Int j simul model 8 (2009) 1, 27-37
ISSN 1726-4529 Original scientific paper

MODEL-BASED TESTING USING REAL-TIME ADAPTIVE SIMULATOR

Soklic, M. E.
Computer Science Department, Florida Gulf Coast University, Fort Myers, FL 33965, U.S.A.
E-Mail: msoklic@fgcu.edu

Abstract
Using modeling and simulation users can apply realistic interactions with their synthetic environments representing real world. In this study the synthetic environment is a hardware railroad model representing a miniature replica of a full-size railroad system under study. The hardware model is tested by means of real-time adaptive software simulator consisting of two subsystems: the virtual railroad which mimics behavior of the hardware railroad model, and the controller which is used to synchronize and control the hardware and the virtual railroads. The controller has a built-in feature to forecast possible railroad hazard conditions and to avoid them by sending corrective commands to both railroads. Testing the model can be observed on the simulator screen and on the hardware railroad model. The simulator also features test-event history log allowing playing back the tests.
(Received in April 2008, accepted in September 2008. This paper was with the author 3 months for 1 revision.)

Key Words: Event-Driven Software Simulation, Hardware Railroad Model, Railroad Traffic Hazards, Adaptive Traffic Control

1. INTRODUCTION

Full behavior and safety properties of large-scale transportation systems, such as railroads, are hard to understand because they contain a large number of possible trajectories where trains can move in different directions and different speeds. The task of understanding railroad traffic behavior is a haphazard and error-prone. Effective safety assurance relies on hazard identification which in turn helps to avoid railroad traffic hazards and accidents. Such a problem clearly exceeds manual capabilities and must be addressed through automation.

The literature on transportation systems shows various approaches dealing with railroad issues. A review of various railroad simulation tools, challenges, and benefits is discussed in [1]. Methodology for rapid prototyping of simulation models for large scale transportation systems is discussed in [2]. Methods for fault detection of railway vehicles and tracks using multi-resolution analysis are described in [3]. A deadlock-free algorithm for dispatching trains in complex rail networks has been devised in [4]. Experiments on railroad crossing problem were addressed in [5]. These techniques are, however, limited in the sense that they mainly center attention to specific railroad traffic conditions.

The focus of this research is to assure safety train movements in a generic railroad system using hazard identification and collision prevention techniques. For that purpose a hardware railroad model and a software simulator were designed and implemented. The model is used to represent a miniature replica of a full-size railroad system under study, whereas the simulator is used to test the model. Since the model and the simulator have components that perform identical functions, components from the model are also referred to as physical, e.g., physical railroad; whereas their matching counterparts in the simulator are also referred to as virtual, e.g., virtual railroad.

Fig. 1 depicts fundamental connection between the simulator and the railroad model. The railroad model, placed in the laboratory environment, features various railroad track configurations populated with trains that can move in forward and in reverse directions at various speeds along the tracks. The railroad model has no notion of software. The simulator consists of two software subsystems: the virtual railroad (VR) which mimics railroad model traffic behavior, and the controller (CO). The controller can run VR stand alone or concurrently with the railroad model. In the latter case, responses from the railroad model are used by the CO to synchronize virtual traffic on the VR with the traffic on the railroad model. As a result, based on the VR traffic conditions, the simulator is able to detect, forecast, and avoid possible traffic hazards.

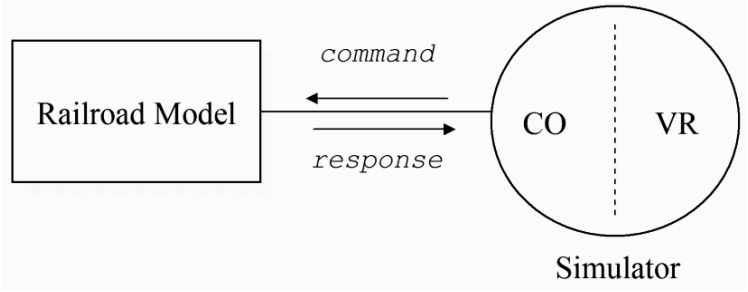

Figure 1: Simulator and the railroad model – basic connection.

Fig. 2 illustrates research equipment assembled around the following subsystems: the railroad model, whose controllable components are track switches, train sensors, and trains; the single-board computer (SBC) hosting real-time software simulator; and the personal computer workstation (PCW), used as a user console of the simulator. The CPW is also used as a platform during the development of the software simulator which is then downloaded to the SBC.

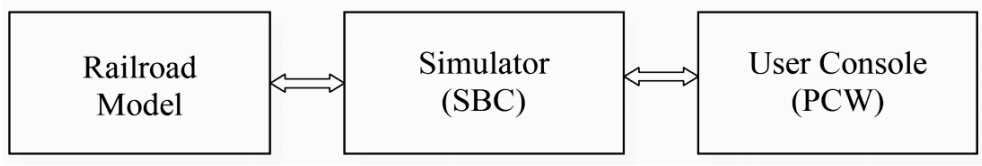

Figure 2: Equipment - arrangement of the subsystems.

The decision to use the railroad model and real-time adaptive simulator, in order to assess traffic safety, is based on the following:

- Since safety is a dynamic system attribute, the model-simulator approach offers more credible estimation of system safety than an approach where a simulator is used stand alone, i.e., without a model.
- Through its adaptive behaviour, the simulator is able to amass additional information from the model, which might be either hard to describe, would make simulator too complex, or is not known at the time of testing; for example, unexpected malfunctioning of trains or their unusual behavior.
- Safety alternatives are tested in a synthetic environment (model-simulator) rather than in actual operation; railroad test scenarios can include cases of extreme variations, as well as typical or average cases.

The following sections present architecture and functioning of the railroad model and the simulator in more detail.

2. THE RAILROAD MODEL

Fig. 3 shows track layout of the railroad model under study. There are 11 track-switches shown as octagons with numbers representing track-switch numbers, and 29 sensor-locations, shown as numbers along the tracks used to detect trains presence. The symbols N, S, E, and W represent the compass orientations of the track layout; they are used to describe the direction of a train movement. The trains are miniature replicas of trains used on actual railroads. All components of the railroad model are manufactured by Märklin Company [6].

The railroad model incorporates three modules which enable interaction with the simulator. Logical organization of the three modules is shown in Fig. 4. The interface module is a gateway for sending commands from the SBC to either the control module or to the track-detection module. Control signals emanating from the simulator are marked with bold arrow-lines.

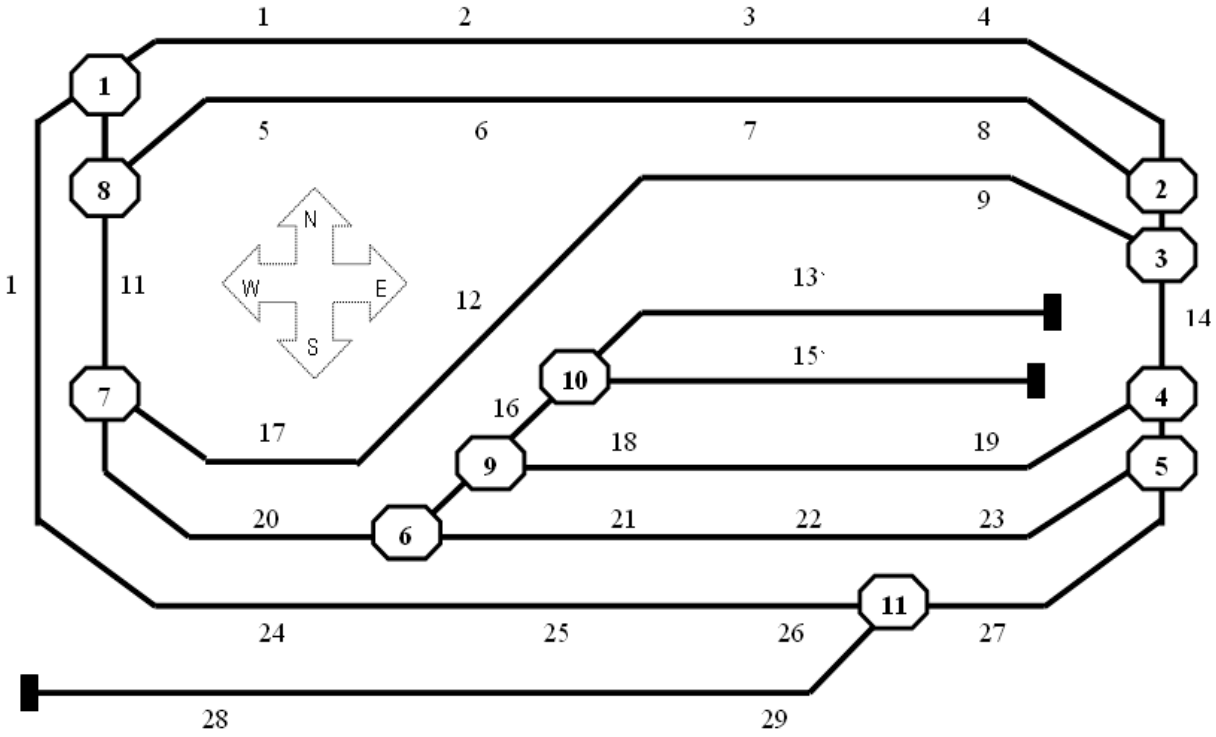

Figure 3: Railroad model - track layout.

The control module broadcasts commands for trains and track-switches over the middle rail of the track, indicated with two dotted lines. These commands cause the following actions on the railroad model: start/stop train, set train speed, accelerate/decelerate train, change train's direction of movement, and set/reset track-switch.

The track-detection module, independently of other modules, gathers data from all sensor locations along the track. The railroad track has three rails, two outer rails and the middle rail. Sensors can be placed either close to the two outer rails, or in the middle rail. As a result, a single sensor location can have one up to three sensors. However, the number of sensors in a given sensor location depends on the test under study. Sensors can only detect moving trains in their close vicinity and cannot identify them. Thus, when a force emanating from a train triggers a sensor, this sensor built-in switch makes a momentary closure on its terminal, sets its corresponding track-detection module latch input to logic high, indicating a train triggered this sensor. Sensors' data stored in the track-detection module are read by the simulator.

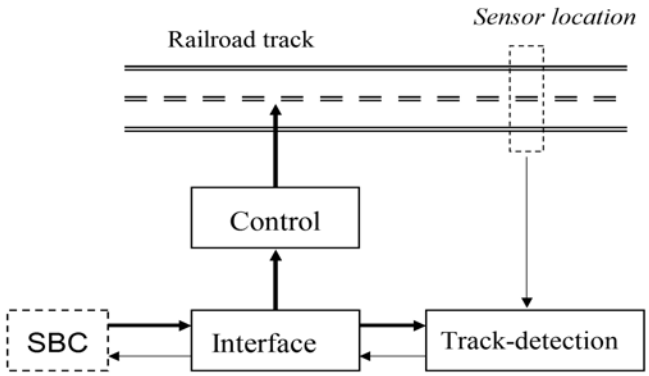

Figure 4: Railroad model - logical arrangement of hardware modules.

The SBC is the Motorola MBX-860 single-board computer [7]. This board is well-proven in industry application, and is particularly suited for development of real-time embedded applications. The simulator hosted on SBC controls the railroad model via its application programming interface (API) representing a collection of software functions designed to control the railroad model.

3. THE SIMULATOR

Software for real-time railroad simulation and testing consists of two major subsystems: the virtual railroad, and the controller. The virtual railroad and the controller together are referred to as the simulator.

Fig. 5 shows the context diagram of the simulator. The user console is used to enter data to the simulator shown as *key_input*. The simulator sends commands to the railroad model seen as *RR_commands*, and reads information about physical train locations as *sensor_data*. Relevant information about both railroads is sent by the simulator to the user console, shown as *sys_out* data.

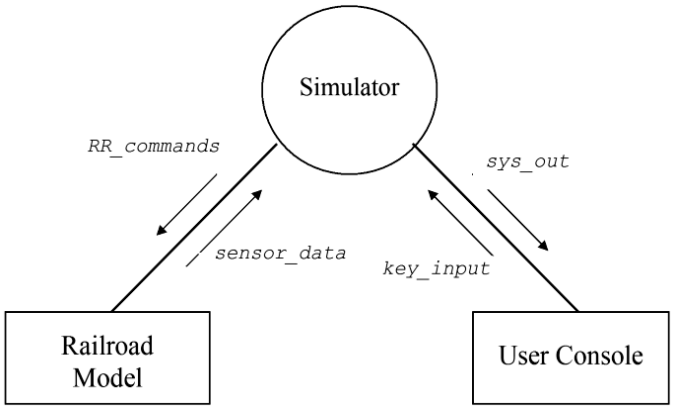

Figure 5: Simulator - context diagram.

Fig. 6 describes the simulator in terms of two high-level internal modules, the controller (CO) and the virtual railroad (VR). The CO interacts with both the VR and the railroad model in the same manner.

The virtual railroad features: (i) virtual track layout, (ii) virtual track-switches, (iii) virtual trains, (iv) ability to dynamically setup virtual track-switches, (v) ability to position and run virtual trains on the virtual tracks, where each virtual train has its own speed and direction of

movement, and (vi) ability to detect train locations on the virtual railroad. In short, the virtual railroad replicates salient features of the (physical) railroad model.

The CO supports two modes of operation: virtual-only mode, and virtual-physical mode. In the virtual-only mode, the CO is used to run the VR stand alone, and to perform simulator self-tests. In the virtual-physical mode, the CO runs VR and railroad model concurrently.

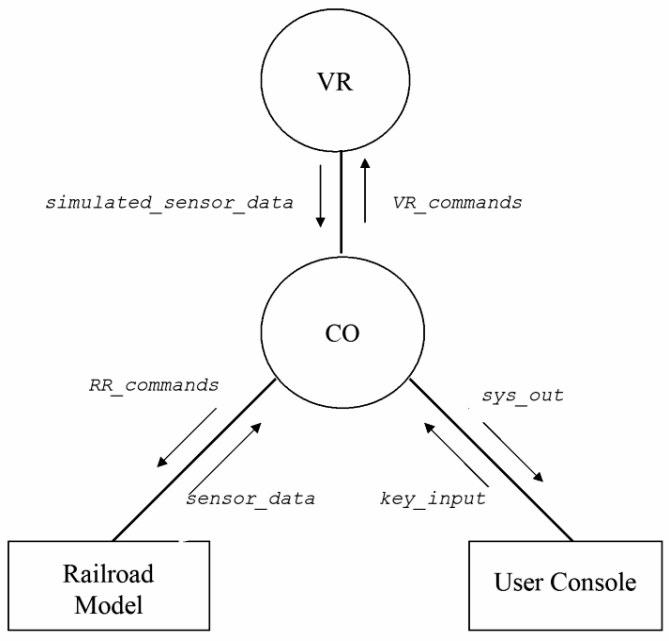

Figure 6: Simulator - level 1 flow diagram.

In the virtual-physical mode of operation, the VR is operating in synchronization with the railroad model. Traffic conditions on the railroad model and on the VR are synchronized when the virtual train locations match their corresponding physical train locations. For that purpose, the CO compares *sensor_data* from the railroad model with the *simulated_sensor_data* from the VR. If there is a mismatch between the *sensor_data* and the *simulated_sensor_data* the CO adjusts locations of virtual trains (that do not match the corresponding *sensor_data*) to their matching physical train locations; and, consequently, accelerates/decelerates virtual train speeds to get adjusted to the speeds of corresponding physical trains. The CO checks for possible railroad hazards on the VR, and, if necessary, sends corrective commands to both railroads to avoid train accidents. Activities of the CO are displayed on the user console, including status information about each train on the VR, such as: train number, train speed, the sensor number recently triggered, the next sensor number to be triggered, and the time left the next sensor number will be triggered; possible hazard conditions that might emerge during that point in time are displayed as well.

To maintain synchronization, the CO updates the sensor data on the VR to be consistent with the sensor data from the railroad model. The updates take place every time the user selected sensor locations on the railroad model are triggered. For instance, if the selected sensor location 14 has been triggered, say, by a train on the railroad model whose number is 35, then, the corresponding virtual train 35 on virtual track will also be set to that virtual location; consequently, if this virtual train was ahead/behind relative to its physical train location, then the virtual train's speed is decelerated/accelerated accordingly.

Fig. 7 shows level 2 flow diagram representing an exploded view of the CO consisting of four sub modules: the User Interface module (UI), the Collision Avoidance Logic module (CALM), the Command Handler module (CH), and the Sensor Monitor module (SM).

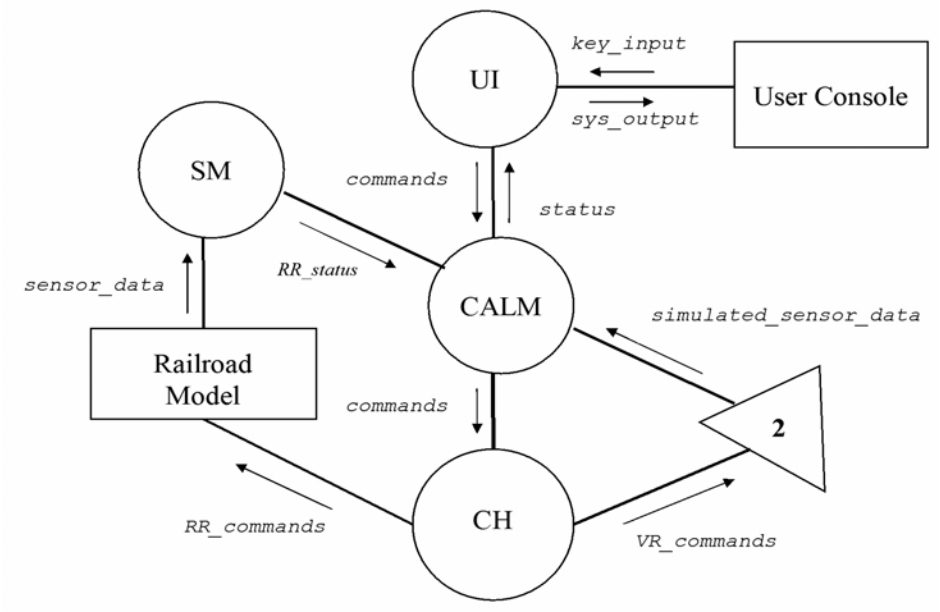

Figure 7: Controller - level 2 flow diagram.

The purpose of the controller sub modules are as follows:

- The UI module function is to enable input/output between the user console and the CO. This module accepts user text commands as *key_input* via the user keyboard and displays simulator information as *sys_output* on the user display.
- The SM module gathers *sensor_data* from the railroad model, via sensor polling, and sends them as *RR_status* to the CALM. The CH module accepts commands from the CALM and sends them to the VR as *VR_commands*, and to the railroad model as *RR_commands* if the CO is in virtual-physical mode of operation.
- The CALM accepts *commands* from the UI module which are then sent to the VR and to the railroad model via CH module. To synchronize both railroads, the CALM directs the received *RR_status* data from the railroad model to the VR via CH module. In turn, the CALM receives *simulated_sensor_data* from the VR which are used to check for railroad safety hazards. If hazard conditions are detected, the module issues corrective *commands* to both railroads via the CH module.

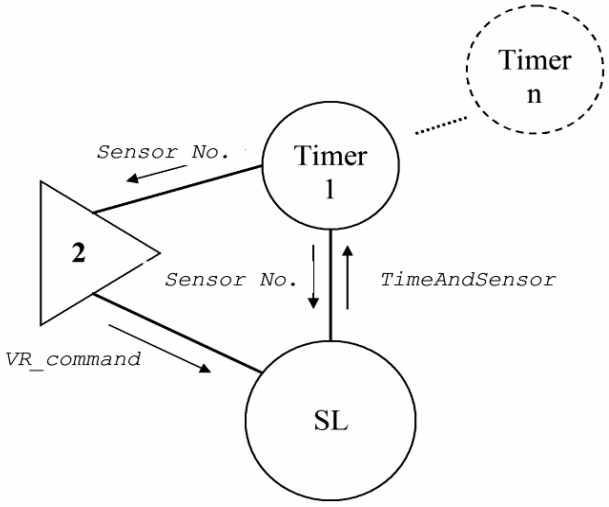

Figure 8: Virtual railroad - level 2 data flow diagram.

Fig. 8 shows level 2 data flow diagram of the VR consisting of the Simulation Logic module (SL), and *n* Timer objects (Timer). The virtual railroad functions are as follows:

- A Timer object contains information about a single virtual train. There are *n* virtual trains on the virtual railroad, designated as Timer 1 through Timer *n*. A virtual train represents an instantiation of the class Timer which creates a sensor object that triggers when the time interval associated with each virtual train expires, indicating that the virtual train has triggered the designated virtual sensor along the virtual track. As a consequence, each virtual train broadcasts its sensor data *sensor_No* to the SL module and to the CALM, which collects sensor numbers of all virtual trains that triggered (represented as *simulated-sensor-data* on Fig.7).
- The SL module represents virtual tracks. These tracks are represented as a collection of all possible virtual railroad track-sections, where a track-section represents a distance between two neighboring sensor locations. From each virtual train, the module receives a sensor number it just triggered. In return, the module responds to the virtual train by sending it *TimeAndSensor* information containing a number of the next virtual sensor location to be reached and the time it has to be reached, i.e. triggered. The time interval for a virtual train to move along the designated track-section is initially estimated based on the (physical) railroad model operating conditions, and later, adjusted by increasing or decreasing speed of the virtual train in question.

Fig. 9.a and Fig. 9.b show modules and actual process names used in implementation of the simulator. Processes involved in interaction are arranged horizontally, whereas time is represented vertically. Thin rectangle on process's lifeline represents the time when the process is the controlling process in the system. A process takes over control at the top of its rectangle and relinquishes control at the bottom of its rectangle.

Fig. 9.a shows interactions of processes controlling the railroad model. The SensorMonitor process periodically polls the railroad model (RailRoad : hardware) for updated sensor data. Once sensor data are found to be available, the SensorMonitor process notifies the CALM. The CALM will not be notified of sensor data if no new data is available. That is the purpose of the self-checking CheckSensorData() function defined within the SensorMonitor process.

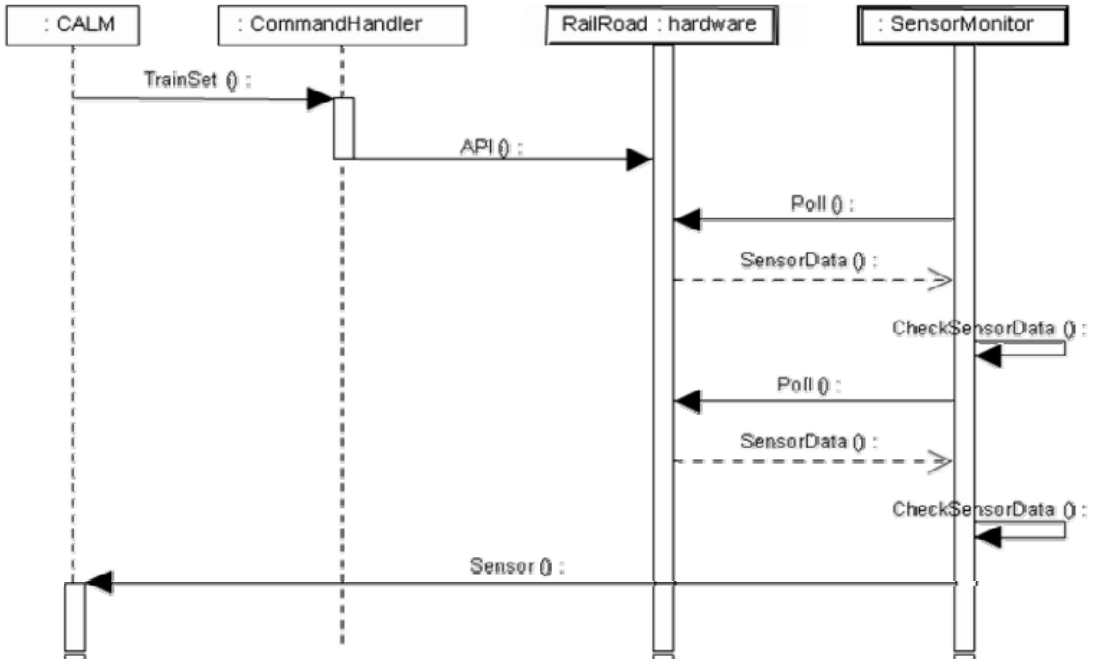

Figure 9.a: Sequence of operations - railroad model.

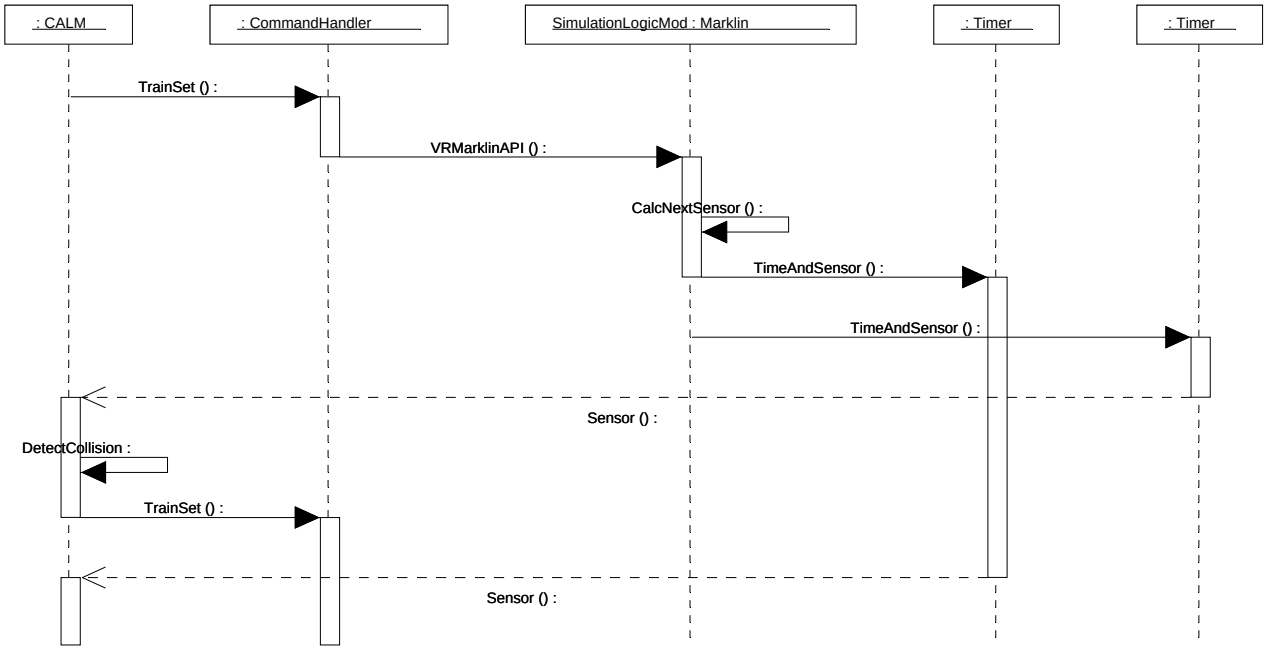

Figure 9.b: Sequence of operations - virtual railroad.

Fig. 9.b shows interaction of modules in the virtual railroad. The SimulationLogic module representing the virtual railroad is not polled; instead, it uses timers to simulate triggered sensor data. The feedback from timers is sent to the SimulationLogic process and to the CALM, seen as horizontal dashed lines.

3.1 Highlights of Simulator Implementation

Real-time software simulator was designed with the following two software qualities in mind: concurrent performance (recognized during system runtime), and modifiability (recognized during system development and maintenance). Efficient concurrent performance was accomplished by proper allocation of functionalities to the simulator modules/processes, with minimal amount of necessary communication among them. Modifiability was achieved with loose coupling between modules so that the user can easily add, change, or remove them.

Fig.10 shows a diagram of object structures that have been created to simulate the railroad model and how they relate to each other. For instance, the Track object is composed of multiple Sections, denoting track-sections. Track-sections are further composed of two Sensor objects.

The software simulator was developed using functional and object-oriented methods and implemented in C++ programming language. Internal data are represented as objects, whereas logical operations are managed procedurally. As a result, this allows greater flexibility in encapsulating data while taking advantage of multi-processing and shared memory capabilities of the VxWorks real-time operating environment [8].

The use of exceptions handling is utilized in all software components. These exceptions are handled by higher-level components which either correct the condition or alert the user. Errors encountered by the simulator fall into two general types: (a) errors that endanger the railroad, and (b) errors that don't put the railroad in immediate danger. Errors type (a) represent unavoidable collision, such errors result in the railroad model being shutdown immediately in order to avoid damaging the model. Errors type (b) don't put the railroad in immediate danger, such errors are seamlessly handled by the CALM.

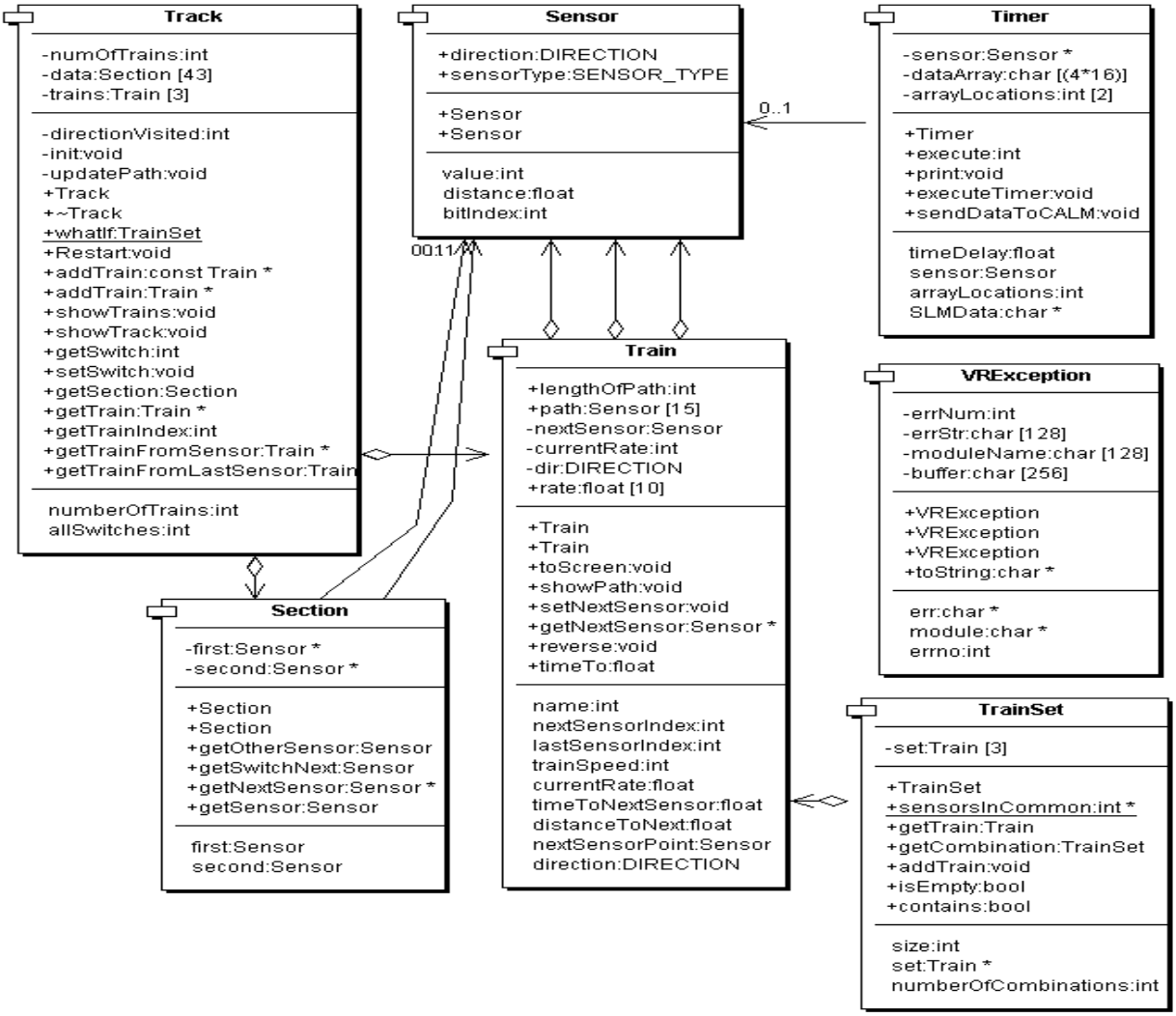

Figure 10: Virtual railroad - UML diagram.

The simulator is built as a true real-time software system, real-time constructs such as semaphores, message queues, and signals have been used in its development. The semaphore construct is used throughout the system to control access to both the railroad model and the virtual railroad, ensuring that race conditions cannot occur. Components that need to communicate utilize the message queue construct. The message queue gives flexibility in how software modules communicate and how these communications are handled by the recipient. The process of designing and implementing real-time simulator follows procedures and steps outlined in the literature on classical and object oriented software engineering [9-11].

4. TESTING THE MODEL

A comprehensive look at the model safety was accomplished by making test cases which utilized different number of trains with varying strain speeds and different railroad-track configurations. To perform tests on the railroad model from the whole systems perspective the CALM was designed which takes charge of the following types safety hazards:

A. train collisions,
B. train entering turn at excessive rate of speed,
C. train running into an end-of-track bumper, and
D. sensor reading timeout.

Train collisons are further distinguished into:

A.1 two trains occupying the same track-segment at the same time,
A.2 a faster train overtaking a slower train,
A.3 two trains heading in opposite directions on the same tarck-section, and
A.4 track-switch settings enabling any of the above.

The hazards of type A, B, and C represents high risk of failure for a railroad system caused by configuration of railroad tracks, speed of trains, and number of trains involved in the traffic. For that reason CALM constantly checks the current states of the virtual railroad and predicts the future states in order to detect possible future hazard conditions. If hazards are detected and railroad failures can be avoided, it issues corrective commands to both railroads (virtual railroad and railroad model). Corrective commands instruct the designated trains to accelerate/decelerate or stop. If a hazard cannot be avoided both railroads are shutdown immediately.

Before corrective commands are issued, CALM uses a look-ahead feature which assures that corrective commands won't create another hazard. This is accomplished by 'knowing' where the trains on the track will go in the next *n* seconds. Using these predicted paths, the CALM can find possible hazards well before they are a danger to the railroad. Once possible hazards have been detected, possible solutions are gathered and 'what-if' scenarios are generated for each of the solutions. The 'what-if' scenarios are predicted train paths that are checked for possible traffic hazards. If a generated 'what-if' scenario alleviates the current predicted hazard and it does not cause any new hazard, then corrective commands are issued.

The likelihood of a fault, due to operator’s commands, is low if intelligent input handling is added to the UI module and to the CALM which ensures that not only valid commands are entered but also that only commands that do not affect the overall safety of the railroad model are followed. The faults can arise from any of the following conditions:

- user error in inputting railroad paramters,
- failure or overload in CALM, and
- loss of physical connectivity to the railroad model model.

The safety measures put in place to alleviate the hazards are alarms and internal checking. Alarms are brought to the attention of the user in conditions, such as: sensor read time outs, message queues time outs, process spawn errors, and any exceptions that get raised.

The model was tested using traffic scenarios where each scenario differs from other scenarios in one or more of the following: number of trains used, train speeds, and railroad track configurations. The scenarios were grouped in two categories, one involving test cases representing typical railroad traffic conditions, and the other one with extreme railroad traffic conditions. Test results on all test scenarios showed that the simulator successfully detected all traffic hazard conditions on the railroad model and avoided all train accidents.

5. CONCLUSION

This paper presents an approach of assessing traffic safety of a railroad system subject to failure caused by number of trains involved in traffic, train speeds, and different configuration of railroad tracks.

The assessment of safety, a primary concern in this study, is accomplished by using a model-simulator approach, where the simulator is used to run and test the model in a loop. The model is a miniature hardware replica of a full-size railroad system under study. The simulator consists of two software subsystems a controller and a virtual railroad, where the virtual railroad represents a replica of the railroad model. This architectural arrangements allows the simulator to run stand-alone (using virtual railroad only), or concurrently with the model.

In this research the model-simulator approach was used for two reasons:

- Virtualization of the mechanical railroad components, e.g., virtual trains and track-switches, have no notion of weight and time, as a consequence, virtual trains cannot derail on sharp turns, and their acceleration and decelerations are instantaneous.
- Operations of real-world trains tend to exhibit a high degree of natural variability.

The effect of weightlessness, timelessness, and unexpected train behavior is eliminated by means of feedback data from the model. As a result, the virtual railroad adapts its status to the status of the physical railroad (model) such that both railroads exhibit the same traffic conditions.

Based on the experience with the developed model-simulator architecture, the following advantages can be identified:

- The model-simulator replaces an actual size system under study.
- The simulator software architecture exhibits qualities of modifiability which allows reuse and modification of the simulator software modules.
- Hazard traffic safety conditions can be evaluated under various traffic scenarios.
- During the testing, traffic conditions can be monitored on the railroad model and on the simulator screen, and off-line using simulator log file.

Test results showed that, based on traffic scenarios used, the simulator did detect all traffic hazards on the railroad model and avoided all train accidents.

Future work on this research is targeted towards higher modularization of the simulator so that it will, among others, assist in communication and education during contract negotiations.

6. ACKNOWLEDGEMENT

This research was partially supported by the author's grant obtained from the Wind River Company, a manufacturer of VxWorks integrated software development environment for embedded and real-time systems.

REFERENCES

[1] Vaillancourt, E.; Vucko, S. J.; Drummie, A. M.; Bekavac, J. (2000). Simulation within the Railroad Environment, *Proceedings of the 2000 Winter Simulation Conference*, 1191-1200

[2] Kavicka, A.; Klima, V.; Adamko, N. (2007). Simulations of Transportation Logistic Systems Utilising Agent-Based Architecture, *International Journal of Simulation Modelling,* Vol. 6, No. 1, 13-24

[3] Hayashi, Y.; Kojima, T.; Tsunashima, H.; Marumo, Y. (2006). Real time fault detection of railway vehicles and tracks, *IET International Conference on Railway Condition Monitoring,* 20-25

[4] Quan L.; Maged D.; Robert C. L. (2004). Modeling train movements through complex rail networks, *ACM Transactions on Modeling and Computer Simulation (TOMACS),* 48-75

[5] Saeed J.; Tarbiat M.; Mehdi M.; Tarbiat M. (2007). RVERL: Run-time Verification of Real-time and Reactive Programs using Event-based Real-Time Logic Approach, *5th ACIS International Conference on Software Engineering Research, Management Applications,* 550-557

[6] Märklin (2000). *Märklin Catalog - Complete Program 1999/2000*, Osfildern

[7] Motorola (2001). *MBX Series Embedded Controller – Programmer's Reference Guide,* Motorola

[8] Wind River Systems (1999). *VxWorks Reference Manual - 5.5,* Wind River

[9] Bass. L.; Clements, P.; Kazman, R. (2003). *Software Architecture in Practice*, Addison Wesley

[10] Schach, S. (2002). *Object-Oriented and Classical Software Engineering*, McGraw-Hill

[11] Burns, A.; Wellings, A. (2001). *Real-Time Systems and Programming Languages*, Addison-Wesley