Specification and Analysis of a Reliable Broadcasting Protocol in Maude*

Grit Denker†, J.J. Garcia-Luna-Aceves‡, José Meseguer†,
Peter Csaba Ölveczky†, Jyoti Raju‡, Brad Smith‡,
Carolyn L. TalcottII

† Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA

{denker,meseguer,peter}@csl.sri.com

‡ Computer Engineering Department, University of California, Santa Cruz, CA 95064, USA

{jj,jyoti,brad}@cse.ucsc.edu

II Computer Science Department, Stanford University, Palo Alto, CA 94305, USA

clt@sail.stanford.edu

1 Introduction

The increasing importance, criticality, and complexity of communications software makes very desirable the application of formal methods to gain high assurance about its correctness. These needs are even greater in the context of active networks, because the difficulties involved in ensuring critical properties such as security and safety for dynamically adaptive software are substantially higher than for more static software approaches.

There are in fact many obstacles to the insertion of formal methods in this area, and yet there is a real need to find adequate ways to increase the quality and reliability of critical communication systems. As a consequence, in spite of the existence of good research contributions in formal approaches to areas such as distributed algorithms and cryptographic protocols, in practice new systems are developed for the most part in a traditional engineering way, using informal techniques, and without much to go by before detailed simulations or an actual implementation except for pseudocode and informal specifications.

The present work reports on an ongoing case study in which a new reliable broadcasting protocol (RBP) currently under development at the University of California at Santa Cruz (UCSC) has been formally specified and analyzed, leading to many corrections and improvements to the original design. Indeed, the process of formally specifying the protocol, and of symbolically executing and formally analyzing the resulting specification, has revealed many bugs and inconsistencies very early in the design process, before the protocol was implemented.

RBP performs reliable broadcasting of information in networks with dynamic topology. Reliable broadcasting is not trivial when the topology of the network can change

*Supported by DARPA through Rome Laboratories Contract F30602-97-C-0312, by DARPA and NASA through Contract NAS2-98073, by Office of Naval Research Contract N00014-96-C-0114, and by National Science Foundation Grants CCR-9505960 and CCR-9633363.

due to failure and mobility. The aim is to ensure that all nodes that satisfy certain connectedness criteria receive the information within finite time, and that the source is notified about it. The protocol should furthermore incur as low latency and as few messages as possible.

The amount of effort required in the formal specification and analysis process has been moderate, and it has been relatively easy for the researchers at UCSC to learn and use the specification formalism involved, so that the formal specification task has indeed been carried out in a joint way by researchers at SRI International, Stanford, and UCSC.

Reflecting upon the reasons for the success of this experiment, we can mention the following:

- *Early insertion of the formal method.* In this way, maximum benefit can be obtained, since the design can be corrected very early, before heavy implementation efforts have been spent.

- *Simplicity and intuitive appeal of the formalism.* The formalism involved, namely rewriting logic [12], is very simple and it is very well suited for specifying distributed systems, in which local concurrent transitions can be specified as rewrite rules.

- *Executability.* Rewriting logic specifications are executable in a rewriting logic language such as Maude [3]. This means that the formal model of the protocol becomes an *executable prototype*, that can be directly used for simulating, testing and debugging the specification.

- *Formal analysis.* Since the behavior of a communications protocol is highly concurrent and nondeterministic, a particular simulation run only exhibits *one* among many possible behaviors. Therefore, although direct execution can already reveal many errors and inconsistencies, a much greater confidence on the correctness of the design can be gained by formal analysis techniques in which *all* possible behaviors—up to termination, or up to a certain depth, or up to satisfaction of a specific state condition—are analyzed in detail. This can be done in Maude by means of exhaustive execution strategies that achieve a form of "model checking" analysis of the state space.

Formal Methodology

The formal methodology underlying our approach can be summarized by stating that *a small amount of formal methods can go a long way.* Approaches requiring full mathematical verification of a system can be too costly. Proof efforts should be used judiciously and selectively, carefully choosing those properties for which a very high level of assurance is needed. But there are many important benefits that can be gained from "lighter" uses of formal methods, without necessarily requiring a full-blown proof effort.

The general idea is to have a series of *increasingly stronger methods*, to which a system specification is subjected. Only after less costly and "lighter" methods have been used, leading to important improvements and corrections of the original specification, is it meaningful and worthwhile to invest effort on "heavier" and costlier methods. Our approach is based on the following, increasingly stronger methods:

1. *Formal specification.* This process results in a first *formal model* of the system, in which many ambiguities and hidden assumptions present in an informal specification are clarified.

2. *Execution of the specification.* If the formal specification is executable, it can be used directly for simulation and debugging purposes, leading to increasingly better versions of the specification.

3. *Formal model-checking analysis.* Errors in highly distributed and nondeterministic systems not revealed by a particular execution can be found by more a sophisticated model-checking analysis that considers all behaviors of a system from an initial state, up to some level or condition. In this way, the specification can be substantially hardened, and can even be formally verified if the system is finite-state.

4. *Narrowing analysis.* By using symbolic expressions with logical variables, we can carry out a symbolic model-checking analysis in which we analyze all behaviors not from a single initial state, but from the possibly infinite set of states described by a symbolic expression. Some of these analyses are already a special type of formal proof.

5. *Formal Proof.* For highly critical properties it is also possible to carry out a formal proof of correctness, which can be assisted by formal tools such as those in Maude's formal environment [4].

Up to now, we used methods 1–3 in the present case study, reaping important benefits from this use. In the future we may use methods 4 and 5 for selected purposes. The above case study is part of a broader effort to use rewriting logic and Maude in the context of active networks and network security. Related studies include: formal specification and analysis of cryptographic protocols [7]; work by Denker and Millen using Maude to specify the CAPSL and CIL specification languages [8]; and joint work with Carl Gunter and Yao Wang at the University of Pennsylvania on the formal analysis in Maude of an active-network algorithm written in the PLAN language [10].

The rest of the paper is as follows. In Section 2 we explain the key ideas of the RBP protocol and discuss its informal and formal specifications. In Section 3 we then discuss our use of methods 2–3 to further increase our assurance by means of execution and model-checking formal analysis. We finish the paper in Section 4 with some concluding remarks about our experience, and about future work.

2 Informal and Formal Specification of the RBP

2.1 Informal Specification

Little work exists on reliable broadcast protocols in dynamic networks. Most broadcast protocols are based on the PI and PIF protocols [17, 16], and all the broadcasting protocols for dynamic topologies extending these are based on the routing protocol by Merlin and Segall [11], which incurs too much communication to be attractive for, say, a wireless network [9]. It is the goal of our specification effort to come up with a more general approach (in terms of mobility of nodes) and more efficient protocols for reliable broadcasting in dynamic networks than exists so far. The starting point of our effort was an informal draft specification of the protocol given in [9]. We will refer to this description throughout the text as the original pseudocode.

In the following we illustrate part of the somewhat abbreviated version of the informal specification given in [9], which describes the proposed protocol for dynamic topologies.

The network is modeled as an undirected connected graph, where each node knows its neighbors but does not know anything more about the topology. In dynamic networks links can fail and come up. The objective of the protocol is to broadcast the message from a source node to all the nodes in the network and to give feedback to the source.

The dynamic case requires the handling of connecting and disconnecting network sections. If a new link (i, b) is established and i and b reside in different, unconnected parts of the network, then messages that have been sent to i must be forwarded to b and vice a versa. For example, if i has heard the latest message from source j, and b has not yet heard it because it had no physical path to j, then i forwards this message. There can be more than one message in a network. For example, different partitions of the network can have different messages which are in transit, and those messages might be forwarded to another partition when a new link comes up. Thus, each node has to store the latest number of a message with respect to a specific source node. Similarly, all other information, like the state of a node or its parent, is parametric in the source node. Thus, most of the node attributes have the source node as parameter. In particular, the following attributes form the data structure of a node: (1) a set of direct neighbors and several attributes which store information with respect to a given source node j. These are: (2) a flag which specifies whether the node is active or passive for j (i.e., a node is active if it has sent the latest message of j to its neighbors and has not yet received all acknowledgements); (3) a parent for messages from j, that is, the node from which it got first the latest message of source j; (4) the latest message number of messages from j; and (5) the state of a neighbor with respect to j, i.e., a flag which specifies whether the node is active or passive for a neighbor with respect to a message from j. A node i being active for a neighbor k with respect to a message from a source node j means that i has forwarded j's message to neighbor k but still waits for an acknowledgement from k.

Because of space limitations we only explain the behavior of network nodes upon the receipt of a message. The full specification tackles the cases of acknowledgements, link deletion and link addition as well, and can be found in [6].

The reaction of node i upon receipt of message m from p depends on the message number. In accord with the pseudocode suggested by in [9], we use the following notation to refer to attributes: N^i denotes the neighbors of node i (`nbs`), and for source node j ST^i_j denotes the state of node i (`state(j,_)`), s^i_j denotes the parent of i (`parent(j,_)`), SQ^i_j denotes the last sequence number of i (`seqNo(j,_)`), and ST^i_{jk} denotes the state of i with respect to neighbor k (`nbState(j,k,_)`). Figure 1 presents a pseudocode specification of the behavior later formalized in Maude for receiving a message: There are three main cases for receiving a message.

1. If a node repeatedly receives the current message (i.e., $SQ^i_j = m$), then it only replies with an acknowledgement to the sender.

2. If a node receives a newer message ($SQ^i_j < m$) or a message from a node for which it has no current message number (as it is the case if a new link comes up), then it stops an earlier diffusion source and starts a new one with the newer message. The node from which it got the latest message becomes the parent ($s^i_j := p$). It forwards the newer message to all neighbors other than p and waits for their acknowledgement. In case there are no more neighbors, it can acknowledge the receipt of the message.

3. If a node receives an older message ($SQ^i_j > m$), then its reaction depends on its state. If the node is still active, then it extends the ongoing diffusion computation

if $SQ^i_j = m$
  send ack$_m$ to p for source j
if ($SQ^i_j < m$ or SQ^i_j is not defined)
  $s^i_j := p$, $SQ^i_j := m$, $ST^i_{jp} := passive$
  if i has at least one more neighbor other than p
    $ST^i_j := active$, $ST^i_{jk} := active$ for all $k \in N^i - p$,
    send message m for source j to all $k \in N^i - p$
  if $N^i = p$
    $ST^i_j := passive$, send ack$_m$ for source j to p
if $ST^i_j = active$ and $SQ^i_j > m$
  ST^i_{jp} := active, send message SQ^i_j for source j to p
if $ST^i_j = passive$ and $SQ^i_j > m$
  $ST^i_j := active$, $s^i_j := i$, $ST^i_{jp} := active$,
  send message SQ^i_j for source j to p.

Figure 1: Receiving a message.

by notifying the sender about the newest message. If the node is already passive, then it starts a local diffusion computation with itself as the source.

2.2 Formalization Process

There are many fuzzy requirements and much implicit knowledge in most informal specifications, such as in the starting point of our specification effort, a specification written in natural language and pseudo-code. The *formalization* part of the specification process aims at clarifying those requirements and making essential implicit knowledge explicit. For example, during the formalization process we realized that it was not clear whether one should take into account scenarios where an acknowledgment from node i to j may be received before the message sent from i to j. In other words, are messages/acknowledgments sent from i to j received in the same order as they are sent? This important aspect was not mentioned in the informal specification and provides an example of how the formalization effort helps to make implicit assumptions explicit. After discussing the matter it became clear that the implicit intention was in fact that the order of messages is preserved along a channel.

2.3 Maude Specification

The protocol is specified in *rewriting logic* [12, 13, 14, 15]. Rewriting logic is an executable logic which extends algebraic specification techniques to concurrent and reactive systems. Among its possible advantages over other executable specification formalisms are its being based upon a natural and well-known formalism, its natural integration of static and dynamic system aspects, the abstract modeling of communication, and its possibility to define execution strategies in the logic itself [1, 3, 2, 5]. All these advantages are fully supported by the Maude rewriting logic language [3]. Rewriting logic seems particularly suitable for specifying communication protocols, including those used for security. Such protocols are complex enough to warrant prototyping and their operational nature fits very well with the executable character of rewriting logic. The use of rewriting logic for the specification and analysis of security protocols is shown in [7].

Maude Basics

We briefly outline the syntactic key features of Maude that are essential for our case study. We model a distributed system configuration as a soup (multi-set) of concurrent objects and messages that behave according to a set of rewrite rules describing the behavior of individual objects. Maude allows to declare object-oriented *classes* with the following syntax

```
class C | a1: S1, ... , an: Sn
```

where `C` is the name of the class, and each `ai` is an *attribute identifier*, and `Si` is the sort inside which the values of the attribute identifier `ai` must range. An *object* in a given state is represented as a term `<O: C|a1: v1, ... , an: vn>`, where `O` is the object's name or identifier, `C` is its class, the `ai`'s are the object's attribute identifiers, and the `vi`'s are the corresponding *values*. Message-constructing operators are introduced with the keyword `msg`. We can think of a system as a "soup" in which objects and messages float, so that any objects and messages can at any time come together and participate in a concurrent transition corresponding to a communication event of some kind.

Defining Network Nodes and Messages

We define the following class in Maude, which specifies the nodes of the network.

```
  class Node | nbs : OidSet, states : IdStatusPFun,
               parents : IdIdPFun, seqNos : IdIntPFun,
               nbsStates : IdIdStatusPFun .
```

The set of neighbors is a set of object identifiers. The other attributes are partial functions, that is, for a given source node identity, they deliver the state (`states : IdStatusPfun`), the identity of the parent (`parents : IdIdPFun`), the latest sequence number (`seqNos : IdIntPFun`), or, for each neighbor the state of the neighbor (`nbsStates : IdIdStatusPFun`). We modeled these data types in Maude by defining pairs and tuples which have as the first parameter always the identity of the source node. Partial functions are sets of pairs or triples. A variety of constants and operators is defined for these sorts. Details can be found in [6].

Assume the following node instance:

```
 < i : Node | nbs : set(k,l), states : state(j,active),
              parents : parent(j,k), seqNos : seqNo(j,5),
              nbsStates : nbState(j,l,active) >
```

Node i has two neighbors, k and l. Currently, i is active in a diffusion computation for source j. The latest message it got from j has the number 5, and i already forwarded this message to its neighbor l without having yet received an acknowledgement. Thus, i is in an active neighbor state for l with respect to source j.

The main messages in the system are: (1) sending and acknowledging a message (number) from a source node between network nodes; and (2) establishing and deleting links between nodes. We can formalize this in Maude by defining messages:

```
msg msg_To_From_Src_ : MachineInt Oid Oid Oid -> Msg .
msg ack_To_From_Src_ : MachineInt Oid Oid Oid -> Msg .
msg newlink : Oid Oid -> Msg .
msg failure : Oid Oid -> Msg .
```

In the following, to give a flavor for how RBP is formalized, we describe one of the rules for the dynamic RBP. The full specification is given in [6].

Receiving a message

As we pointed out before, there are three main cases for receiving a message.

If a node repeatedly receives the current message (i.e., $SQ^i_j = m$), then it only replies with an acknowledgement to the sender. This corresponds to the Maude rule

```
  rl [RepeatRecCurrentMsg] :
       < A : Node | nbs : nbs(B,OIDSET),
                    states : IDSTATUSPFUN,
                    parents : IDIDPFUN,
                    seqNos : seqNos(seqNo(C,M),IDINTPFUN),
                    nbsStates : IDIDSTATUSPFUN >
       (msg M To A From B Src C)
     =>
       < A : Node | nbs : nbs(B,OIDSET) >
       (ack M To B From A Src C) .
```

The lefthand side of the rule defines the precondition for the firing of the transition. In the above rule we require that at least one node and one message have to be in the network. The node with name `A` (`A` is a variable) has a neighbor `B` and the latest message number `A` heard from a source `C` was `M`. If `M` is also the currently received message number from source `C`, then it reacts by sending an acknowledgement back to `B`. The receiving node doesn't change its state.

3 Analysis Techniques using Maude

3.1 Validation with the Default Interpreter

A major advantage of rewriting logic specifications is that they can be *validated* immediately by executing test cases to provide quick feedback on the specification. This prototyping possibility comes for free. We used this feature every time the specification was modified, and often encountered errors quite easily on quite simple test examples (such as a network of three nodes). In this validation effort, we executed the test cases using Maude's default interpreter, which simulated some arbitrary run of the protocol for a given initial state of a network. The validation effort helped eliminate errors of syntax and of thought; furthermore, the built-in Maude facilities for tracing an execution were useful for discovering where the error occurred. This validation and correction cycle led to substantial improvements on, and a clear formalization of, the basic ideas of the starting informal protocol. In particular, it was agreed upon that the following follows from the informal specification:

1. A node should acknowledge all its "siblings" when it has received something (message or acknowledgment) from all its neighbors.

2. A node should acknowledge its "parent" node when it has received acknowledgments from all its neighbors except the parent.

As a first form of analysis, our final version of the protocol based on these ideas was validated using Maude's default interpreter. The extensive testing always returned the expected (and hoped-for) result.

3.2 Formal Analysis of the State Space

To substantially increase our confidence in the specification before any costly attempt at a formal proof of correctness, the specification can be subjected to close formal analysis using the meta-programming features of Maude to explore all states and behaviors that can nondeterministically reached from an initial state. Since the specification should be terminating, one could apply a strategy that explores all possible rewrite paths from some given initial state. In particular, we wrote a strategy for finding every non-rewritable state reachable from the initial state. For non-terminating systems, this setting can be modified to give e.g., every state which is reachable in less than 50 one-step rewrites from some initial state.

We experimented with different, increasingly complex, versions of the protocol. For example, executing the specification using an exhaustive strategy on a clique with three nodes did *not* produce the hoped-for result, namely a singleton set of irreducible states. Instead, the set of irreducible configurations reachable from the clique of three nodes included a term which indicated a deadlock before the expected end of one round of the protocol. A simple analysis of the output explained the error. If a node `a` is both "grandparent" and "sibling" to a node `b`, then, according to the ideas underlying the protocol, there is a deadlock as follows: Node `c` cannot acknowledge its parent `a` before it has received the "parent-acknowledgment" message from its child `b`. Node `b` cannot acknowledge its parent `c` before it has received an acknowledgment from its sibling `a`. And `a` cannot acknowledge `c` before it has received something (which in this case only can be an "parent-acknowledgment" message) from `c`!

Using Maude's meta-programming features, the user may himself define the rewrite strategies, thereby analyzing the specification in various ways. Although we only needed the quite straightforward exploratory analysis to invalidate our first version, this capability is very useful for analyzing various executions and extracting the relevant information from these automatically. Using the information from the exploratory analysis, the group came up with a new improved version of the protocol.

4 Concluding Remarks

The effort of formally specifying RBP for dynamic networks using the executable specification language Maude brought to light several weaknesses of the original pseudocode [9]. However, the analysis is not yet finished. For the moment, the following problems were identified and solved:

1. As described above we could eliminate a deadlock situation in the given protocol.

2. The pseudocode on which we based our first specification of the dynamic RBP was incomplete. In several places it was not clear what are the assumptions about node attributes.

3. A description of initial state was missing, and so was the termination condition for the protocol.

4. Moreover, the original pseudocode was incomplete with respect to how attributes are updated.

5. Some cases were left out in the original specification.

6. Other essential errors could be detected using the strategies. For example, given a test scenario with three nodes a, b, and c where b has the neighbors a and c, a is a source nodes which sends message number one and the link between a and b breaks down. Running the protocol with this initial configuration using an exhaustive search strategy delivers three different states of which one is a correct state, the second one reveals an undesired behavior and the third one showed an error in the original pseudocode which has been corrected in the current version.

Details can be found in [6]. The analysis is not finished yet. Our current implementation works fine for general test cases using a default strategy. But abnormal behavior appears when we run the protocol with an exhaustive search strategy which traverse all possible behaviors. This test cases are currently under investigation and will lead to changes in the protocol pseudocode. Moreover, further test cases in which link additions and deletions are combined with sending one or more messages are currently under investigation.

We have some further suggestions concerning other cases in the pseudocode which we have not implemented yet. We are currently discussing those suggestions.

References

[1] M. Clavel. *Reflection in general logics, rewriting logic, and Maude.* PhD thesis, University of Navarre, 1998.

[2] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Martí-Oliet, and J. Meseguer. Metalevel computation in Maude. In *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, Electronic Notes in Theoretical Computer Science. Elsevier, 1998.

[3] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. *Maude: Specification and Programming in Rewriting Logic.* Computer Science Laboratory, SRI International, Menlo Park, 1999. `http://maude.csl.sri.com`.

[4] M. Clavel, F. Durán, S. Eker, and J. Meseguer. Building equational proving tools by reflection in rewriting logic. In *Proc. of the CafeOBJ Symposium '98, Numazu, Japan.* CafeOBJ Project, April 1998. `http://maude.csl.sri.com`.

[5] M. Clavel and J. Meseguer. Reflection and strategies in rewriting logic. In J. Meseguer, editor, *Proc. 1st Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1996.

[6] G. Denker, J. J. García-Luna-Aceves, J. Meseguer, P. C. Ölveczky, Y. Raju, B. Smith, and C. Talcott. Specifying a reliable broadcasting protocol in Maude. Technical report, Computer Science Laboratory, SRI International, Menlo Park, 1999. Available at `http://www.csl.sri.com/~denker/pub_99.html`.

[7] G. Denker, J. Meseguer, and C. Talcott. Protocol Specification and Analysis in Maude. In N. Heintze and J. Wing, editors, *Proc. Workshop on Formal Methods and Security Protocols, 25 June 1998, Indianapolis, Indiana*, 1998.

[8] G. Denker and J. Millen. CAPSL Intermediate Language. In N. Heintze and E. Clarke, editors, *Workshop on Formal Methods and Security Protocols (FMSP99)*, Trento, Italy, 1999. Available at `http://www.csl.sri.com/~denker/pub_99.html`.

[9] J. J. Garcia-Luna. Reliable broadcasting in computer networks, 1998. Manuscript, UC Santa Cruz.

[10] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, pages 86–93. ACM, 1998.

[11] P. M. Merlin and A. Segall. A failsafe distributed routing protocol. *IEEE Trans. Commun.*, 27(9):1280–1288, 1979.

[12] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[13] J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In G. Agha, P. Wegner, and A. Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT Press, 1993.

[14] J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Proc. Concur'96*, volume 1119 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[15] J. Meseguer. Research directions in rewriting logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany, July 29 – August 6, 1997.* Springer-Verlag, 1998.

[16] A. Segall. Distributed network protocols. *IEEE Trans. Info. Theory*, 29(1):25–35, 1983.

[17] A. Segall and B. Awerbuch. A reliable broadcast protocol. *IEEE Trans. Commun.*, 31(7):896–901, 1983.