Journal of Universal Computer Science, vol. 6, no. 11 (2000), 1145-1163
submitted: 16/6/00, accepted: 19/9/00, appeared: 28/10/00

A Survey of Formal Methods Applied to Leader Election in IEEE 1394

Savi Maharaj
University of Stirling
FK9 4LA
United Kingdom
sma@cs.stir.ac.uk

Carron Shankland
University of Stirling
FK9 4LA
United Kingdom
ces@cs.stir.ac.uk

Abstract: We present a survey of formal specification techniques applied to the Tree Identify Protocol of the IEEE 1394 High Performance Serial Bus[1]. Specifications written in a variety of formalisms are compared with regard to a number of criteria including expressiveness, readability, standardisation, and level of analysis.
Key Words: formal methods, comparative case study, standards, networks, leader election protocol, concurrency
Category: D.2.1,F.3.1

1 Introduction

Formal Methods are increasingly becoming accepted within the system development process. There is growing recognition of the advantages of using a precise, formal description technique which is amenable to formal analysis. However, there is also a plethora of available formalisms, and it is difficult to keep up with developments or to decide which formalism to choose for a particular application. Case studies therefore have an important role to play at several stages in the development of a formal method. Initially they are useful in assessing the capabilities of a new formalism. Once the method becomes stable case studies are a convenient way to disseminate information about the method. They may also illustrate typical circumstances in which the method performs well (or perhaps in which it does not perform well), and give design patterns for solutions to certain kinds of problem.

[1] The diagram in Section 4.1 reprinted with permission from IEEE Std 1394-1995 "IEEE Standard for a High Performance Serial Bus" Copyright 1996, by IEEE. The IEEE disclaims any responsibility or liability resulting from the placement and use in the described manner.

Comparative case studies, in which different formal methods are applied to the same specification problem, are especially illuminating because the similarities and differences between the methods are thrown into sharp relief. These studies can also function as a Rosetta stone for specification languages, making it easy for the practitioners of each method to become conversant in other languages, and thereby broadening the range of techniques at their disposal. Finally, these studies can also serve as benchmarks to be used in assessing new developments in formal methods. For all these reasons, there has been considerable interest in comparative case studies, leading to such examples as the steam boiler control problem [ApL96], the invoicing case study [AAH98], and the robot production cell [LL95]. Our aim in this paper is to add to this literature with a comparative study based on specifications of part of the IEEE 1394 standard.

The IEEE 1394 Multimedia Serial Bus [IEE95] ("FireWire") is a particularly useful source of case studies for several reasons. It is an international standard in communications with a clear, widely available statement of its definition. It is an important and ubiquitous component of modern multimedia systems so that there is widespread commercial interest in its correctness: companies such as Sun Microsystems, Apple, Philips, Microsoft, Sony and many others have been involved in its development. Finally, it is a complex system with a variety of aspects which pose a challenge to the capabilities of formal description techniques.

In this paper we focus on a specific component of the FireWire system: the network leader election protocol, or Tree Identify Protocol. This protocol has been specified and analysed in a variety of formalisms, including I/O automata, E-LOTOS, and μCRL. We examine four different (groups of) specifications of this protocol, comparing the advantages and disadvantages of each approach, and assessing each specification on a number of criteria including expressiveness, degree of standardisation, means of analysis, and readability.

The paper is organised as follows: we begin by describing the system to be specified, the Tree Identify Protocol of the IEEE 1394 FireWire. In Section 3 we state the criteria by which specifications are to be assessed, and explain the method used to arrive at each assessment. In Section 4 we describe the formal specifications and assess each of them according to our criteria. We conclude by discussing the limitations of our survey and outlining our plans for further development.

2 The FireWire and its Tree Identify Protocol

The Firewire is a serial bus used to transport digitized video and audio signals within a network of multimedia systems and devices. It has a scalable architecture, and it is "hot-pluggable", meaning that devices and peripherals can easily be added or removed from the network at any time.

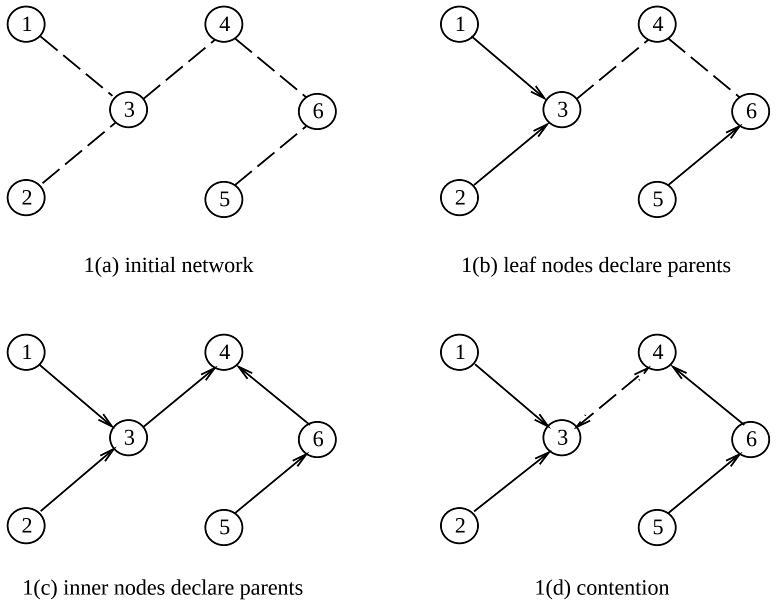

Figure 1: Some snapshots of network evolution during the Tree Identify phase

The system as a whole is complex, comprising a number of distributed communicating components and using a number of different protocols for different tasks (e.g. data transfer between nodes in the network, bus arbitration, leader election). The IEEE standard [IEE95] provides a layered description in the style of OSI, with physical, link, and transaction layers. Each layer is in turn split into different phases. In this paper we are concerned only with the Tree Identify phase of the physical layer.

In essence, the Tree Identify phase of IEEE 1394 is a leader election protocol which takes place after a bus reset in the network (i.e. when a node is added to, or removed from, the network). Immediately after a bus reset all nodes in the network have equal status, and know only to which nodes they are directly connected. A leader (root) needs to be chosen to act as the manager of the bus for subsequent phases of the 1394. The protocol is designed for use on connected networks, will correctly elect a leader if the network is acyclic, and will report an error if a cycle is detected.

The picture in Figure 1 shows a network evolving through several stages of the Tree Identify Protocol. Diagram 1(a) shows the initial network, with connections between nodes but no child/parent relations identified (shown by the dashed lines). Essentially, each node has two phases based on the number of children, c, and the number of neighbours, n. If $n - c > 1$ the node waits for "be my parent" requests from its neighbours. If $n - c = 1$ then the node sends a "be my parent" request to the neighbour which isn't a child, provided that it hasn't already received a "be my parent" request from that neighbour. In diagram 1(b) all the leaves (which by definition have $n - c = 1$ initially) have successfully requested to be the child of their only neighbour. The established child/parent link is shown as a solid line, with the arrow pointing to the parent. In diagram 1(c), nodes 3 and 6 have requested to be the children of node 4, which will then be the root of the network. The node which has $n - c = 0$ is the leader.

The protocol may not proceed this straightforwardly because "be my parent" requests are not atomic and may not be sucessful. Actually, a request must be followed by an acknowledgement, and an acknowledgement of the acknowledgement! Diagram 1(d) illustrates the situation in which node 4 has become the parent of node 6, and then *contention* has arisen as nodes 3 and 4 simultaneously send "be my parent" requests to each other. Since only one node can be the leader contention must be resolved; this is done by a scheme which uses timing. Each node chooses nondeterministically whether to wait for a long or short time. If, after the wait period is over, there is a message from the other node saying "be my parent" then the node becomes the root. If there is no such message then the node resends its own "be my parent" message (and contention may result again).

This protocol is similar to a leader election protocol presented in [Lyn96], page 501, though it uses a different method of contention resolution (in [Lyn96] the node with the highest identifier becomes the root). This approach will not work for IEEE 1394 because the nodes do not have identifiers during the Tree Identify phase. The two protocols were developed independently.

3 Criteria for Comparison

In the IEEE standard [IEE95] the Tree Identify Protocol is defined using a mixture of formal techniques (state machine diagrams and C++ code) and informal text. The protocol has also been formally specified using the methods of E-LOTOS [SV99a], I/O automata [DGRV00, GV98, Rom99, SV99b], and μCRL [SvdZ98]. In order to effectively compare all of these approaches, we must first establish some general criteria by which the specifications can be judged. We have formulated these criteria based on our own experience of formal methods, but, rather pleasingly, they turn out to be very similar to the criteria proposed by Wing [Win90] and Bowen and Hinchey [BH99].

Our list of criteria is far from exhaustive: there are many interesting criteria which are not considered in our survey, either because they could not be assessed based on the available data, or because they were not relevant to this case study. We shall say more about these in Section 5.

The criteria upon which our assessment is based are *expressiveness*, *standardisation*, *readability*, and *type of analysis*.

3.1 Expressiveness

Although the Tree Identify Protocol is a relatively small part of the IEEE 1394, there are many variables in the way in which it can be described. The criterion of expressiveness reflects the aspects of the protocol captured in each specification. The results of the comparison based on expressiveness are summarised in Figures 2 and 3 and discussed at greater length in Section 4.

The tables in Figures 2 and 3 are divided into 4 horizontal sections. Each corresponds to a different specification method: IEEE 1394 standard, I/O Automata, E-LOTOS and μCRL. There is only one description associated with the IEEE 1394 standard, therefore it only has one row in the table. The remaining sections have a number of rows: each one relates to a particular named specification in that formal method (usually covering different aspects of the protocol as described below). For example, TIP3 is an I/O Automata specification. The names are taken from the original references describing the specifications. Each specification is described in more detail in Section 4.

The columns of Figures 2 and 3 relate to particular aspects of expressiveness of the description of the protocol. A $\surd$ in a column indicates that the specification models that feature of the protocol. For example, the SyncImp description of E-LOTOS has nodes communicating synchronously, denoted by a $\surd$ in column 6 of Figure 2.

The first aspects of expressiveness we consider are the way in which the basic components of the system, and communication between them, are modelled. For example, at the most abstract level, all aspects of internal communication between nodes can be ignored and the system behaviour described as a whole (some node declares "leader" but the identity of the node is not important). This corresponds to column 1 of Figure 2. In the remainder of Figure 2 a lower level of detail is considered where the system is described by the interaction of identical communicating nodes. Columns 2 and 3 of Figure 2 relate to the means of communication, which can be concrete (voltages) or abstract (messages), respectively. Columns 4 and 5 of Figure 2 consider how the network itself is modelled; as a set of nodes and edges (column 4 of Figure 2) or more concretely as a set of devices and their ports (column 5 of Figure 2). Lastly, the communication mechanism can be synchronous or asynchronous, (columns 6 and 7 of Figure 2).

Moving on to Figure 3, columns 1, 2 and 3 relate to contention resolution. If asynchronous message passing is used then contention and its resolution must also be described. Resolution of contention for leadership can be modelled abstractly (by nondeterminism) or more accurately by adding time and/or probability measures. These three different methods of modelling contention resolution are indicated in columns 1, 2 and 3 of Figure 3 respectively.

In column 4 of Figure 3 we consider the extra complexity introduced to the system by considering real time aspects affecting system decisions. In particular, the standard mentions a parameter called FORCE_ROOT which is used to increase the probability of a particular node becoming root (by adding timing conditions to the point when a node moves from phase 1 (gaining children) to phase 2 (gaining a parent)). A $\surd$ in column 4 of Figure 3 indicates that the description successfully models the FORCE_ROOT parameter.

Lastly, columns 5, 6 and 7 of Figure 3 summarise how the different descriptions model cycle detection in the network. This is important because the presence of loops and unconnected components in the network cause the algorithm to behave incorrectly. These erroneous network configurations can be excluded from consideration by initial assumptions, indicated by column 5 of Figure 3, labelled *implicit* because the loop detection is assumed to be carried out prior to the Tree Identify Protocol. More realistically they can be explicitly checked in the specification, either by looking at network properties (column 6 of Figure 3), or by using timeouts (column 7 of Figure 3).

It should be noted that the formalism may also introduce new information which is not present in the original standard. In particular, in order to talk about the nodes in the network they must be given unique names, but these names are not part of the standard. The specifiers must be careful not to use this information unfairly by, for example, exploiting distinctions between nodes which are indistiguishable in the standard. We did not detect any instances of such unfair use.

3.2 Readability

Readability is of course a very subjective criterion, depending greatly on the reader's familiarity with the notation being used. Our main concern is with readers who are trained software engineers, with some mathematical background but with no previous experience of formal methods. We have not attempted a scientific experiment to assess readability, but have instead conducted an informal survey of five computer scientists (systems administrators and research assistants) all of whom fit the description above. These subjects were given a verbal explanation of the Tree Identify Protocol, and then shown fragments of specification in each of the four formalisms. The fragments were matched so that they covered roughly the same section of the protocol at about the same level of abstraction. The subjects then filled in a questionnaire in which they recorded how readable and understandable they found each fragment to be.

	single	messages		network		communicating nodes	
	“leader”	volts	abstract	edges	ports	synch.	asynch.
IEEE 1394		√	√		√		√
I/O Automata							
SPEC/Spec	√		√	√			
TIP			√	√			√
TIP1	√		√		√		
TIP2	√		√		√		
TIP3			√		√		√
TIP4			√		√		√
Impl			√	√			√
E-LOTOS							
Spec	√		√	√			
SyncImp			√	√		√	
AsyncImp			√	√			√
TimedImp			√	√			√
μCRL							
Spec	√		√	√			
ImpA			√	√		√	
ImpB			√	√			√

Figure 2: Different Levels of Expressivity: Messages and Communication

3.3 Standardisation

This criterion is really a property of the formal method rather than the individual specification. For each method, we ask what is the current degree of stability or standardisation of that method. This is an important consideration, because no method will achieve widespread acceptance if its definition or semantics is in a state of flux or constant revision. Standardisation may mean that the formal method is the subject of an ISO, IEEE, or other standard, or that a *de facto* standard has arisen through common usage. Stability is related, and has to do with whether the syntax or semantics of the method is still currently under development. Finally, we also look at popularity, by estimating the number of users of each method.

3.4 Analysis

For this criterion, we consider what kinds of analysis have been applied to each specification. This has two aspects: first we must determine what kinds of anal-

	contention				cycle detection?		
	nondet	time	prob	FORCE_ROOT	implicit	network	time
IEEE 1394		√	√	√			√
I/O Automata							
SPEC/Spec					√		
TIP	√				√		
TIP1						√	
TIP2						√	
TIP3	√					√	
TIP4	√						√
Impl		√	√			√	
E-LOTOS							
Spec					√		
SyncImp					√		
AsyncImp	√	√			√		
TimedImp	√	√		√			√
μCRL							
Spec					√		
ImpA					√		
ImpB	√				√		

Figure 3: Different Levels of Expressivity: Contention, Timing and Errors

ysis are made possible by each formal method, both in terms of having a theoretical foundation and in terms of the existence of tool support. Second, we also look at which of the possible analysis techniques have actually been used, and what results were obtained.

The kinds of analysis that are possible include simple syntax checking, type checking, animating or executing specifications, testing of implementations, or verification through traditional proof, interactive machine-checked proof, or fully automatic procedures such as model-checking. For each of these, and especially for the more sophisticated kinds of analysis involving proof, it is interesting to consider the question of what exactly is being proved. If a specification is shown to be "correct", how does this relate to the "real-world" correct behaviour of the Firewire? We do not attempt a complete answer to this question, but we make some observations.

4 Formal specifications of the Tree Identify Protocol

The existing specifications of the IEEE 1394 Tree Identify protocol can be grouped into four main approaches: the semi-formal approach of the standard itself [IEE95], the program-like, time-oriented approach of E-LOTOS [SV99a], the many refinements using different kinds of I/O automata [DGRV00, GV98, Rom99, SV99b], and the mathematical verification-oriented approach of μCRL [SvdZ98]. In each section below we try to give a flavour of each approach, and analyse how each performs with respect to the criteria outlined in Section 3 above.

4.1 IEEE Standard

The main description of the Tree Identify Protocol is in Section 4.4.2.2 of the IEEE standard [IEE95]. The major transitions of the protocol are described using state machine diagrams, and the actions of the states themselves are described by C++ implementations and informal text. Other information about the protocol (such as the physical realisation of messages as voltages, and timing information can be found elsewhere in Section 4 (Cable PHY Specification). There is an informative example of operation of the protocol in Annex E of the standard [IEE95].

Expressiveness of the method is measured by which aspects of the protocol are described, and at what level of abstraction. As one would expect, the standard itself covers all concrete aspects of the protocol as shown in Figures 2 and 3; however, the standard does not present more abstract levels of description of the protocol. For example, although the state machine diagram of Figure 4[2] describes the major transitions of a single node in an abstract way, the states T0, T1 and T3 are also associated with C++ code (the functions tree_id_start_actions, child_handshake_actions, and root_contend_actions). Therefore, the behaviour of the protocol cannot be understood without also understanding the accompanying C++ code, knowing the values of the timing parameters, how the network is constructed, and how the abstract names map to voltages.

The IEEE description rated highly on our readability exercise, although that was more due to the C++ code, which is certainly understandable by most programmers, than to the state machine notation. A problem of the standard is that information concerning behaviour is spread throughout the 372 page document, which makes understanding the system more difficult. Also, the lack of an abstract level of specification causes the reader to be easily confused by, for example, issues of cable and physical connector make-up, or of the services and protocols of other layers of the standard.

[2] From IEEE Std 1394-1995. Copyright 1996 IEEE. All rights reserved.

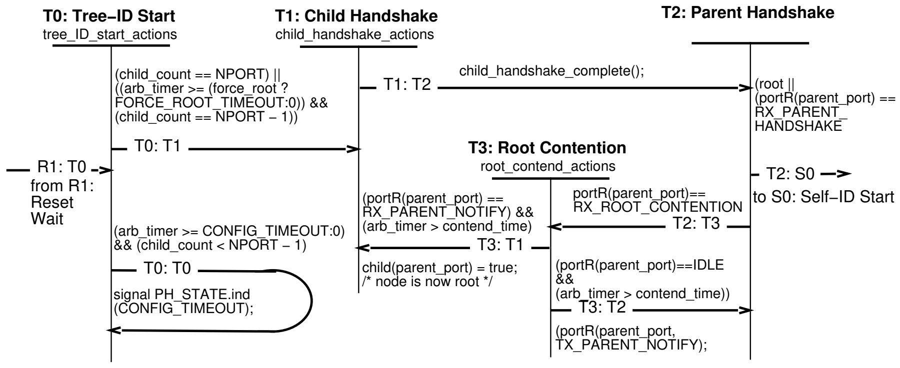

Figure 4: The state machine of a node in IEEE 1394

We now consider the question of the degree of standardisation of the notations used in the IEEE standard. A mixture of techniques are used to describe the protocol, some informal and therefore not standardised, and some formal. For example, the state machine diagrams are a formal method, and the C++ programming language can certainly be regarded as a formal description which is stable and extremely popular.

Development of IEEE standards proceeds by peer review, and there were at least 6 independent groups involved with development of the PHY layer of IEEE 1394. Since they all came to similar conclusions about fundamental behaviour of the protocol there is substantial evidence that the solution proposed in the standard is correct. However, no formal verification is known to have taken place, nor is it known if any rigorous approach to testing was used.

4.2 E-LOTOS

E-LOTOS [ISO98] (also known as Enhancements to LOTOS [ISO88]) is a new formal description technique currently undergoing the ISO (International Standards Organisation) standardisation process. E-LOTOS allows the specification of behaviour using process algebra, with some features borrowed from imperative programming languages, and data using abstract data types, described in a style similar to ML and Haskell. A goal of the development of E-LOTOS was to create a language for describing large timed computer systems which was easily accessible to software engineers. For this reason E-LOTOS has a module system to structure larger specifications, and operators to describe real time events.

The Tree Identify Protocol is described at four different levels of abstraction in [SV99a]. These are named `Spec`, `SyncImp`, `AsyncImp` and `TimedImp`. The motivation behind these descriptions was to demonstrate that E-LOTOS can describe a system at many different levels of abstraction, and that it might be easier to understand the protocol behaviour at a high level before dealing with low-level details. `Spec` is just the simple process which declares a leader without knowing which node was declared leader. `SyncImp` models a network of nodes which communicate synchronously (therefore there is no possibility for contention to arise). This is very similar to *ImpA* of [SvdZ98] and *TIP2* of [GV98] in its level of abstraction; however, the style is very different because E-LOTOS emphasises programming language operators. See below for an example of E-LOTOS.

`AsyncImp` also models a network of nodes, this time communicating asynchronously via a network of `Buffer` processes. This is similar to *TIP* of [DGRV00] and *ImpB* of [SvdZ98], but with the addition of timing parameters to resolve contention. Further timing details regarding delays introduced by passing messages in buffers (modelling the cables) are added in the final description `TimedImp`. This enables more detailed description of the loop detection mechanism, in the same fashion as *TIP4* of [Rom99], and also introduces the FORCE_ROOT parameter. The expressiveness of these descriptions is summarised in Figures 2 and 3.

The module features of E-LOTOS allow the descriptions to be split into three modules. The first is shared by all descriptions and specifies the data types and associated functions. The second module describes the process behaviour and is different for each level of abstraction. The third module describes the timing constraints of the system and is used only by `AsyncImp` and `TimedImp`.

The emphasis in the design of E-LOTOS was on increasing readability and easy understanding. As a result E-LOTOS borrows many features from programming languages. For example, the following fragment describes how contention is resolved:

```
process Contention [leader, s:comm, r:comm](id:iden, p:connections) is
 (* There will be only one j in p. *)
 var j:iden, b:bool, t:time in
   ?b := any bool;
   ?j := any iden [isin(j,p)];
   if b then ?t := ROOT_CONTEND_SLOW
        else ?t := ROOT_CONTEND_FAST
   endif;
   (  wait(t); s(!id, !j, !parent); WaitParent[...](id, p)
   [] r(!j, !id, !parent); SendAcks[...](id, {}, {j}) )
 endvar
endproc
```

Here `any` is a wildcard allowing nondeterministic choice of a value from a range of values, and `wait(t)` blocks the `s` actions until after time `t` has passed.

The example above shows how much like a programming lanaguage E-LOTOS can be. However, there are also some less familiar operators. For example, the following extract uses the `par` and synchronization operators to make `n` interleaved instantiations of `TreeIDAsync`, corresponding to the members of a list `l`, and combine those processes in parallel with a network of Buffers:

```
  ?n := sizeof(NW);
  ?l := (0 upto (n-1));
  hide s:comm, r:comm in
       par ?id in l
         ||| TreeIDAsync[...](id, neighbours(NW,id), {}, false)
       endpar
    |[s,r]|
       Buffers[s,r](n)
endhide
```

E-LOTOS was preferred amongst the formal methods in our readability test.

E-LOTOS is a proposed international standard. Its predecessor, LOTOS [ISO88], an ISO standard, gained some level of popularity and has a number of tools to support development of specifications. Therefore, E-LOTOS has potentially many users; however, delays in the standardisation process, a lack of specific tool support, and some complex features have hindered its take-up. Currently, although participants from ten different countries were involved in developing the E-LOTOS standard, that language is used only at two main sites (Stirling and Madrid). However, variant extensions of LOTOS are used elsewhere, notably at INRIA (where, incidentally, their version of E-LOTOS was used to demonstrate an error in a different part of the IEEE 1394 [SM97]).

The authors of [SV99a] stress that the four descriptions given constitute *informal* refinements of the system. Each of the descriptions `Spec`, `SyncImp`, `AsyncImp` and `TimedImp` were executed by hand to increase confidence in their correctness, but lack of tool support prevented any further, more rigorous analysis. A further barrier to analysis is that there is currently no theoretical framework on which to base analysis (for example, refinement or equivalence relations, related logics, theories of testing). The close relation between LOTOS and E-LOTOS means that such a framework should be easily constructed once standardisation is complete.

4.3 I/O Automata

Several papers [DGRV00, GV98, Rom99, SV99b] contribute descriptions of the Tree Identify Protocol using I/O Automata. Each paper has a different emphasis, concentrating on a different level of abstraction from the others, and some use I/O automata extended by time and/or probability.

I/O automata were developed by Lynch and Tuttle [LT89] as a labelled transition system model for components in asynchronous concurrent systems. Although the semantics is similar to that of process algebras, the standard notation is to specify operations in terms of their preconditions and effects. Thus the method has a more declarative feel than other operational techniques. The behaviour of an I/O automaton can be described in terms of traces (or fair traces). I/O automata have been used for describing and reasoning about many different types of systems, including network resource allocation algorithms, communication algorithms, concurrent database systems, shared atomic objects, and dataflow architectures.

The Tree Identify Protocol is described at many different levels of abstraction in the four papers selected. In [DGRV00] there are two descriptions: the very abstract *SPEC* which only allows a single *root* action to be performed, and the more concrete *TIP* which assumes a network of nodes communicating via message queues to achieve the leader election. The main focus of [GV98] is the verification technique of normed simulations, and the IEEE 1394 is used as an illustrative example. Three specifications, *TIP1* through *TIP3*, model the network as a set of devices and ports and define operations over the network as a whole. The abstract *TIP1* performs a single leader action after checking for loops in the network (and hence is slightly more complex than *SPEC*). *TIP2* models a global `child` set and is thus similar in behaviour to *ImpA* of [SvdZ98] and `SyncImp` of [SV99a], although no inter-device communication takes place. Finally, *TIP3* models inter-device communication via message queues. This is similar to *TIP* but more operational in style as it focuses on devices and their ports rather than the network as a graph. *TIP3* is roughly equivalent to *ImpB* of [SvdZ98] and *AsyncImp* of [SV99a] in level of abstraction.

The two remaining works deal with timed and probabilistic descriptions. A timed extension to *TIP3*, called *TIP4*, is given in [Rom99]. The main differences are the addition of delays between messages sent and received, and a timer to detect loops in the network. Root contention is solved in a single step and does not involve time; however, the final model *Impl* of [SV99b] does incorporate time and probability into contention resolution, making it a much more accurate model of the IEEE standard. Intermediate descriptions are introduced to ease the verification process. The simple *Spec* is equivalent to *SPEC*, but the others (*I1*, *I2*, *I3*) include probability details and are unlike any of the other specifications considered here.

The I/O automaton *Impl* has two advantages over the E-LOTOS `TimedImp`. First, probability is used to evenly weight the choice between a long and a short delay, instead of relying on assumptions of fairness to guarantee that eventually contention will be resolved. Second, the behaviour of the standard says the device should wait for the chosen time, then check the appropriate port for an incoming "be my parent" message from the other node, then resend "be my parent" if no incoming message is present. The model of communication in process algebras generally does not allow a test to see if a communication is ready. The solution to this problem presented in the E-LOTOS specification `AsyncImp` is to use a `wait` action. Only the receive action is enabled during the wait, but as soon as the time expires the send action is also enabled, therefore the receive action no longer has priority. Since message queues are used in the I/O automaton *Impl* it is possible to wait for a specified time and then check the message queue to see if a message has arrived at a port.

The different levels of expressiveness achieved in the I/O automata descriptions are summarised in Figures 2 and 3.

An example of the style of I/O automata specification is given below. This describes the preconditions and effects of the action of *RESOLVE_CONTENTION*. Several such definitions make up the automaton.

RESOLVE_CONTENTION (e: **E**)
  Precondition:
    $\wedge$ *contention*[source(e)]
    $\wedge$ *contention*[target(e)]
  Effect:
    child[e] := 1
    contention[source(e)] := 0
    contention[target(e)] := 0

The network is described as a collection of edges e, with source and target nodes, *contention* is a predicate indicating if a particular node is in a contentious state, and *child* indicates if the edge leads to a child node. This is nondeterministic resolution of contention because each node has this description therefore either of the nodes *source(e)* or *target(e)* may perform the action *RESOLVE_CONTENTION*.

This style of specification was deemed to be "prettiest" or most readable by one of our survey respondents, but not understandable without further training. Otherwise, I/O automata came out in the middle of our readability exercise.

I/O automata are standard in the sense that there is a formal definition and users do not deviate from that definition; however, there are many different extensions of the basic I/O automata model to choose from. I/O automata are used by researchers at a (small) number of sites, including the home of I/O automata at M.I.T. and several in the Netherlands.

The standard proof technique for I/O automata is to show a refinement relation between two automata. Different relations are used depending on the features of the I/O automata model. There are a number of generalised theorem provers which have been customised to support reasoning about refinements between I/O automata.

For example, in [DGRV00] verification means showing that the fair traces of *TIP* are included in the fair traces of *SPEC*, i.e. that each run of the automaton *TIP* can be simulated by the automaton *SPEC* (which can only perform a single

$ROOT$ action), and that all executions of *TIP* are finite. The proof proceeds by showing a number of invariants hold over the operations defined in *TIP*. PVS was used to check the proofs, thus avoiding problems with book-keeping errors.

This verification demonstrates that the more complex automaton performs the basic action $ROOT$ (or *leader*) exactly once. This corresponds to the safety and liveness properties that a single leader is chosen, and a leader is eventually chosen. The other I/O automata verifications also prove the same properties hold.

The normed simulations developed in [GV98] are equivalent to branching bisimulation, which is the equivalence relation used in the μCRL verification. The proofs of [GV98] are carried out by hand.

The proofs in [Rom99] used timed trace inclusion, and were carried out by hand. The emphasis in that paper is on checking that the timed approach to detecting cycles in the network worked correctly.

Finally, although the correctness proofs for the timed probabilistic automata of [SV99a] were carried out by hand, automated proofs of very similar systems have been carried out in Uppaal [SS00].

4.4 μCRL

Like LOTOS and E-LOTOS μ-CRL [GP95] is also a process algebra extended with a formal treatment of data. The style of μCRL descriptions is much more mathematical than the others here, and the descriptions rather terse. The emphasis in the μCRL approach in general is to keep the descriptions small and therefore easier to reason about.

The Tree Identify Protocol is described at three different levels of abstraction in [SvdZ98]. These are named *Spec*, *ImpA* and *ImpB*, and correspond to the others in expressiveness as shown in Figures 2 and 3. *Spec* is the simplest process which merely declares leader. Both *ImpA* and *ImpB* are parallel collections of nodes and correspond broadly to `SyncImp` and `AsyncImp` of the E-LOTOS descriptions.

Each node is designed as a state machine parameterised by identifier, set of connected nodes and state variable for *ImpA* with the addition of a set of children for *ImpB*. Each process has a number of possible branches, but depending on the value of the state variable and the parent set only some of those branches are available at any given time.

For example, the branch in *ImpA* which corresponds to being able to declare oneself leader is written:

$$\mathit{leader} \cdot \mathit{NodeA}(i, p, 1) \lhd s = 0 \land \mathit{empty}(p) \rhd \delta$$

where p is the set of connections and s is the state variable. This should be read as “if the state is 0 and there are no more connections left to be made, then perform a leader action and behave as *NodeA* in state 1, otherwise terminate (δ)".

A more complex example occurs when contention must be resolved in *ImpB*. The following expresses the choice that if the state is 3 (contention) and the remaining connection is j, then either a "be my parent" request can be sent ($\bar{s}$), or one can be received (r).

$$\begin{array}{l}\sum_{j:N} r(j,i,par) \cdot NodeB(i, p \setminus \{j\}, c \cup \{j\}, 1) \lhd s = 3 \wedge p = \{j\} \rhd \delta + \\ \sum_{j:N} \bar{s}(j,i,par) \cdot NodeB(i,p,c,2) \lhd s = 3 \wedge p = \{j\} \rhd \delta\end{array}$$

where c is the set of children.

The state variable here corresponds broadly to the state numbers of the state machine of the standard (see Figure 4).

In addition to the node descriptions, *ImpB* also has a lattice of buffers between nodes, to model asynchronous communication. The size of the buffers is one; larger buffers are unnecessary since only one message can be sent between a pair of processes at a time.

This is the same model of communication between nodes as used in the E-LOTOS `AsyncImp`, and is interesting because it automatically excludes some methods of introducing cycles in the network. There can be only one connection between nodes using these buffers, but in reality devices can be connected by more than one cable. This may be regarded as an additional assumption imposed on the initial structure of the network. This over-simplification does not arise in the I/O automata of [GV98, Rom99] since devices and ports model the network more accurately.

It can be seen that the μCRL descriptions are extremely compact, but this must be weighed against the rather mathematical notation which might not appeal to a software engineer. In our readability study μCRL was rated very low; however, for a software engineer with no formal methods background this is to be expected. Of all the notations examined in this paper, μCRL is the one which relies the most on concise syntax, and if that syntax is unfamiliar there are very few cues to help interpret it.

μCRL is standard in the sense that it has a stable formal definition which is not deviated from by its practitioners. It is not yet widely used; most users of μCRL are based in Amsterdam.

It is formally shown in [SvdZ98] that the three specifications are equivalent using branching bisimulation and the cones and foci technique. An important part of the development of μCRL has been the development of a supporting proof system. The cones and foci technique evolved from earlier, more complex and tedious proof techniques based on algebraic manipulation using the laws of branching bisimulation. The proof is carried out by hand; however, tools for manipulating μCRL specifications have since been developed. Branching bisimulation seems to be an appropriate choice when using equivalence as a measure of correctness of a system since it ignores internal actions, but preserves information about the point at which choices are made. Additionally, if two processes are branching bisimilar then they have the same deadlock properties.

Refinement does not play a part in [SvdZ98]. Although the three descriptions can be seen as increasingly complex, all three were developed independently.

5 Conclusions

In summary, we have examined a variety of specifications of the Firewire Tree Identify Protocol, as well as the IEEE standard in which this protocol was defined, and compared them according to four criteria. In this section we outline the main results of our comparison, and then discuss the limitations of the present survey and our plans for further development of this case study.

All the formal methods considered scored highly on expressiveness, in that they all made it possible to write both highly abstract specifications as well as very concrete descriptions capturing most of the aspects of the protocol as described in the IEEE standard. No specification completely captured all of the details of the description in the standard, though the more detailed I/O automata specifications came closest.

None of the formal methods examined has been officially standardised, though E-LOTOS is in the process of becoming an ISO standard. All of the methods are, however, stable, and may be thought of as having attained a *de facto* standard status through common use. None of them are widely used, with I/O automata probably scoring highest on this criterion by a small margin.

All of the formal methods scored poorly on readability in our informal survey. E-LOTOS was judged to be the most readable, because of its programming language-like syntax, while μCRL was generally thought to be the least readable, perhaps because of its heavy reliance on mathematical notation. These results on readability should not be taken too seriously, since a better measure would be given if software engineers were first trained for a short time in the formalism and then shown the specifications.

The main type of analysis used in the examples surveyed was formal proof. Apart from E-LOTOS, all of the formal methods are backed by the theoretical infrastructure to enable some form of equivalence or refinement relation to be proved. In all cases, this proof was done manually, either on paper or with the help of an interactive proof checker. None of the methods have purpose-built proof tools or any substantial support for automatic proof.

There are many interesting criteria which we were not able to cover in this survey. For example, we do not have data on the length of time it takes to learn to use each formal method or the length of time taken to write each specification. In general, when examining a specification, it would also be interesting to know the role which it plays within the system development lifecycle, and the degree to which it is complemented by accompanying descriptions in other formal, semi-formal, or informal notations. Considerations such as these did not seem relevant to the specifications considered in the survey, as the answers were clearly going to be the same in all cases (the specifications were devised as academic exercises, outside the normal developmental lifecycle, and no integrated formal or other methods were used).

The biggest disadvantage of the current survey is the very narrow range of formal methods which is covered. According to Bowen and Hinchey [BH99], formal methods can be broadly grouped into three types: model-based methods such as Z and VDM; property-based descriptions, which may be further classified as being axiomatic descriptions (e.g. Larch) or algebraic descriptions (e.g. algebraic data types); and process algebras. Of the formal methods we have covered, E-LOTOS and μCRL are both essentially process algebras, while I/O automata can be seen as having both model-based and process algebra aspects. We have no specifications in the more popular model-based languages, and none at all from within the category of property-based languages.

In future work, we aim to remedy the shortcomings of this survey by gathering data on a larger selection of criteria and broadening the range of formal methods which we examine. To achieve this, we are organising a workshop based upon the Firewire case study, in the style of Abrial's Steam Boiler Specification workshop [ApL96]. Practitioners of a wide variety of formal methods will be invited to specify the Tree Identify Protocol in their favourite method, and asked to self-assess their work with respect to the broader range of criteria indicated above. We hope that the results of this workshop will serve to expand upon the contribution made by the present survey.

References

[AAH98] M. Allemand, C. Attiogbé, and H. Habrias, editors. *International Workshop on Comparing Systems Specification Techniques*. IRIN Press, March 1998.

[ApL96] J-R. Abrial, E. Börger, and H. Langmaack, editors. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Computer Science*. Springer-Verlag, October 1996.

[BH99] J.P. Bowen and M.G. Hinchey. *High-Integrity Systems Specification and Design*. FACIT. Springer-Verlag, 1999.

[DGRV00] M.C.A. Devillers, W.O.D. Griffioen, J. Romijn, and F. Vaandrager. Verification of a Leader Election Protocol - Formal Methods Applied to IEEE 1394. *Formal Methods in System Design*, 16(3):307–320, 2000.

[GP95] J.F. Groote and A. Ponse. The Syntax and Semantics of μ-CRL. In *Proceedings of Algebra of Communicating Processes, Utrecht 1994*, Workshops in Computing. Springer-Verlag, 1995.

[GV98] W.O.D. Griffioen and F. Vaandrager. Normed simulations. In *Proceedings of CAV'98*, number 1427 in Lecture Notes in Computer Science, pages 332–344, 1998.

[IEE95] Institute of Electrical and Electronics Engineers. *IEEE Standard for a High Performance Serial Bus. Std 1394-1995*, August 1995.

[ISO88] International Organisation for Standardisation. *Information Processing Systems — Open Systems Interconnection — LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, 1988.

[ISO98] International Organisation for Standardisation. *ISO/IEC JTC1/SC21 WG7: Enhancements to LOTOS*, May 1998. Final committee draft.

[LL95] C. Lewerentz and T. Lindner. *Formal Development of Reactive Systems: Case Study Production Cell*, volume 891 of *Lecture Notes in Computer Science*. Springer-Verlag, January 1995.

[LT89] N. Lynch and M. Tuttle. An Introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219–246, September 1989.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc, 1996.

[Rom99] J.M.T. Romijn. A Timed Verification of the IEEE 1394 Leader Election Protocol. In *Fourth International Workshop on Formal Methods for Industrial Critical Systems*, 1999. To appear as a special issue of Formal Methods in System Design.

[SM97] M. Sighireanu and R. Mateescu. Validation of the Link Layer Protocol of the IEEE-1394 Serial Bus (FireWire): an Experiment with E-LOTOS. In *2nd COST 247 International Workshop on Applied Formal Methods in System Design (Zagreb, Croatia)*, June 1997. The full version of this paper is available as INRIA Research Report RR-3172.

[SS00] D.P.L. Simons and M.I.A. Stoelinga. Mechanical Verification of the IEEE 1394a Root Contention Protocol using Uppaal2k. Report CSI-R009, Computing Science Institute, University of Nijmegen, Nijmegen, 2000. Submitted for publication.

[SV99a] C. Shankland and A. Verdejo. Time, E-LOTOS, and the FireWire. In M.A. Marsane, J. Quemada, T. Robles, and M. Silva, editors, *Workshop on Formal Methods and Telecommunications*, pages 103–119. Prensas Universitarias de Zaragoza, 1999.

[SV99b] M.I.A. Stoelinga and F.W. Vaandrager. Root Contention in IEEE 1394. In *5th AMAST Workshop on Real-Time and Probabilistic Systems*, LNCS 1601. Springer-Verlag, 1999.

[SvdZ98] C. Shankland and M. van der Zwaag. The Tree Identify Protocol of IEEE 1394 in μCRL. *Formal Aspects of Computing*, 10:509–531, 1998.

[Win90] J. Wing. A Specifier's Introduction to Formal Methods. *IEEE Computer*, 23(9):8–24, September 1990.