in this section on the way to handle infinite traces. We firstly recall the classic algorithm to analyse the main part of the trace. Then, we provide the end of trace algorithm. Finally, we use a pragmatic approach to handle border effect caused by the transformation of the finite trace.

4.3.1 Büchi automaton execution

Execution of the Büchi automaton is performed in two steps: nominal case and end of trace. The nominal case consists in computing the set of next current states by evaluating the formulas for each possible transition. The nominal case is composed of 6 steps:

1. The next state of the trace is loaded, if there is one.
2. If the next state does not exist, then the nominal case algorithm ends and the end-of-trace algorithm starts.
3. The formula of each transition for all states of the Büchi automaton in the set of current states is evaluated. If the formula is true, then the state following the transition is added to the new set of current states.
4. The new set of current states becomes the set of current states. In this set, each state only occurs once to avoid combinatorial explosion.
5. If the current state is empty, then the property is false. The nominal case algorithm ends.
6. Loop to the first step.

σ is an execution trace. $E_{\mathcal{A}}(i)$ is the set of current states of the Büchi automaton $\mathcal{A}$ at state i of σ. The initial state of $\mathcal{A}$ is singleton $I_{\mathcal{A}}$. For a given state ϵ, T_ϵ is the set of transitions with ϵ as origin. $\nu(\sigma_i, T_\epsilon)$ returns the set of next states pointed by the transition where the formula is evaluated to true.

The formalized algorithm is, for all n in $[\![0, |\sigma| - 1]\!]$:

$$E_{\mathcal{A}}(0) = I_{\mathcal{A}}; E_{\mathcal{A}}(n) = \bigcup_{\epsilon \in E_{\mathcal{A}}(n-1)} \nu(\sigma_i, T_\epsilon)$$

When there is no next state in the trace, we switch to the end of trace algorithm, defined to handle the execution of the Büchi automaton at the end of the trace. This algorithm answers the following question: is there an infinitely often accessible final state? This algorithm performs:

1. Computing the strongly connected components of the Büchi automaton, by only taking into account transitions where the formula is true.
2. Defining a direct acyclic graph (dag), equivalent to the Büchi automaton, from the strongly connected components computation.
3. Browsing the direct acyclic graph for each element of the current state.
4. Determining for each state of the dag accessed from an element of the current state if it is a final state of the Büchi automaton.

Because of the looping over the last state of the trace, the Büchi automaton is equivalent to a graph. Then verifying a property on this end of trace is equivalent to looking for a cycle in the graph with acceptant states. And finally, looking for a cycle with accepting states is equivalent to browsing the associated direct acyclic graph and to finding an equivalent accepting state during this process. Consequently, the algorithm is validated.

Pattern	$\Box\Diamond\ (P)$	$\Box\ (P \Rightarrow Q)$	$\Box\ (P \Rightarrow\Diamond\ Q)$
Metric	Number of P	Number of P	Number of P, Number of Q, min/max Number of P before an occurrence of Q

Table 1: Statistical information computed with the three patterns.

4.3.2 Computation of additional statistical information

This method allows the treatment of big traces and gives a result of an analysis depending on a temporal property. However, the expected result is only $true$ or $false$. When the property is $false$, this result is not enough, because the origin of the problem is not always obvious. Therefore, we propose to compute additional statistical information to target the origin of the property invalidation. These metrics can be used to detect if the property is $false$ because of a border effect. We propose two kins of statistical information:

- for each kind of properties, providing the number of the state and the timestamps where the error occurs,
- for some pattern of properties, providing specific metrics

In the previous non-publishable analysis, three kinds of pattern of properties were often used :

- $\Box\Diamond\ (P)$ which means that propositional property P often occurs on a trace ;
- $\Box\ (P \Rightarrow Q)$ which means that when propositional property P occurs then propositional property Q occurs at the same time ;
- $\Box\ (P \Rightarrow\Diamond\ Q)$ which means that when propositional property P occurs, propositional property Q will occur.

The interest of this approach is to provide to the verifier a quantitative aspect related to the trace and the property. Indeed, this kind of information is helpful to detect an anomaly even if the property is $true$. For instance, the fact that $\Box\ (P \Rightarrow Q)$ is $true$ does not mean that P occurs several times. Perhaps P never occurs. The statistical information ensures that P will occur a reasonable number of times. Table 1 defines the different statistical information computed, when a pattern is recognized.

5 Conclusion and future works

This position paper is focused on the last part of the PERFECT project which aims to study the ERTMS implementation on the French territory troughouth the LGV-Est line. In this last part is studied the collaboration between the formal validation and tests in an integrated approach. Tests are performed on an ERTMS railway 3D simulator which records all events of the scenario played into a trace. In this paper, we define a method to automatically analyse the execution traces. The method consists in verifying temporal properties throughout the formalism of Büchi automata. Temporal properties are expressed using LTL or regular expression for sequence properties. In addition, this approach provides a method to compute metrics about the verified trace according to a specific pattern of properties.

Experiments have to be performed on the platform to provide trace to be analysed. Firstly, we hope to analyse trace coming from a scenario that simulates a normal journey. Secondly we will focus on an accident scenario. As a future work, we propose to couple the system of metric computation to a study related to the human factors. Indeed, it is possible to study the behaviours of a driver using the 3D simulator. In addition, computation of statistical information could be performed using a specific automaton instead of a "hard encoding solution".

Acknowledgement

The work is funded by the French national research agency (ANR) in the context of the PERFECT project and RE(H)STRAIN project. PERFECT is also supported by the French I-TRANS competitive pole.

References

[1] Accellera. The accelera psl language reference manual. Technical report, Accellera, 2004.
[2] H. Barringer, A. Groce, K. Havelund, and M. Smith. Formal analysis of log files. *Journal of aerospace computing, information, and communication*, 7(11), 2010.
[3] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Eagle does space efficient ltl monitoring. Technical report, Nasa, 2003.
[4] A. Bauer, M. Leucker, and J. Streit. Salt—structured assertion language for temporal logic. In *Formal Methods and Software Engineering*, volume 4260 of *LNCS*. Springer, 2006.

[5] Marcelo d'Amorim and Klaus Havelund. Event-based runtime verification of java programs. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, May 2005.

[6] Marcelo d'Amorim and Grigore Rosu. Efficient monitoring of omega-languages. In *CAV'05*, pages 364–378, 2005.

[7] Doron Drusinsky. The temporal rover and the atg rover. In K. Havelund, J. Penix, and W. Visser, editors, *SPIN Model Checking and Software Verification*, volume 1885 of *Lecture Notes in Computer Science*. Springer, 2000.

[8] M. Dwyer, G. Avrunin, and J. Corbett. Property specification patterns for finite-state verification. In *Proceedings of the second workshop on Formal methods in software practice*, FMSP '98. ACM, 1998.

[9] A. Ferlin and V. Wiels. Combination of static and dynamic analyses for the certification of avionics software. In *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*, pages 331–336, Nov.

[10] Antoine Ferlin. *Vérification de propriétés temporelles sur des logiciels avioniques par analyse dynamique formelle*. PhD thesis, 2013. In French, Thèse de doctorat,Univ. Toulouse, ISAE.

[11] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*. Springer, 2001.

[12] D. Giannakopoulou and K. Havelund. Automata-based verification of temporal properties on running programs. In *Automated Software Engineering*, 2001.

[13] K. Havelund and G. Rosu. Monitoring programs using rewriting. In *Automated Software Engineering*, pages 135 – 143, 2001.

[14] Klaus Havelund and Kestrel Technology. A rewriting-based approach to trace analysis. *Automated Software Engineering*, 12:2005, 2002.

[15] PatrickO'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. An overview of the mop runtime verification framework. *International Journal on Software Tools for Technology Transfer*, 14:249–289, 2012.

[16] A. Pnueli and A. Zaks. Psl model checking and run-time verification via testers. In *FM 2006: Formal Methods*, volume 4085 of *LNCS*. Springer, 2006.

[17] Volker Stolz and Eric Bodden. Temporal assertions using aspectj. *Electronic Notes in Theoretical Computer Science*, 144(4):109 – 124, 2006. Proceedings of the Fifth Workshop on Runtime Verification (RV 2005).

Office National d'Études et de Recherches Aérospatiales
2 avenue Edouard Belin - BP 74025
31055 TOULOUSE Cedex 4
Tél. : +33 5 62 25 25 25
http://www.onera.fr