INTEGRATION OF TWO SIMULATION SOFTWARE PACKAGES
IN THE CONTEXT OF A LRT SYSTEM STUDY

Eric Vaillancourt
Senior Specialist – Simulations –
Railway Operations
Hatch Mott Macdonald
5 Place Ville-Marie, Suite #200
Montreal, QC H3B 2G2 Canada
Tel : (438) 266-0815
eric.vaillancourt@hatchmott.com

Aleksandra Trpkovic
Rail Systems Engineer
Hatch Mott Macdonald
2800 Speakman Drive
Mississauga, ON L5K 2R7 Canada
Tel: (905) 403-3781
Aleksandra.trpkovic@hatchmott.com

Michael Samario
Specialist – Rail Operations & Simulations
Hatch Mott Macdonald
5 Place Ville-Marie, Suite #200
Montreal, QC H3B 2G2 Canada
Tel : (438) 266-0808
michael.samario@hatchmott.com

Number of Words: 3942 including figures (250 words per table or figure)

ABSTRACT
In the fields of road and transit transportation planning, simulation tools have evolved to achieve
great levels of sophistication. Simulation has become the predominant tool in the expert’s toolset
and an essential extension of his or her ability to apply domain knowledge. Meanwhile, issues
requiring analysis increasingly involve multiple areas of expertise. In such situations, where
issues to be studied lie at the intersection of two or more disciplines, the integration of multiple
approaches and their dedicated simulation packages becomes relevant to the quality and cost of
the study. Street-running LRT systems are an example of this situation. They involve two
interacting modes of transportation regulated by control systems which may be separate in some
areas and overlapping or interacting in others. Road/Rail intersections are nodal points of such
systems. Simple interactions to be considered include road traffic obeying grade-crossing signals
and street-running trains obeying traffic signals. More complex interactions include partial and
absolute traffic signal pre-emption by approaching trains as well as rail signals reflecting the
status of road traffic at intersections.

This paper describes a process to analyze such a system using two concurrent modelling
approaches, road and rail traffic, in a dynamically integrated manner. A description of the runtime
integration methodology is presented, where separate simulators are integrated through a
specifically designed “hub” program using techniques borrowed from parallel discrete event
simulation of asynchronous systems strategies. Benefits and challenges associated with this dual
approach are discussed and compared to alternate methodologies. Results for one sample
application are discussed.

INTRODUCTION
In the domains of railroad and transit transportation planning, as in most other areas involving
detailed planning to minimize capital expenditures, simulation tools have evolved over the past
decades to achieve great levels of sophistication. In fact, simulation in a broad variety of
engineering areas has become a predominant tool in the subject matter expert’s toolset and can
now be considered an essential extension of his or her ability to apply domain knowledge to
engineering planning problems. Meanwhile, as accepted approaches apply a more and more
integrated systems view to engineering projects, issues requiring analysis increasingly involve more than one single area of expertise. Examples of such multi-domain systems interfacing with railroad operations abound: road traffic for street-running LRTs, Mine site loading and unloading, passenger train loading and unloading at station platforms, industrial processes, to name a few.

In 1946, S.V. Smith laid the foundation for the use of simulation in the railroad business when he filed patent application No. 674,540 entitled the *Train Performance Calculator and Recorder* for a *device for continuously calculating and recording train performance data*. This was not long after Alan Turing’s then little known accomplishments in the area of computing and consisted of an electro-mechanical device made of resistors, potentiometers and electric motors. Clearly, this device was not designed to interact with any other calculator. From this basic building block, calculator, or simulator development evolved both macroscopically, encompassing larger and larger components of a rail network, and microscopically, depicting more and more detailed representations of the train’s dynamic behavior.

Current railroad simulation software packages cover all scales of railroad related phenomena, from the micro- effects of one additional orifice to hydraulic cushion coupler to the macro-parametric models of rail line capacity. The integration of various simulators remains often vendor-specific and does not easily allow the seamless integration required to model complex railroad-centric systems that include non-railroad components.

This paper outlines the steps taken to create a hub acting as the interface between multiple simulation programs, mediating the synchronization and information exchange between the programs. It describes the first implementation of this hub, mediating a railroad network simulator and a highway traffic light simulator in the context of a street-running LRT simulation project.

**Parallel Discrete Event Simulation**

One approach to the integration of two or more distinct simulation programs running concurrently can be called Parallel Discrete Event Simulation. The approach has been developed since the 1980’s in response to the interest (that) arises from the fact that large simulations in engineering, computer science, economics, and military applications, to mention a few, consume enormous amounts of time on sequential machines. In the case presented here, the objective of distributing the computing is not motivated by a possible performance increase as much as by the requirement to involve two or more distinct expert teams, and their separately developed simulation tools, within one planning project.

**System Architecture**

Our approach to resolving the various issues at the interface of two or more simulation software programs requires a third program to carry out a number of mediation tasks. Among those, the communications function, the information storage function and the process synchronization function. Additionally, the system may be called upon to perform ancillary functions such as system reporting and performance metric logging.

In figure 1, the main components of the proposed integrated system are depicted. For the proposed arrangement to be successful, the ability for the simulator programs to send and/or receive messages, as well as the option given to third party programs to somehow affect the pace at which the simulator is being run, are crucial. Without messaging functions, the simulator operates as if inside a black box. Without pacing features, there is no way to synchronize the simulator with its counterparts.

In figure 1, visible from all components of the system is the *Common Data* area, containing all the information shared by either module, in this case information on the road and traffic signal status from the road network simulator to control train movements as well as detector statuses from the rail network simulator that act as triggers for all manners of preemption to occur in the road network simulator.
Next farthest from the center of the system are the individual simulator programs, each being stand-alone but having some ability to interact with the system at run-time.

The series of light green circles designates **Communicator modules** specifically designed to interact with the each simulator. Each is capable of using the same communications protocols as its assigned simulator program, recognizing and translating messages coming from the simulators and being able to send messages that are recognizable by it. The **Communicator** modules need access to the common data area for storage and retrieval. They also require access to a central clock and the means to enforce some form of synchronization with the simulators.

The central circle contains the **Coordinator Hub**, the module which is intimately linked to the Common data area but whose other main contribution is to manage the synchronization between the simulator programs via the communicator modules. For this purpose, the hub logic core may be expanded to perform complex tasks, correcting synchronization inaccuracies or similar issues.

---

**COMMUNICATIONS**

The concept of successfully running two or more specialized simulations concurrently to resolve an issue that neither one addresses satisfactorily on its own hinges on the system’s ability to allow communication to flow between its components. Communication throughout the inner circles of the system, the coordination hub and communication modules, is relatively easy to implement as it amounts to having components of a program *designed* to communicate together do so. Managing the communication between this central system and third party software is more challenging.
Coordinator Hub - to - Communicator Module
The Communicator modules are designed as Dynamic Link Libraries (dll) to be Hub program extensions. A simple dll common interface is used to allow direct communication using common memory areas between the hub and the communicator. With this arrangement, communication on the hub side is relatively straightforward since direct access to the Common Data area is built into the dll interface. The result is seamless data access and very fast communication.

Communicator Module - to - Simulator
Communication methods and protocols are specific to each third party simulator package, and there are a variety of techniques that may be used: a simulator program may provide a DLL interface that allows direct access to a number of key program variables and allows communication to happen directly. Another typical arrangement is for the simulator program to provide an Application Program Interface allowing a set of commands that can be accepted by the software over the internet as well as a set of reporting options for information retrieval from the software.

The structure successfully implemented here is an XML based message format, Simple Object Access Protocol (SOAP) using a TCP/IP connection. This arrangement has several advantages, and a few drawbacks. Being XML based, the message constructs are simple and hierarchical, which makes the tailoring of the communications routines more streamlined. The TCP/IP connection allows for distributed computing, which can be a significant gain in terms of processing performance, but which may also alleviate software license issues: each simulator software package may remain located on each expert's computer and communicate with the hub over the internet. Overall messaging performance, for large data exchange requirements, may become an issue with this arrangement, however.

In the vendor-provided dll interface option, the communication occurs through a dll designed to interact with the simulator program directly. It is far more efficient from a data throughput perspective, but requires an additional communicator module to reside on the simulator computer if distributed computing is to be considered.

Development Environment
Microsoft .NET environment was selected for development of the Hub and Communicator modules. This environment provides development options where languages may be relatively freely mixed for flexible development regimes, i.e. a professional program developer may be called upon to develop process intensive modules in high-performance native languages such as C++ while less critical development issues may be left to code-knowledgeable rail practitioners for Rapid Application Development (RAD) solutions in Basic or C#, for example.

System Specifics
Information Sharing
Two levels of information are required in the common data area: data specific to each simulated world, in this case the rail network and the road network that is relevant to the other, as well as data specifically required to allow the meshing of the two worlds.

Simulation-specific data
In the case of the street-running LRT, the data required to pass relevant information onto the other system can be relatively simple.

Road-To-Rail: The train will follow signals that can be considered entirely controlled by the road-traffic simulator; in this case a representation of signal phases, or aspects, needs to be held in the common data area. These states may, in some cases, enter rail interlocking implementations.

Rail-To-Road: To enforce preemption of any ilk, information on approaching train movement needs to be provided in the field to the traffic light controller. This information is captured through
virtual *instrumentation* within the rail network simulator, much as it is in the real world: AEI readers, in-track inductive loops, track-circuits. Either way, train occupancy detection data needs to be stowed and made available to the road-traffic signal simulator.

**Meshing data**

This section makes reference to the data required to run the integrated simulation system itself. It basically consists of tables of relations between same object identifiers in the various simulators that have different representations of them. It is the means by which the Hub programs knows which loop in the traffic simulator program needs to be notified that the same element was triggered in the rail network simulator. Also needed is information data such as simulation time offsets, desired time increments, or paces, for each simulator, etc...

**Synchronization Issues**

Several cases need to be addressed in combining distinct simulation packages. For instance, each program may not have a similar control on the passage of simulated time, or simulation *pace*. If neither one has any control over its pace, and if the simulators do not operate at identical “simulated time to real time” ratios, synchronization may prove impossible. If only one of the two simulators offers sufficient control over its own pace, synchronization may be achieved as long as the fastest pace of the controlled pace simulator is faster than that of the uncontrolled one.

Two means of pace control are prevalent among many simulators: The first method is the fixed time-step function, where the simulator is given a command to advance by a number of seconds no matter the pace. In this case, for each “time step”, the simulator runs for a given time increment and then stops, allowing other processes to terminate before the following time steps occur. This method is the easiest to manage provided that time increments are compatible between simulators. Another is direct control over the simulated-to-real time ratio of the simulation being run. In this case, as long as the selected pace is achievable given performance constraints, synchronization may be achieved through adjusting the paces to be identical in practice.

**The Rail Network Simulator**

OpenTrack, from OpenTrack Railway Technology Ltd., is a powerful, flexible and easy-to-use railroad network simulation software package that was developed at the Swiss Federal Institute of Technology for Transportation Planning and Systems located in Zurich. The program was designed to assist professionals in the evaluation of rail capacity constraints and analyzing current and future operations of simple or complex systems. OpenTrack uses supplied infrastructure, train consist, schedules, timetables, incidents and work-program information to model selected scenarios. The software is well suited for light rail (LRT) systems, but can also be used to simulate many different types of dedicated or mixed passenger and freight networks. OpenTrack was selected, among other reasons, for its proven track record on LRT simulation projects as well as its ability to communicate with 3rd party applications over the Internet.

**The Road Traffic Simulator**

SmallSim-Roads is a simple street traffic light controller simulator software program developed in-house by Hatch Mott MacDonald. The program was designed to simulate, track and manage the phase changes of traffic lights at intersections in a network of roads and highways. It allows for inbound messages indicating train-occupancy driven triggers and can send signal phase state messages. It is flexible enough to accommodate multiple variations of traffic signal pre-emption. It can also receive time step messages which facilitate synchronization.

**Simulated Case**

The system was tried on a test case presenting examples of each typical road/rail control system interaction. Figure 2 presents the OpenTrack screen showing the simulated rail network. In this depiction, the grey horizontal lines represent tracks, where the green overlay signifies signal
block reservation and the red overlay means physical occupancy by the train identified in the grey text box. The track section has 13 intersections, 11 of which are instrumented, i.e. fitted with virtual detectors allowing for train location messages to be sent to the Rail Network Communicator Module. Most recent virtual detector information is represented in the grey text boxes on either side of the track.

Figure 2  Simulated Case

Figure 3 highlights the passage of train “NB 104” through Intersection 8, and some of the information being sent by the instruments. This snapshot, taken at simulated time $T_s = 7:14:32$, shows that occupancy of detector “Int 8 NB Setup” by train NB 104 started at $T_s = 7:14:14$, and occupancy of the intersection itself started 13 seconds later. This information is sent at runtime to the communication hub, which in turn updates corresponding statuses in the system Common Data area. The updated information will be used by the road traffic light simulator as is relevant given the type of traffic light interaction that is implemented in this location.

**Full preemption:** if by definition this means absolute priority to rail traffic, this information, which is presumably received long enough in advance of the train’s arrival, triggers changes to traffic signal phase durations such as the train is guaranteed to approach the intersection on clear, or green, signals. In other words, the information about the train passage received by the system when the train passed Int 8 NB Setup detector, and the processing by the traffic light controller, ensured that the light was green for the train, and presumably red for all cross-traffic. Only RAIL -> ROAD communication is relevant in this case.
Figure 3 Opentrack instruments (detectors)

No preemption: when the traffic light controller receives no train movement input, trains have no impact on traffic lights. In this case, if detectors are present, their output is not used and traffic lights follow their cycle irrespective of the approach of trains. Only ROAD -> RAIL communication is relevant.

Partial preemption: this case is the most complex case to represent, and the only one where information needs to flow in both directions. The train approaching the intersection triggers the detector that sends the information to the traffic light controller. The traffic light controller processes the information and determines whether priority is given to the approaching train or not. Depending on the decision, the signal aspect displayed to the approaching train is either a green or a red. Both RAIL -> ROAD and ROAD -> RAIL communications are relevant in this case.

Future Enhancements and Other Applications
The system presented here is a simple implementation of the dynamic linking of two simulators in addressing operating issues at intersections for a street-running LRTs operation. Significant extensions are being considered to further the development of the concept. Among these:

- Replacing the Smallsim-Roads simulator with a commercially available road traffic simulator will make the approach more useful for addressing complex road traffic issues not handled in SmallSim-Roads, and will allow the simulation of the impact of the rail traffic on vehicular traffic as well.
- Identifying other commercially available simulators that could be linked to the rail network simulator in the area of industrial processes, pedestrian movements, in-train dynamic force simulation, passenger platform allocation, etc.

Conclusion
Rail networks do not exist in a vacuum. The ability to integrate elements of adjacent systems that affect or are affected by rail operations into the analysis of rail network issues is an obvious benefit to the quality of any planning study. The additional benefit of doing so by incorporating domain specific knowledge and experts into the analysis process is evident. The system described in this paper allows experts from various fields to contribute to the simulation-based analysis while using their own sophisticated simulation tools.

This system is a first step in a novel approach to rail operation engineering and planning projects where the power of simulation-based analysis is applied to problems affected by more than simply rail related issues.
REFERENCES


TABLES AND FIGURES

Figure 1  System Architecture
Figure 2  Simulated Case
Figure 3  Opentrack instruments (detectors)
INTEGRATION OF TWO SIMULATION SOFTWARE PACKAGES IN THE CONTEXT OF A LRT SYSTEM STUDY

Eric Vaillancourt
Aleksandra Trpkovic
Michael Samario
Agenda
- Back in Alan Turing’s day
- Expert Silos
- The Street-Running LRT problem
- Building bridges with “Parallel discrete event simulation”
- The SmallSim Hub solution / Test case
- Conclusion

Back in Turing’s day
- Turing’s machine

Back in Turing’s day
- S. Smith’s 1946 Train Performance Calculator (TPC)

Expert Silos
- Specific field experts
- With specific needs
- Develop specific tools
The Street-Running LRT problem

Building bridges with “Parallel discrete event simulation”
- Family of techniques for communication and synchronization of simulator processes
- Developed for distributing processing purpose
- Deals with issues of varying time steps and time scales

The SmallSim Hub Solution

The SmallSim Hub Solution
The SmallSim Hub Solution

Hub tasks
• Provide comm. infrastructure
• Coordinate simulator runs
• Store common data
• Pass data between simulators

Test Case
• OpenTrack rail network simulator
• SmallSim-Roads street traffic light controller simulator

Test Case
• OpenTrack Instruments (Detectors) collect train location information
• Here, “Int 8” detectors show current train NB104 as having approached and entered Intersection 8 area

Test Case
• New traffic light status is determined by SmallSim-Roads and sent to Main Controller
• Main controller sends on the data to the rail network simulator as input

Test Case
• OpenTrack receives information from Main Controller and detects that Intersection 7 lights are now green (clear)
• NB104 will be allowed to continue on due to traffic light preemption
In conclusion

• Rail networks do not exist in a vacuum
• Integration of adjacent systems is key to improving quality of rail system analyses
• Infrastructure that facilitates cooperation between domain experts is desirable as complexity of engineering problems increases

Thank you!