Founding FireWire Bridges through Promela Prototyping

Izak van Langevelde[1]
email: izak@cwi.nl

Judi Romijn[2]
email: j.m.t.romijn@tue.nl

Nicu Goga[2]
email: n.goga@tue.nl

[1]Centrum voor Wiskunde en Informatica
P.O. Box 94079
1090 GB Amsterdam, The Netherlands

[2]Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven, The Netherlands

Abstract

The standardisation procedure of the IEEE P1394.1 Draft Standard for High Performance Serial Bus Bridges is supported through the use of the state-of-the-art model checker Spin, which has been used to simulate the complex net update procedure of the standard, and the use of which will eventually be refined to obtain a solid model checking analysis of the standard. A concise description of net updates is formalised in terms of spanning trees, and it is shown how Spin was used to track down errors in the standard and to gather support for the solutions proposed.

1 Introduction

This paper describes how the IEEE standardisation of serial bus bridges benefits from the use of Promela prototypes analysed using the Spin model checker [5, 6]. The development of the IEEE P1394.1 Draft Standard for Serial Bus Bridges [9] was initiated in 1996 to facilitate the building of networks of serial buses [7, 8] connected through bridges. The imposed requirement that plugging and unplugging bridges leaves the network integrity intact as much as possible led to the inclusion of a *net update procedure*, which appeared to be too complex to be shown correct by mere human wit and was significantly different from algorithms known in the literature.

Experience with the use of formal methods in the verification of IEEE 1394 [12, 10] led to the belief that the approach might be successful once more. However, IEEE P1394.1 had not stabilised yet, so verifying this standard-in-creation resembled hitting a moving target. Nevertheless, the challenge was taken.

The net update problem IEEE 1394 [7, 8] standardises the high-performance *serial bus* as an efficient and flexible network of at most 63 nodes. To be able to build a larger *network*, the *bus bridge* is standardised as a means to connect two serial buses. In the network, each node can be addressed with the identification of the serial bus it resides on and its identification on this bus.

The IEEE 1394.1 protocol is responsible for guaranteeing that each node has a unique address and data can be routed towards any address. For the former, it suffices to ensure that each bus in the network has a unique identification, since the IEEE 1394 protocol already ensures that each node on this bus has a unique identification; this task will be referred to as *bus enumeration*. For the latter, each bus bridge must have information on how to reach a bus by its identification; this information is stored in its *route map*.

The net update problem is defined as guaranteeing the correctness of bus enumeration and route maps, while bus bridges are added or deleted.

Our goal The goal of the research reported here is to contribute to the IEEE 1394.1 standardisation through the use of formal methods and, secondary, to demonstrate the use of formal methods to the standardisation committee. However, the nature of standardisation implies a caveat.

The IEEE 1394.1 standardisation is a long-term cooperative effort of a great number of parties concerned, including manufacturers of network appliances and consumer electronics, each having a specific interest in this standard. The requirements imposed onto the standard-in-creation range from basic properties, like correctness of routing, via complex criteria like performance and fault-tolerance, to contextual issues like backward compatibility and business politics. Some of these are made explicit and easily understood; some are implicit and obscure. Some are consistent, in that they can both be satisfied; some are inconsistent and liable to be subject of complex trade-offs. However tempting it may seem for researchers to concentrate on one such requirement in isolation, this approach is doomed to fail, since

0-7695-1926-1/03/$17.00 (C) 2003 IEEE

almost certainly conclusions which apply to one such aspect will appear incompatible with others, and solutions accepted by one party might be unacceptable for others. The key is in close interaction with the IEEE 1394.1 community.

Interaction with the standardisation community was effectuated through active participation in the IEEE 1394.1 Working Group's *reflector mailing list*, during the creation of the initial draft standard, and active membership in the *Ballot Response Committee* whose task is to resolve the 508 comments issued by those who voted in the first ballot. It was this interaction which made us shift our focus from the work on correctness of route maps to the part of the problem that deals with loop elimination.

Approach The challenge taken is to analyse an IEEE standard in creation, of which both the actual standard and the properties to be verified are incomplete and subject to change. A suitable approach needs to satisfy the requirements that incomplete systems can be verified, and that the verification method is efficient enough to be easily repeated for variants of the standard analysed before. The former requirement suggests a prototyping approach, which facilitates fast and small-scale experiments with promising variations on a theme. The latter makes model-checking [2] an excellent candidate, in that model checking is supported by efficient and completely automated tools.

The method of our choice is the language Promela and the tool Spin [5, 6], for a number of reasons. First, this combination has a good reputation when it comes to practical applications, which is well documented (e.g. [1]) and has gained official recognition by winning the ACM System Software Award in 2001. Second, the use of Spin is beyond checking properties to be verified, it is also useful as a simulation tool which is valuable in the prototyping approach we have in mind. Third, the resemblance of Promela to programming languages is expected to facilitate actual implementations.

Outline of this paper Section 2 describes the net update problem, and Section 3 formalises the relevant notions. Section 4 gives the obtained results and Section 5 presents the conclusions of this verification study.

2 IEEE 1394.1 net updates

This section overviews the IEEE P1394.1 draft standard [9] and the underlying IEEE 1394 standard [7, 8], at a level of abstraction which gives some intuition of the net update fragment of the former (i.e. Chapter 10 of [9]).

The 1394.1 functionality The IEEE 1394 standard specifies how devices, identified through their 64-bit hardware addresses, can be interconnected through cables, without users having to worry about configuring or bringing down these devices. The IEEE abstraction as presented through the standard consists of *nodes*, identified through 6-bit *physical ids*, connected through a *serial bus*. This abstraction is illustrated in Figure 1.

IEEE 1394 restricts serial buses to a maximum of 63 nodes attached, in order to meet its high-performance needs. Larger networks can only be built by connecting serial buses through *bridges* which selectively route network traffic originating from a node on one bus destined to a node on another bus.

The IEEE 1394.1 standard specifies how buses can be interconnected through bridges, consisting of a pair of directly coupled nodes called *bridge portals*, without users having to worry about configuring or bringing down any of these buses. The two portals which constitute one bridge, called each other's *co-portal*, communicate by a medium which is beyond the 1394 protocol. It is important to note that as a consequence, each of the two co-portals resides on its own separate bus. The abstraction as presented through the standard consists of a *network* of serial buses, identified through 10-bit *bus ids*. This abstraction is illustrated in Figure 2.

The main services offered through the standard are the assignment of unique bus ids to the connected buses, or *bus enumeration*, and directing network traffic from a node on one bus to a node on another bus, or *routing*.

We now give a short summary of key concepts that play a role in bus enumeration and routing. The central player in bus enumeration is the *prime portal*, which is responsible for assigning bus ids to buses. It is essential that each portal

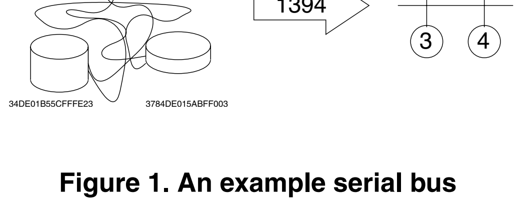


Figure 1. An example serial bus

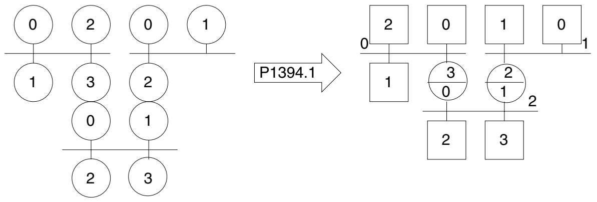


Figure 2. An example IEEE 1394.1 network

is able to communicate with the prime portal; to this end, on each bus the *alpha portal* is the one portal directed towards the prime. In order to guarantee a unique route between each pair of nodes, a *loop elimination* algorithm is used to cut *routing loops*. All information on the network topology is stored in *route maps*.

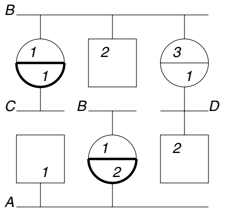
In this section bus enumeration, routing, and route maps are explained. We defer the explanation of the detection and elimination of routing loops to the next section.

Net update The unhampered continuation of IEEE 1394.1 services is taken care of by the *net update procedure*. The goal of net update is therefore to ensure correctness of the bus enumeration procedure and the routing mechanism under changes involving the addition or deletion of bridges. This boils down to updating the prime portal, route maps, and alpha portals, each time a bridge is added or deleted.

The immediate cause for the net update is the addition or deletion of a bridge portal on a bus, which is noticed by all other nodes on the bus. The bridge portal with the highest physical id on the bus is chosen as *coordinator*, being the local manager of the net update procedure. First, the coordinator detects and breaks *routing loops*. Second, it selects from the prime portals in the network one prime portal for the updated network, according criteria which are too elaborate to be described here. Third, it gathers the route maps of all portals on its bus, in order to generate for each prime portal on the network a *clan map* of the bus ids assigned. Fourth, these clan maps are combined into a *net map*, describing for each bus id in the network whether it is used exactly once or more than once, marking the latter as controversial. Fifth, it sends the new prime and the net map to all portals on the bus, with a request to update their route maps and to hand this information to their co-portals.

When a portal receives new information from its co-portal, as a result of the net update procedure on the co-portal's bus, the receiving portal starts the net update procedure on its own bus. In this way, the net update procedure and the resulting information spread through the net like oil slick on water. Care is taken that the net update procedure does not return to buses where it was already executed, unless the resulting information has changed.

An example network We now explain the notions of routing and bus enumeration with an example: two networks, totaling five buses and three bridges.



In this example, the bus ids are represented by letters and the physical ids are identified by numbers. At first sight, there is a problem in that there are two buses 'B', but closer examination reveals that these buses are in different networks. Within each network each node can be uniquely identified by the combination of its bus and its physical id, for instance B1, where the assignment of physical ids is taken care of by IEEE 1394 while the assignment of bus ids is taken care of by IEEE 1394.1. In the above picture, the two prime portals are marked by bold lines.

Routing Suppose node B2 (this node is necessarily on bus B in the top network) sends a message to node D2, which is in principle reachable through the bridge consisting of portals B3 and D1. For the message to effectively and efficiently make it to D2, bridge portal B3 must know that bus D is somewhere 'behind its back' while bridge portal B1 must know that bus D is not behind its back. This knowledge is stored in route maps.

A portal's *route map* stores for each possible bus id whether traffic towards this portal should be transferred to the co-portal or not, or whether this bus id is controversial (these options are denoted by '$+$', '$-$' and '$\perp$', respectively). The route maps for the bridges in the upper network of the example are as follows.

B1	
B	−
C	+
D	−

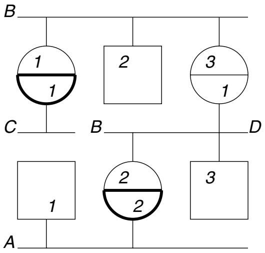
C1	
B	+
C	−
D	+

B3	
B	−
C	−
D	+

D1	
B	+
C	+
D	−

It is intuitively clear that these route maps must satisfy a number of requirements: (i) no bus ids are controversial; (ii) on each bus, exactly one bridge forwards traffic towards a distant bus; (iii) no bridge forwards traffic for the local bus. These requirements will be discussed in Section 3.

An example of net update Suppose bridge portal B1 is connected to bus D. The IEEE 1394 protocol assigns unique physical ids to the nodes on bus D:



At this point, there are two buses B in the network, there is a bus which appears to have been assigned two bus ids B and D, the remaining buses A and C are not known throughout the network, and there are *two* prime portals. In this inconsistent situation, IEEE 1394.1 net update kicks in.

In the running example, the connection of portal B1 is first detected at the bus with ids B and D; for notational convenience, this bus will be denoted BD. Here, portal BD2 is designated coordinator. As this coordinator detects no routing loops, it gathers for each of the two primes on the network the bus ids assigned, and compiles a net map:

C1	
A	−
B	+
C	+
D	+

+

A2	
A	+
B	+
C	−
D	−

=

net	
A	+
B	⊥
C	+
D	⊥

So, bus id B is known to both prime portals C1 and A2, rendering it controversial in the net map. Bus B and D are both assigned to the local bus, which also renders D inconsistent. As a result, only the bus ids A and C, which are both known to exactly one prime, remain valid. Next, the coordinator sends this net map to portal BD1, and subsequently both portals update their route maps:

BD1	
A	−
B	⊥
C	+
D	⊥

BD2	
A	+
B	⊥
C	−
D	⊥

So, traffic towards bus C is routed by bridge portal BD1 and traffic towards bus A is routed by the portal BD2. Having updated their route maps, each portal passes the net map to its co-portal, which incorporates this in its route map:

B3	
A	+
B	⊥
C	−
D	⊥

A2	
A	−
B	⊥
C	+
D	⊥

Having updated their route maps, the co-portals realise that something has changed in the network. By the oil slick mechanism, the net update process spreads through the network, finally yielding the following route maps:

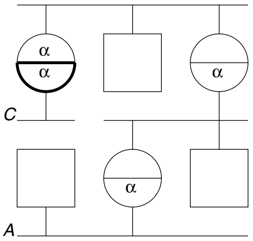
B1		C1		B3		BD1		BD2		A2	
A	−	A	+	A	+	A	−	A	+	A	−
B	⊥	B	⊥	B	⊥	B	⊥	B	⊥	B	⊥
C	+	C	−	C	−	C	+	C	−	C	+
D	⊥	D	⊥	D	⊥	D	⊥	D	⊥	D	⊥

The resulting network configuration is sound, in the sense that no traffic will be routed to the wrong destination. However, it is not complete, in that the bus ids B and D are marked as inconsistent, rendering unreachable all nodes connected to buses formerly assigned these ids. The process of assigning new bus ids to buses that have 'lost' their id is called *bus enumeration.*

Bus enumeration The crux of bus enumeration is to have one node on a bus with an inconsistent id send a request for a new bus id to the network's prime portal. There are, however, two complications: the requesting node has to be able to route its request to the prime portal but it may not know the bus or physical id of the prime portal, and the requesting node lacks a valid bus id, so there is no way the response from the prime portal can be routed back to it. The solution of this problem is in the alpha portals.

The alpha portal has the task of checking whether its bus id is marked inconsistent. The first complication can be circumvented by sending the request for a new bus id to the prime portal via the alpha portal of each bus in between. The second complication is dealt with by having the original alpha requester ask its co-portal to send a request to the prime portal, and to hand over the prime's response. If the alpha's co-portal lacks a consistent bus id, it defers the alpha's request until its bus is correctly reassigned.

In the example above, the structure of alphas and primes is as follows:



Once an alpha portal has a new bus id, it once more initiates the net update procedure, finally leading to the configuration of route maps towards the newly assigned bus id.

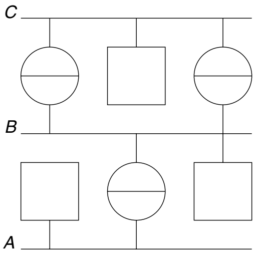
An example of bus enumeration In the running example, the buses formerly known as B and BD have lost their bus id, so their respective alpha portals ask their co-portal to request a new bus id. The co-portal of the alpha of the former happens to be the network's prime portal, which directly returns 'E' as the first free bus id and the co-portal hands this over to the requesting alpha. The co-portal of the alpha of the latter bus now has a bus id assigned, so it may directly address the prime portal and request a new bus id. The prime portal now returns 'F' as the first free bus ID which, upon receipt, is handed over to the requesting alpha.

3 Modeling

This section explains our models of net update in two steps. First, we determine the desired properties of the net update procedure by considering buses and bridges in terms of graphs and spanning trees. Second, we give an overview of our Promela models.

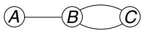
3.1 Graph interpretation

As running example the following network will be used, which differs from the example of the previous section in that the network contains a loop.



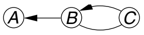
It is instructive to recall the division of tasks over the standards IEEE 1394 and IEEE 1394.1. The former handles the interconnection of nodes within a bus, while the latter handles the interconnection of buses through bridges, taking the former for granted. So, on the bridge level a bus can be considered an atomic unit.

If this abstraction is applied to the example network, the network of three buses and three bridges can be represented as an undirected graph of three nodes and three edges, where each bus corresponds to a node, each bridge corresponds to an edge and each alpha portal corresponds to one of the two ways an edge can be directed.



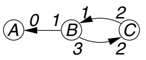
The route map of a bridge portal specifies for the corresponding directed edge whether it is a route for a destination bus id or not. Alternatively, the route map entries for each portal in the network towards some bus id define a directed graph imposed on the underlying undirected network. For reasons of efficiency, IEEE 1394.1 demands a unique route from each source bus id towards each destination bus id. So, correctly configured route tables towards a given bus id define a spanning tree with this bus id as its root.

The prime and alpha portals also define a directed graph on the underlying undirected graph, which is a spanning tree if primes and alphas are configured correctly.



So the net update of IEEE 1394.1 can be characterised as a distributed algorithm for spanning directed trees over an undirected graph; i.e. each node in the graph is the root in one spanning tree for its bus number, and there is one more spanning tree with the prime portal on this tree's root node.

Routing loops, detection and elimination A routing loop is a loop in the directed graph for a bus id. Obviously, such a loop must be caused by a loop in the graph structure itself: the topology of the network. This is detected in 1394.1 by observing the alpha portals, i.e. the directed graph for the prime portal. The loop detection criterion, in terms of graphs, is satisfied whenever there are two outgoing edges on one bus for the same prime portal, or whenever there is an outgoing edge for a prime portal on the same bus as that prime portal itself. In terms of 1394.1, the criterion is satisfied whenever there are two alpha portals on one bus for the same prime portal. In order to decide how to cut such a loop, and be able to cut it by only changing the information of a bridge connected to the bus on which net update is being executed, it is convenient to have information on the distance of the bridge portals to the prime portal. It is safe to cut a loop by undirecting the edge which is furthest from the prime portal, see the following example.



In 1394.1 terms, the loop is cut by turning one of the alpha portals into a non-alpha portal. This portal must also inspect all bus ids in its route map, and change the entries of those bus ids for which it is en route from '+' to '−'. Since the route maps are inspected by the coordinator only after loops have been cut, these bus ids can be taken care of appropriately, and do not become controversial unnecessarily. In this manner, routing loops are eliminated.

Spanning trees in the literature There are two kinds of algorithms in the literature for creating and maintaining spanning trees in a graph. The first kind assumes that the graph is weighted, and is based on Kruskal's algorithm or Prim's algorithm (e.g. [3]). In the spanning tree computed, the sum of the weights is minimal. Since in 1394.1 networks the edges do not have a weight, it seems unnecessarily restrictive to use this algorithm.

The second kind of algorithm does not assume that the graph is weighted: the spanning tree to be computed has a minimal number of edges. An example is the algorithm in the IEEE 802 standard for transparent bridges, (see [11]), which is theoretically sound and widely used for interconnecting networks. But this algorithm and others derived from it are very different from the 1394.1 situation: for example, messages are sent periodically in order to detect topology changes and update the tree, while this is not necessary for 1394.1 networks, which detect topology changes automatically on the bus level.

3.2 Promela models

Two Promela models have been constructed: one modeling the part of net update that maintains a spanning tree for the prime portal and the alpha portals, and one modeling the part of net update that maintains the route maps.

Prime and alphas In this Promela model, there are ten bridge portals, constituting five bridges, which are all active all along. There is a generic process definition of the coordinator part of a portal's behaviour, and another generic process definition for the rest of a portal's behaviour. Of both of these, 10 separate copies are executed in parallel to model the ten bridge portals. One more process is present, which models the changes in the network topology, by nondeterministically connecting and disconnecting bridge portals to buses. A bus is not modeled separately in a process, it can be identified through the connection state of the portals.

The coordinator process, when active, collects information of all bridge portals on its bus, observes whether the loop criterion is true, and if so, cuts the loop by undirecting appropriate bridges. It then selects the winning prime portal, and informs all bridge portals of this choice. The bridge portals give information when requested, and pass on the new information to their co-portals, which can in turn then start net update on their bus to process the new information.

Safety properties are added for which the successful termination of net update, and the spanning tree are checked. These properties are all defined in terms of what should hold for each bus and for each bridge.

Route maps This Promela model specifies the maintenance of the route maps and bus enumeration throughout the net update procedure. A complication here is that the latter relies on the former, in that a request for a new bus id is routed towards the prime by the IEEE 1394.1 protocol. In other words, the maintenance of route maps cannot be studied in isolation from the use of these same route maps. So, the Promela model for net updates also covers the routing mechanism, which makes it more complex than expected. The model consists of a number of processes. Each serial bus is modeled by one process, and each bridge portal is specified by two processes: a protocol stack and a portal control. Here, the IEEE 1394 fragment which is relevant to IEEE 1394.1 is modeled in the serial bus protocol and in the protocol stacks, and the actual IEEE 1394.1 fragment is modeled in the protocol stack and the portal controls. Also, there is one environment process, from which bridge portals can be added to or removed from serial buses.

Even for a simple example as given in Section 2 the resulting Promela model encompasses as many as five bus processes, six protocol stack processes, six portal control process, and one environment process. Taking into account that the protocol stacks and the portal controls model calculations on route maps, it becomes apparent that the complexity of the underlying model is astronomical.

The results reported in this paper were obtained through simulations of the model, in which selected scenarios were subjected to manual analysis. As will be described in Section 4, this relatively simple technique already revealed a number of weaknesses and bugs in the protocol. However, it is expected that the model used so far can be simplified with specific target aspects in mind to support a more thorough analysis of problematic issues.

4 Results

The modeling of IEEE 1394.1 in Promela and the subsequent analysis of the resulting models revealed a large number of obscurities, omissions, minor mistakes, as well as some real errors. The mere attempt to model the standard was hampered by the fact that the draft standard is not complete in listing all actions required by a bridge portal upon reception of messages, which also depends on the state of the bridge portal; more than once these 'holes' in the draft had to be filled by laborious trial and error. However, not all shortcomings of the draft were just omissions.

4.1 Maintaining the prime and alphas

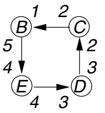
The Promela model describing the prime/alpha maintaining part of net update has led to the following items to be identified and fixed.

- (Omission) The draft standard did not mention when a bridge without direction may be directed again, this is now included: upon change of the prime portal.

- (Improvement) The two portals in one bridge must now always change their information in synchronisation with each other. This greatly simplifies the description and nature of net update.
- (Unclarity) The behaviour of a portal and the coordinator task are considered to be two separate processes. This greatly simplifies the description and nature of net update.
- (Error) Not all loops are detected with the current criterion, and net update may not terminate at all.

We now explain the last item in more detail. This was the most important error that we found. The current solution was constructed in cooperation with the members of the IEEE P1394.1 ballot response committee.

Error in loop detection The basic criterion for detecting that a loop exists in the network is that on a bus, there are two alpha portals, i.e. the bus has either two bridges pointing away from that bus into the direction of the same prime portal or the bus has a prime portal and a bridge pointing away to that same prime portal. It is perhaps not surprising that, after some network topology changes, it can also be the case that a loop exists in the network in which this criterion is not met: all buses on the loop have exactly one alpha portal for a prime portal, but the prime portal itself isn't there. This can be recognised by observing the information of the distance of each portal to the prime portal. There must be at least one bus that has an outgoing bridge with a distance larger than a corresponding incoming bridge. See the following figure.



In this situation, the net update algorithm as presented in the current draft does not terminate, but keeps being executed on consecutive buses in the loop, and the distances keep increasing. This is a major error. It was discovered by extensive simulation with Spin. It has not been discovered with Spin verification, because this takes much CPU time and memory. Even the approximate verification algorithms in Spin are not able to detect the error.

We have tried to fix the algorithm, by undirecting the incoming bridge with the erroneous distance. However, the adjusted Promela model still contained the same erroneous behaviour, which could be discovered neither with verification nor with simulation. Being suspicious and aware of what type of behaviour would cause the error, we tried to guide the simulation by applying an input/output paradigm, and did find the error. See [4] for details on this method.

The solution Finally, at the suggestion of the ballot response committee, we added a separate algorithm to the net update functionality, acting as a kind of *network reset*, with the property of resetting all information in each bridge portal in the network to the initial state. This means that each bridge is directed and one of the two portals in each bridge is a prime portal. Obviously, upon finishing this new algorithm, the network state is not correct, hence net update must be started again to sort out a single remaining prime portal, as well as the spanning tree. This solution is going to be included in the IEEE P1394.1 standard.

The network reset condition is met whenever the alternative looping situation mentioned above occurs, or when the distance to the prime portal exceeds the maximum network size. In this case, the bridge portal signaling the condition starts the network reset algorithm, which spreads a message through the entire network. The effect of this network reset message is that portals first stop acting as a bridge and inform their co-portal, and only initialise and restart their bridge activities when the message has been processed by all portals on their own bus and on the co-portal's bus. The message also carries a parameter indicating the distance to the bus on which the algorithm was originally started. In this way the algorithm is guaranteed to stop by respecting the maximum size of the network.

An interesting observation is that in the spanning tree algorithm for IEEE 802 transparent bridges there is a similar network-wide reset. A difference is that in the case of IEEE 802, each bridge must wait a time proportional to the diameter of the network for allowing the information to spread, whereas the 1394.1 reset algorithm was conceived without any unnecessary waiting time: the bridges wait only for the information to spread at the level of both connected buses, which happens rapidly.

The current simulation and verification results are encouraging, but have not yet given complete confidence in the adjusted tree spanning procedure.

4.2 Maintaining the route maps

The complexity of the Promela model did not allow more than extensive simulations of the net update procedure. Nevertheless, a number of weak point were found:

- An early version of the draft standard defined the initial value of a portal's route map to be 'vendor dependent'. Experiments revealed that an initialisation which expects a portal to 'learn' the bus id of its co-portal renders inconsistent the bus ids of both the portal and the co-portal, after which both buses are to be assigned a new id. Although this situation finally results in a correct configuration for both portals, the lesson learnt is that some initialisations are better than others. Later, the initialisation of the route maps was changed into a value where no bus id is routed.

- Several simulations pointed out problems with 'concurrent net update', i.e. net updates originating from bridges that were added or removed in different parts of the network, leading to 'waves of net updates' simultaneously spreading through the network and clashing somewhere in between. The problem was addressed by adding one semaphore to each bridge which prevents two co-portals from simultaneously handling a net update. Although this semaphore seems to solve the problem locally, it is as yet unknown whether it is a global solution.
- Bus enumeration relies on the co-portal of the alpha on a bus with an inconsistent bus id to send a request to the prime portal on the network. For this request to be effectively routed, this co-portal must know the bus id of the prime, but in the current draft standard this bus id is not known throughout the network. Attempts to write a solid Promela model revealed this problem, which has not been solved yet in the latest draft standard.

5 Conclusions

The net update fragment of the IEEE P1394.1 Draft Standard for High Performance Serial Bus Bridges was presented and formalised in two Promela models. This standard in creation, was analysed through the award-winning tool Spin. With the standard being subject to change, this analysis has taken the shape of extensive simulations so far. However, with the draft converging into a solid standard, and the Promela specifications stabilising into a slim model which allows efficient model checking, the character of the analysis is expected to change from common-sense simulation and testing to exact formal verification.

The results obtained include, apart from a variety of omissions and obscurities, a number of relevant problems signaled. Most noticeable is the discovery of a fatal error in the loop detection algorithm, and the design of a alternative algorithm. The correctness of this algorithm was made plausible through simulations, but its correctness is still a subject of study.

The experience of the verification of IEEE 1394.1 through Spin suggests that the standardisation of a complex protocol can indeed benefit from a close interaction with formal methods research. The key is in a short feedback loop, where fast and practical simulations generate the direct consequences of changes made in the draft standard. Once the draft converges into a relatively stable standard, the verification regime may change towards longer feedback loops, in which the now well-understood standard can be subjected to a thorough model-checking analysis.

Future work on the application of formal methods in standardisation must go beyond the classical research themes of expressiveness and algorithmics, and focus on the interaction between research and standardisation. The research reported here is just one encouraging step in researchers meeting the dynamical complexity of standardisation, and standardisation people getting to know the analysing powers of formal methods tools.

References

[1] D. Bošnački and S. Leue, editors. *Proceedings of the 9th SPIN Workshop*, volume 2318 of *Lecture Notes in Computer Science*. Springer Verlag, 2002.

[2] E. W. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, USA, 1990.

[4] N. Goga, J. M. T. Romijn, and G. Tagviashvili. Using the I/O paradigm for Spin simulation. 2002. Available via `http://www.win.tue.nl/~jromijn`.

[5] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[6] G. J. Holzmann. The Spin model checker. *IEEE Trans. on Software Engineering*, 23(5):279–295, May 1997.

[7] The Institute of Electrical And Electronics Engineers, Inc. *IEEE Standard for a High Performance Serial Bus*, Aug. 1996. IEEE Std 1394-1995.

[8] The Institute of Electrical And Electronics Engineers, Inc. *IEEE Standard for a High Performance Serial Bus – Amendment 1*, Dec. 2001. IEEE Std 1394a-2000.

[9] The Institute of Electrical And Electronics Engineers, Inc. *IEEE P1394.1 Draft Standard for High Performance Serial Bus Bridges*, Apr. 2002. Version 1.02.

[10] S. Maharaj, J. M. T. Romijn, and C. Shankland, editors. *Formal Methods Applied to IEEE 1394, a special issue of Formal Aspects of Computing*. 2002. To appear.

[11] R. Perlman. An algorithm for distributed computation of a spanning tree in an extended LAN. *ACM SIGCOMM Computer Communication Review*, 15(4), September 1985.

[12] J. M. T. Romijn. A timed verification of the IEEE 1394 leader election protocol. *Formal Methods in System Design*, 19(2):165–194, 2001.