An Executable Model for a Family of Election Algorithms

Wei Shi[1] , Jean-Pierre Corriveau[1]

[1]School of Computer Science, Carleton University
5302 Herzberg Building, 1125 Colonel By Drive,
Ottawa, Canada
{swei4,jeanpier}@scs.carleton.ca

Abstract

In this paper, we present an executable model for a family of algorithms dealing with leader election in a ring topology. We follow the traditional approach of system family engineering [7]. That is, we develop a feature model that captures variability across these algorithms. We then proceed to produce a generator. This generator receives as inputs specific values for each of the variation points (i.e., features) we identify. And it produces the behavior corresponding to the specific configuration of features at hand. Contrary to existing generative programming literature, we do not resort to C++ meta-programming but instead develop an executable model using Rational Rose RT. More precisely, we have designed a single State Chart that can model all the algorithms of the family we studied. We focus here on how to obtain such a State Chart, rather than on the identification of the features we used, or on ROSE-RT semantics. We do believe however that our approach can be reused to provide a semantically unified and executable modelling approach for other families of algorithms.

1 Introduction

In traditional algorithm discovery and optimization approaches, researchers usually design and improve a single algorithm. That is, they either come up with a new algorithm to solve a new problem, or improve an existing algorithm to get better performance. This kind of single-solution approach comes at a very high cost, each algorithm being typically highly customized. From a software engineering viewpoint, *reuse* is presented as a solution to lower design costs[6]. System family engineering (SFE) [7] proposes that we address variability across a domain in order to maximize reuse. In this paper, the domain we choose is a family of algorithms for leader election in a ring. One fundamental aspect of SFE is that we are to develop a *generative* model: family members are not exhaustively modelled. Quite on the contrary, we are to build a generator that can be configured in terms of its variation points (called features). Given a configuration of feature values, a generator can produce the corresponding family member. For example, a generator for a car would input the car's options (such as color, number of doors, type of engine, etc.) and generate the requested vehicle.

Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)

IEEE COMPUTER SOCIETY

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

We believe we are the first to apply the ideas of SFE to algorithms. And we contend this combination offers significant advantages. First, a domain viewpoint will naturally lead to a better understanding of the differences between existing algorithms, as well as to facilitating exploring new combination of features corresponding to new algorithms. And we can hope to develop test suites that hold across a domain, as opposed to being specific to a single algorithm. Furthermore, in research on algorithms, functional requirements (i.e., expected behavior) are expressed very precisely. As are non-functional requirements such as space and time complexity. Such precision readily provides an objective method to compare the different members of the family we model (both in terms of functionality and performance). Such a comparison method often lacks in ill-defined domains.

In this paper, we do not focus on comparison between the different members of the family we modelled: an application engineering environment [1, 3, 7, 9] is required for such a task. Instead, we focus on modelling the domain we chose. Let us elaborate.

In distributed computing, leader election is one of the most often used solutions to solve several recurring problems. In particular, algorithms for the ring topology have received considerable attention. Though a multitude of such algorithms have been proposed, we will limit ourselves to the ones presented by Santoro[10]. This group of algorithms includes most algorithms proposed since G. LeLann published his first famous ring leader election algorithm in 1977[8].

We will first briefly explain the set of features we use to model the variability inherent to this domain. We then explain how we develop our generator. The latter takes the form of an executable State Chart implemented and tested in Rational Rose Real Time (hereafter RRRT). Whereas SFE has produced generators in complex C++ or Prolog, we believe our generator is much simpler to obtain and presents significant advantages. Specifically, whereas the metaprogramming techniques used in SFE all but prevent debugging, using a commercial case tool and the well-known concept of state machines promotes ease of design and ease of debugging and testing. Indeed, Binder [2] has presented several systematic testing techniques for state machines.

2 Feature Modelling of Leader Election in a Ring

2.1 Leader Election in a Ring

Whether used as a solution for simplifying many complex distributed computing problems, or because of the nature of the problem itself, the idea of selecting a single coordinator from a population of autonomous symmetric entities plays a crucial role in distributed computing. The task of selecting a single coordinator is known as the problem of Leader Election [10].

Formally, the task consists in moving the system from an initial configuration where all entities are in the same state (usually called available) into a final configuration, where all entities are in the same state (traditionally called follower), except one that is in a different state (traditionally called leader). There is no restriction on the number of entities that can start the computation, nor on which entity should become leader. Leader Election can happen in different topologies, such as tree, ring and general graph. The ring is of particular interest to many researchers and practitioners.

A ring consists of a single cycle of length n. In a ring, each entity has exactly two neighbors (whose associated ports are) traditionally called left and right. Rings form networks with the sparsest topology, namely m = n. However, unlike trees, rings have a complete structural symmetry (i.e., all nodes look the same) as opposed to the inherent asymmetry of trees (e.g., the existence of internal and leaf nodes)[10].

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

2.2 Feature Model for Leader Election in Ring

Feature modelling is the key step in engineering a family of systems to capture the commonalities and variability of this domain [7]. Feature discovery, however, lies beyond the scope of this paper.

2.2.1 Brief Feature definitions

After studying the target family, we grouped our features into three groups. They are: Topology features, Node Behavior features and Protocol features. Under each feature group, we have several features and associated feature values. We summarize our work below. Further details are available in[11] and [12].

- Topology group

The Topology group consists of two features: Ring Type and Direction of the Ring. 'Ring Type' is a feature to describe whether each node in the ring can send messages into both directions or only one. This feature has two feature values: "Unidirectional Ring" and "Bidirectional Ring". The feature 'Direction' captures whether a ring has a unique sense of direction or not. Its two feature values are "Oriented" and "Unoriented". In an unoriented ring, left does not necessarily mean the same thing for every node!

- Protocol group

The protocol group has five mandatory features and three optional feature.

Election Scope

- Among all nodes: election is required to be executed among all the nodes
- Among initiators: Sometimes the election do not need to be executed among all the entities in the ring. Only those that send an 'initiate' message participate. In this case, the nodes that do not send this initial message are called relay nodes.

Election Strategy

- Elect minimum: the node with the smallest value is elected as the leader
- Elect maximum: the node with the greatest value is elected as the leader
- Elect "min-max": the algorithm will execute "minimum" or "maximum" election at different stages (see below).

Stages

- Single stage: A node makes only one attempt to become leader: it sends a message containing its id only once.
- Multiple stages: A node makes several successive attempts to become a leader. Each time a node originates a message, it starts a new stage. The nodes will elect a leader after more than one stage.

Send Message Strategy

- Randomly choose direction: if a node can send messages to both of its neighbors, then each time it must send a message, it randomly chooses one neighbor as the destination.
- Send message to both neighbors: each node sends each message to both of its neighbors.
- Send to the open direction: each node sends each message to the unique open direction.

Terminate Passing Message Strategy

- Complete: each message travels along the ring until it comes back to its originator
- Block: each message travels until it arrives at a node with a smaller ID in "Elect minimum" strategy, or until it arrives at a node with a greater ID in "Elect maximum" strategy
- Originator controls the distance: the originator of the message controls how many nodes the message should travel along the ring
- To the next candidate: the message will travel along the ring until it arrives at the next candidate node (possibly through several relays).

Keep Message Order Strategy

- Queuing management: use a queue to keep the message arrival order, when different parts of the ring have different speed.
- "Ignore" management: only process the message with the greater stage number, and ignore all the messages from the lower stage, when different parts of the ring have different speeds.

Stage Advance strategy

- Do not use feedback: Do not require or

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

use a feedback message to decide whether a candidate should advance to the next stage.
- Use a feedback mechanism: If a message successfully reaches its target, the target will send a feedback message back to the message originator, the originator will decide whether to advance to the next stage accordingly.

Send Message Mode
- Simultaneous send: each node sends the message to both sides along the ring at the same time
- Alternate send: each node sends the message to one side in one stage, then the other side in the next stage.

- Node Behavior group
The Node Behavior group has two mandatory features: Notification feature and Behavior feature. After the leader is elected, all the nodes in the ring should stop the election immediately. There are two ways to make the entire ring get into this 'terminate' stage: one is to have the leader send a notify message to make all the nodes know who has been elected: this is called "Need to notify". The other way is to have each node decide itself whether it is the leader according to all the messages it got. Each node should also figure out who is the leader. This is called "Do not need notify".

2.2.2 Feature Combination Rules

There are nine combination rules for features in our feature model for leader election in a ring. Here we just give one example to illustrate such combination rules:

- If the ring type is Unidirectional, then feature value "Alternate send" must be chosen for feature "Send Message Mode".
This combination rule describes a relationship between features "Ring Type" and "Send Message Mode".

3 Integrated State Machine for Leader Election in Ring

3.1 Why State Machine

One of the key contributions of the Unified Modelling Language (UML), which is supported by RRRT, is its rich and precise semantics and its extensive support (e.g., simulation, code generation) for State Charts. Such state machines are essential for the construction of executable models that can effectively react to incoming events in a timely fashion.[4] [5]

3.2 Integrated State Machine

As Binder points out in his book[2], a state machine is an abstraction composed of events (inputs), states, and actions(outputs). A transition typically takes a system from one state to another. The initial state is the state in which the first event is accepted. State machines have sequential behavior.
From the previous section, we identified the feature "Stage". In the single stage algorithms, each node only originates a message once for election to take place. Each node will thus go through Initial state, Electing state, finally it goes into state Won (becomes leader) or state Follower (becomes follower), then end up to state Terminate. see Figure 1.

Here the 'Electing' state will be triggered by the "Receiving Message" transition, or by the "Start" transition (which means the node originated a message and sent it to one of its neighbors). Getting a notify message from the elected leader is the trigger to terminate a node by having it become a follower. This will happen when feature value "Need notify" was chosen for feature "Notification". If the feature value "Do not need notify" is chosen for feature "Notification", then after each node gets all the messages in the network, the node with the smallest ID (if we have feature value "Elect minimum" for feature "Election Strategy") or the greatest ID (if "Elect minimum" for feature "Election Strategy") will step into the "Terminated" state. This is the other trigger to terminate an electing node– "terminateddirectely". We will return to this

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

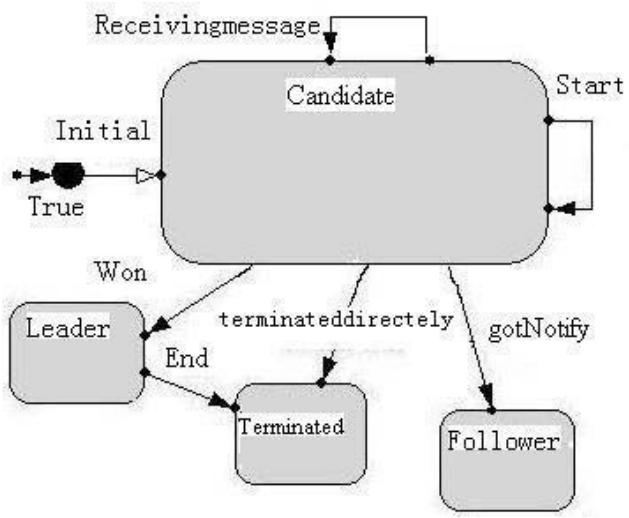

Figure 1. State Diagram for Leader Election in Ring–Single Stage

point later.

Leader election using multiple stages is handled in our integrated state machine as follows: choosing feature value "Multiple stage" from feature "Stage" is the trigger event for state "Candidate". This is because from the nature of the problem itself, after each stage of an election, some nodes should be eliminated. The nodes that are eliminated are called defeated, and the ones that are going to originate new messages or are going to advance to the next stage are called Candidate. As candidate and defeated nodes have different behaviors, we define six states for multiple stage algorithms: Initial state, Candidate state, Defeated state, Won state, Follower state and Terminated state.

At this point we have already integrated all the existent algorithms in the domain into two groups and built two corresponding state machines: one for "single stage" group, one for "multi-stage" group. Now we must integrate these two state machines into one. Recall that a candidate node is a node that can originate messages. A key observation is that the Electing state for single stage algorithms is similar to the Candidate state in multiple stage algorithms (though the latter handles more functionality). We also note that the Terminated state is shared across our two initial state machines.

To continue the integration, let us consider an example. Among "Single stage" algorithms, there are two ways of terminating the entire Leader Election algorithm. This is what feature notification is about: an algorithm can be terminated either by receiving a notify message (feature value "Need notify"); or the algorithm will make sure each node in the ring can calculate who is the leader and terminate itself automatically after that (feature value "Do not need notify"). And if feature value "Multiple stage" is chosen for feature "Stages", then only feature value "Need to notify" can be chosen for feature "Notification". The part of the state machine for "Multiple stage" is shown in Figure 2.

The state machine for "Single stage" is shown in Figure 1. In order to integrate the two state machines, we need to add one transition from state "Candidate" to state "Terminated" to describe the combination of feature value "Single stage" and "Do not need notify". One important detail: in the new integrated

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

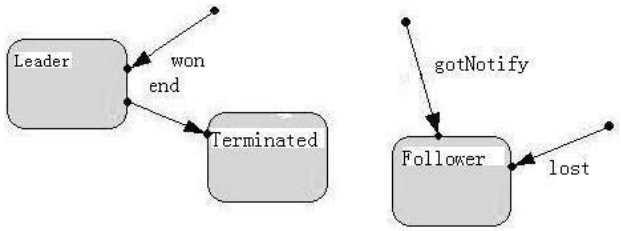

Figure 2. Partial state Diagram for Leader Election in Ring–Multiple Stage

state machine we must also change all trigger events on all transitions! For example, in the partial integrated state machine (Figure 3), in transition "terminateddirectly", the trigger event becomes:

return((Notification == NoNotifyMessage)&&(Stage == SingleStage)&&(MyID !== wd →ID))

Here "(NOTIFICATION == NoNotifyMessage)" and "(Stage == SingleStage)" illustrates how we carry feature and feature values into the integrated state machine: features are made to control which transitions execute. More details are available elsewhere[12, 11].

4 Using Rational Rose RT to Implement the Integrated Model for Leader Election in Ring domain

4.1 What is Rational Rose RT

Rational Rose RT is a graphical tool from IBM for modelling real-time systems. It supports executability of state machines, debugging and testing of distributed applications.

4.2 Running model for Leader Election in Ring Using Rose RT

As we described in the previous section, we carry features and their values directly into the integrated state machine. In this section, we present 1) how to use Rational Rose RT to implement the integrated state machine, and how to generate the new algorithms in the form of a state machine. 2) how to bridge feature combination rules to the running integrated state machine; and 3) the complete integrated domain model for Leader Election in a Ring.

According to the feature model we built, we can have many feature combinations. Ideally we want to have a generator capable of generating all the legal combinations, be they existing algorithms or new algorithms. Our integrated state machine is aiming at using all the features as inputs to configure our generator. Rational Rose RT provides a modelling environment to realize this.

We use a configuration of feature values as the initial input for the integrated state machine. This group of feature values might be legal combinations or not. Hence, we provide a "Verification" function to check the validity of the input feature values. If the feature values correspond to a legal combination, the integrated state machine is able to execute according to the selected configuration. See Figure 4. Interestingly enough, the simulation functionality of RRRT allows us to follow state by state, transition by transition the execution of an algorithm.

We can also run the integrated model on a distributed network using Rational Rose RT. After synchronizing all nodes to their initial state on different machines, we have successfully tested several of the algorithms of the family.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

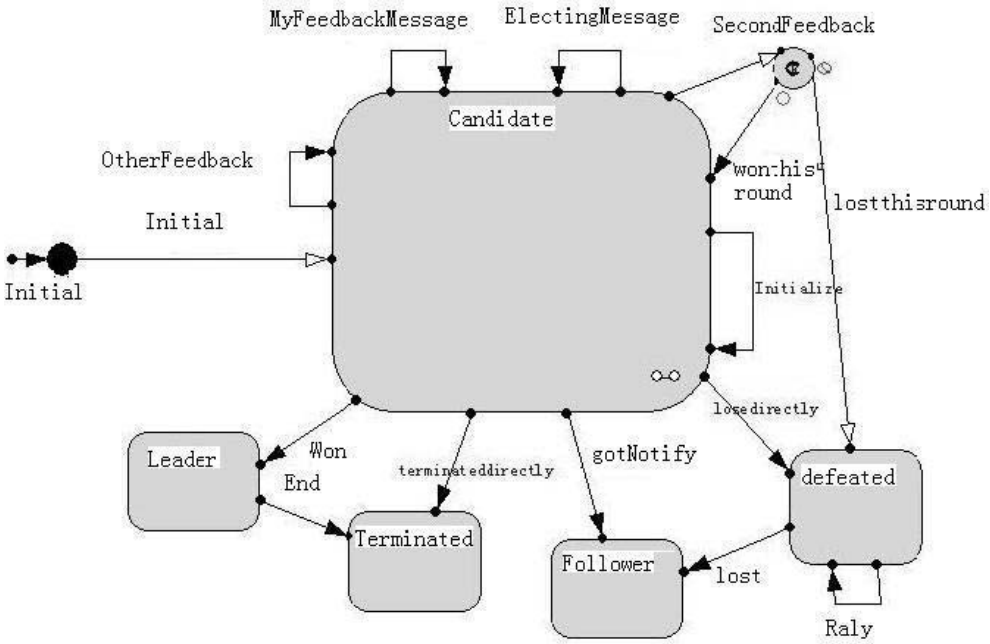

Figure 3. Partial Integrated State Diagram for Leader Election in a Ring

5 Conclusion

We propose adopting System Family Engineering as a productive modelling approach for families of distributed algorithms. Furthermore, we believe that our original idea of a configurable state machine in ROSE Real Time is important as it opens the door to a commercial simulation environment with precise semantics (which does handle distribution), executability, reusability and testability. Our future work includes:

1) to finish testing all existent combinations (which form the existent algorithms) of the feature model; to explore new algorithms by using new feature values combinations of the feature model. And compare the performance of each algorithm, hopefully we will come up with some more optimal algorithms.

2) to expand the domain: ideally we want to start adding the Tree topology into our existing domain.

3) to build a systematic test model for the target domain.

References

[1] D. Batory and S. OMalley. The design and implementation of hierarchical software systems with reusable components. *In ACM Transactions on Software Engineering and Methodology, vol. 1, no. 4, pp. 355-398.*, October 1992.

[2] Robert V. Binder. *Testing object-oriented systems – Models, Patterns, and Tools.* Reading, MA: Addison-Wesley,, 1999.

[3] J. C. Cleaveland. Building application generators.e. *In IEEE Software, no.4, vol. 9, July 1988, pp. 25-33.*

[4] Bruce Powel Douglass. Statecharts: a visual formalism for complex systems,. 1998.

[5] David. Harel. Real-time uml: Developing efficient objects for embedded systems . *Science of Computer Programming. Vol. 8, p. 231.*, 1987.

[6] Jacobson. Oose and book on reuse. 1999-2000.

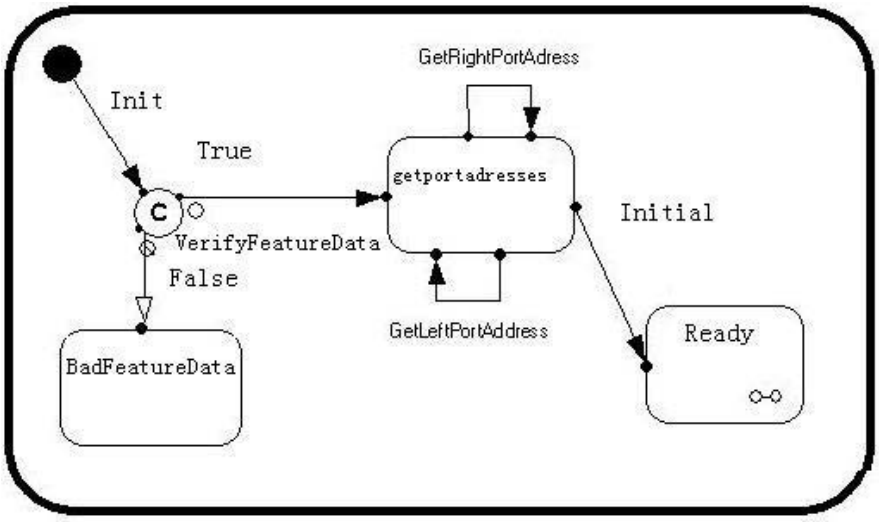

Figure 4. Top Level Integrated State Machine

[7] Ulrich W. Eisenecker K. Czarnrcki. *Generative Programming - Methods, Tools, and Applications*. Addison Wesley, 2002.

[8] Gerard Le Lann. Distributed systems - towards a formal approach. 1977.

[9] J. Neighbors. Software construction using components. *Ph. D. Thesis, (Technical Report TR-160), University of California, Irvine,*, 1980.

[10] N. Santoro. *Design and Analysis of Distributed Algorithms*. unpublished, Manuscript, 2002.

[11] W. Shi and J.-P. Corriveau. System family engineering on leader election in ring topology. *The IASTED International Conference on Parallel and Distributed Computing and Networks*, 2003.

[12] Wei Shi. System family engineering on leader election in ring. *School of Computer Science, Carleton University, Dissertation*, 2003.