Asynchronous Router Specification

Faron Moller
Computing Science Department

ABSTRACT

We describe the application of three formal design tools to a case study in the design of a distributed system. The case study in question involves the specification of an asynchronous message router; the three design tools are process algebra (specifically Milner's Calculus of Communicating Systems CCS), the modal μ-calculus and the Edinburgh Concurrency Workbench (CWB). We demonstrate how an informally-presented specification can be formalised within the language of the modal μ-calculus, allowing for a rigorous mathematical analysis of the correctness of our proposed implementation. For modest-sized versions of the router, this correctness proof has been carried out using the CWB.

INTRODUCTION

There has been a great deal of effort spent on developing formal methodologies for specifying and reasoning about the logical properties of systems, be they hardware or software. Of particular concern are issues involving the verification of safety and liveness conditions, which may be of critical importance to the correct functioning of a system. However, there remains a wide gap between theory and practice, as practitioners are reluctant to adopt new formal technologies to replace long-standing intuition-based design methodologies. The reason for this reluctance is many-fold, but one aspect is the perceived difficulty in applying mathematical techniques.

In this paper we address this perception by carrying out a case study in the use of three formal design tools, concentrating our effort on an intuitive introduction to these tools. More specifically, we present a case study in the use of process algebra (specifically Milner's Calculus of Communicating Systems CCS) and the modal μ-calculus as specification languages, and the Edinburgh Concurrency Workbench (CWB) as a verification tool. To this end, we informally describe the behaviour of a system in much the same way as we might expect the specification to be given by an informal human end-user. We then show how to make this specification precise within the language of the modal μ-calculus. From here, we can perform a rigorous correctness proof of a proposed implementation, specified in CCS, showing that it satisfies (the formalised version of) the informal specification. This verification can be undertaken by the CWB.

The example we use for the case study is an asynchronous message router. This is a system which has some collection of input lines and output lines (each line with a corresponding acknowledgment line), with the property that messages are input on some input line, with each of them addressed to some output line, and the router correctly delivers the messages to their intended

Author's Current Address:
Computing Science Department, Uppsala University, P.O. Box 311, S-751 05 Uppsala, Sweden.

Based on a presentation at COMPASS '96.

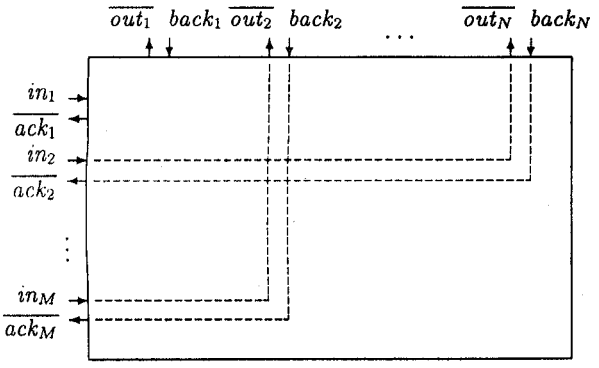


Fig. 1. The M by N Router

target output lines. The externally visible physical description of the router is thus as given in Figure 1.

By using components with acknowledgments, our circuit implementation will furthermore satisfy a delay-insensitivity property, which will ensure that the implementation performs its task correctly regardless of the delays in the circuit. The circuit will be free of *computational interference*: any time a component is ready to send a signal, the receiving component will be in a state in which it is ready to accept that signal. This will be immediate, as every component will initially be receptive, and will be receptive upon producing an acknowledgment, and no component will produce two outputs without receiving an intervening acknowledgment. Computational interference can also be checked using techniques of the modal μ-calculus as described in this paper.

In our analysis, we ignore the actual messages being relayed, and also ignore any details as to how addresses and messages are encoded. We thus assume that input values range over output addresses OA = {1, 2, . . ., N} indicating the destination of the message, and output values range over input addresses IA = {1, 2, . . ., M}, indicating the source of the message being output. The inclusion of the source of a message in the output is simply to facilitate the analysis. Thus for values ∈IA and j ∈OA, we have events $in_i(j)$ representing the input of a message on input channel i destined for output channel j; $\overline{out}.j(i)$ representing the output of a message from input channel i on output channel j; $back_j$ representing the input of an acknowledgment of the reception of a message output on channel j; and $\overline{ack}_i$ representing the output of an acknowledgment that the message most recently input on input channel i has been delivered. (Following CCS convention, we take output events to be designated by overbarred labels and input events to be designated by undecorated labels.)

In this paper we provide only the bare minimum description of each of CCS, the modal μ-calculus and the CWB in order for the uninitiated reader to understand our specifications. For a more detailed description of each of these systems we refer the reader to the following: [6] for a definitive treatment of CCS; [9] for an explanation of the modal μ-calculus and its use for expressing properties of systems; and [2, 7] for a description of the CWB and its functionalities.

ACKNOWLEDGMENT

I am grateful to Jan Tijman Udding for pointing out the design of the asynchronous message router, as well as motivating my study of it. The design itself is proposed in [5] as a method for connecting INMOS transputers together, and alternative specifications of the design are provided in [3] and [4]. This case study was developed as an example in the use of process algebra, the modal μ-calculus and the Edinburgh Concurrency Workbench for the use in lectures presented during a 10-week research visit to the University of Calgary, hosted and financed by Graham Birtwistle.

SPECIFYING THE ROUTER

When modelling systems within process algebra, the formal verification of a concurrent system typically entails presenting an intuitively clear description of the state space of the system in the same algebraic language in which the implementation is expressed, and then proving the two to be in some sense equivalent. The difficulty here arises when the state space really cannot be easily described, or is too big to be described with complete confidence. Furthermore, the specification may want to leave vague some details of an implementation, allowing several behaviourally distinct implementations to satisfy the specification, rather than specifying the behaviour of the "correct" implementation uniquely.

In specifying the behaviour of our router, we assume that the user "knows what he wants," and can explain it in "formal English." This is often the extent to which specifications of components are currently made, with the task of the further formalisation of these specifications left to the requirements engineer. Our purpose here is to show that such an informally-presented specification can be made rigorous within the language of the modal μ-calculus.

To this end, we describe the specification of the router by insisting that it satisfy the following properties.

1. Inputs and corresponding outputs occur in alternating order. That is, an output $\overline{out}_j(i)$ on output channel j of a message from input channel i cannot occur unless a corresponding input $in_i(j)$ had occurred previously; and when an input event $in_i(j)$ on input channel i occurs, a further input event $in_i(j^1)$ on the same input channel cannot occur until after the occurrence of an output event $\overline{out}_j(i)$ corresponding to the delivery of the first message input on input channel i.

2. The only possible impediment to the input ini(j) of a message on input channel i is if the router has yet to provide an acknowledgment $\overline{ack}_i$ of the delivery of a message previously input on that input channel.

3. The only possible impediment to the output $\overline{out}_j(i)$ of a message on output channel j in response to an input event $in_i(j)$ is the possibility of the router delivering some other message to output channel j at the same time. If several messages are being delivered to the same output channel, then their order of delivery is not specified; the router is free to arbitrate between the messages using any discipline, so long as it is always possible that at least one of these messages can be delivered at any given moment.
4. Upon outputting a message $\overline{out}_j$ (i) from input channel i on output channel j, the router should be immediately prepared to accept an acknowledgment $back_j$ from the environment, and to be able to deliver that acknowledgment $\overline{ack}_i$ to input channel i without any impediment.

The above list provides a complete informal specification of the router, as may be given by an end-user. The specification is informal, in the sense that it is a purely English description of the system. However, we shall demonstrate that it can be naturally translated into a temporal logic, thus allowing for a rigorous analysis of an implementation. It is complete in the sense that it specifies all that the user insists of the behaviour of a correct implementation. Different implementations may satisfy this specification (and we shall briefly discuss variations on the implementation which we provide in this paper), but they will all be deemed correct.

For example, the above specification describes nothing of the type of arbitration which goes on inside the router between messages competing for the same output line. If a message is being delivered to a particular output channel, then the router might be designed so as to allow that output to be processed ahead of all other outputs to the same channel, or at the other extreme the router may force it to await the processing of all other outputs currently being sent to that channel (and hence possibly prevent it from ever being processed). The specification only stipulates that a message must be able to be output if there are no other messages currently being delivered to the same output channel, and that if several messages are competing for an output channel, then at least one of these can be output.

FORMALISING THE SPECIFICATION

We now have a specification of a router which the person providing the specification may consider perfectly adequate. However, it is clearly necessary to formalise this specification in order to do a rigorous mathematical analysis of our proposed implementation. It is in fact rather straightforward to translate this specification into the notation of the propositional modal μ-calculus.

Briefly, the modal μ-calculus is a simple propositional logic with modalities for possibility (•) and necessity [•], as well as maximum and minimum fixpoints, νZ.P and μZ.P respectively, defining recursive properties for perpetual processes. Hence we have the following syntax:

$$P, Q \;::=\; true \mid false \mid P \wedge Q \mid P \vee Q \mid \langle K \rangle P \mid [K]P \mid \nu Z.P \mid \mu Z.P \mid Z$$

where Z ranges over some set of propositional variables and K ranges over sets of events taken from some set ∈of all possible events. We also allow modalities to be either lists of events, for example reading (a, b)P as ({a, b})P, or the negations of modalities, for example reading (-a)P as (ξ\ {a})P and [-]P as [ξ]P (so that "—" acts as a *wildcard* modality). Finally we consider as valid only *closed* formulae, that is, those formulae in which every occurrence of a variable Z is bound by a fixpoint, by being contained within a subterm of the form νZ.P or μZ.P.

We interpret the formula (K)P informally as "it is possible to do an event from the set K and evolve into a state satisfying P," and the dual formula [K]P as "if it is possible to do an event from the set K, then necessarily upon so doing we will evolve into a state satisfying P." Hence, for example, the formula (a) (b) true stipulates that it is possible to do an *a* event followed by a *b* event (that is, it is possible to do an *a* event and evolve into a state from which it is possible to do a *b* event and evolve into a state satisfying *true*, the proposition which is satisfied by any state); and the formula [—]*false* stipulates that no event is possible (that is, necessarily upon doing any event we must evolve into a state satisfying *false*, the proposition which is satisfied by no state); in other words, the state is deadlocked.

Fixpoints can be informally understood by considering their approximations through their unfoldings. The maximum fixpoint equation νZ.P can be interpreted as the infinite conjunction

$$P^0 \wedge P^1 \wedge P^2 \wedge P^3 \wedge \cdots$$

$$\text{where} \quad P^0 = true$$

$$\text{and} \quad P^{i+1} = P\left\{P^i/_Z\right\}.$$

Here we use the substitution notation P { Q/z } meaning the formula P with all free occurrences of the variable Z simultaneously replaced by the formula Q.

Maximal fixpoints are useful for expressing *safety* properties, such as the temporal logic box operator "□P" which stipulates that the property P holds in every (reachable) state. This operator is defined as the maximal fixpoint of the equation

$$\Box P = P \wedge [-](\Box P),$$

that is, □P $=^{def}$ νZP A [—]Z. Informally, □P says that P holds now, and no matter what event occurs, the system will be in a state in which □P (recursively) holds.

The dual minimal fixpoint equation μZ.P can be interpreted as the infinite disjunction

$$P_0 \vee P_1 \vee P_2 \vee P_3 \vee \cdots$$

where P_0 = false
and $P_{i+1} = p\ \{P_i/Z\}$

Minimal fixpoints are useful for expressing *liveness* properties, such as the temporal logic diamond operator "◇P" which stipulates that the property P holds in some (reachable) state. This operator is defined as the minimal fixpoint of the equation

$$\Diamond P \ = \ P \ \vee \ \langle - \rangle (\Diamond P),$$

that is, $\Diamond p \stackrel{\text{def}}{=} \mu Z.P \vee (-) Z$. Informally, ◇P says that either P holds now, or else it is possible to perform some event and be in a state in which ◇P (recursively) holds. By being a minimal fixpoint, we are stipulating that the recursive unrolling must "bottom out" at some point: the property P must hold within a finite (but not an a priori specified) number of steps.

We shall allow τ to represent internal communications which are unobservable to the environment. We can then define weak modalities as follows (where K is a set of observable events).

$$\langle\!\langle K \rangle\!\rangle P \stackrel{\text{def}}{=} \mu Z.\langle K \rangle \Big(\mu Y.P \ \vee \ \langle \tau \rangle Y \Big) \ \vee \ \langle \tau \rangle Z$$

$$[\![K]\!] P \stackrel{\text{def}}{=} \nu Z.[K] \Big(\nu Y.P \ \wedge \ [\tau] Y \Big) \ \wedge \ [\tau] Z$$

These weak modalities allow any number of internal τ's to occur before and after an event from the set K. Hence we interpret ((K))P informally as "it is possible to observe an event from the set K and evolve into a state satisfying P," and the dual formula $[K]^P$ as "if it is possible to observe an event from the set K, then necessarily upon so doing we will evolve into a state satisfying P." The sense in which we observe events is that any sequence of unobservable internal events may precede and follow the observed event.

We now have enough logical machinery in place to formalise our specification. Corresponding to the first clause of the specification, we can define the following macro:

$$Alt(a,b) \stackrel{\text{def}}{=} [b]false \ \wedge \ [a]\Big(Alt(b,a)\Big) \ \wedge \ [-a]\Big(Alt(a,b)\Big)$$

Taking the maximum fixpoint to this equation gives us a safety property which stipulates that events *a* and *b* must occur alternately in a system. The first clause of our specification then translates to the following.

$$P \stackrel{\text{def}}{=} \bigwedge_{\substack{1 \le i \le M \\ 1 \le j \le N}} Alt(in_i(j), \overline{out_j}(i))$$

Note that the property P in itself says nothing about the necessity of the events $in_i(j)$ and $\overline{out}_j(i)$ occurring; indeed the deadlocked process which exhibits no behaviour at all satisfies this property! The necessity of the occurrence of the input, output and acknowledgment events is dealt with by the remaining clauses. These are formally interpreted as follows. Firstly, we can present a property $out_j(i)$ which describes when the router is in the process of delivering a message from input channel i to output channel j:

$$\begin{aligned} Out_j(i) \stackrel{\text{def}}{=} & \ \langle\!\langle \overline{out_j}(i) \rangle\!\rangle true \\ \vee & \ \langle\!\langle back_j \rangle\!\rangle Out_j(i) \\ \vee & \bigvee_{\substack{1 \le i' \le M \\ i' \ne i}} \langle\!\langle \overline{out_j}(i') \rangle\!\rangle Out_j(i) \end{aligned}$$

Taking the minimal fixpoint to this equation gives us a liveness property which stipulates that after possibly processing the outputs of any other messages currently being delivered to output channel j (as well as some finite amount of internal computation), the output of the message from channel i must be possible. This property adheres to clause 3 above.

We can then formulate the following property:

$$Q \stackrel{\text{def}}{=} \bigwedge_{1 \le i \le M} \Box \Big(Q_1^i \vee Q_2^i \vee Q_3^i \vee Q_4^i \Big)$$

where

$$Q_1^i \stackrel{\text{def}}{=} \bigwedge_{1 \le j \le N} \langle in_i(j) \rangle true$$

$$Q_2^i \stackrel{\text{def}}{=} \langle\!\langle \overline{ack_i} \rangle\!\rangle true$$

$$Q_3^i \stackrel{\text{def}}{=} \bigvee_{1 \le j \le N} Out_j(i)$$

$$Q_4^i \stackrel{\text{def}}{=} \bigvee_{1 \le j \le N} \langle back_j \rangle \langle\!\langle \overline{ack_i} \rangle\!\rangle true$$

This property captures the intent of clauses 2, 3 and 4 above, in that it claims that at any point, the router must be able to do one of the following:

- accept an input $in_i(j)$ on input channel i destined for some output channel j;
- provide an acknowledgment $\overline{ack}_i$ signalling the delivery of a message previously input on input channel i;
- deliver a message $out_j(i)$ previously input on input channel i to output channel j;

or

- receive an acknowledgment $back_j$ from the environment signalling the reception of a message from input channel i, and then deliver that acknowledgment to the input channel i.

Hence our formal specification of the router is given as the conjunct of these two propositions:

$$Spec \stackrel{\text{def}}{=} P \ \wedge \ Q.$$

THE IMPLEMENTATION

In this section, we describe an implementation of the router in CCS which satisfies the specification given above. The proof of correctness has been carried out for modest-sized instances using the CWB.

CCS is a language for specifying concurrent communicating processes. Syntactically CCS terms take the following form

E, F	::=	$a.E$	(prefix)
	|	$E + F$	(choice)
	|	$E \mid F$	(composition)
	|	$E\backslash L$	(restriction)
	|	*if b then E else F*	(branching)
	|	X	(variable)

where X ranges over some set of process variables, a ranges over some collection of atomic events ∈including a single unobservable event τ representing an internal communication, L ranges over sets of observable events and b ranges over boolean expressions. We also generalise the binary choice and composition operators to arbitrary choices and compositions, which we write $\Sigma_{i\in I}E_i$ and $\Pi_{i\in I}$ E_i respectively.

Briefly the meaning of a CCS term is provided by the following interpretation of the constructs. Firstly, an observable action may be either an input event or an output event, possibly involving the input or output of a value. Again input events are represented by undecorated labels whereas output events are represented by overbarred labels. a.E then represents the process which performs the event *a* and subsequently evolves into the state *E*. The process E + F represents the process which may behave either as *E* or as *F* with the choice being made at the time of the first event. The process E / F represents the parallel composition of the component processes; its behaviour is that of the two component processes interleaving their events, and allowing for a handshake communication at any time if one ever has the capability of outputting (a value) with an overbarred label which equals the undecorated input label of the other process, resulting in a τ event representing the unobserved internal synchronisation. Hence for example we may have the process (a(χ)-E | a(5).F) perform the event τ and evolve into the state (E {5/χ} | F). The process E\L behaves as E but does not allow it to ever perform an event present in the restriction set L. The process if *b* then *E* else *F* is a simple decision, behaving either as *E* or as *F* depending on the evaluation of the boolean *b*. Finally *X* represents the behaviour of whatever process the variable *X* is bound to, and we shall always bind process variables to processes.

Returning to our router specification, one possible implementation would be to have one process P_i for each input chancel i dedicated to processing messages sent from that channel. Each output channel j would then have a semaphore sem_i regulating the mutual exclusion of outputs on that channel and corresponding acknowledgments, to assure that two outputs on a given channel cannot occur without an intervening acknowledgment by the environment of the reception of the first. We could describe this implementation quite simply as follows.

$$\left(\prod_{1\le i\le M} P_i \mid \prod_{1\le j\le N} Sem_j \right) \backslash L,$$

where

$$P_i \stackrel{\text{def}}{=} \sum_{1\le j\le N} in_i(j).get_j.\overline{out_j(i)}.back_j.\overline{put_j}.\overline{ack_i}.P_i,$$

$$Sem_j \stackrel{\text{def}}{=} \overline{get_j}.put_j.Sem_j,$$

$$L \stackrel{\text{def}}{=} \{get_j, put_j : 1 \le j \le N\}.$$

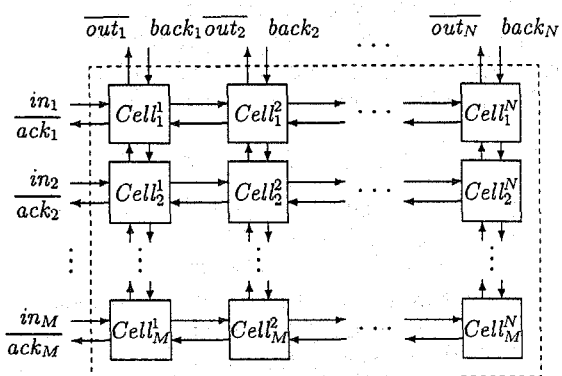


Fig. 2. Implementation of the Router

It is fairly clear that this process does in fact satisfy our specification. It is even possibly a desirable implementation, as a message being delivered to a given output channel could never be hindered from its delivery by any other message being sent to the same channel. However, the connections between the input and output channels in this implementation are fixed, requiring very many and very long connections, as each input channel must be directly connected to each output channel.

Instead, we wish to consider a more dynamic configuration of processes, where communication links between input and output lines are established only as needed. Our proposed implementation uses an array of cells, as depicted in Figure 2, which implement a simple passing/switching function. A message will travel along its source row until it reaches its destination column, at which point it will be sent upwards toward the destination output of the message, as is implied in Figure 1.

The cells are built from a simple switching element and an arbiter as described in Figure 3, on next page. Their specifications are given as follows (with the bottom row and rightmost column of cells being suitably simplified versions).

$$Cell_i^j \stackrel{\text{def}}{=} \left(C_i^j \mid Arb\right)\backslash\{s\}$$

$$C_i^j \stackrel{\text{def}}{=} in_L(x).\ \text{if } x=j \text{ then } \overline{s}(i).back_R.\overline{ack_L}.C_i^j \text{ else } \overline{out_R}(x).back_R.\overline{ack_L}.C_i^j$$

$$Arb \stackrel{\text{def}}{=} \left(s(x).A(x) \mid in_B(x).B(x) \mid Sem\right)\backslash L$$

where

$$A(x) \stackrel{\text{def}}{=} get.\left(\overline{out_T}(x).back_T.\overline{put}.\overline{back_R}.s(x).A(x) + \overline{put}.A(x)\right)$$

$$B(x) \stackrel{\text{def}}{=} get.\left(\overline{out_T}(x).back_T.\overline{put}.\overline{ack_B}.in_B(x).B(x) + \overline{put}.B(x)\right)$$

$$Sem \stackrel{\text{def}}{=} \overline{get}.put.Sem$$

$$L \stackrel{\text{def}}{=} \{get, put\}$$

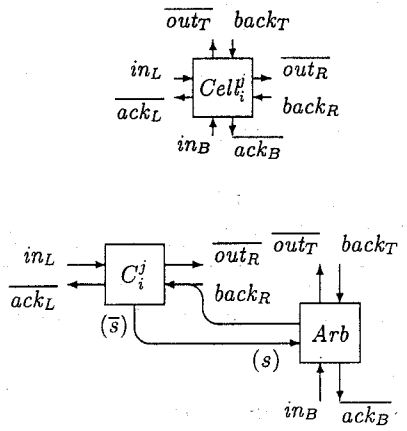


Fig. 3. Implementation of a Cell

The Arbiter is designed so it can pass on a value from its *s* line or from its in_B line to its $\overline{out}_T$ line, and arbitrate between the two if both have supplied values. This arbitration is not committed until the output takes place. The controller C_i is designed to pass messages straight through if the message is not directed to the column in which the controller resides; otherwise, the controller passes the message into the arbiter to redirect it.

The router implementation is provided by combining these cell processes in parallel, after suitably relabelling the channels, and restricting away (the observation of) the occurrences of the internal communications. The resulting implementation can be demonstrated to satisfy the formalised specification in the previous section.

CONCLUSIONS

In this paper, we have taken an informal specification of an asynchronous message router, and translated it into the formal language of the modal μ-calculus. The intent was to carry out this translation in an intuitively clear fashion, based on a simplistic view of the calculus which we tried to exploit for the benefit of the system designer who calls out for clear and intuitive formalisms. We next described a particular implementation of the router within the language of CCS which we also attempted to present in an intuitive fashion rather than as a mathematical theory. Finally we pointed out a particular tool — the Edinburgh Concurrency Workbench — with which our implementation was shown to satisfy our specification, at least for modest-sized instances of the router (of around 16 cells).

We carried out this experiment as a tutorial introduction to these three formal tools which avoided the mathematical overhead associated with a precise study of the tools and their semantic underpinnings. We feel that such an introduction is possible and desirable, as an enticement for engineers for using formal methods, and for designers of formal theories for making their theories accessible to the working engineer. It has in fact been used successfully as a teaching example.

There are other CCS-based case studies which use the CWB as an analysis tool and which have been used successfully for teaching concepts. Two particularly delightful examples are the analysis of an electronic mail system by Brebner [1] and the verification of a CSMA/CD-protocol by Parrow [8]. However, the former study is concerned only in the problem of checking for deadlocks, while the latter is interested only with demonstrating equivalence between a specification and an equivalence; both of these problems are carried out by the CWB using dedicated routines which make no use of the model-checking facility. Our present example on the other hand is aimed at demonstrating that the logic is understandable and useful as well.

REFERENCES

[1] Brebner, G., (1993),
A CCS-Based Investigation of Deadlock in a Multi-Process Electronic Mail System,
Formal Aspects of Computing 5(5):467-479.

[2] Cleaveland, R., J. Parrow and B. Steffen, (1993),
The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems,
ACM Transactions on Programming Languages and Systems 15:36-72.

[3] Creveuil, C. and G-C. Roman, (1994),
Formal Specification and Design of a Message Router,
ACM Transactions on Software Engineering and Methodology, 3(4):271-307.

[4] Josephs, M.B., R.H. Mak and T. Verhoeff, (1991),
Asynchronous design of a router,
IEEE Symposium on Circuits, Systems and Signal Processing.

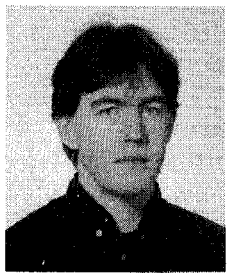
[5] May, D. and P. Thompson, (1990),
Transputers and routers: components for concurrent machines,
In Transputer/occam Japan 3,
T.L. Kuniis and D. May (editors),
1SO Press, pp. 3-20.

[6] Milner, R., (1989),
Communication and Concurrency,
Prentice-Hall International.

[7] Moller, F., (1991),
The Edinburgh Concurrency Workbench (Version 6.0),
Department of Computer Science Technical Note LFCS-TN-34,
University of Edinburgh,
Edinburgh, Scotland.

[8] Parrow, J., (1988),
Verifying a CSMA/CD-Protocol with CCS,
In Protocol Specification, Testing, and Verification VIII,
S. Aggarwal and K. Sabnani (editors),
North-Holland,
pp. 373 384.

[9] Stirling, C., (1996),
Modal and temporal logics for processes,
In *Logics for Concurrency: Structure vs. Automata,*
F. Moller and G. Birtwistle (editors),
Lecture Notes in Computer Science,
1043:149-237.



Faron Moller joined the Computing Science Department at Uppsala University in 1996. Before this he was Vikariat Professor of Distributed Systems in the Department of Teleinformatics at the Royal Institute of Technology (KTH), Stockholm; Senior Research Fellow at the Swedish Institute of Computer Science (SICS), Stockholm; Senior Research Fellow in the Laboratory for the Foundations of Computer Science (LFCS), Edinburgh; and Lecturer at the University of Strathclyde, Glasgow. He has also held visiting positions at the University of Calgary, Canada; the Electrotechnical Laboratories (ETL), Japan; and the Software Verification Research Centre (SVRC), Australia. He has been working in concurrency theory for over 10 years. His current interests are three-fold: algorithmic results for infinite-state automata; temporal models of concurrency; and the theory and (semi-) automated application of concurrency semantics. He worked on the development of the Edinburgh Concurrency Workbench (CWB), an automated tool for CCS, and the Mobility Workbench (MWB), an analogous tool for the pi-calculus, as well as developed the Temporal CCS calculus; most recently he has been concentrating on decomposition and decidability results for infinite-state systems. He has over 30 papers in refereed journals and conference proceedings.