The Formal Specification of the Fieldbus Foundation Link Scheduler in E-LOTOS

Nicholaos Petalidis   Deshinder S. Gill

The University of Brighton
School of Engineering
Communications Research Laboratory
Brighton, BN2 4GJ, UK
E-mail: n.petalidis@computer.org
d.s.gill@brighton.ac.uk

Abstract

This paper examines the applicability of the new specification language E-LOTOS *in the description of real time applications.* E-LOTOS *is the new extended version of* LOTOS*, currently under consideration by the ISO/IEC committee. The paper presents the complete process of producing the formal specification of a real time scheduler from its informal description. During this process, a semi-formal model of the scheduler is first built that helps identify the critical parts of the scheduler. This model is then used as the basis for the formal specification of the scheduler. Finally, the degree at which* E-LOTOS *has succeeded in describing the critical parts of the scheduler is examined. The chosen, real life, paradigm is the Link Active Scheduler (LAS) of the Fieldbus Foundation (FF) Data Link Layer (DLL) protocol. Its formal specification has not been presented before.*

1. Introduction

Previous work [13] has identified the formal specification language LOTOS [5] as being particularly suited for the specification of OSI communication services. However, LOTOS lacked the necessary operators for describing the time critical aspects of real life protocols. E-LOTOS [6] is a formal specification language, currently considered by ISO/IEC for standardisation. E-LOTOS extends the capabilities of the well known formal description technique LOTOS by providing predefined data types, modularisation facilities and operators that are sensitive to time passage. The language builds upon the success of its predecessor and therefore is a promising candidate for the description of real time systems. However, the final criterion for the success of a specification language lies in its applicability to real life problems. This paper examines the suitability of E-LOTOS in specifying real life, time-critical protocols and shows how the language can be employed for their description. The chosen paradigm is the link scheduler of the Fieldbus Foundation (FF) Fieldbus protocol [4]. Fieldbus protocols are time critical protocols used for the interconnection of sensors, actuators and other control field devices at the process-control level of the factory hierarchy [11].

An informal description of the scheduler of the Fieldbus Foundation protocol is presented in section 2. This is followed by a presentation of the rationale of the formalisation process. The features of E-LOTOS are then briefly introduced in section 4. The proposed formal specification of the scheduler in E-LOTOS is found in section 5. Conclusions and pointers for future work are presented in the final section.

2. Description of the scheduler

The Fieldbus Foundation protocol uses layers seven, two and one of the OSI reference model. The lower layer uses a token bus type protocol. A specific node of the network, the Link Active Scheduler (LAS), distributes the token. There can be only one LAS at any particular instant. Any node that has the necessary functionality for becoming a LAS is called a Link Master (LM).

Any node in possession of a token has the right to transmit at that time. There are three different types of tokens:

The scheduler token

This token is held by the LAS. It can be sent to another LM to transfer the activation of LAS functions to that receiving node.

The delegated token

The LAS creates and sends this token to a node on the local link. The token carries a duration parameter, which denotes the period for which the token should be used. The token is returned upon completion of its use to the LAS or, alternatively, its return is assumed by the LAS upon its expiration.

The reply token

This is created by the current delegated or scheduler token holder when there is a need to query a node on the local link. The token is sent to that node requesting an immediate reply to the query. It is returned with that immediate reply, or acquired by the current token holder at the expiration of the reply period.

Only one type of token can be the dominant token of the link at any time. A reply token is the dominant token on the link during the period after its creation and before its expiration or return. Otherwise, when no reply token exists, a delegated token is the dominant token on the link during the period after its creation and before its expiration or return. At all other times, the scheduler token is the dominant token on the link [4].

The reply token is responsible for handling periodic traffic and it is normally sent to a node according to some schedule. The schedule contains information on the times and the period that each node needs to be queried. During the idle intervals of the periodic traffic, the delegated token is circulated among the nodes. Non-periodic events are handled using this second token. The token is circulated according to some predefined order.

The LAS is responsible for executing the schedule. This involves passing the reply token as the schedule requires, and delegating the second token around the nodes. It should make sure that no node exceeds the time assigned to it.

Moreover, the LAS is responsible for polling inactive nodes to allow for the situation when they became active. In such a case, it records them in its *Live List*. This list contains all the nodes to which the token should be passed. The LAS also acts as the *Time Master* for the bus, making sure that the clocks of all the nodes, as well as their rates of advance, are synchronised.

Since the disruption of the communication mechanism in a Fieldbus network will probably have catastrophic results, the protocol makes a provision that, in the event of a breakdown of the LAS, the bus will continue operation as normal. If, for any reason, the LAS fails, one of the LMs takes its place and becomes the new LAS. Thus, there is a need to keep a consistent record of the Live List and of the schedule across all the LMs. Hence, procedures exist for the distribution of the schedule and the Live List. The distribution of the schedule is not of concern here, as it is done via the network management protocol. The procedure of distributing the Live List usually includes a LM requesting it and the LAS replying when time permits.

Since the operations of a LM and of a LAS are closely related, the term LM/LAS will be used to denote a Data Link Layer Entity (DLE) that can function as a LM and eventually become a LAS.

3. The formalisation process

This section describes the process of formalising the scheduler's specification. Its main purpose is to identify the constituent components of the LM/LAS DLE and produce a design scenario for the formal specification, which is presented in section 5. Furthermore, it provides a list of characteristics the formal specification should respect.

The formalisation process consisted of a number of small steps that gradually changed the informal description to a formal one. During these steps, a semi-formal model of the scheduler was devised based on finite state machines (FSM). Finite state machines provided in this case a natural platform for the transition from the informal specification to the formal. This is because the informal English-language description is very similar to that of a state machine.

The data link layer of Fieldbus is divided into levels that closely parallel the partitioning of the ISO/IEC LANs into a Media Access Control (MAC) and a Logical Link Control (LLC) sublayer. The lower level is responsible for accessing the physical layer, assembling, disassembling and validating packets whilst the higher level is responsible for tasks such as the interaction with the upper layers and the ordering of packets. The first prerequisite of the formal specification would thus be to maintain this separation of interests.

The points of contact of the scheduler with the rest of the world need to be defined next. Since the scheduler is a hidden part of the data link protocol that only occasionally communicates with the network management protocol, there is need for only one visible interaction point. The scheduler also needs to interact with the lower level. However, as this interaction is hidden from the external world, the introduction of a second, invisible, contact point is necessary. Figure 1 visualises the separation of interests and the definition of interaction points: `DLM` is the visible interaction point with the network management layer and `MAC` is the hidden interaction point with the lower level.

Figure 2 presents a simple FSM where the different phases that a LM/LAS DLE goes through are depicted. In the Activation phase, the LM/LAS DLE is incapable of transmitting, unless requested to do so via a *Probe Node* Protocol Data Unit (PDU). During this phase, the necessary operations for learning the link's parameters and establishing the node's address are carried out. Since non-LM/LAS DLEs also go through this phase the objectives of the formal specification need to include the ability to re-use such

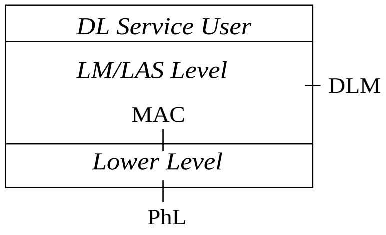

Figure 1. Structure of the scheduler

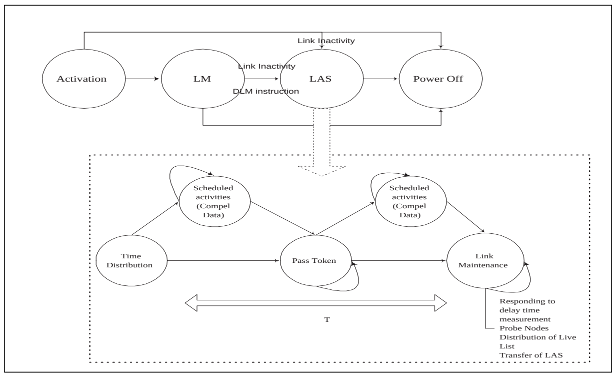

Figure 2. The phases of a LM/LAS DLE

fragments in the specification of these DLEs.

In the Link Master phase, the node listens to the network in order to update its schedule and Live List database. It also watches for the case when the link's LAS has stopped operating. In such a case, it claims LAS responsibilities and, provided the claim is successful, it becomes the link's LAS.

The LAS phase represents the core of the scheduler and draws most of the attention in the rest of this paper. When the LM/LAS DLE enters this phase it acquires the scheduler token. Figure 2 shows that the activities of the scheduler are divided into periodic ones (scheduled activities) and non-periodic ones as described in section 2. The correct timing of these activities is a major requirement of the formal specification. When a periodic activity takes place, the scheduler passes the reply token to the node that should perform this activity. The delegated token is passed by the scheduler during a non-periodic (Pass Token) activity. A careful consideration of the FSM in Figure 2 reveals that each of the periodic and non-periodic operations actually constitutes a separate protocol, independent of the operations of the other ones. Maintaining this independence in the formal specification is another prerequisite since the specification of independent modules will produce a clearer model that is easier to verify.

Analysis of the scheduler in lower levels reveals two additional important aspects. The first is that the decision for the activation of an operation is internal; it is entirely based on the current time, the contents of the transmission queue, the time taken up by each operation and the time of the next scheduled activity. All these are parameters known to the scheduler for which no external synchronisation is needed.

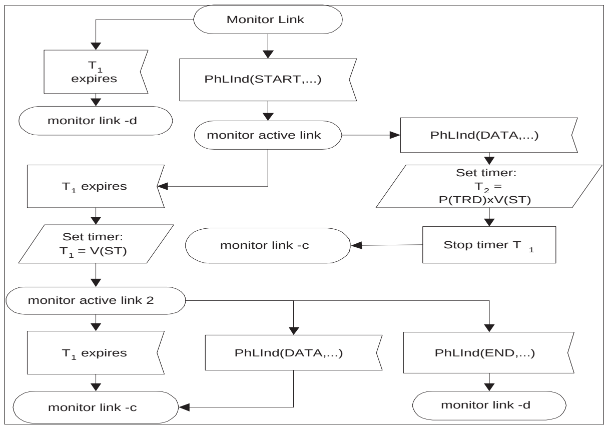

Figure 3. The monitor link-activity FSM

The second is the existence of a function, present in all of the scheduler's phases, namely the monitoring of the link's activity. Monitoring of the link is done at two levels, depending on the particular operation the LM/LAS is performing. At the higher level, the LAS monitors the reception of whole PDUs whereas at the lower level it monitors the presence of data on the link. The lower level monitoring is used for determining if a functioning LAS exists and if the node that currently holds the token is alive. Figure 3 presents the semi-formal flowchart produced for the description of the link monitoring activity. Monitoring of the link is an operation that spawns across all levels since it reports information from the physical layer back to the LM/LAS DLE. It should be therefore treated with care, as otherwise the separation of interests described in the beginning of this section will collapse. For the moment it will be considered that a separate interaction point exists, `Link`, from where information from the physical layer can be communicated. This extra interaction point should be added to the model of Figure 1 at the boundary between the LM/LAS and the lower level.

Finally, since the Fieldbus protocol is a real time protocol, its formal model should correctly specify the timing of its actions.

The previous analysis has therefore identified the following important characteristics that the formal model should consider:

- Separation of concerns
- Re-usability

- Independence of modules
- Correct timing of events

4. E-LOTOS

The features of E-LOTOS relevant to the description of the scheduler will be described here. It is assumed that the reader has some knowledge of LOTOS. Tutorials for LOTOS can be found in [1] and [9]. An extensive tutorial in E-LOTOS can be found in [6].

E-LOTOS extends the formal description technique LOTOS by providing facilities for modularisation, data typing, typed gates and time. In E-LOTOS a system is specified as a process possible made up of several interacting subprocesses each of which is a process too and may itself be divided into subprocesses. The observer of the system can be thought of as a process ready to observe any observable action. Interaction among processes takes place at communication points called *gates* or in rare occasions via *signalling*. In general a process can be thought of as being a black box communicating via a number of predefined gates.

A number of different operators enrich the language and make it suitable for the specification of distributed systems (see Table 1).

Table 1. Some of the E-LOTOS operators

Operator	Description
stop	Denotes the inactive (deadlocked) behaviour.
exit	Denotes successful termination.
; (action prefix)	$a; B$ denotes a behaviour that performs a and then behaves like B.
i	Denotes an urgent, unobservable action.
$B_1[]B_2$	Denotes the choice between two behaviours. It is usually resolved by the environment
$P_1\|[S]\|P_2$	Denotes two processes that run in parallel and synchronise at the gates present in the set S. $\|\|\|$ is another form of parallel operator that denotes true interleaving.
hide S **in** B	The gates of the set S are hidden inside the behaviour B, i.e. they are renamed to **i**.
$pname[S](D)$	Denotes the instantiation of the process $pname$ with parameters the set of gates S and the values of variables in D.

For example, a customer ($cstmr$) who waits to be served by any of three cashiers ($cash$) at some bank at the teller (tlr) can be modelled as:

$$cstmr[tlr]\ |[tlr]|\ (cash_1[tlr]|||cash_2[tlr]|||cash_3[tlr])$$

Values can be communicated or generated during a synchronisation. The expression $a?x : Natural$ denotes that at gate a a natural number may be given as an input (or generated) and the expression $a!Test$ denotes the output of the value $Test$ at gate a. Since the paper is mainly interested in the behaviour part of a real time scheduler, the single most important feature of E-LOTOS is time.

E-LOTOS has introduced a delay operator, **wait**(d), that delays the offering of an event by d time units. A time unit is of *sort* time. The time data type is a total order data type with support for addition. Furthermore actions have been augmented with an additional operator @ sensitive to time which can record the time the action took place or put constrains on the time an action will take place. The following expression will offer at gate b the time at which the synchronisation at gate a took place:

$$a@?t; b(!t)$$

A simple timeout construct can be specified as:

$$a; stop[]wait(timeout); i; b; stop$$

The time is measured from the moment the event was offered until the moment the event took place. There is no way to enforce that an observable event will take place as soon as all interested parties are ready to participate in that event. Only hidden events take place immediately. A detailed analysis of the timing features of E-LOTOS can be found in [8].

Another operator that has been extended is the disable $[>$ operator of LOTOS. In LOTOS it is possible to specify that a behaviour B_2 can interrupt at any time a behaviour B_1, unless B_1 has exited successfully. This is done by the expression:

$$B_1[> B_2$$

The E-LOTOS operator [X> does the same thing but the interrupting behaviour B_2 can raise an exception X and resume behaviour B_1. B_2 restarts after the signal is raised. The two behaviours cannot communicate via any shared variables. Moreover, when the behaviour B_2 is activated, time does not pass for behaviour B_1.

The sequential composition operator of LOTOS $>>$ has been replaced. Behaviours can be combined in E-LOTOS using the ; operator as with gates.

E-LOTOS has also support for functions. Functions are similar to processes in E-LOTOS but they can not synchronise at any gates and don't have any real time behaviour. Functions can have input and output parameters as in most programming languages.

Finally, E-LOTOS has a number of syntactic conveniences that allow for example a choice over data values and iteration. Choice is achieved via the **case endcase** construct and iterations via the **loop endloop** statement.

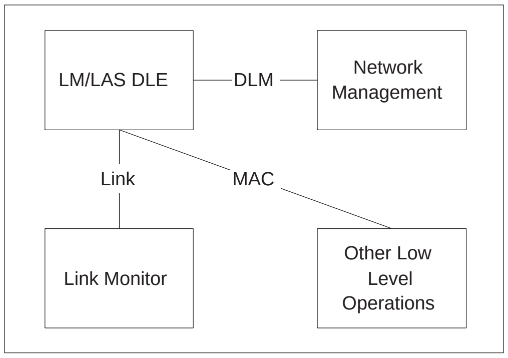

Figure 4. Top view of the specification

5. The formal specification

Figure 4 presents the external view of the LM/LAS DLE. As described in section 3 there are three gates, which are visible from other entities: `DLM`, `MAC` and `Link`. The LM/LAS DLE runs in parallel with the other modules presented in the figure, synchronising with them at the corresponding gates.

An LM/LAS DLE can enter the LAS phase at any time if no PDU is received for an extended period. In such a case, the LM/LAS DLE claims the scheduler token. The formal specification thus needs a process that measures the time elapsed from the last received PDU, claims the scheduler token and forces the LAS phase to start in case the claim is successful. Since the time at which the LAS phase must be activated is not constant but varies according to the time the last PDU was received, a combination of the parallel and disable operator is used as depicted in Figure 5.

This combination is used throughout the specification whenever a timer is needed and the expiration period of the timer is not constant. In order to highlight the use of the parallel and the disable operator the specification was kept simple and recursion was eliminated. However, it is not difficult to specify a scheduler that restarts in the LM phase whenever the scheduler token is dropped. The figure also shows the specification for the `Claim_LAS` process. Note that since the `wait` operator of E-LOTOS measures the time for which the action was available, the timer `t1` restarts after an interaction at the `MAC` gate takes place. Also note that in case of a successful claim the process offers an event at gate `Token` which forces the parallel counterpart of `Claim_LAS` to be disabled and start process `FF_LAS`. Finally, it should be noted that the `Activation` phase is kept separate and can be reused elsewhere.

The overall design of the LAS phase is presented in Figure 6. There is one process, the `Scheduler`, that controls, via gate `Switch`, the activity that will be initiated. All

```
process FF_Scheduler[DLM:DLM_SP, MAC:PDU, Link]
(Valid_Addresses: DL_Addresses, Sum:Schedule, ... )
(* a list of parameters needed by the LM/LAS DLE *)
is
 hide Token in
  (
   ( Activation[DLM, MAC]; LM[MAC] [> Token );
     FF_LAS[MAC, Link] (0, Valid_Addresses, Sum)
  )
   |[MAC, Token]|
  Claim_LAS[MAC, Token](GetNodeAddress())
 endhide
endproc (* FF_Scheduler *)

process Claim_LAS[MAC:PDU, Token](V_tn:DL_Address) is
 var
  count: Natural := 0,
  t1: Time := V_tn,
  Packet: PDU
 in loop
   ( wait(t1); i;
     MAC (?Packet) [IsCL(Packet)];
      ?count:= count + 1;
     case count is
       1 -> ?t1 := V_tn
       2 -> ?t1 := any(time) [t1 <=3]
       3 -> ?t1 := V_tn; ?count := 0; Token
      endcase
   )
   [] MAC(?Packet)
  endloop
 endvar
endproc (* Claim_LAS *)
```

Figure 5. Top level specification of the Activation and LM phase

the activities synchronise at the `MAC` gate while the activities that monitor the link will also synchronise at the `Link` gate.

Figure 7 and Figure 8 present the details of this model, whereas the corresponding specification can be found in Figure 9 and Figure 10. Note that all the activities, periodic and non-periodic, are independent from each other. New ones can easily be added as required.

Function `Next_NonPeriodic_Event` (Figure 10) decides which non-periodic activity should be initiated next. This decision does not depend on any external synchronisation, hence the use of a function instead of a process. The events at gate `Switch` control the activa-

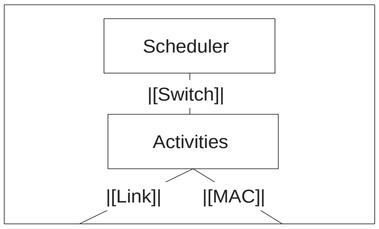

Figure 6. Decomposition of LAS phase

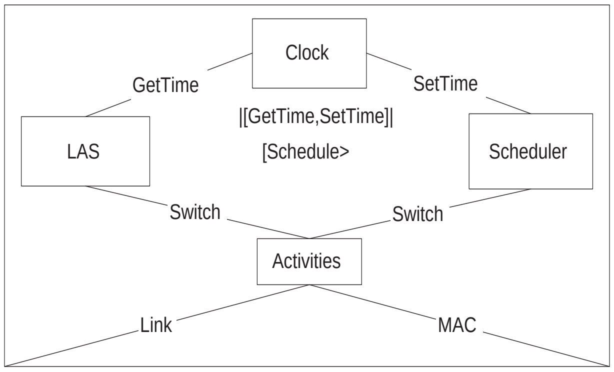

Figure 7. Decomposition of FF_LAS

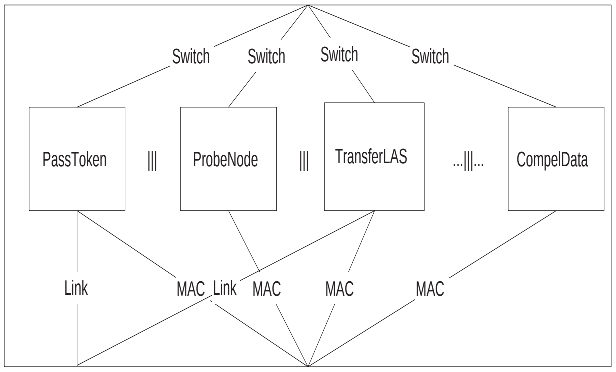

Figure 8. Decomposition of Activities

```
process FF_LAS [MAC: PDU, Link]
(Tnsa:Time, Valid_Addresses:DL_Addresses,
  Sum:Schedule)
is
 hide Switch, GetTime, SetTime in
  ( Clock[GetTime, SetTime]
    |[GetTime, SetTime]|
   ( LAS[Switch, GetTime, SetTime](Valid_Addresses,
                                          Sum, EmptyQ)
     [Scheduled_Event>
    Scheduler[Switch, SetTime](Tnsa,Sum)
   )
  )
  |[Switch]|
   Activities[MAC,Switch,Link]
 endhide
endproc (* FF_LAS *)

process Activities[MAC,Switch,Link]
is
PassToken[MAC, Switch, Link]( ... (*parameter list*))
||| SendPN[MAC,Switch] ||| SendTL[MAC,Switch,Link]
||| SendDT[MAC,Switch] ||| SendCD[MAC,Switch]
||| SendTD[MAC,Switch] ||| SendIDLE[MAC,Switch]
||| SendNothing[MAC,Switch]
endproc (*Activities*)
```

Figure 9. Specification of LAS phase (a)

tion. If, for example, the token must be delegated, then the events offered at the Switch gate will be of the form Switch(!Start,!PT_Activity) and consequently Scheduler will synchronise with the PassToken process. Since E-LOTOS adopts the maximal progress assumption for hidden events, the processes will synchronise immediately. The scheduler will then move to offer the event Switch(!End,!PT_Activity). The PassToken process, however, does not offer this event until all of the token delegation operations take place. Consequently, by measuring the time for which the Switch(!End,!PT_Activity) was offered by Scheduler, the duration of the token delegation operation can be measured. Up-to-date settings for the time can be maintained in this way.

Also, the *resume/suspend* operator, ([Scheduled_Event>) proves very useful in the specification of process FF_LAS. Whenever the time for the next scheduled activity arrives, the left-hand part (process LAS) is suspended and the scheduled activity is performed. The operator ensures that an important requirement of the informal description, the correct timing of the scheduled activities, is met. Due to the semantics of the *resume/suspend* operator time does not evolve in the left-hand part of the operator, when the right hand (process Scheduler) is activated. Thus, the new time settings must be communicated to the left hand part. Process Clock reads the time at which a scheduled activity has completed so that the left-hand side can update, when activated, its sense of time.

The specification in Figure 9 and Figure 10 might look a bit complicated and it would be helpful to see how it can model a simple execution sequence. Figure 11 presents a possible scenario for the LAS machine, taken from [12]. At time t_1 the schedule requires the LAS to instruct a node to compel data (CD). The node eventually transmits some data (DT). However, the time remaining until the next scheduled activity which takes place at t_2 is adequate to send a Probe Node (PN) PDU. The node though does not reply and there is enough time for an IDLE PDU.

The starting time t_1 can be taken to be zero (0) time units from the activation of the LAS (process FF_LAS). So, Scheduler will be activated with Tnsa (time to next scheduled activity) equal to zero and consequently it will interrupt LAS due to the presence of the internal action i. Scheduler will then synchronise with the SendCD component of the Activities process, via the Switch gate. These two processes will synchronise again at the same gate when SendCD concludes its cycle. After the necessary calculations of the value of the current time and the clock setting via SetTime, LAS will be activated by the **raise** command.

The LAS process will act similarly to Scheduler, only this time non-scheduled activities will take place.

```
process Scheduler[Switch, SetTime]
  (Tnsa:Time, ScheduleSum:Schedule)
raises [Scheduled_Event]
is
 var
   Time_now, Duration:Time
 in wait(Tnsa); i;
  Switch(!Start,!CD_Event);
   Switch(!End,!CD_Event)@?Duration;
    ?Time_now:=Tnsa+Duration;
     SetTime(!Time_now);
      ?Tnsa:= Next_Periodic_Event
          (Time_now,ScheduleSum);
       raise Scheduled_Event
 endvar
endproc (* Scheduler *)

process LAS[Switch, GetTime, SetTime]
 (ValidAddress:DL_Addresses, Sum:Schedule, Q_UR:Queue)
is
 var
  Next_Event: Event
  Time_Now, Duration: Time
 in loop
  GetTime(?Time_Now);
   ?Next_Event:=Next_NonPeriodic_Event
                              (Time_now, Sum, Q_UR);
     Switch(!Start,?An_Event,?Duration)
                                  [Event=Next_Event];
      Switch (!End, ?An_Event)
                        [Event=Next_Event]@?Duration;
       ?Time_Now:=Time_Now+Duration; SetTime(!Time_Now)
  endloop
 endvar
endproc (* LAS *)
```

Figure 10. Specification of LAS phase (b)

Overall, Scheduler will perform the following synchronisations before raising Scheduled_Event.

$$\begin{aligned} Scheduler ::\ & wait(0) \xrightarrow{0\ time\ units} i \rightarrow \\ & \rightarrow Switch(!Start, !CD\_Event) \rightarrow \\ & \xrightarrow{d_1\ time\ units} Switch(!End, !CD\_Event) \rightarrow \\ & \rightarrow SetTime(!d_1) \end{aligned}$$

Note that just before the signal is raised, $Time\_now = d_1$. After the signal is raised, LAS will be activated and the following synchronisations will take place:

$$\begin{aligned} LAS ::\ & GetTime(?Time\_Now) \rightarrow \\ & \rightarrow Switch(!Start, !PN\_Event, ?Duration) \rightarrow \\ & \xrightarrow{d_2\ time\ units} Switch(!End, !PN\_Event) \rightarrow \\ & \rightarrow SetTime(!(d_1 + d_2)) \rightarrow GetTime(?Time\_Now) \rightarrow \\ & \rightarrow Switch(!Start, !IDLE\_Event, ?Duration) \rightarrow \\ & \xrightarrow{d_3\ time\ units} Switch(!End, !IDLE\_Event) \rightarrow \\ & \rightarrow SetTime(!((d_1 + d_2) + d_3)) \end{aligned}$$

At this point the wait construct in the Scheduler process will timeout and cause the re-activation of the Scheduler. An alternative specification could have removed the

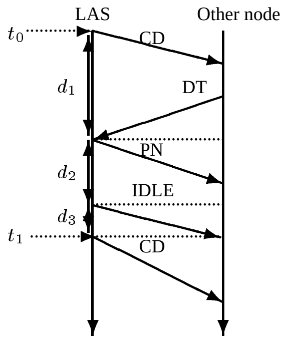

Figure 11. An example scenario for LAS

function Next_NonPeriodic_Event and kept a separate local scheduler for each activity. This local scheduler would then offer a synchronisation event when all the requirements for the instantiation of the activity were met. A global scheduler would then be responsible for choosing one of the offered events and communicating that event to the lower level. This approach is presented in Figure 12. The drawback in this approach is that E-LOTOS does not support priorities and there is no way to ensure that the most urgent activity will be performed.

Finally, Figure 13 presents the model and Figure 14 the specification of the PassToken process. The characteristic of this process is its dependence on the activity present

```
process DistributeTime[SwitchTD] (Period: Time)
is
   loop
      wait(Period);SwitchTD
   endloop
endproc (* DistributeTime *)

process PassToken[SwitchPT]
(Tpt: time, Sum: Schedule, Time_now: Time)
is
 loop
   var Tnsa: time in
    ?Tnsa := Time_To_Scheduled_Activity (Sum, Time_now);
    if Tnsa>Tpt then
     SwitchPT(!Start); SwitchPT(!End)@?Time_now
    endif
   endvar
 endloop
endproc (* PassToken *)

process Scheduler[SwitchTD,SwitchPT, ... ] is
 loop
   var Pack: PD in
      SwitchTD; MAC(?Pack)[IsTDPDU(Pack)]
       []
      SwitchPT(!Start); MAC(?Pack)[IsPTPDU(Pack)]; ...
        ;SwitchPT(!End) ...
      endvar
 endloop
endproc (* Scheduler *)
```

Figure 12. Alternative design of LAS phase

not only on the MAC gate (as in the Activation process) but on the Link gate as well.

According to the protocol, the physical layer marks the beginning of data with a PhLInd(START) primitive and the end of data with a PhLInd(END-OF-ACTIVITY) or a PhLInd(END-OF-DATA-AND-ACTIVITY) primitive. In order to eliminate the effects of noise, the reception of a PhLInd(START) and the consequent reception of a PhLInd(END-OF-ACTIVITY) with no PhLInd(DATA) in between does not constitute link activity.

To show how PassToken evolves, two different scenarios can be considered. One where the LAS passes the token and the link remains silent for the whole monitoring period and one where the LAS passes the token and the receiving node reacts before the expiration of the monitoring period.

In the first case PassToken will synchronise as follows:

$$
\begin{aligned}
PassToken ::\ & Switch(!Start, !PT\_Event, ?MaxLife) \rightarrow \\
& \rightarrow MAC(Pass\_Token\_Packet) \rightarrow \\
(now\ in\ Monitor\_Link) & \rightarrow wait(IIRD) \rightarrow \\
& \xrightarrow{IIRD\ time\ units} i \rightarrow Link(!False) \rightarrow \\
(now\ in\ PassToken) & \rightarrow Switch(!End, !PT\_Event)
\end{aligned}
$$

On the other hand, in the second case the time out construct wait(IIRD) will not expire and the sequence of synchro-

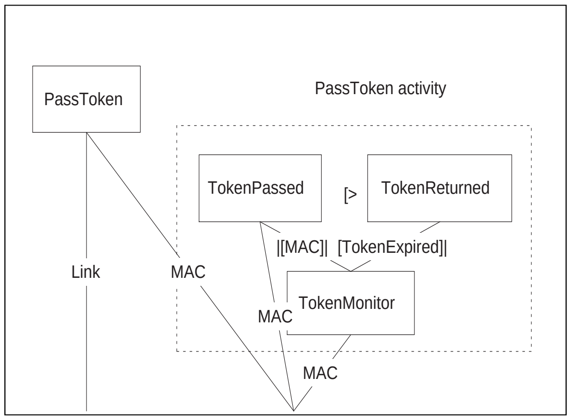

Figure 13. Design of PassToken

```
process PassToken[MAC: PDU, Switch, Link]
  (IIRD:Time, ...  (*List of relevant params here*))
is
  loop
  var
    DropToken :bool
    Packet: PDU
    MaxLife: Time, ActualLife: Time
  in
    Switch(!Start, !PT_Event, ?MaxLife);
     FormPT(MaxLife, (*other prms*), ?Packet,
                                         ?ActualLife);
      MAC(!Packet);
        ?DropToken:= Monitor_Link[Link](IIRD);
        if DropToken = false then
          Token_Traffic[MAC](ActualLife)
        endif;
        Switch(!End,!PT_Event);
  endvar
  endloop
endproc (* PassToken *)

process Monitor_Link[Link]
        (in IIRD: Time, out DropToken:Bool)
is
  var Activity: LinkActivity in
  ( wait(IIRD);i;
    Link(?Activity)[Activity != True];
    case Activity is
    False -> ?DropToken:=True
    Started-> wait(1);
          Link(?Activity)[Activity != True];
            case Activity is
              False -> ?DropToken:=True
              Started-> ?DropToken:=False
            endcase
    endcase
  ) [> Link (?Activity)[Activity = True];
      ?DropToken := False
  endvar
endproc (* Monitor_Link*)

process Token_Traffic[MAC:PDU] (ActualLife: Time) is
  hide TokenExpired in
    (TokenPassed[MAC]
           [> TokenReturned[TokenExpired])
      |[MAC, TokenExpired]|
    TokenMonitor[MAC,TokenExpired] (ActualLife)
  endhide
endproc (* Token_Traffic *)
```

Figure 14. Specification of PassToken

```
process StartPhData[PhL, Link] is
PhL(?PhLInd) [PhLInd = Start]; PhData[PhL,Link]
[] Link(!False);StartPhData[PhL, Link]
endproc

process PhData[PhL,Link] is
PhL(?PhLInd)[PhLInd = Data]; PhMoreData[PhL,Link]
[] Link(!Started);PhData[PhL,Link]
endproc

process PhMoreData[PhL,Link] is
PhL(?PhLInd)[PhLInd = Data]; PhMoreData[PhL,Link]
[] Link(!True);PhMoreData[PhL,Link]
endproc
```

Figure 15. Counterpart of `Monitor_Link`

nisations will be as follows:

$$\begin{aligned} PassToken ::\ & Switch(!Start, !PT\_Event, ?MaxLife) \rightarrow \\ & \rightarrow MAC(Pass\_Token\_Packet) \rightarrow \\ (now\ in\ Monitor\_Link) & \rightarrow Link(!True)\text{(due to } [>]) \rightarrow \\ (now\ in\ PassToken) & \rightarrow \text{call to } \texttt{Token\_Traffic}\ldots \end{aligned}$$

The link is considered inactive if the last received indication reported `END-OF-ACTIVITY` and no other indication has been received from the physical layer within a predefined interval. It is clear therefore, that the process `Monitor_Link` needs to have a direct knowledge of the events at the physical layer. On the other hand, if the process `Monitor_Link` synchronised at the `PhL` gate, the separation of levels described in Figure 1 would have collapsed. The solution is to introduce the gate `Link` at the boundary between the LM/LAS level and the lower level. `Monitor_Link` assumes the existence of another process at the lower level with which it synchronises at gate `Link`. A possible realisation of this process is presented in Figure 15. The solution might seem as over-specifying a simple problem; there is a perfectly good reason for this. On the one hand, the monitoring of the link is an observation that does not need to interact with the link since the link is unaware of its presence and it is not affected by it. This observation should have led to a simple model where the monitoring would not have been split into two processes running at different levels. On the other hand however, the theoretical model on which E-LOTOS is based, namely CCS [10], requires that *observation is equivalent to interaction*. Thus the notion of monitoring is not present in the language and it has to be explicitly modelled through the procedures described previously.

The rest of the activities present no real differences from the ones reviewed here. Process `Send_Nothing` of Figure 9 simply idles when there is not enough time to perform a non-periodic activity before the next scheduled event.

6. Conclusions

This paper has presented the E-LOTOS formal specification of the Fieldbus Foundation Link Active Scheduler. The informal description of the scheduler has first been analysed and the critical parts of the specification have been identified. The formal specification emphasised the behavioural part of the scheduler, as this part affects most the timing of the scheduler's actions.

During the analysis phase, presented in section 3, four key aspects this specification should respect have been identified.

- Separation of concerns

  The term *separation of concerns* referred to the requirement to separate the specification and the verification of the functions of the lower and the upper levels (section 3), for these levels concern themselves with different operations. The presence of the link-activity monitor has made such a separation not entirely feasible although a solution has been provided.

  The extent to which E-LOTOS has assisted to the task was limited by the language's view on *observation*. The assumption that observation and interaction are essentially the same thing, has unnecessarily increased the complexity of the resulted specification and therefore the complexity of the validation model.

- Re-usability

  The components of the resulted formal specification can certainly be employed in the specification of other parts of the Fieldbus protocol. For example, the part that specifies the Activation phase of the LM/LAS DLE (Figure 5) is the same for all the DLEs of the protocol. Furthermore, the same is true for the other components. The `CompelData` activity, for example, can be re-used in the specification of a peer DLE entity as is, with no modifications.

  Therefore, in this case E-LOTOS has managed to meet the imposed requirement.

- Independence of modules

  The independence of the different modules of the specification is a requirement for the effective validation of the model. It can certainly be argued that each of the different modules that constitute the specification is independent. For example each of the processes in `Activities` does not synchronise with the others and can therefore be considered as being independent. It can also be argued that the specification of the LM/LAS DLE is independent from the specification of, for example, the Network Management DLE (Figure 4).

However, the maximal progress assumption taken by E-LOTOS produces a different semantics for actions taking place at hidden gates and actions taking place at non-hidden gates. If the LM/LAS DLE is considered as a separate independent specification then the `DLM, MAC, Link` gates are not hidden and therefore events at these gates do not take place immediately, as they should. The model thus behaves differently when it is considered alone and when it is considered with respect to its surrounding entities, something not entirely satisfying. It seems that the maximal progress assumption should have been taken for a more broader class of events, maybe for the class of events considered in [2].

- Correct timing of events

  It appears that this requirement has also been dealt with. Although a formal validation of the scheduler has not yet taken place there are aspects of the specification that build confidence on the produced model. The first is that all the calculations related to time are performed by functions (e.g. `Next_NonPeriodic_Event`) which do not interact with the environment and therefore are not prune to deadlocks or livelocks. The proof of the correctness of such functions are a simple matter of real-time analysis. The second aspect is the presence of the powerful *resume/suspend* operator which enforces the activation of `Scheduler` and therefore the correct timing of the periodic activities.

  However, the specification could have been a lot simpler had E-LOTOS supported a prioritised choice mechanism. The *resume/suspend* operator which has been used in this case study, is not a panacea and in some cases may be too powerful. The reason is that although the expression A[X$>B$ can be used to give priority to the execution of B, it will do it in a destructive manner; A might be already executing and disrupted by the activation of B. Extra attention must be paid thus, so that non-intentional disruptions of this kind do not take place. The presence of a prioritised choice mechanism could ensure that B takes precedence over A only if A has not started execution.

It is, finally, true that no matter how powerful or not a formal language is, a major criterion for its acceptance, is the time needed to get familiarised with it and employ the full extend of its capabilities. In this respect the following two conclusions can be drawn from the project.

1. For persons having some experience in LOTOS, the learning curve for E-LOTOS is not long. The formal specification of the LAS/LM DLE was produced in less than a month, despite the shortage of E-LOTOS productivity tools. The introduction of predefined data types has also greatly reduced the time spent in the formalisation of the LM/LAS DLE. Most of the problems encountered were related to the interpretation of the informal specification.

2. Although the team was experienced in both LOTOS and process algebras it was felt that an intermediate step, where the informal specification was first mapped to a set of finite state machines, was necessary. Finite state machines provided a very good platform for the understanding of the LM/LAS DLE. In this respect the results coincide with that of the MEDAS project ([7]) where the SDL [3] formal description technique is proposed as an intermediate step to formalisation.

Validation of the specification has not yet been performed due to the lack of appropriate validation tools. Such tools are expected to appear once the language has been standardised.

References

[1] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[2] T. Bolognesi, F. Lucidi, and S. Trigila. Converging towards a timed LOTOS standard. *Computer Standards & Interfaces*, 16:87–118, 1994.

[3] CCITT. *Functional Specification and Description Language,(SDL)*, 1989.

[4] FF-94-822. *Fieldbus Specification Data Link Protocol Specification*. Fieldbus Foundation, August 1995.

[5] ISO. *ISO 8807 LOTOS A formal description technique based on the temporal ordering of observational behaviour*, 1989.

[6] I. JTC1/SC21/WG7. Final committee draft on enhancements to LOTOS. Technical report, ISO/IEC, February 1998.

[7] G. León, J. Carracedo, J. C. Moreno, J. C. Yelmo, J. C. Gil, C. Sánchez, and F. J. Carrasco. An industrial experience on development with LOTOS and SDL. *IFIP Transactions on C-Communication Systems*, 22:219–234, 1994.

[8] L. Léonard and G. Leduc. An introduction to ET-LOTOS for the description of time-sensitive systems. *Computer Networks and ISDN Systems*, 29(3):271–292, 1997.

[9] L. Logrippo, M. Faci, and M. HajHussein. Introduction to LOTOS. Learning by examples. *Computer Networks and ISDN Systems*, 23(5):325–342, 1992.

[10] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[11] E. L. Pageler. FIELDBUS: Overview, benefits and impact. In *ChemAsia/Instrument Asia International Conference*, pages 1–9, 1993.

[12] N. Petalidis. The test cases for the data link layer protocol of the Fieldbus Foundation. Technical report, University of Brighton, 1995.

[13] N. Petalidis and D. S. Gill. Modelling of a service in LOTOS and SDL - A comparison. In *First International Symposium on Communication Systems & Digital Signal Processing*, volume 1, pages 275–278, April 1998.