Simulated Validation of Autonomous Traffic Control Systems*

V. S. Alagar, O. Ormandjieva
Department of Computer Science
Concordia University
Montreal, Quebec H3G 1M8, Canada
E-mail: {alagar,ormandj}@cs.concordia.ca

M. Zheng
Department of Computer Science
University of Wisconsin-LaCrosse
La Crosse, WI, 54601, USA
zheng.mao@uwlax.edu

Abstract

The paper describes a simulation approach to the validation of autonomous traffic control systems. Traffic control systems are safety-critical real-time reactive systems, whose correct behavior must be validated before deploying them. The simulator uses only the design specification of the system and does not require an implementation of it for validation. Using the simulator the developer can debug the design, validate specific scenarios, and reason about the possible future status knowing its execution history, before implementing the system.

1 Introduction

Traffic control systems are *safety-critical real-time reactive systems*, characterized by continuous interaction with their environment through stimulus-response behavior. Processes in the system are responsible for synchronization with their environment. The *reactive behavior* refers to two types of synchronization in the system, namely *stimulus synchronization* when a process reacts to every stimulus from the environment, and *response synchronization* when the system responds in a *timely* fashion so that the environment can make use of the response. Factors contributing to the complexity of such embedded systems are criticality, concurrency, and the time-dependent nature of artifacts they control. The complexity of system requirements permeates into several stages of development, and may lead to faulty designs and unpredictable failures. This type of complexity should be resolved at early stages. The main contribution of this paper is a solution to deal with such a complexity: we discuss a simulated validation of the design of the formal model of an autonomous traffic control system, before the system is implemented. We illustrate the results of validation on the formal model developed in [1].

This work is partially supported by grants and fellowships from the Natural Sciences and Engineering Research Council (NSERC) of Canada and Le Fonds pour la Formation de Chercheurs et l'Aide a la Recherche (Fonds FCAR) of Quebec.

1.1 Traffic Controller Problem

The traffic controller model developed in [1] assumes that at the proximity of the intersection, each road is divided into six lanes; there are three lanes for *incoming* traffic in each of the northbound, southbound, eastbound, and westbound directions. In every direction, vehicles in the right lane turn right, vehicles in the middle lane go straight, and vehicles in the left lane turn left. Vehicles approaching in a lane enter the crossing on a first-in-first-out basis. Vehicles cross the intersection in a finite amount of time; no vehicle stops in the intersection. Incoming vehicles in the four *right* lanes are allowed inside the crossing independent of any other lane; they are collision-free. However, traffic lights regulate all lanes. Incoming vehicles in the *middle* and *left* lanes need to wait until they are granted permission to enter the crossing. Vehicles in at most two middle or left lanes can be granted access simultaneously, only if the two lanes are collision-free.

When a lane requests for access rights, the

lane queues up for allocation. Access is granted to lanes in the order in which they request for permission, subject to collision-free property. That is, a vehicle v_2 that arrives at the intersection later than another vehicle v_1 may be given access to the intersection if the vehicle v_2 satisfies the collision-free property, whereas the vehicle v_1 has failed to satisfy the property. When a lane obtains access rights, it must surrender these rights within a certain time interval.

1.2 Properties for Simulated Validation

The intersection is a *shared resource* that is allocated to vehicles in such a way that the following properties are satisfied.

> *Liveness*: every vehicle at the intersection obtains the resource within a finite amount of time; there is neither deadlock nor starvation.
> *Safety*: vehicles do not collide while crossing the intersection.

The goal of simulated validation is to check that the above two properties are satisfied by the design. The outcome from the simulator for each run will be evaluated for the above properties. In principle, there are an infinite number of traffic patterns and hence an exhaustive enumeration and validation is ruled out. To contain this complexity, we construct a set of scenarios of interest and validate all scenarios in this set. Our approach also enables us to assess the *performance* of the system, allowing us to compare different designs and choose the best possible one for an implementation.

2 Traffic Controller Model

The design of the formal model of the traffic controller in [1] is based on the following design principles: the modeling elements are *vehicles*, *traffic lanes*, *traffic lights*, *controllers*, and an *arbiter*. There are twelve lanes, three for *south-north* traffic, three for *north-south* traffic, three for *east-west* traffic, and three for *west-east* traffic. In each of the twelve incoming lanes, a sensor detects vehicles approaching the intersection, a traffic light regulates the flow of vehicles, and a controller monitors the traffic. An arbiter allocates access rights to the controllers if collision-free property holds for vehicles in that lane.. The vehicles and the traffic lights are environmental entities; the controllers and the arbiter are system entities. For each lane, a subsystem models the communication configuration for the vehicles, the traffic light, and the controller. The overall system consists of twelve such subsystems, and one arbiter interacting with the middle and left lane controllers.

On approaching, each vehicle informs the respective controller through the sensor, and either receives an immediate go-ahead to traverse the crossing or waits for clearance. The waiting time for a vehicle depends on the status of the lights, the number of vehicles in front of it in the lane, and the number of controllers that are waiting for resource allocation. Vehicles in the four right lanes cross the intersection simultaneously and independent of vehicles in other lanes.

When there is no vehicle approaching the intersection along a lane, the traffic light at that lane remains *red*. The controller for the lane switches the light from red to *green* when it holds access rights and there is an approaching vehicle in the lane. The controller switches the light from green to *yellow* when the allocation time for the lane is about to expire, or when all incoming vehicles in the lane have crossed the intersection, whichever occurs earlier; the light turns *red* when the time expires. The timing constraints on the behavior of the traffic lights are the same for the three kinds of lanes.

The role of the controller is to detect approaching vehicles through the sensor, and to manage the status of the traffic light. The timing constraints for controllers monitoring the flow of traffic approaching the intersection in the middle and left lanes are different from those for the right lanes. Right lane controllers have no interaction with the arbiter. Controllers for middle and left lanes request for access rights to the intersection from the arbiter whenever there is an approaching vehicle in the lane. Upon receiving

the resource, the controller turns the light green, and gives the go-ahead to vehicles in the lane during the allocated time. The time-dependent interaction between the controllers and the lights is subject to the flow of traffic. For instance, the controller may switch the light from red to yellow before the end of the time-out period if there is no other vehicle approaching in the lane.

The arbiter optimizes resource utilization with concurrent allocation, ensures collision-free access, and prevents starvation. It processes requests for the resource from controllers for the middle and left lanes. It allocates the resource to at most two lanes concurrently, and on a first-in-first-out basis. The resource is to be released by the lane within a certain time interval. The arbiter maintains a queue of identifiers for requests from the lanes. It uses a table for determining the compatibility of lanes accessing the resource concurrently. The table enumerates the twelve scenarios for concurrent collision-free crossing of the intersection by vehicles in two different lanes. The main role of the arbiter is to prevent collision situations in the intersection and to allocate access rights with fairness and according to certain timing constraints.

2.1 Abstract Behavior

The time-dependent behavior of the system conforms to the following rules.

- *Traffic lights:* A light remains red as long as there is no vehicle approaching or waiting at the crossing. The controller switches the light from red to green, and subsequently from green to yellow. The light switches to red within a period of 3 to 4 time units after turning yellow.
- *Vehicles:* A vehicle sends a message to the respective controller on approaching the intersection. The controller subsequently sends a go-ahead message to the vehicle when it holds access rights to the crossing. The vehicle enters the crossing within 2 time units of receiving the go-ahead message. The vehicle leaves the crossing within 5 time units of receiving the go-ahead message.
- *Right lane controllers:* A right lane controller remains idle as long as there is no vehicle approaching or waiting at the crossing. The controller switches the light from red to green within 2 time units of receiving a message from the first vehicle approaching the intersection. The controller sends a go-ahead message to the first vehicle at the intersection within 1 to 2 time units of switching the light to green. The controller receives messages from other approaching vehicles while the light is green. The controller records the identifiers of the approaching vehicles in a queue as they are received. The controller authorizes vehicles to cross the intersection during the allocation period of 10 to 40 time units, and removes the identifiers of the processed vehicles from the queue. The controller switches the light from green to yellow within 1 time unit of being timed out. If there is no vehicle in the queue, the controller goes back to idle; otherwise, it reactivates and switches the light from red to green within 20 to 30 time units of switching the light to yellow. This time gap may be used to allow for pedestrian crossing in an extended version of the system.
- *Middle and left lane controllers:* A middle or left lane controller remains idle as long as there is no vehicle approaching or waiting at the crossing. The controller requests for access rights from the arbiter within 2 time units of receiving a message from the first vehicle approaching the intersection. Upon receiving permission from the arbiter, the controller switches the light from red to green within 2 time units. The controller sends a go-ahead message to the first vehicle at the intersection within 1 to 2 time units of switching the light to green. The controller receives messages from other approaching vehicles while the light is green. The controller records the identifiers of the approaching

vehicles in a queue as they are received. The controller authorizes vehicles to cross the intersection during the allocation period of 10 to 40 time units, and removes the identifiers of the processed vehicles from the queue. The controller switches the light from green to yellow within 1 time unit of being timed out. The controller returns the resource to the arbiter within 6 to 8 time units of turning the light from green to yellow. If there is no vehicle in the queue, the controller goes back to idle; otherwise, it reactivates and sends a new request for access rights to the arbiter within 20 to 30 time units of returning the resource. This time gap allows for other lanes to be processed without starvation.

- *Arbiter:* The arbiter remains idle as long as there is no request for access rights from middle and left lane controllers. When it receives a first request, it allocates the resource immediately. If the resource is returned before any other request is received, the arbiter goes back to idle. The arbiter records requests for access rights in the queue as they are received. If the arbiter receives another request before the first controller returns the resource, the arbiter either allocates the resource concurrently to the second controller if the two are compatible, or waits for the first one to return the resource if the two are not compatible.

  If one controller holds the resource, and there are requests in the queue, then when the controller returns the resource, the arbiter allocates the resource to the first one in the queue, and allocates the resource to the second one concurrently if the two are compatible. Otherwise, it waits for the first one to return the resource. If two controllers hold the resource concurrently, and there is no request in the queue, then when one of the controllers returns the resource, the arbiter goes into the state where only one controller is holding the resource and there is no pending request. If two controllers hold the resource concurrently, and there are requests in the queue, then when one of the controllers returns the resource, the arbiter allocates the resource to the first one in the queue if it is compatible with the controller still holding the resource. Otherwise, it waits for this controller also to return the resource.

2.2 Design Models

A reactive entity is modeled as a class with local clocks for measuring time. A finite state machine augmented with ports, attributes, logical assertions on the attributes, and timing constraints. Objects communicate using a synchronous message passing mechanism. An external event, either input or output, can only occur at an instance of a specific port type; an internal event occurs at the null port. Thus a port type symbolises the events that can occur through its instances; events label the transitions between states. The type of an attribute can be either a port type, or an abstract data type. Logical assertions on the attributes specify a port condition, an enabling condition, and a post condition on each transition. Timing constraints are associated with a transition to describe the time-constrained response to a stimulus. An abstract model of a system includes instances of classes, each with port type instances, and links connecting ports for compatible communication. A port link is an abstraction of communication mechanism between the objects associated with the ports connected by the link. The port links effectively determine the set of all valid messages that can be exchanged among the objects in a subsystem. Fig. 1 shows the class diagram for the structure of the system.

3 Validation Environment

The validation environment is TROMLAB [2] framework that we have built as a platform for object-oriented design and development of real-time reactive systems. Figure 3 is an overall architectural view of TROMLAB.

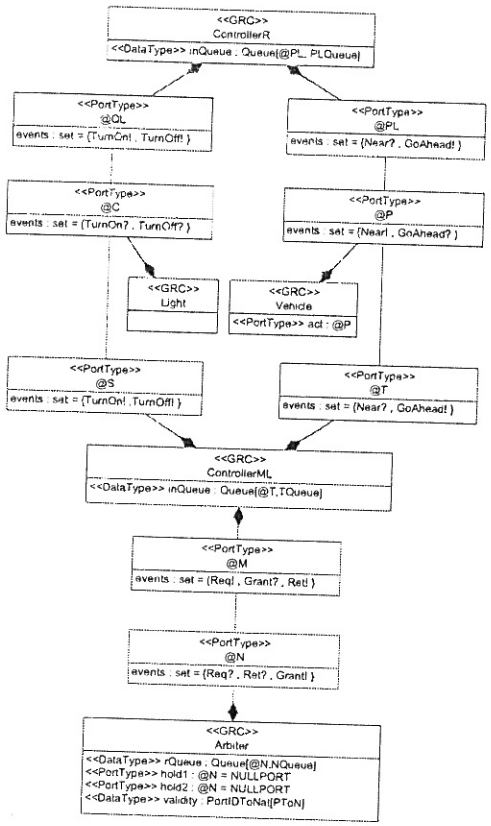

Figure 1. Class Diagram for Road Traffic Control System.

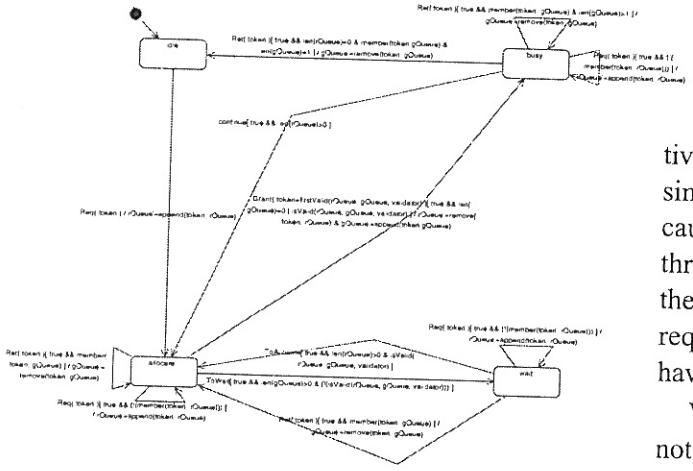

Figure 2. Statechart Diagram for New Arbiter.

The *Simulator* tool animates the formal model of a system. The tools *UML/ROSE*, *Interpreter*, *Graphical User Interface* (GUI) assist a developer in constructing a graphical representation of the model, and get a formal specification of it. The graphical models are constructed using the interface to Rational Rose provided by our tool *UML/ROSE*. The models are translated by the same tool to their textual descriptions, which are then interpreted by the tool *Interpreter*. An internal representation constructed by the *Interpreter* tool provides the interface for the *simulator* tool. This sequence of activities are enabled through GUI tool. This process is completely automated.

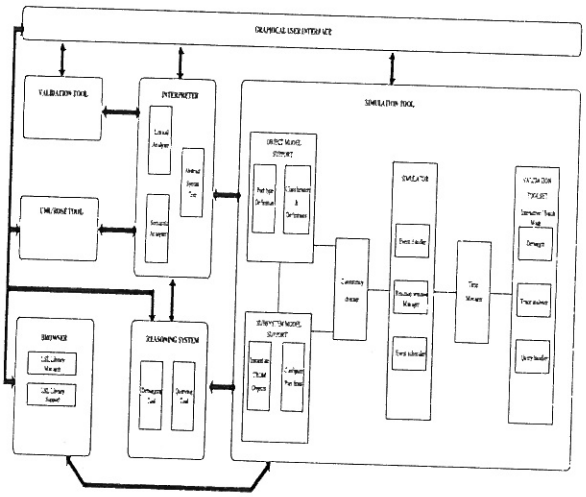

Figure 3. Simulation Environment

To contain complexity, the user may interactively conduct design validation by simulating a single computational step and then analyzing the causes of unexpected results. Going backwards through the history maintained by the simulator, the user can debug the design making use of the requirements and knowledge about expected behavior.

We simulated the model developed in [1] and noticed deadlock for some traffic patterns. After modifying the design of one controller and arbiter in their design, we removed the deadlock. We have considerably simplified the Arbiter design, by reducing the number of states, and tightening some of the transition specifications. The new Arbiter statechart diagram is shown in Figure 2. The new Arbiter prevents starvation in the system and satisfies the safety property. The

modified design has been simulated extensively and seems to satisfy both the safety and liveness properties. Section 4 discussed the results of the simulation run.

4 Simulation Results

We came up with four distinct traffic patterns that are either likely to cause a deadlock or a collision. A summary of our analysis of the design based on simulation results is given below:

Experiment 1. Several vehicles in one lane, say lane x, are waiting to get access to the intersection: this happens when a vehicle in lane y is in the crossing and the lanes x and y are not collision-free lanes.

Experiment 2. Every middle lane has a vehicle waiting to cross the intersection: this scenario happens because of the independence of traffic flow in the lanes. This scenario introduces conflict, which is resolved more efficiently by the collision avoidance scheme of the new Arbiter.

Experiment 3. Several vehicles may wait in one lane and every middle lane having at least one vehicle. The simulation for this scenario is to validate the situation which combines the two situations independently validated in the previous two experiments. The simulator runs to completion ensuring collision-free passage, and avoids starvation in the system.

Experiment 4. This is the same as Experiment 3, except that we increased the number of vehicles, decreased the frequencies of their arrival at the intersection, and forced some of them to arrive at the intersection at the same instant. The goal of this simulation is to test the ability of the system for handling high-volume traffic concentrated over short periods of times, and validate the system for successful handling of concurrency. The simulator uses a random order to sequentialize concurrency in the system. The simulation ran to completion, validating the safety and liveness properties. Every vehicle successfully crosses the intersection with no collision, thus the simulator validates the safety and liveness properties.

5 Conclusion

We have proposed simulation of the design as a means of validating safety and liveness properties in a traffic control system. The basis of the formal design has been explained in [1]. Simulation revealed the flaws in the original design, which were corrected. The new design successfully passed four types of experiments, which covers all possible traffic patterns. The number of scenarios for each pattern is potentially infinite. By successfully simulating one scenario from each pattern we claim that in principle all scenarios have been validated. We are developing a formal verification of the system based on the above notion.

References

[1] V.S. Alagar, D. Muthiayen. *A Rigorous Approach to Modeling Autonomous Traffic Control Systems.* ISADS2003, Pisa, Italy, pp.193-202.

[2] V.S. Alagar, A. Achuthan, D. Muthiayen. TROMLAB: *A Software Development Environment for Real-Time Reactive Systems,* (first version 1996, revised 2001), Technical Report, Department of Computer Science, Concordia University, Montreal, Canada.

[3] S.H. Liu. *Simulated Validation of Real-Time Reactive Systems with Parameterized Events.* Master of Computer Science Thesis, Concordia University, Montreal, Canada, October 2003.

[4] O. Ormandjieva. *Deriving New Measurements for Real-Time Reactive Systems.* Ph.D. thesis, Concordia University, Montreal, Canada, April 2002.