Efficient representation for formal verification of time performances of networked automation architectures

Silvain Ruel, Olivier de Smet, Jean-Marc Faure

Efficient representation for formal verification of time performances of networked automation architectures

S. Ruel [*] O. de Smet [*] J-M. Faure [*]

[*] *LURPA ENS de Cachan, Cachan, France, (e-mail: {silvain.ruel, olivier.de_smet, jean-marc.faure}@lurpa.ens-cachan.fr)*

Abstract: Networked automation architectures with Ethernet-based fieldbuses instead of traditional fieldbuses are more and more often used in industry, even for critical systems such as chemical or nuclear power plants. The strong safety requirements of these processes impose to evaluate the time performances of these complex architectures. Formal verification techniques are promising solutions to reach this objective. Hence, this paper focuses on the applicability of formal verification techniques to check time performances. On the basis of a case study, it is shown how formal models of networked automation architectures which are simple enough to be checked by existing timed model-checkers while yielding meaningful results can be developed.

1. INTRODUCTION

Networked automation architectures (NAA) are used in many industrial domains and even in critical industries such as chemical or power plants. Ensuring the dependability of these architectures requires not only to check their functionalities but also to evaluate their performances, and in particular their time performances.

Several methods have been developed to evaluate time performances. Some of them consist in testing the real system (measurements as in Viegas et al. [2006] or Lee and Lee [2002]). Other ones need a representation of the architecture and can therefore be used during the design phase. These "a priori", i.e. prior to implementation, methods are mainly simulation (Pereira et al. [2004], Zaitsev [2004] or Marsal [2006]), analytic methods (Worst Case evaluation Hung et al. [2004], Network Calculus Georges et al. [2002]) and timed model-checking (Witsch et al. [2006]). Model-checking is the most interesting method for critical applications because it is based on an exhaustive analysis of the state space and then seems to be the most appropriate method when safety constraints are considered.

In spite of this possibility, timed model-checking is not used to evaluate time performances of NAA in industry, because of its well-known sensitivity to combinatory explosion. Its advantage, the exhaustivity of the analysis, becomes very quickly a limitation for large-sized models.

The aim of this paper is to contribute solving of this issue. More precisely, it will be shown how to construct and analyse formal models of NAA which are simple enough to be checked by existing timed model-checkers while yielding meaningful results.

Section 2 presents the kind of automation architecture that is considered and the time performance which is focused on; a simple but non-trivial example illustrates this presentation and will be used as a case study in the remainder of the paper. The modelling and verification principles which were adopted during this study are given in section 3. Then different models of the architecture and the components are depicted respectively in sections 4 and 5, whilst section 6 explains how the verification of time performance is carried out. Finally, the results which were obtained with the different models are presented and compared in section 7.

2. NETWORKED AUTOMATION ARCHITECTURES

2.1 Considered class of architecture

Several Ethernet-based industrial solutions (Ethernet IP, Modbus TCP, Powerlink, Profinet, . . .) can be selected to implement a NAA. This paper considers only NAAs which rely on the Modbus TCP protocol, while the methodology which is presented can be applied to other kinds of networks. More precisely, focus is put on architectures where logic controllers and remote input-output modules (RIOMs) communicate to carry out automation functions; with the selected protocol, controllers are clients and RIOMs are data servers. Figure 1 shows an example of such an architecture that will be used as a case study in this paper.

The main features of the physical components of these architectures are:

- Controllers (Programmable Logical Controllers (PLCs) or industrial computers) are modular. Within each controller, a calculus processor runs a program cyclically, while a communication processor performs a periodic scanning of some RIOMs, termed I/O scanning. It matters to underline that the cycles of these two processors are asynchronous, data exchanges being made by means of a shared memory.
- The network includes Ethernet switches and Ethernet links and is dedicated only to communications between the PLCs and RIOMs; there is no other additional traffic.

- Inputs and outputs from/to the plant are gathered in RIOMs which are directly connected to the network. One RIOM may be shared by several PLCs.

Moreover, it will be assumed that there is no frame loss. This explains why probabilistic model-checking Greifeneider and Frey [2007], an interesting verification technique for architectures which require probabilistic modelling, was not considered in this work.

2.2 Example description

The architecture of the case study encompasses three PLCs and nine RIOMs. PLC 1 communicates with four RIOMs (RIOMs 1 to 4), PLC 2 communicates with five other RIOMs (RIOMs 5 to 9); thus, these two PLCs do not share any RIOM. PLC 3, which runs monitoring functions, communicates with all RIOMs and consequently shares respectively four and five RIOMs with PLC 1 and PLC 2. To evaluate the time performances of this architecture, an input event and two output events are introduced. The input event, which could correspond to a global "start" or "stop" event, is observed by two RIOMs: RIOM R1, which is scanned by PLC 1, and RIOM R5, scanned by PLC 2. The responses of the architecture to the input event, from PLC 1 and PLC 2, are respectively output1 and output2 and are sent to the plant through the same RIOMs.

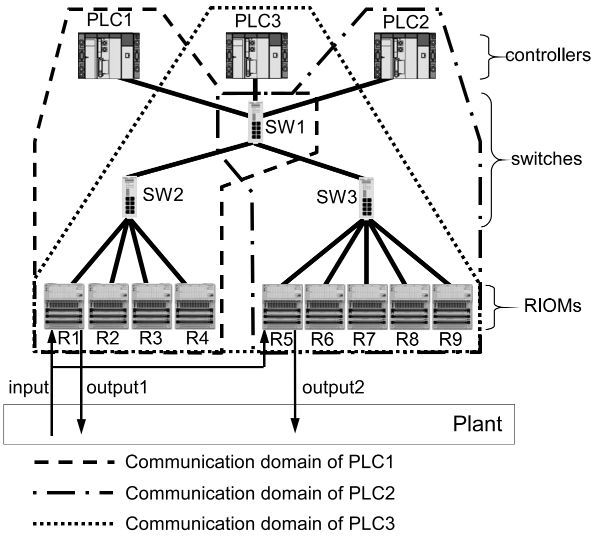

Fig. 1. Studied architecture

The configurations of the 3 PLCs are given in table 1.

Table 1. Configurations of the 3 PLCs

	PLC 1	PLC 2	PLC 3
calculation duration	2 to 3 ms	3 to 4 ms	5 to 6 ms
I/O scanning cycle time	10 ms	10 ms	50 ms
RIOMs scanned	R1 to R4	R5 to R9	R1 to R9

2.3 Required performance

There are several global time performances of a NAA, such as response time (delay between an input event and the output event which is its consequence) or the minimal duration of an input event which can be always detected by the automation architecture, for example.

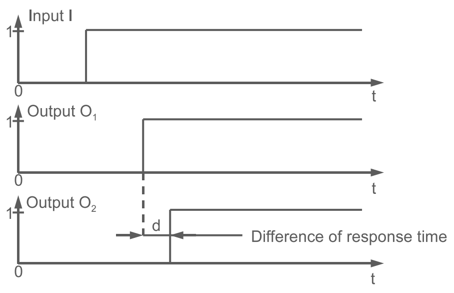

Fig. 2. Difference of response times

This paper focuses on a particular time performance: the delay between 2 output events which are both consequences of the same input event (fig. 2). This performance is worth checking when the synchronisation between two processes must be ensured; it will be called difference of response times and noted d. Hence, the property to check can be written, in a informal way, as follows:

$$d \text{ is always lower than } \tau,$$

where τ is the maximal difference of response times which ensures system dependability.

3. MODELLING PRINCIPLES

Two principles have been defined to produce and analyse formal models which are small enough to be processed by timed model-checker UPPAAL, without combinatory explosion and in reasonable times, while yielding meaningful results. The first principle consists in lessening the complexity of the models of both the architecture and the components (PLC, Network, RIOM). The second principle is to use state space approximation to speed up properties verification; such an approximation is an option of UPPAAL (section 3.2).

To study the benefits of these 2 principles on the basis of the case study, three models were designed and compared (table 2). The first model is the fine reference model on which the first principle is applied to obtain model 2. Model 3 is the consequence of the application of the second principle on model 2.

Table 2. Presentation of the models

	model 1	model 2	model 3
Architecture modelling	fine	coarse	coarse
Components modelling	fine	coarse	coarse
Time approximation	without	without	with

3.1 Overall description of the models

Model 1, called fine model, is the model which represents the NAA as precisely as possible. All the components are taken into account and the behaviour of each component is modelled in a detailed way; no state space approximation is planned to check this model.

Model 2 is called coarse model. The models of components which do not directly impact the studied performance are removed and the behaviours of the remaining components are simplified, provided that this does not reduce the confidence level in the obtained results. It matters to note that it does not make sense to use fine models of components with a coarse model of architecture because, in this case, the fine components models express behaviours which are not necessary for the coarse architecture model. Model 3 is an over-approximation of model 2, as explained below.

3.2 State space representation

Uppaal (Larsen et al. [1997]) is based on the theory of timed automata (Alur and Dill [1994], Cassandras and Lafortune [1999]) and uses a subset of CTL (Clarke and Emerson [1981]) for formal properties writing. This model-checker proposes different state space representations: with or without approximation.

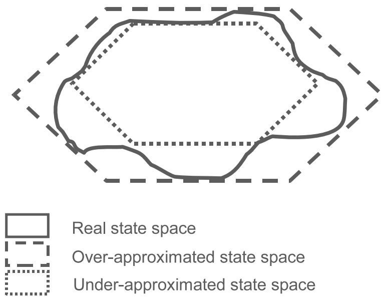

Fig. 3. State space representation

Two representations with approximation are possible: over-approximation and under-approximation. Fig. 3 shows the principles of these approximations. The approximated state space is represented by a simple model, an hexagon, which is either totally included in the real state space (under-approximation) or which contains the whole real state space (over-approximation). For room reasons, the technical details of the internal representation of state space will not be given.

For the studied time performance (the difference of response times), to guarantee the dependability of the system, it is necessary to over-estimate the value of the difference of response times. Over-approximation will thus be used instead of under-approximation.

4. ARCHITECTURE MODELS

In this section, the fine and coarse models of the architecture are sketched. An architecture model is an Uppaal model, in the form of a set of communicating timed automata. Each timed automaton, named component model, describes precisely the behaviour of one component of the architecture, e.g. a communication processor, and includes parameters that represent features of this component, such as duration of the I/O scanning cycle. The structure of an architecture model can be represented by a graph where nodes are components models and edges represent communications between these models, which are implemented by communication channels and shared variables. The choices which were made to obtain the coarse model from the fine model are also explained.

4.1 Fine model

The fine model of the architecture includes the models of all the components, in the form of timed automata. CALi and COMi are respectively the models of the calculus processor and the communication processor of PLCi. Network and RIOj are respectively the models of the network and remote input-output modules.

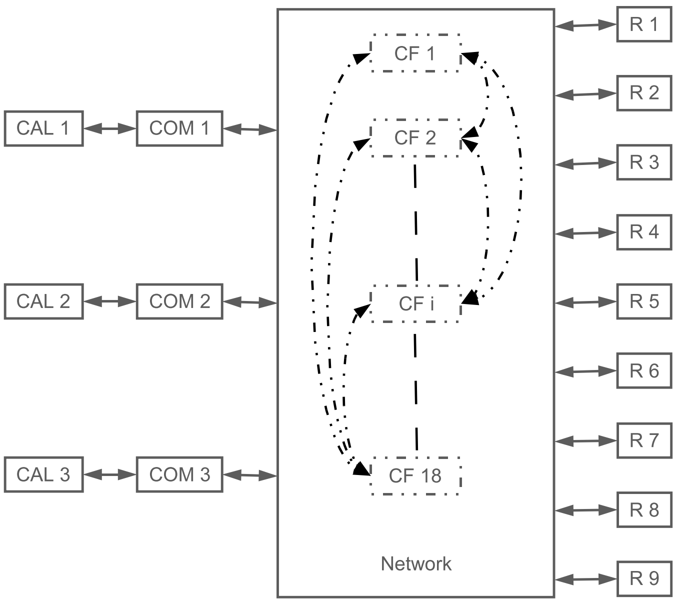

Fig. 4. Fine model of the case study

The fine model of the automation architecture is made of 16 models of components: 3 CALs, 3 COMs, 1 network and 9 RIOMs. The model of the network is very complex; it describes the communications between 3 COMs and 9 RIOMs, which corresponds to 18 "communication functions", each one modelling the communications between 1 PLC and 1 RIOM. Each model has its own clock. The clocks of models CALi and COMi evolve independently, in an asynchronous way. For the other models, this is not the case. The clock of each other model j is initialized by the reception of a message coming from another model k; hence the clocks of models j and k are synchronized.

4.2 Model simplification

To obtain the coarse model of the architecture (fig. 5) from the fine model (fig. 4), two kinds of simplification have been made, on the structure of the architecture model and on the components models.

The simplification of the structure of the architecture model consists in keeping only the components models that introduce delays (treatment times, waiting times for resource availability or synchronization) which impact directly the property to prove; all the other components models are removed. When the property to prove refers to a difference of response times, the components models that must be kept can be easily found, they are "on the route of data". This means that they generate, modify or propagate data (frames or variables) which are functions of the input or output events which the considered time performance is based on.

However, to provide meaningful verification results which can be used for safety/dependability analysis, the effects of the removed components must be included within the coarse models of the components which are kept.

For instance, the delay that is introduced by the I/O scanning of COM3 must be included in the formal model of the remaining RIOMs models, as it will be shown in section 5. The same comment applies for the delays in COM models that represent the communications with the removed RIOM models.

Moreover, previous experimental results can justify a strong simplification of the network model. This simplification is also discussed in the next section.

4.3 Coarse model

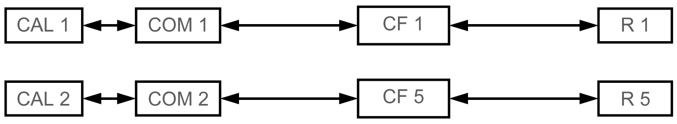

Fig. 5. Coarse model of the case study

Thus, for the property that must be checked, the coarse model includes only 8 automata: 2 CALs, 2 COMs, 2 "communication functions" that model, in an abstract way, the network, and 2 RIOMs. Each model has its own clock. These clocks behave as in the fine model; the clocks of CALi and COMi evolve independently, whereas the other ones are synchronized.

5. COMPONENTS MODELS

This section explains the simplifications which were made on the components models. Two reasons justify these simplifications:

- experimental results, regarding the network model.
- the consequences of the simplification of the architecture model, for the other components.

5.1 Network

The fine model of the network details all its internal mechanisms: queues management, switching process, possibilities of congestions, ... To ease the understanding of this complex behaviour, this model is organized in 18 "communication functions", each one depicting the communications between 1 COM and 1 RIOM. Each one of these 18 "communication functions" interacts with the other ones, as illustrated in Fig. 4.

However, results obtained by measurements (Denis et al. [2007]) on this kind of architecture showed that, in spite of this complexity, the load of the fieldbus does not impact the transmission delays. Thus, for the communication between two components, the network can be assumed perfect (no limit of throughput, no congestion) and modelled as a constant delay. This explains why the coarse model of the network includes no more interactions between the communication functions which are mere delays.

5.2 Other components

As mentioned in section 4, the effects of the components whose models have been removed from the coarse model of the architecture must be included within the coarse models of the remaining components. This will be illustrated on the basis of the RIOM models. During this study, Uppaal models have been obviously designed; however, for room reasons, only the coarse RIOM Uppaal model is shown figure 6, whereas a simplified form of the fine model is depicted in figure 7.

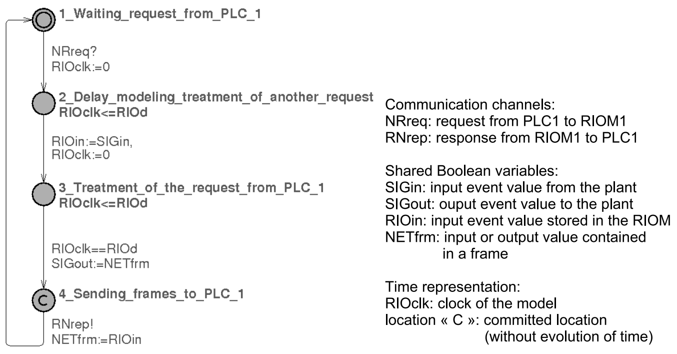
1_Waiting_request_from_PLC_1
NReq?
RIOclk:=0
2_Delay_modeling_treatment_of_another_request
RIOclk<=RIOd
RIOin:=SIGin,
RIOclk:=0
3_Treatment_of_the_request_from_PLC_1
RIOclk<=RIOd
RIOclk==RIOd
SIGout:=NETfrm
4_Sending_frames_to_PLC_1
C
RNrep!
NETfrm:=RIOin
Communication channels:
NRreq: request from PLC1 to RIOM1
RNrep: response from RIOM1 to PLC1
Shared Boolean variables:
SIGin: input event value from the plant
SIGout: ouput event value to the plant
RIOin: input event value stored in the RIOM
NETfrm: input or output value contained in a frame
Time representation:
RIOclk: clock of the model
location « C »: committed location (without evolution of time)

Fig. 6. Coarse model of a RIOM

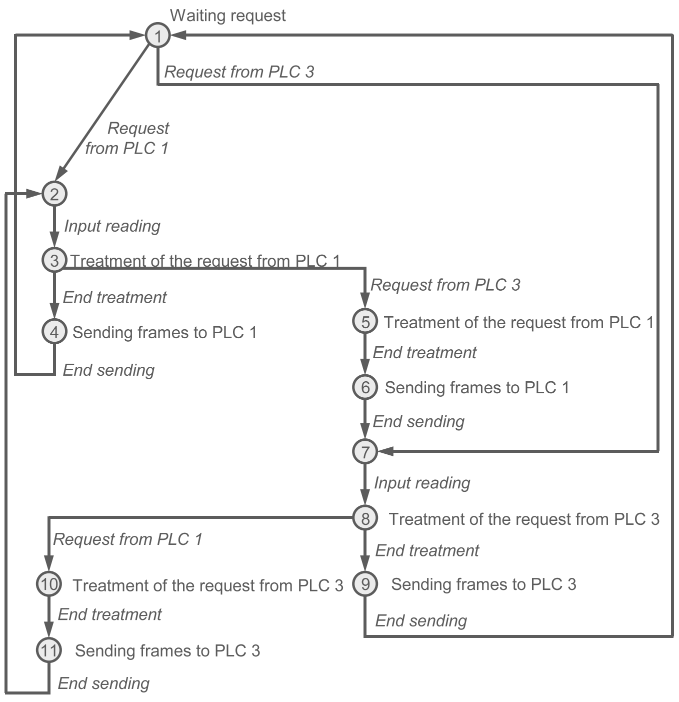
1 Waiting request
Request from PLC 3
Request from PLC 1
2
Input reading
3 Treatment of the request from PLC 1
End treatment
4 Sending frames to PLC 1
End sending
Request from PLC 3
5 Treatment of the request from PLC 1
End treatment
6 Sending frames to PLC 1
End sending
7
Input reading
8 Treatment of the request from PLC 3
Request from PLC 1
End treatment
9 Sending frames to PLC 3
End sending
10 Treatment of the request from PLC 3
End treatment
11 Sending frames to PLC 3
End sending

Fig. 7. Fine model of a RIOM

The coarse model (fig. 6) includes only 4 states and 4 transitions and is very simple, whereas the fine model (fig. 7) owns 11 states and 14 transitions and is more complex. States 1, 3 and 4 in the fine model are the same as states 1, 3 and 4 in the coarse model. States 5 to 11 are replaced by state 2 in the coarse model. State 2 (in the coarse model) models the delay due to a possible communication with another PLC, in our case, PLC 3. Thus, each request sent to the RIOM may be delayed or not by a possible communication with PLC 3, according to the current model-checker interpretation. This behaviour allows keeping the same confidence in the verification results. This model can be considered as a worst case model because each request received by a RIOM can be delayed that is not necessarily the case with the fine modelling. For example, in the case study, the I/O scanning cycle time of PLC 1 is 10 ms whereas the I/O scanning cycle time of PLC 3 is 50 ms. With the fine model, if one request coming from PLC 1 is delayed by a request coming from PLC 3, it is impossible that the four next requests would be delayed, what is not the case with the coarse model.

The other models are also simplified using the same methodology. For the CAL and COM models, the delays due to the removed RIOMs are preserved in the coarse models, for instance.

6. PROPERTY CHECKING

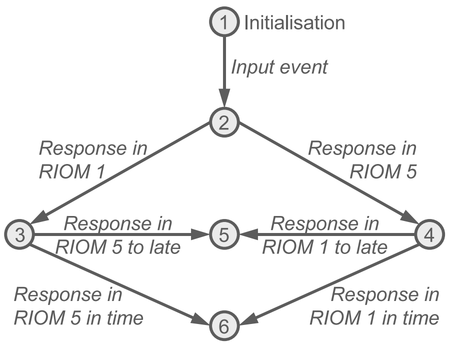
1 Initialisation
Input event
2
Response in RIOM 1
Response in RIOM 5
3
Response in RIOM 5 to late
5
Response in RIOM 1 to late
4
Response in RIOM 5 in time
Response in RIOM 1 in time
6

Fig. 8. Observer automaton

For all the models, the verification of the property (d is always lower than τ) is made thanks to an observer automaton and a property in a subset of CTL.

Fig. 8 shows a simplified representation of a model, named OBS, which generates the input event to excite the architecture and observes its evolutions. From the initial state, the input event can appear at any time (transition from state 1 to state 2). The automaton moves to state 3 if the output event appears in RIOM R1 first and to state 4 if the output event appears in RIOM R5 first.

State 5 is reached only if the 2^{nd} output event appears after the wished delay (SIGmd). In the other cases (the 2^{nd} output event appears before the wished delay), state 6 is reached. Thus the time property can be replaced by a simple reachability property: if state 5 is reached, the difference of response times is over the wished limit.

The formal property to check is therefore:

$$\text{A[] not OBS.5,}$$

i.e. for all future evolutions, state 5 is not reached. This property is evaluated to true if and only if every reachable state is different from state 5. It is interesting to note that this property is non-timed whereas a time performance is considered.

To determine the limit value of the difference of response times, it is necessary to find τ the lowest value of SIGmd for which the property is *satisfied*. This result is obtained by dichotomy, which imposes calculus iterations, each calculation being made with a different value of SIGmd. It is necessary to define a validity area for SIGmd; for the case study, the lowest limit "l" is 0 ms and the uppermost limit "u" is 100 ms. With UPPAAL it is impossible to use parameters which are not integers, thus all parameters are multiples of 10 μs. Algorithm 1 explains the method to determine τ.

```
**Data**: $l = 0$ and $u = 10000$
**Result**: $\tau$ the lowest upper bound of d
**while** $u - l > 1$ **do**
    $SIGmd = int((u - l)/2)$;
    check property;
    **if** *property is satisfied* **then**
        $u = SIGmd$
    **else**
        $l = SIGmd$
$\tau = u$
```

Algorithm 1. Dichotomy algorithm

It is interesting to note that the duration of one property checking (inside the while loop) is function of the value of SIGmd. When the value of SIGmd is lower than the lowest upper bound of d (τ), a counter-example can be find before the whole state space is explored. Thus, in that case, the result can be obtained faster than when SIGmd is higher than τ, that requires to explore the whole state space.

7. OBTAINED RESULTS

This section presents and discusses the results that were obtained with the three models (table 3). All the resolutions were made on the same computer, with a 2.8 GHz Pentium 4 processor and 4 GB of RAM, so that calculation durations could be compared.

7.1 Results with model 1

Resolution is impossible with this model, because the memory size is not sufficient to perform the whole calculation. The fine model cannot be checked.

7.2 Results with model 2

Resolution is possible by using a representation without state space approximation. Because of the resolution by dichotomy, it is necessary to make calculus iterations to obtain the result. The longest iteration, with SIGmd higher than 2140, what corresponds to 21.4 ms, lasts 28 hours. Thus, the value of τ is obtained in some days.

The property is *false* (state 5 of OBS is reached) for a value of τ of 21.4 ms and is *satisfied* (state 5 is not reached) for a value of τ over 21.4 ms. Thus, the difference of response times is always lower or equal to 21.4 ms. This value is the lowest upper bound of the difference of response time.

7.3 Results with model 3

As with the previous model, resolution is possible. The longest iteration lasts about 1 second, which is much faster than with model 2. The lowest upper bound (τ) is obtained in a few minutes.

The maximal difference of response time is estimated to 21.4 ms, which is the same value as with model 2. However, the result with approximation does not exactly have the same meaning. Indeed, the property is said *maybe satisfied* for a limit of 21.4 ms and is said *satisfied* for a limit over 21.4 ms. These results show that the difference of response times is never over 21.4 ms but there are no information about a difference of response times below this value. Then this value is an upper bound of the difference of response times but, contrary to the value obtained with model 2, it is not necessarily the lowest upper bound.

7.4 Discussion

Table 3. Comparison of the results

	fine model	coarse model	coarse model with approximation
Verification duration	impossible	28 hours	1 second
Area obtained		always $\leqq 21.4\ ms$ never $> 21.4\ ms$	never $> 21.4\ ms$

These experiments clearly highlight the impact of the size of the models. The formal verification of time performances of networked automation architectures requires to develop compact models that keep only the information that are helpful for the property that must be checked.

The comparison of the results which are obtained with model 2 and model 3 shows that:

- Model 3 leads to a very short verification time, about 100,000 times smaller than that necessary with model 2.
- Model 2 gives the lowest upper bound of the difference of response times whereas model 3 gives only an upper bound.
- In the case study, the upper bound which was given by model 3 is the lowest, but this can not be generalized for other models.

The potential difference between the upper bound which is obtained with model 3 and the lowest one is not necessarily an issue. In all cases, resolution with state space approximation can be used to determine quickly an approximate value of the performance which will be used later for a resolution without approximation. This approximate value can be seen as a limit which reduces the area to explore to find the result without approximation.

8. CONCLUSION

This paper has presented a methodology to use the formal verification techniques so as to evaluate the time performances of networked automation architectures.

This methodology is based on two main principles:

- The first one is to remove the components which have no direct impact on the studied property, i.e. only the components which are on the routes of the data that are used in the property are kept. Moreover, the models of the remaining components are simplified, while integrating the influences of removed components.
- The second one is to use state space over-approximation, which is a possibility of model-checker Uppaal. This approximation is chosen in order to keep confidence in the obtained results, which is of primary importance for safety/dependability analysis.

On the basis of these principles, three models of the same architecture were developed and tested. The comparison of the results showed the benefits of the two principles.

On-going works are aiming at extending this approach to other time performances such as the minimal duration of an input event which can be always seen by the automation architecture, and to other kinds of automation architectures.

REFERENCES

R. Alur and D. L. Dill. A theory of timed automata. 126: 183–235, 1994.

C. G. Cassandras and S. Lafortune. *Introduction to Dioscrete Event Systems.* Kluwer Academic Publishers, 1999.

E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs Workshop, Yorktown Heights, New-York*, 1981.

B. Denis, S. Ruel, J.-M. Faure, G. Marsal, and G. Frey. Measuring the impact of vertical integration on response times in Ethernet fieldbuses. In *Proc. of ETFA'07*, 2007.

J.-P. Georges, E. Rondeau, and T. Divoux. How to be sure that Ethernet networks will satisfy the real-time requirements? In *Proc. of IEEE Int. Symposium on Industrial Electronics*, July 2002.

J. Greifeneider and G. Frey. Probabilistic timed automata for modeling networked automation systems. In *proc. 1st IFAC DCDS, Cachan*, pages 143 – 148, 2007.

M.-H. Hung, J. Tsai, F.-T. Cheng, and H.-C. Yang. Development of an Ethernet-based equipment integration framework for factory automation. *Robotics and Computer-Integrated Manufacturing*, 20:369–383, 2004.

K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

K. C. Lee and S. Lee. Performance evaluation of switched Ethernet for real-time industrial communications. *Computer standards & interfaces*, (24):411–423, 2002.

G. Marsal. *Evaluation of time performances of Ethernet-based automation systems by simulation of high-level Petri nets.* PhD thesis, ENS Cachan (France) and Kaiserslautern University (Germany), December 2006.

N. Pereira, E. Tovar, and L. M. Pinho. Timeliness in COTS factory-floor distributed systems: what role for simulation? In *Proc. of 9th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, September 2004.

R. Viegas, R.A.M. Valentim, D.G. Texeira, and L.A. Guedes. Analysis of protocols to Ethernet automation networks. In *Proc. of International Joint Conference SICE-ICASE*, pages 4981–4985, 2006.

D. Witsch, B. Vogel-Heuser, J.M. Faure, and G. Poulard-Marsal. Performance analysis of industrial Ethernet networks by means of timed model-checking. In *Proc. of 12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2006)*, pages 101–106, 2006.

D. A. Zaitsev. Switched LAN simulation by colored Petri nets. *Mathematics and Computers in Simulation*, 65(3): 245–249, 2004.