Computing and Informatics, Vol. 28, 2009, 1001–1018, V 2009-Oct-4

VERIFICATION OF A FIELDBUS SCHEDULING PROTOCOL USING TIMED AUTOMATA

Nicholaos Petalidis

Department of Informatics and Communications
TEI of Serres
Terma Magnisias
62100 Serres, Greece
e-mail: n.petalidis@computer.org

Manuscript received 5 September 2007; revised 3 November 2008
Communicated by Jolana Sebestyénová

Abstract. This paper deals with the formal verification of a fieldbus real-time scheduling mechanism, using the notion of timed-automata and the UPPAAL model checker. A new approach is proposed here that treats the set of schedulers that regulate access on a fieldbus as a separate entity, called the scheduling layer. In addition a network with a changing topology is considered, where nodes may be turned on or off. The behaviour of the scheduling layer in conjunction with the data link, the medium and the network management layer is examined and it is proved that it enjoys a number of desirable properties.

Keywords: Fieldbus, formal verification, protocol verification, timed automata, UPPAAL

1 INTRODUCTION

Factory communication protocols that are designed to operate at the plant level are usually token-bus protocols built around a scheduler that distributes a token across the network's nodes. Examples of such protocols are the Foundation Fieldbus [1] and the ISA Fieldbus [2]. These fieldbus protocols, as they are called, perform time-critical transactions which must be concluded within a pre-specified time window. Fieldbuses are usually employed in hazardous environments (e.g. nuclear plants) and their error-free operation is very important.

During the previous decade a number of attempts were made to confirm that these protocols performed in accordance to their designer's intentions. However, the lack of industrial strength automated tools at the time, prompted researchers to resort to simplifications of the underlying mechanisms in the protocols, failing thus to provide convincing proofs of correctness.

For example in [3] a fieldbus scheduler was presented and analysed but only as a separate entity and the resulting verification results were valid only under normal operating conditions, i.e. with a single scheduler operating on the link. A formal approach was also taken in [4] but the timing constraints of the protocol were not taken into consideration. Similar approaches can be found elsewhere too. In [5] a simple consumer-producer scenario was analysed but again with no reference to the scheduling characteristics of the protocol.

The first attempt to analyse a fieldbus protocol with a mature modelling tool, namely UPPAAL [6, 7], was made in [8]. However, in there, emphasis was placed on the "bus coupler" and the transmission of data over the data link layer. The protocol under investigation did not include any scheduling mechanism.

A different method using UML Statecharts is reported in [9], where the PROFIsafe fieldbus protocol is specified by means of UML Statecharts and modelled using the VALID toolset. However, the protocol itself is a master-slave protocol with no token circulation. Similarly in [10] where a verification via simulation was attempted the scheduling mechanism is not taken under consideration.

In [11] a model of the hardware redundancy mechanism of the Ethernet PowerLink fieldbus protocol appears. The mechanism allows for devices to connect simultaneously to two independent links and to receive frames from both but process frames from only one. This way if one link fails, the network can still operate. The presented model is again based on timed automata. The model, however, does not include the redundancy mechanisms that relate to the link's arbiter.

A fieldbus that has attracted much attention is the Controller Area Network (CAN) protocol, and various versions of it have been analysed in a number of articles, e.g. [12, 13, 14, 15]. CANs do not have a dedicated scheduler for the link and thus their modelling does not include such a feature. Instead there is a time-master node that is responsible for maintaining a common sense of time. Of particular interest is [12] because the authors investigate the problem of modelling a system with clock drifts. The authors propose calculating the maximum drift ϵ of a clock and then offering a synchronisation within the $[t-\epsilon, t+\epsilon]$ interval in order to model the possible variance that may be present due to the existence of the drift. Unfortunately, this technique can only be realistically used when ϵ is in the same order of magnitude as t, for otherwise most modelling tools cannot handle the resulting difference between t and ϵ.

Finally in [16] another real-time network, RTnet, is modelled and its operation is verified. Although RTnet is not a true fieldbus protocol it nevertheless has a lot in common with fieldbus protocols, such as real-time transmissions with a token distributed around the network. Unlike a fieldbus, however, RTnet follows a distributed approach in scheduling real time traffic. The paper proves a number of properties that the protocol enjoys, for example that at most one node has the role of the token holder, at most one node has the role of the monitor and so on.

Similarly, here the focus is placed on a time-critical protocol but with a scheduling operation that is different from that of RTnet. Also, in order to provide the proof a different approach is proposed that treats the set of schedulers as a separate layer, the *scheduling layer*. This is in contrast with the conventional view that scheduling is just a part of the data-link layer. Furthermore, our purpose is not to simply verify the absence of deadlocks and livelocks, but to validate that the operating requirements hold even in a network with a changing topology. During this process the methodology of specifying and validating each layer is also presented. We believe that the treatment of the set of schedulers as a layer and the breadth of the approach that includes a changing topology of schedulers and data nodes constitute the novel characteristics of this paper.

The real-time scheduler adopted here is a version of the scheduler found in the Foundation Fieldbus data-link layer protocol that abstracts away from all unnecessary details like for example the exact format of the packets the scheduler transmits or receives. The modelling and validation is performed using the UPPAAL tool which itself is based on the notion of timed automata [17, 18].

2 THE UPPAAL MODELLING TOOL

The following is meant to be a brief introduction to the UPPAAL tool. The interested reader can find a more extensive tutorial on UPPAAL in [19].

UPPAAL is a set of tools that allows for the validation and verification of real-time systems.

One can use the tool to model a system using timed automata. A timed automaton is essentially a finite state machine extended to include the notion of time via the adoption of clock variables. Clocks use a dense-time notion and evaluate to a real number. More formally:

Definition 1. A *timed automaton* (TA) T is a tuple $\langle L, l_0, C, A, E, I\rangle$ where L is a finite set of locations and l_0 is the initial location. C is the set of clocks, A is a set of actions, co-actions and the internal τ action. $E \subseteq L \times A \times B(C) \times 2^C \times L$ is a set of edges between locations with an action, a guard, and a set of clocks to be reset and $I : L \rightarrow B(C)$ is a function assigning invariants to locations [19].

In the above definition, the set of possible guarding expressions typically includes conjunctions over simple conditions involving clocks or integer variables and simple relations $(<, \leq, =, \geq, >)$ to natural numbers.

Locations can be ordinary, urgent or committed. An automaton in an urgent or committed location cannot delay. Furthermore, an automaton in a committed location cannot interleave with other automata.

Timed automata may form a *network*, i.e. a system of more than one automata that communicate with each other via synchronisation actions.

Definition 2. A network of timed automata is the parallel composition $A_1|\dots|A_n$ of a set of timed automata $A_1, \dots, A_n$, called processes, combined into a single system by the CCS parallel composition operator with all external actions hidden. Synchronous communication between the processes is by hand-shake synchronisation using input and output actions; asynchronous communication is by shared variables.

Time progresses at the same pace for each automaton in the system.

UPPAAL allows for one to define a system, simulate it, verify invariants and perform reachability analysis. UPPAAL provides with a number of queries that can be performed on the system under question and prove the reachability, safety and liveness properties that the system respects. More specifically, the queries available in the verifier can be categorised as follows:

- Queries used to prove reachability properties, i.e. that there exists a path starting at the initial state, such that p is eventually satisfied along that path.
  - $E{<>}p$: there exists a path where p eventually holds
- Queries used to prove safety properties, i.e. that something bad will never happen
  - $A[\,]p$: for all paths, p always holds
  - $E[\,]p$: there exists a path where p always holds
- Queries used to prove liveness properties, i.e. that something good will eventually happen
  - $A{<>}p$: for all paths, p will eventually hold
  - $p \rightarrow q$: whenever p holds, q will eventully hold

where p and q are simple formulas with basic arithmetic and Boolean operators.

Using this language we are allowed to prove important properties that the fieldbus protocol respects, for example that as long as there is a scheduler online, a node will be offerred a token.

3 DESCRIPTION OF THE PROTOCOL

The protocol described here is a token-bus real-time protocol based on the Foundation Fieldbus data link layer protocol [1]. Nodes are connected on a bus-type network. A node can transmit on the bus if it has authorisation to do so. Authorisation to transmit is granted via the reception of a token. The token goes around in address order from one node to another. A node may request extra time within the same cycle of token circulation, in which case if time permits it will receive the token for another time, after all nodes in the link have been served at least once.

The link should not be inactive for more than a pre-specified period of time and thus nodes are forced to transmit at least every $V_{\mathrm{MRD}} < P_{\mathrm{TRD}}$ slot times (V_{ST}). V_{MRD} denotes the maximum response delay allowed to be observed and is a constant value set by the network management and P_{TRD} is the token recovery delay. The slot time is equal to twice the maximum delay in the transmission of one octet plus a safety factor. In here, it is assumed that the minimum packet size is 8 octets long. If a node does not have a packet to transmit during the next V_{MRD} slot times then it should return the token.

The particular node which is responsible for distributing the token across the other nodes of the link will be called a Link Active Scheduler (LAS), or simply an active scheduler. An active scheduler is said to hold the *scheduler token.* There can be only one LAS at any particular instant. Any node that has the necessary functionality for becoming an LAS is called a Link Master (LM). A node that can become neither an LAS nor an LM is a Data Link Entity (DLE).

In order to make the protocol robust there are mechanisms, in the event of failure of the current active scheduler, for enabling an LM to become an LAS:

i) Each LM is assigned an address i and listens to the bus. If there is no activity for a period of $i \times V_{\text{ST}}$ then the LM transmits a CL (Claim LAS) packet to claim LAS responsibilities. If again no activity is heard for another period of $i \times V_{\text{ST}}$, the LM sends the same packet again, after which it chooses a uniformly-distributed random integer in the range $(0 \ldots 3) \times V_{\text{ST}}$, and monitors the medium for that many slot-times. If no activity is again heard the sending LM assumes scheduling responsibilities and transmits a packet so that there is no more than $14 \times V_{\text{ST}}$ octet-durations of inactivity on the local link.

ii) Once an LM has become the active scheduler it passes the token, via the PT (Pass Token) packet, to the nodes of the link. A node that receives this token is said to hold the *delegated token.* When a node finishes transmitting it returns the token via a RT (Return Token) packet or an RI (Request Interval) packet. In the latter case, the node is requesting for extra time within this cycle of token circulation. Only one node at a time can hold the delegated token.

iii) An LM node may request to become an active scheduler, in which case the LAS transfers scheduling responsibilities through a TL (Transfer LAS) packet before it starts a new cycle of circulating the token. The receiving LM may choose to explicitly deny assumming responsibilities through an SR (Status Response) packet, or it may accept responsibilities by starting itself a new token circulation.

iv) When the active scheduler is awaiting for a node to reply (through an RT, RI, or SR packet) it should make sure that the link should not go idle for more than V_{IRRD} slot times. V_{IRRD} denotes the value of the *immediate response recovery delay.*

v) Finally, in order to minimise clock drifts the LAS distributes a TD (Time Distribution) packet that allows the clocks of the DLEs to be syncrhonised with the clock of the scheduler. This packet is transmitted at least once in a $0.95 \times V_{\text{TDP}}$ period, and it takes priority over any other transmission.

The default values for the variables introduced so far can be found in Table 1.

Finally, it is assumed that a node that can act as an LM receives the ON signal by the network management before it goes on-line and an OFF signal before it goes off-line.

SNODES=$\{10_{16},...,FF_{16}\}$	The set of scheduling nodes
DNODES	The set of data link entities (DLEs)
$V_{\mathrm{ST}}\in\{1,...,4\,095\}$	The slot time. Its unit is the transmission duration of one octet and its suggested value is 8. In a link with a transmission rate of 1 Mbps it is 512 μsecs
$V_{\mathrm{MRD}}\in\{V_{\mathrm{ST}},...,11\times V_{\mathrm{ST}}\}$ $P_{\mathrm{TRD}}\in\{V_{\mathrm{MRD}}+3,...,14\}$	The maximum period a token holder can go without transmitting and the associated recovery delay. Suggested values are 10 or $V_{\mathrm{MRD}}+3$.
$V_{\mathrm{IRRD}}\in\{2\times V_{\mathrm{ST}},...,12\times V_{\mathrm{ST}}\}$	The maximum period of inactivity which an active scheduler allows. Its default value is $V_{\mathrm{MRD}}+V_{\mathrm{ST}}$
V_{DTHT}	The default token holding time. Its default value is set to $278\times V_{\mathrm{ST}}$
$\Lambda_i=i\times V_{\mathrm{ST}}$, $i\in$SNODES	The maximum period of link inactivity, for which LM i waits, before claiming scheduling responsibilities
$V_{\mathrm{TDP}}\in\{5\,\mathrm{ms},55\,\mathrm{sec}\}$	The time distribution period. Determines the minimum frequency of time distribution on the local link and was set to 51.2 sec.
V_{MDC}	The minimum delay between successive network-management commands

Table 1. The parameters of the system

3.1 Assumptions

Some assumptions and simplifications were made in order to produce a model with a manageable state-space. In particular

i) Only packets that are transmitted from/to an LM or a LAS are modelled. The rest, since they are not identifiable by an LM or a LAS, are modelled as VAR packets and only their presence is detected.

ii) The RQ (Round-trip delay query) and RR (Round-trip delay response) packets are not modelled, since they are used to measure the physical characteristics of the link and these cannot be interpreted realistically on the model.

iii) Transferring of the LAS takes place after an explicit request from an LM. In this model transferring of a LAS through a TL packet takes place non-deterministically.

iv) Inclusion of the standardised request/response sequence for node activation through the transmission of PN (Probe Node), PR (Probe Response) packets increased the generated state-space prohibitevely. Instead to model the effect of a changing token circulation list, the LAS would non-deterministically choose to include or exclude nodes from this list.

v) Clock-drifts are inevitable in real life. Fieldbus devices are expected to drift in the range of μsecs for every sec. However, UPPAAL cannot model variable rate clocks. Stopwatches can be used to the same effect but that would require to include invariants that check for values in the $\{1\,\mu\text{sec}, \ldots, 51.2 \times 10^6\,\mu\text{sec}\}$ range, which again is not possible in the tool. It was proved, however, that a time synchronisation packet (TD) was transmitted at least once in the required intervals.

4 DESIGN AND METHODOLOGY

4.1 The Design of the Specification

One of the peculiarities of specifying a scheduler is that it cannot be directly thought of as representing a layer that provides services to the layer above it using the layers below it. In fact, the scheduler's functions appear to be an isolated part of the data-link layer protocol. Even the standards describe the scheduler's functionality in a separate part of the data-link layer specification. Adopting this approach during formalisation leads to faithful interpretations of the informal specification and formal models suited for deriving conforming implementations.

However, if one wants to specify the properties that the specification should honour and examine its behaviour this is not the most appropriate approach. In such a case the specification depends upon the formulation of predicates. In the case of layered protocols, these properties are part of the specification of the service definition and can be easily deduced from there. Scheduler specifications, however, do not have such service definitions, at least in the form suggested in [20] or [21]. This creates the problem that it is difficult to decide against what properties to verify the protocol.

In order to cope with this problem, a list of properties that each scheduler should have is formulated and it is considered that each scheduler provides a service to the data-link layer. In other words, it is assumed that the set of active and inactive schedulers forms a layer which provides a service to the data-link layer using the physical layer. The benefit of this view is that arguments can be made about the behaviour of the scheduling in general; the arguments are not restricted to proofs about the absence of deadlocks or livelocks in a particular scheduler node. Whereas in the classical layering of fieldbus standards the properties of the schedulers are implicit, in this view they are explicit. Thus the job of the verification engineer is made easier. An additional benefit is that by un-coupling the scheduling from the data-link layer, the process of specification and verification is modularised and thus simplified.

Another design decision made is to consider not only the interactions of the scheduler with the data-link layer but with the network management as well. In real-time protocols it is crucial to be able to argue about events which should happen. For example, it is important to insist that at any time there is an LAS operating on the link. Such a statement, though, without a qualifying assumption is usually too strong and sometimes unrealistic. Thus the previous statement would better be as follows: *at any time, provided there is at least one LM on-line, there is an LAS operating on the link*. Therefore, a scheduler is viewed as operating in two dimensions. Above and parallel to the scheduling layer there is the physical layer and vertical to the scheduling layer there is the network management layer. The network management instructs a scheduler to go either on-line or off-line. Figure 1 presents this view together with the view which conventional approaches take.

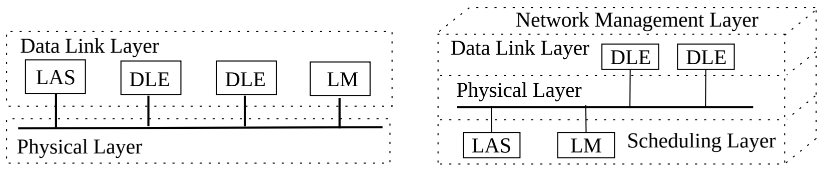

Fig. 1. The Fieldbus layers as viewed conventionally (left) and as seen here (right)

Finally, it is assumed that the functions of the lower level, such as framing, validation of packets and communication with the physical layer are provided by the medium which interconnects all the nodes of the link. A similar decomposition of functions is performed for the data-link entities of the network. Thus, the term *medium layer* will be used from now on to denote a layer with the combined functionality of the physical layer and the lower sub-level of the network's nodes.

Having a separate medium layer gives us not only better decoupling between the upper level and the lower level functions of a node but also allows us to explicitly model the primitives exchanged between the scheduling and the physical layer. In particular the scheduler needs to detect the presence of activity on the link and it does this by detecting the occurence of the primitives START, DATA and END which are issued by the physical layer. Similarly, in our model the medium layer "translates" the occurrence of a packet in the link by issuing the same primitives. Finally, a separate medium layer also allows us to explicitly model link characteristics such as packet loss or corruption. The decomposition of the layers and the packets they exchange is presented in Figure 2.

4.2 The Adopted Methodology

In order to follow a consistent method in the development of the specification for the layers presented above the following methodology was used.

The set of messages accepted by each layer was divided into two subsets: a subset Σ_{out} containing all the messages that the automaton transmits and a set Σ_{in} containing all the messages that the automaton is ready to accept. The union of

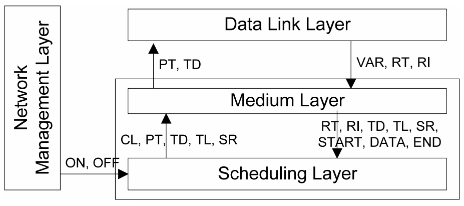

Fig. 2. Decomposition of the system under investigation

these forms the set of messages Σ of the automaton: $\Sigma = \Sigma_{\text{in}} \bigcup \Sigma_{\text{out}}$. This basically incorporates what a layer might send or receive.

After that each safety property was formulated as follows: For every message $X \in \Sigma_{\text{out}}$ a requirement of the form "if X occurred then $Y \in \Sigma$ also occurred", was added.

Similarly each liveness property was formulated as follows: If $Z \in \Sigma$ occured then X will (eventually) occur or if a timer expired then X will (eventually) occur.

Because the query language of UPPAAL only allowed to prove properties involving locations and variables but no actions, each automaton was annotated with a local variable indicating the last message received. It was then possible to express, whenever necessary, requirements in the form

$$\texttt{msg.pType} == \texttt{PT} \rightarrow \texttt{SL.lastMessage.pType} == \texttt{RT}.$$

Here a message is a structure defined as follows:[1]

```
typedef struct {
  int [PT,VAR] pType; //The type of the packet
  int [0,numberOfNodes-1] sender; //the address of the sender
  int [0,numberOfNodes-1] receiver; //the address of the receiver
  int alottedTime; // the time alotted for token usage
  int usedTime; // the time actually used
} Message;
```

After the requirements were laid out in the form of the query language of UPPAAL, each automaton was constructed separately and proved that it satisfied the properties. This initial proof was made against an "observer", i.e. an automaton that at any time is ready to synchronize with the automaton under investigation.

[1] Shared variables were used to communicate values over a single synchronisation channel.

5 THE AUTOMATA

5.1 THe Data-Link Layer

The set of messages of the data-link layer can be defined to be the set of events (packets) that it can understand:

$$\Sigma_{\mathrm{DL}} \overset{def}{=} \{\mathrm{TD}_j, \mathrm{PT}_j, \mathrm{RT}_i, \mathrm{RI}_i, \mathrm{VAR} \mid i \in \mathrm{SNODES}, j \in \mathrm{DNODES}\}.$$

The set of messages that the data link-layer generates can then be defined as follows:

$$\mathrm{Out}(\Sigma_{\mathrm{DL}}) \overset{def}{=} \{\mathrm{RI}_i, \mathrm{RT}_i, \mathrm{VAR} \mid i \in \mathrm{SNODES}\}$$

where the notation RI_i denotes a packet sent via the medium to a node with address i.

The timed automaton template representing one DLE is presented in Figure 3, and some of the formalised properties that the automaton respects can be found in Table 2. The template was parameterised by the address of each DLE.

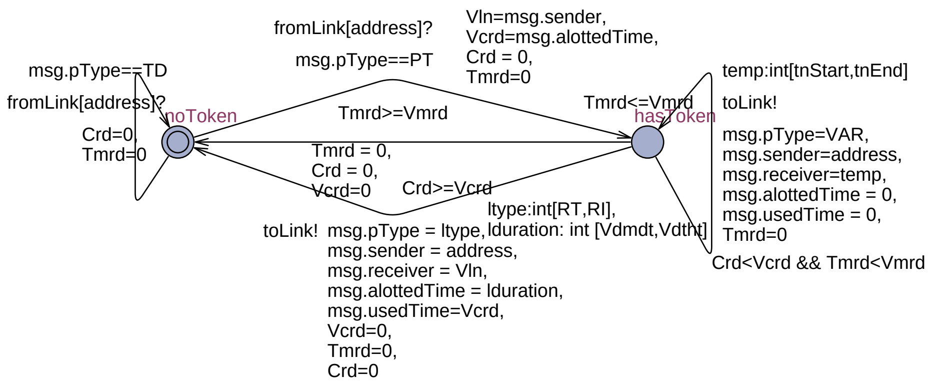

Fig. 3. The timed automaton representing the Data Link Layer

The automaton has only two states, noToken and hasToken. It can move to the hasToken state only if it receives a token. It can stay in the hasToken state transmitting VAR packets as long as it respects the timing constraints. At any time before the expiration of the token holding time it may return the token via a RT or an RI packet. While in the noToken state it may receive a TD packet.

Note that specifically for the data-link layer there was no requirement of the form *if* $Z \in \Sigma$ *occurred then* X *will (eventually) occur* because the schedulers should continue to operate even if a DLE node fails to act.

`A[] (msg.pType==RT imply (DLE1.lastMessage.pType==PT || DLE1.lastMessage.pType==VAR) && DLE1.Tmrd<=Vmrd && DLE1.Crd<=Vcrd)`	If the DLE has sent an RT packet then either the previous packet was an PT packet or a VAR packet and the transmission respected the relevant time limits.
`A[] (msg.pType==VAR imply (DLE1.lastMessage.pType==PT || DLE1.lastMessage.pType==VAR) && DLE1.Tmrd<=Vmrd && DLE1.Crd<=Vcrd)`	If the DLE has sent a VAR packet then either the previous packet was an PT packet or a VAR packet and the transmission respected the relevant time limits.

Table 2. Some basic safety properties that a DLE automaton respects

5.2 The Scheduling Layer

The set of messages of the scheduling layer can be defined to be the set of messages that it may send or receive:

$$\Sigma_{\mathrm{SL}} \overset{def}{=} \{\mathrm{ON}, \mathrm{OFF}, \mathrm{CL}, \mathrm{TL}_i, \mathrm{TD}_j, \mathrm{PT}_j, \mathrm{RT}_i, \mathrm{RI}_i, \mathrm{START}_i, \mathrm{DATA}_i, \mathrm{END}_i \mid j \in \mathrm{DNODES}, i \in \mathrm{SNODES}\}.$$

The set of messages that the scheduling layer sends can then be defined as follows:

$$\mathrm{Out}(\Sigma_{\mathrm{SL}}) \overset{def}{=} \{\mathrm{PT}_j, \mathrm{CL}, \mathrm{TL}_i, \mathrm{SR}_i, \mathrm{TD}_j \mid i \in \mathrm{SNODES}, j \in \mathrm{DNODES}\}.$$

The automaton for the scheduling layer is presented in Figure 4. The automaton can be in any of thirteen different locations, one of them being the ‘off-line’ location, which denotes that network management has instructed the automaton to turn off. There are three more locations where a scheduler lies during the arbitration procedures (LM1, LM2, acquiringToken) and three locations for the case where the scheduler circulates the token (hasSchedulerToken, monitoringPT, noSchedulerToken). While in state hasSchedulerToken the scheduler may transmit a TD or a PT packet. One more location denotes the state where the scheduler has transmitted a TL packet (transmittedTL) and one where the scheduler has received the TL packet (receivedTL). Finally the rest of the locations (not shown in the figure) are used to eliminate activity due to noise on the link and detect corrupted packets.

This automaton is parameterised by the address of the scheduler.

Some of the major properties that the scheduling layer should respect are summarised in Table 3.

5.3 The Medium Layer

The medium is basically the interface between the scheduling and the data link layer. The medium guarantees certain properties to the data link layer, provided that it can rely upon the scheduling layer for other properties and vice versa.

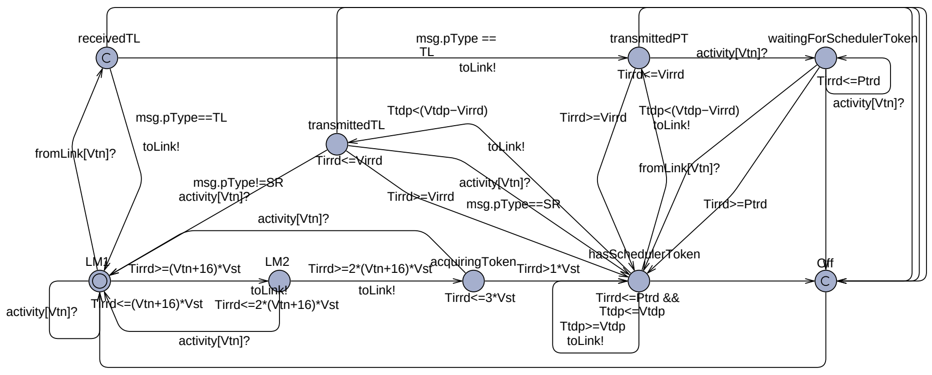

Fig. 4. The timed automaton representing an LM (simplified for presentation purposes)

`E<> msg.pType==SR && msg.sender==1 && SL2.LM1`	An LM can send an SR packet (rejecting an attempt to transfer the LAS)
`E<> SL1.hasSchedulerToken`	It is possible for an LM to acquire the scheduler token
`A[] (msg.pType==PT and msg.sender==0 and Medium.Receiving) imply (SL1.transmittedPT && SL1.Tirrd<=Ptrd)`	If a PT packet was transmitted by a scheduler, then that scheduler was in the transmittedPT state and the link did not remain inactive for more than P_{TRD} time
`A[](SL1.Tirrd>=16*Vst and Medium.Receiving) imply msg.pType==CL`	Always, if at any time there is link inactivity of more than Λ_i time then a CL packet is transmitted
`(SL1.Tirrd>=16 and msg.sender==0 and Medium.Receiving and not SL1.Off) -> msg.pType==CL`	If the link is inactive for Λ_i slot times, then a scheduler that was online during this period should offer a CL event
`msg.pType==TL && msg.receiver==1 && Medium.Receiving -> ((msg.pType==SR && msg.sender==1 && Medium.Receiving) or (msg.pType==PT && msg.sender==1 && Medium.Receiving) or (SL1.Tirrd>=Virrd && SL1.hasSchedulerToken) ||SL1.Off)`	If an attempt to transfer LAS responsibilities is made, then the receiving LM will either reject the transfer, or transmit a PT or nothing will happen and the current LAS will resume scheduling responsibilities, unless it is turned off
`SL1.hasSchedulerToken && msg.pType==RI && msg.sender==1 && SL1.Vrtha[0]>=Vdmdt -> (msg.pType==PT && msg.receiver==1 && SL1.newCycle==false && SL1.transmittedPT) || SL1.Off`	If a DLE has requested for another interval in the current cycle of token circulation then it will receive it provided that there is enough remaining time (V_{RTHA}) and that the scheduler will not be turned off
`not (SL1.LM1 || SL1.LM2 || SL1.Off || SL1.acquiringToken)-> (SL1.Ttdp==Vtdp imply msg.pType==TD)`	If a scheduler has the token then it will transmit a TD at least every V_{TDP} time

Table 3. Some basic properties of the scheduling layer

The set of messages that the medium can transmit or receive is basically the union of the messages of the data-link and the scheduling layer without the network management messages.

$$\Sigma_{\mathrm{ML}} = \Sigma_{\mathrm{DL}} \bigcup \Sigma_{\mathrm{SL}} - \{\mathrm{ON}, \mathrm{OFF}\}$$

$$\mathrm{Out}(\Sigma_{\mathrm{ML}}) \overset{def}{=} \{\mathrm{TL}_i, \mathrm{TD}_j, \mathrm{SR}_i, \mathrm{PT}_j, \mathrm{RT}_i, \mathrm{RI}_i, \mathrm{START}_i, \mathrm{DATA}_i, \mathrm{END}_i \mid j \in \mathrm{DNODES}, i \in \mathrm{SNODES}\}$$

The timed automaton for the medium is represented in Figure 5. The medium receives packets, generates the appropriate activity events to all but the originating node, and transmits packets to the appropriate destination with the appropriate time delay. TD packets are broadcasted to all nodes. A synchronisation on the toLink channel will set the fields of the shared variable msg to the appropriate values, e.g. a token returned with an RI packet might set the variable as follows:

```
msg.pType=RI; //The type of the packet
msg.sender=20; //the address of the sender
msg.receiver=16;//the address of the receiver
msg.alottedTime=556; // the time alotted for token usage
msg.usedTime=556; // the time actually used
```

The medium forces a delay in the transmission of packets of at least V_{ST} time and after that delay a synchronisation will be offered on the toLink channel and the listening node will receive the message.

The medium delivers packets by indicating a START, carrying on with three or more DATA indications and concluding with an END. The synchronisation channel Activity is used for this purpose together with a shared variable that indicates the type of activity occurring. Sometimes noise might be generated in which case a START will be followed immediately by an END indication. Finally messages may be corrupted in which case one or more DATA indications will set the variable corrupted to true.

Some basic properties that the medium layer respects are presented in Table 4. Similar properties hold for all packets transmitted through the Medium layer.

`A[] Medium.Transmitting and msg.pType==PT imply Medium.lastMessage.pType==PT and Medium1.lt>=Vst)`	If a token is transmitted, then someone has offerred it before V_{ST} time
`Medium.Receiving and msg.pType==PT -> Medium.Transmitting and msg.pType==PT and Medium.lt>=Vst`	Alternatively, if a token is offerred, then it will eventually be transmitted

Table 4. Some basic properties of the medium layer

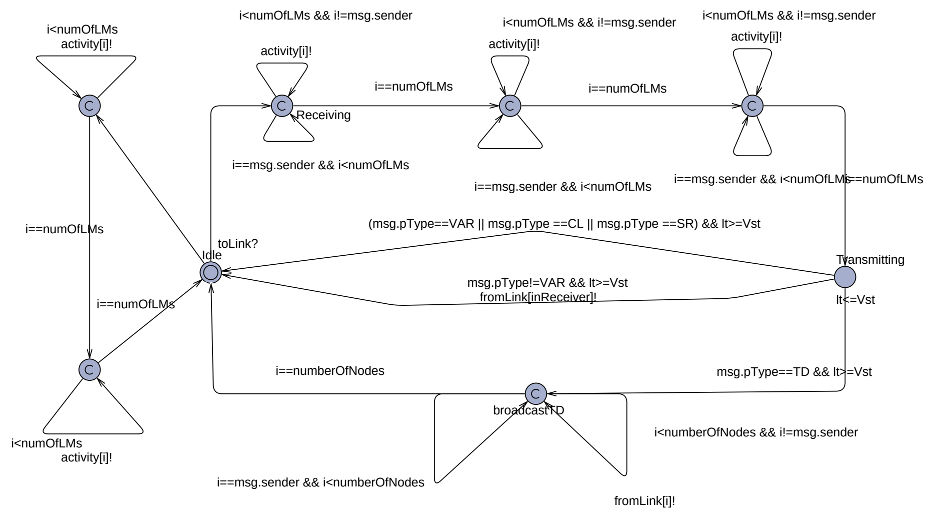
i<numOfLMs
activity[i]!
i==numOfLMs
i==numOfLMs
i<numOfLMs
activity[i]!
i<numOfLMs && i!=msg.sender
activity[i]!
Receiving
i==msg.sender && i<numOfLMs
i==numOfLMs
i<numOfLMs && i!=msg.sender
activity[i]!
i==msg.sender && i<numOfLMs
i==numOfLMs
i<numOfLMs && i!=msg.sender
activity[i]!
i==msg.sender && i<numOfLMs
i==numOfLMs
toLink?
Idle
(msg.pType==VAR || msg.pType ==CL || msg.pType ==SR) && lt>=Vst
msg.pType!=VAR && lt>=Vst
fromLink[inReceiver]!
Transmitting
lt<=Vst
i==numberOfNodes
msg.pType==TD && lt>=Vst
broadcastTD
i<numberOfNodes && i!=msg.sender
i==msg.sender && i<numberOfNodes
fromLink[i]!

Fig. 5. The timed automaton representing the Medium Layer

5.4 The Network Management Layer

The network management layer transmits only two messages and thus its set of messages is:

$$\Sigma_{\mathrm{NL}} = \{\mathrm{ON}, \mathrm{OFF}\}$$

and

$$\mathrm{Out}(\Sigma_{\mathrm{NL}}) = \Sigma_{\mathrm{NL}}.$$

The automaton itself is very simple. It consists of two states only. The automaton can turn on or off a scheduler. The only requirement is that it does not do that infinitely often, but there is a delay of V_{MDC} before turning off a scheduler.

6 FORMAL VERIFICATION

In the previous section the formal specification in the form of a timed automaton for each layer was given. Initially the automata were proved to satisfy their respective requirements. Their combined behaviour was examined next.

If there had been a separate definition for the service which the data-link layer ultimately expects from the scheduling layer, the verification process would have tested whether the combined behaviour of the scheduling and the medium provided that service. Unfortunately such a definition does not exist. In any case, it is worth investigating the service which the data-link layer can expect.

The important properties that probably constitute what can be considered to be the core of the service expected by the scheduling layer are stated informally below and their corresponding formulas can be found in Table 5.

1. There can be no deadlock on the system.
2. If there is a scheduler online, then a DLE will eventually be offered a token.
3. At anytime, at most one DLE is in possession of token.

`A[] not deadlock`
`A<> (!SL1.Off || !SL2.Off)` `imply (DLE1.hasToken || \ldots || DLE10.hasToken)`
`DLE1.hasToken -> (DLE2.noToken and \ldots and DLE10.noToken)`
`\ldots`
`DLE10.hasToken -> (DLE1.noToken and \ldots and DLE9.noToken)`

Table 5. Some basic properties that the fieldbus network respects

Although no major problems were discovered it was nevertheless necessary to interpret the informal description in order to decide how to best formalise a concept. For example the informal specification would state that *immediately* after a scheduler assumes scheduling responsibilities it will start circulating the token without clearly defining that immediately actually meant just after the minimum inter-PDU delay (V_{MID}) has expired. In addition, the specification was not clear about the timing restrictions concerning the delegation of a token immediately after the reception of a TL packet and only after cross-referencing several sections of the documentation it was possible to extract the necessary information. Furthermore, the specification did not make clear in some cases that activity resulting from noise on the link should not be taken under account.

Various network configurations were examined, with emphasis on the more common configurations of up to three LM nodes, since most fieldbus topologies rarely have more than two link masters. Models that allow for the presence of noise on the link and packet corruption were also analysed. Some of the configurations tested are presented in Table 6. Note how the resulted state-space is effected by changing the allowed values of token holding time: V_{DMDT} denotes the minumum token holding time, and when it differs from the default token holding time V_{DTHT} the range of the periods for which a node can hold the token greatly increases and subsequently both the state-space and the memory required to keep the state increase.

Regarding the verification procedure itself, several notes can be made. For a start the ability of token delegation more than once in the same cycle of token circulation to the same node greatly increased the resulted state-space of the model. The reason for this was the introduction of arrays (e.g. V_{RTHA}) that stored for each DLE the time remaining for a subsequent token delegation. Similarly, inclusion of the request/response protocol for node activation required arrays to keep track of the link's live list and increased the size of the resulted state-space. One solution was to remove the arrays from the state-space by declaring them as meta-variables where possible. This of course assummed that they were not used in guard expressions. The difference in the generated space is presented in Table 6[2]. Another technique that proved useful during validation was the explicit declaration of the allowed ranges for all integer variables. Thus, during the development of the model, any resulted out-of-bounds assignments indicated errors in our implementation which would otherwise be difficult to uncover. Finally, the allowed integer range of UPPAAL $[-32\,768, 32\,767]$ hindered the use of stopwatches for modelling clock drifts. The reason was that to model a clock drift of $i\,\mu$sec for every sec it would have been necessary to use a timer t and add an invariant such that $t \leq i$ in a location. But then when another invariant was required to say for example that a TD packet needed to be transmitted every 51.2 sec then that invariant would have to state that $t \leq 51.2 \times 10^6 \mu\,$sec; but that would have exceeded the allowed range of integer variables.

LMs	DLEs	V_{RTHA}	Corruption Noise Detection	$V_{\text{DTHT}} \neq V_{\text{DMDT}}$	States	Memory Resident/Virtual (MBytes)	Time
2	6	Y	N	N	47 299 374	829/1 670	2640.9 s
2	6	N (meta)	N	N	102 266	7.8/28.2	6.147 s
2	6	N (meta)	Y	N	230 239	104.3/246	9.730 s
2	6	N (meta)	Y	Y	1 855 548	185/424	1410.4 s
2	20	N (meta)	N	N	339 374	30.6/63.3	83.92 s
2	20	N (meta)	Y	N	854 309	52.6/109.7	144.05 s
2	20	N (meta)	Y	Y	8 201 003	818/1085	10 394.4 s
3	20	N (meta)	N	N	3 354 546	142/286	897.03 s
3	20	N (meta)	Y	N	11 469 908	378.9/765	3 462.35 s
3	20	N (meta)	Y	Y	108 642 391	1 630/2 330	86 252 s
3	30	N (meta)	N	N	4 851 107	245/496	2 369.02 s
3	30	N (meta)	Y	N	16 919 216	671.7/1 340	9 761.04 s

Table 6. Some of the configurations modelled and relative statistics proving *A[ ] not deadlock*

7 CONCLUSIONS

This paper has presented a proof for the scheduling procedures found in a number of fieldbus systems. The proof was derived by a formal model where the set of schedulers was considered as a layer that provides services to the layers above. A network with a changing topology was also assumed.

Clearly, any formal model and formally specified system relies upon a set of assumptions. The assumptions made for this model were described previously. Furthermore two other assumptions were made: that the schedulers are not continuously turned on and off and that packets (and thus transmission times) have equal length. The first hypothesis is not unreasonable and the second one can be relaxed affecting, of course, the resulted state-space. Verification was also carried out with the medium corrupting messages but it would have been more realistic to try a medium

[2] All measurements were made on a T8300 Intel Processor, with 3GB of RAM, running Windows Vista Business Ed. SP1 and UPPAAL ver. 4.1

that only corrupts messages with certain probabilities. To do that we expect to experiment in the near future with models based on weighted probabilistic timed automata.

REFERENCES

[1] Fieldbus Foundation, Ed. FF-822: Data Link Layer Protocol Specification. Fieldbus Foundation, 1995.

[2] ISA, Editor. ISA-50.02, Part 4-1997 Fieldbus Standard for Use in Industrial Control Systems. Part 4: Data Link Protocol Specification. ISA, 1997.

[3] Durante, L.—Sisto, R.—Valenzano, A.: Formal Specification and Verification of the Real-Time Scheduler In FIP. In IEEE International Workshop on Factory Communication Systems, 1995, pp. 99–106.

[4] Mariño, P.–Poza, F.–Domínguez, M. A.—Nogueira, J. B.: Link Level Formal Specification for Industrial Communication Networks. In IECON Proceedings, Vol. 1, 1998, pp. 226–231.

[5] Juanole, G.—Gallon, L.: Formal Modelling and Analysis of a Critical Time Communication Protocol. In IEEE International Workshop on Factory Communication Systems, 1995, pp. 107–115.

[6] Larsen, K. G.—Pettersson, P.—Yi, W.. UPPAAL in a Nutshell. Int. Journal on Software Tools for Technology Transfer, Vol. 1, October 1997, No. 1–2, pp. 134–152.

[7] David, A.—Behrmann, G.—Larsen, K. G.—Yi, W.: A Tool Architecture for the Next Generation of UPPAAL. In 10th Anniversary Colloquium Formal Methods at the Cross Roads: From Panacea to Foundational Support, LNCS 2003.

[8] David, A.—Yi, W.: Modelling and Analysis of a Commercial Field Bus Protocol. In 12th Euromicro Conference on Real-Time Systems, 2000, pp. 165–172.

[9] Malik, R.—Mühlfeld, R.: A Case Study in Verification of UML State-Charts: The PROFIsafe Protocol. Journal of Universal Computer Science, Vol. 9, 2003, No. 2, pp. 138–151.

[10] Brandao, S.—da Cunha, M. J.—Pinotti, M.: Fieldbus Control System Project Support Tool Based on Experimental Analysis and Modelling of Communication Bus. In IEEE International Conference on Industrial Technology, Vol. 2, 2004, pp. 787–792.

[11] Limal, S.—Potier, S.—Denis, B.—Lesage, J. J.: Formal Verification of Redundant Media Extension of Ethernet Powerlink. In IEEE Conference on Emerging Technologies and Factory Automation, September 2007, pp. 1045–1052.

[12] Rodriguez-Navas, G.—Proenza, J.—Hansson, H.: Using UPPAAL to Model and Verify a Clock Synchronization Protocol for the Controller Area Network. In 10th IEEE Conference on Emerging Technologies and Factory Automation, Vol. 2, September 2005, pp. 495–502.

[13] Leen, G.—Heffernan, D.: Modelling and Verification of a Time-Triggered Networking Protocol. In International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006, pp. 178–188.

[14] Rodriguez-Navas, G.—Proenza, J.—Hansson, H.: Modelling and Verification of Master/Slave Clock Synchronization Using Hybrid Automata and Model-Checking. In ICFEM, 2007, pp. 307–326.

[15] Bonet, M.—Donaire, G.—Proenza, J.: Modelling MajorCAN with UPPAAL. In IEEE Conference on Emerging Technologies and Factory Automation, 2007, pp. 1404–1407.

[16] Hanssen, F.—Mader, A.—Jansen, P. G.: Verifying the Distributed Real-Time Network Protocol RTnet using UPPAAL. In 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, September 2006, pp. 239–246.

[17] Alur, R.—Dill, D.: Automata for Modelling Real-Time Systems. Theoretical Computer Science, Vol. 126, April 1994, No. 2, pp. 183–236.

[18] Bengtsson, J.—Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In W. Reisig and G. Rozenberg (Eds.), Lecture Notes on Concurrency and Petri Nets, LNCS 3098. Springer-Verlag 2004.

[19] Behrmann, G.—David, A.—Larsen, K. G.: A Tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini (Eds.), Formal Methods for the Design of Real-Time Systems, 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004, LNCS, No. 3185, pp. 200–236, Springer-Verlag 2004.

[20] Lam, S. S.—Udaya, S. A.: Understanding Interfaces. In K. R. Parker and G. A. Rose (Eds.), Formal Description Techniques IV, pp. 165–184, IFIP 1992.

[21] Vissers, C.—Logrippo, L.: The Importance of the Service Concept in the Design of Data Communications Protocols. In M. Diaz (Ed.), Protocol Specification, Testing and Verification, Volume V, pp. 3–17, 1986.

Nicholaos Petalidis is a consulting software engineer. Since 2004, he has been teaching software engineering at the Technological Educational Institute of Serres, at the Department of Informatics and Communications. He received his B. Sc. degree in computer science from the University of Crete, Greece in 1994, and his Ph. D. degree from the University of Brighton, UK in 1999. His research interests include software development methodologies and formal methods for protocol specification and verification.