Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

TOWARDS SIMULATOR INTEROPERABILITY AND MODEL INTERREPLACEABILITY IN NETWORK SIMULATION AND EMULATION THROUGH AMINES-HLA

Erek Göktürk

Department of Informatics
University of Oslo
Blindern, N-0371 Oslo, NORWAY

ABSTRACT

Simulation interoperability and model interreplaceability still remain as goals to be attained in network simulation. We have previously developed a high level architecture for network simulators, named AMINES-HLA. In this paper, we discuss AMINES-HLA with respect to IEEE HLA standard, DEVS, and the architectures of OMNeT++ and NS-2. We describe two scenarios which demonstrate how simulation interoperability and model interreplacement can be realized through AMINES-HLA. In the first scenario, we show how a virtualized program and protocol stack can be used to replace a specific model in a simulator built on AMINES-HLA. In the context of the second scenario, we discuss how two simulators can be interoperated.

1 INTRODUCTION

Simulations have been one of the main tools for understanding, tuning, and therefore designing networks and network protocols. In addition to the natural interest from network experimenters, network simulation has also drawn more and more attention from the simulation community. The main driving force for the simulation community's continuing interest is that the networks to be simulated grow bigger and bigger, and new technologies, such as wireless and sensor networks, require more and more complex models.

Today there is a proliferation of network simulators. In addition to the mainstream network simulators such as NS-2, OMNeT++, OPNET Modeler, and GloMoSim, many experiment or project specific simulators, emulators, and testbeds are reported in the literature. Although there have been considerable progress with works such as the Dynamic Simulation Backplane (Xu et al. 2001), to this date, model level interreplaceability and simulator interoperability for model reuse still remain as goals to be achieved in network simulation. In this paper, we shortly describe a high level architecture for network simulators, called AMINES-HLA, and describe two scenarios that demonstrate how this architecture can be used for model replacement and simulator interoperability. These scenarios provide the grounds for our current efforts in re-factoring some widely used open source network simulators.

AMINES-HLA can be regarded as a component-based architecture, where our focus is mainly on the architecture of the run-time representation (RTR) of the model of the target network (MoTN), and the simulator. In AMINES-HLA based simulators, the management of the RTR of the model, such as construction and transformation, is separated from the RTR of the MoTN. We believe that time management, which is traditionally considered a function of the simulation executive, should be a part of the RTRs of the simulation models. Furthermore, a simulation executive which provides simulation model related functionality such as time management, and which is not itself sufficiently componentized, is one of the major obstacles towards simulation model interreplaceability.

The organization of the paper is as follows: in Section 2 we introduce AMINES-HLA shortly. In Sections 3, 4, and 5 we discuss AMINES-HLA in comparison to the IEEE-HLA, the DEVS formalism, and the architectures of network simulators OMNeT++, and NS-2. Then, in Sections 6 and 7, we describe the scenarios for model replacement and simulator interoperability, and we conclude in Section 8.

2 AMINES-HLA

Although high level architecture has become a term that refers to the High Level Architecture Modeling and Simulation Standard (IEEE 2000), we will use high level architecture (HLA) as a term to refer to an architectural specification using which the architecture of a particular system can be formulated. We will also use the term run-time infrastructure (RTI) to refer to the necessary code or binaries that support the software environment described by a high level architecture. In the rest of the paper, we will refer to the High Level Architecture Modeling and Simulation Standard as IEEE-HLA, and its RTI as IEEE-HLA-RTI.

AMINES-HLA is a higher level architecture designed to be used as basis for network simulator architectures. The design goals targeted by the AMINES-HLA are the following:

- Creating a run-time environment where the components are as loosely coupled as possible, in order to support interoperability of simulators, and replacement or interreplacement of simulation model parts.
- Providing an architecture that ensures separation between the code responsible for simulation management and initialization, and RTR of the simulation model.
- Imposing and exposing a meta-model for simulation models that can be simulated by network simulators.
- Providing an architecture where components can be distributed transparently, with respect to the perspective of the experimenter. Current processor trends towards multi-core architectures imply that concurrent simulation will become more common even in simulators designed to run on low-cost PCs and workstations.

A more detailed description of the design of an earlier version of AMINES-HLA, which does not include the concurrency extensions, can be found in Göktürk (2006a).

As discussed in Göktürk (2006b), there are two stakeholders in network simulation and emulation: the simulation system developers, and experimenters. Exposing the meta-model related assumptions made by the simulation system or library developers for a particular simulation system or library, should make the simulation system easier to use for the experimenters. Furthermore, a meta-model that is in good agreement with the modeling approach that appears in the expert knowledge of the experimenters, which they use in their day-to-day research tasks, would make it easier for the experimenters to adapt to and learn to use the network simulator. Our approach in using this similarity of knowledge structure differs from other approaches based on the same similarity idea, such as GTNetS (Riley 2003), in the fact that our focus is on a relatively more abstract level, solely on the architectural aspects, not on providing entities that have real-network counterparts such as nodes, protocols, etc. We see providing appropriate models that have sufficiently real-world-like counterparts as an issue to be handled by the simulation system or simulation library designers.

The meta-model assumed in the design of AMINES-HLA is inspired by the actor model (Agha 1986, Agha 1996). In this meta-model, a simulation model is formed by instances of unit models. The unit models represent the basic sub-models, whose further division appears infeasible with respect to the granularity targeted for a particular simulation model. These unit models are represented as customized behaviors of actor like entities, which result in a model composed of homogeneous simple elements with customized behavior.

The AMINES-HLA does not describe a complete simulation system. It only provides an architectural common ground for network simulators and emulators.

There are two types of units in AMINES-HLA: unit models (UMs) and constructor units (CUs). A simulator that is built on AMINES-HLA is composed of instances of these two kinds of units (UMIs and CUIs), and links between instances of unit models.

A unit model in AMINES-HLA represents a basic indivisible sub-model. Each unit model instance (UMI) has a UMI identifier that is assigned when the UMI is created at run-time. This UMI identifier is guaranteed to be unique among the UMI identifiers created during a particular run. The UMIs are connected to other UMIs with uni-directional data paths, which we refer to as links. A link between two UMIs is uniquely defined by the four-tuple originating out-link identifier, originating UMI identifier, target in-link identifier, and target UMI identifier. The out-link and in-link identifiers are meaningful only locally to a particular UMI.

In a UMI, the behavior is left blank to be provided by the simulator developers. We refer to this provided behavior as the customized behavior of the UMI. AMINES-HLA imposes the restriction that such customized behaviors must work on local information only. Any non-local interaction is to be established via employing services provided by the AMINES-HLA run-time infrastructure (AMINES-HLA-RTI) to the UMI customized behavior. These services are provided to each UMI customized behavior by a dedicated UMI base entity, as shown in Figure 1. The only service provided to the UMI customized behavior in the current design of the AMINES-HLA is the `SendMessage` service, which takes the locally meaningful identifier of an out-link and a message, and sends the message to the instance that has previously been connected to the out-link. The UMI customized behaviors receive two callbacks. The `ReceiveMessage` callback happens when a message is received, and passes the received message and the identifier of the in-link the message was received from, to the UMI customized behavior. The `UmiBaseCreated` callback is used to inform the UMI customized behavior when a UMI base is created and assigned to the UMI customized behavior.

As the reader will notice, the UMIs lack the capability to create new UMIs or links. The reason is that the UMIs are intended to carry customized behaviors that are executable models out of which the model to be simulated will be composed. The functionality about managing the RTR of the model to be simulated, which we will refer as the Simulation Management Functionality (SMF), is to be provided by the

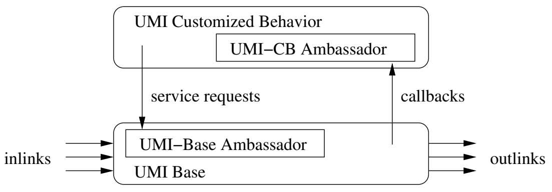

Figure 1: Logical Structure of a UMI

instances of the construction units (CUIs) in an AMINES-HLA based simulator. Examples of the functionality that falls in SMF are construction and destruction of the RTR, or transforming the topology of links between the UMIs.

The logical structure of the CUIs resembles the structure of the UMIs depicted in Figure 1. Each CUI has a unique identifier just like the UMIs have. The CUI customized behaviors are under the same restriction that they can freely work on local information, but any interaction with the other instances in the simulator should be through AMINES-HLA-RTI. However, the services provided to the CUIs are more extensive. The CUIs can:

- create and delete UMIs or CUIs,
- create or delete links between UMIs, or from UMIs to CUIs,
- send and receive messages to and from UMIs and CUIs,
- create and delete execution units, related to concurrent extensions of the AMINES-HLA.

The communication services provided to the CUIs differ from communication services provided to the UMIs. When a message is to be sent, the only allowed addressing for the message is the use of the locally meaningful out-link identifier. The UMI has no way of knowing the other party to the link. The same is true for incoming messages, where the UMI is informed only about the locally meaningful in-link identifier of the link the message was received from. However, since links are created by CUIs using instance identifiers, the CUIs use directly the UMI and CUI identifiers for addressing their messages instead of having to constantly create and delete links. The CUIs get informed about the UMI and CUI identifiers when they create new UMIs or CUIs, or when they exchange messages with other CUIs. Otherwise, the AMINES-HLA-RTI does not provide any centralized repository of UMIs or CUIs created in the simulator.

An example of how a simulator based on AMINES-HLA is typically organized and how it works, is presented in Section 6.

The concurrency support in AMINES-HLA is provided by partitioning the set of instances into execution units (EUs). Within an EU, the execution is sequential. Therefore an EU can be regarded as the equivalent of a logical process (Fujimoto 2000). Like UMIs and CUIs, an execution unit has a unique identifier. The UMIs and CUIs are assigned to EUs by providing the creation service with the identifier of the EU in which the CUI or UMI is to be created. Message sending service workis in a non-blocking manner, both in and accross EUs. Blocking message sending semantics can be implemented in a simulator by using additional UMIs.

The careful reader might notice that AMINES-HLA, unlike other simulation systems, does not include a simulation executive that provides scheduling services. This is intentional in AMINES-HLA's design. We regard the schedulers as part of the model to be simulated. They are therefore expected to be represented in the executable model by one or more UMIs and their links in AMINES-HLA. By granting schedulers and event scheduling mechanisms a normal model element status, we gain additional flexibility when making different simulators or models interoperate. We are currently preparing a more detailed discussion of scheduler designs on AMINES-HLA.

3 AMINES-HLA VS. IEEE-HLA

In this section, we will shortly compare the AMINES-HLA and the HLA IEEE Standard for Modeling and Simulation (IEEE 2000), in order to outline the differences in goals and design.

The AMINES-HLA provides a component-based architecture to be used in building the executable models of network systems to be simulated, as well as the simulators which build and execute the RTR of these models. It is not necessarily to be used along with distributed simulations. It provides the architectural structure for organizing the sub-model instances in a sequential or distributed network simulation.

The IEEE-HLA, on the other hand, provides a runtime infrastructure and architectural specification targeted towards interoperating simulators. IEEE-HLA regards the federates as logical processes, provides simulation executive functionality to the federates, and services at the federate-federate boundary. It does not provide or impose any particular architecture on the internal structure of the individual simulators.

Therefore, the AMINES-HLA and IEEE-HLA are independent tools that can supplement each other. An RTI implementation that is based on the AMINES-HLA for IEEE-HLA seems possible. Our recent and ongoing experience from designing a multi-agent systems based RTI architecture for the IEEE-HLA suggests that the approach is not impractical, nor infeasible.

Providing an IEEE-HLA-RTI interface to the simulators built on AMINES-HLA, is as good an idea as it would be for any simulator that can be used as part of a federated simulation experiment. How this interface can be designed, is out of the focus for AMINES-HLA. This is because the services that need to be translated between the simulator and the IEEE-HLA, are in fact not a part of AMINES-HLA.

Although implementing an IEEE-HLA-RTI based on AMINES-HLA seems possible, the other way around, namely implementing the AMINES-HLA-RTI using IEEE-HLA-RTI, does not look so promising. The most apparent way of using IEEE-HLA-RTI in implementing AMINES-HLA-RTI is mapping execution units to federates. But doing so only results in publishing on the IEEE-HLA-RTI the data passing the execution unit boundaries. Further work is needed in determining whether various services, especially the time management services, can be encapsulated as UMIs in order to be made available in a simulator based on AMINES-HLA.

4 AMINES-HLA VS. DEVS

Discrete Event System Specification (DEVS) is a formalism for specifying models to be simulated (Zeigler 1976, Chow and Zeigler 1994). DEVS provides a meta-model for model developers, but does not describe an architecture for simulators simulating DEVS based models. It should be noted that DEVS is not specifically targeted for network simulators. However, there is an apparent similarity between the architectural aspects of DEVS models and the structure of AMINES-HLA based simulation architectures. Therefore we want to comment on DEVS shortly in this section.

The DEVS based models are composed of basic (atomic) models that receive messages from input ports, create messages at the output ports, and change state. In this respect, the DEVS basic models can be implemented as UMIs in AMINES-HLA.

DEVS also defines what is called “coupled models”, which are composite models built by composing basic or other coupled models. The inputs and outputs of a coupled model are mapped to the inputs and outputs of its component models. AMINES-HLA does not support composed entities as basic entities. However, coupled models are in essence a tool for increasing the manageability of the modeling process from the human modeler’s point of view. Such an abstraction can be employed in a simulator design made using AMINES-HLA, where the tool for model development may provide representations of such coupled models to the modeler, and instruct the CUI of the simulator to create the proper RTR of the coupled model as made up of a set of connected UMIs. This draws onto the fact that the model creation and RTR creation are two different processes.

We believe that an implementation of a DEVS simulator on AMINES-HLA is possible. As an example to the benefit of doing so, we believe one may have largely avoided the need for the abstract model defined by Wutzler and Sarjoughian (2006) in order to interoperate different DEVS simulators, if the implementations of the simulators were based on AMINES-HLA.

5 AMINES-HLA VS. THE ARCHITECTURES OF OMNeT++ AND NS-2

In this section, we will shortly compare the AMINES-HLA to architecture of the network simulators OMNeT++ (Varga 2001), and the NS network simulator (Breslau et al. 2000). A comparative study of AMINES-HLA and GloMoSim (Zeng et al. 1998, Gerla et al. 1999), and AMINES-HLA and SSFNet, which is based on Scalable Simulation Framework (Nicol and Liu 2001), remains as future work.

The OMNeT++ models consist of modules that communicate with message passing. The modules can be simple, or compound, similar to atomic and compound models in DEVS. They can be parametrized, which are initialized when a module is instantiated in the RTR of a particular simulation model. Modules are connected via gates, and connections between gates. Connections can have simulation related properties such as propagation delay, data rate, and bit error rate. OMNeT++ modules have the option of bypassing the gates and connections, and sending messages to other modules directly. It is also possible to modify the run-time configuration of how the modules are connected, dynamically during execution of the simulation.

It is apparent that the OMNeT++ simulator is based on an architecture similar to the AMINES-HLA. The main reason why the scenarios described in Sections 6 and 7 can not be easily realized using OMNeT++, lies in the fact that OMNeT++ modules depend on method calls between objects for data flow between modules. Data flow is strictly and explicitly confined to the links between instances in AMINES-HLA. The gate and connector abstraction provides the same locally meaningful addressing of messages as links in AMINES-HLA provides. However, allowing modules to directly communicate with each other would introduce problems about hidden dependencies between modules. Furthermore, although it can be provided in the AMINES-HLA based simulator architectures using CUIs, we doubt the benefits of allowing changes in the configuration of how the models are connected during execution of the simulation, once the RTR is constructed.

The NS-2 network simulator is widely used in the network research community. It uses a dual-programming approach to employ the TCL scripting language as the model description method. NS-2 is developed by a large community, and the code has traces of this collective effort. The architecture is very diffuse. We believe that the architecture being so diffuse adds up to the steep learning curve of NS-2. There seems to be no limitation, hard or suggested, on the communication between objects that make up the model. However, we believe that the `NsObject` class along with class `Event`, forms a basis for re-factoring the ns-2 simulator using AMINES-HLA. We have very recently started a project to look into this issue.

6 SCENARIO FOR MODEL REPLACEMENT

In this section and the next, we are going to describe two scenarios. In the first one, we will give an example of how models can be replaced in simulation systems built using the AMINES-HLA. In the second scenario, which will be presented in the next section, we will introduce how different simulators built using the AMINES-HLA can interoperate.

In our first scenario, we have a simulator, which we will refer to as *SIM*, whose architecture conforms to AMINES-HLA. We also have a program and an associated protocol stack, which we have virtualized on a particular machine architecture and operating system. We would like to use this virtualized program and protocol stack as part of a simulation model built to be simulated in SIM. The machine architecture or the operating system the virtualization of the program and protocol stack was made in, does not necessarily have to be the same as the machine architecture or the operating system SIM was designed to run on. Such an assumption is unnecessary as long as we have implementations of AMINES-HLA-RTI, which can communicate with each other, on these architectures and operating systems.

The first observation that we will point to, is that by interoperating a virtualized program and protocol stack along with the simulator SIM, we are in effect using SIM in an emulation setting. Using simulators in emulated settings is a source of potential problems (Göktürk 2005). A discussion of how emulation related problems can be avoided using various design principles in cases as we have in our first scenario, is out of the scope of this paper. However, at the very least, either the simulator SIM must be capable of managing a virtual time synchronized to the wall-clock time, or the virtualization of the program and protocol stack must be capable of virtualizing the time of the program and protocol stack as well.

The second preliminary condition that should hold for our solution in this first interoperation scenario, is that in the simulation system SIM, the creation of the RTR of the executable simulation model must precede execution of that model. What we refer to as the creation of the RTR of the executable simulation model, is the creation of unit models as UMIs and the links between them, through a set of CUIs. The topology of the UMIs must be constructed and fixed before the execution of the composite model they constitute starts. New UMIs or links must not be created during execution. Although this condition appears as overly restrictive, situations where the experimenter can not predict an upper bound on the number of model instances (for example entities), are seldom encountered in network simulations.

Since the simulator SIM is assumed to have an architecture derived from AMINES-HLA, it will have one CUI that is created as the first unit instance when SIM is executed. We will refer to that CUI as the main CUI of simulator SIM, or SIM-CUI. The simulator SIM may employ other CUIs when constructing the RTR of the executable model, but the main CUI presents a single point of contact for the whole simulation system. Our third preliminary condition is that the SIM-CUI must support the following as three separately invokable services: construction of the RTR of the executable model, configuration of the individual model instances, and execution of the composite simulation model. We will assume that the SIM-CUI accepts three different messages, `CONSTRUCT`, `INITIALIZE`, and `RUN`, as invocation requests for these services.

In this scenario, we would like to substitute a virtualized program and protocol stack, for some part of a simulation model. After the construction of the RTR of this simulation model, the parts we would like to substitute will correspond to a subset of the RTR. As the simulator SIM is using AMINES-HLA, this subset of the RTR will constitute of one UMI, or more than one UMIs and links between them. The fourth and last preliminary condition that our solution will require, will be that there must exist a mechanism to identify this subset of RTR and obtain the identifiers of the relevant UMIs. Such a mechanism must be accessible through the SIM-CUI, as it is the single point of contact for simulator SIM. Using symbolic names that relate UMIs to the role they play in the simulation model, the SIM-CUI can provide such a mechanism by supporting queries for mapping these symbolic names to UMI identifiers. Such a mapping can not be provided by AMINES-HLA-RTI alone, since the role of a particular UMI in a simulation model can rarely be deduced solely from how the UMIs are connected to each other, which is the only information available to AMINES-HLA-RTI. The SIM-CUI must also support queries for obtaining the topology of the UMIs that have been created by SIM for the simulation model at hand. This topology information is necessary for correctly accounting for the connections of the to-be-substituted set of UMIs to the rest of the RTR of the model.

Before we describe how to connect the virtualized program and protocol stack to the simulator SIM, we must look into how SIM would run the simulation model if we did not interfere with it. The sequence of events that we describe in this paragraph, are exemplified graphically in Figure 2. In such a normal run, the main CUI for the simulator SIM (SIM-CUI) is created by some initialization code (step 1 in Figure 2). This initialization code is specific to the programming environment that is used for implementing the simulator SIM. For example, if the programming environment is C or C++, the SIM-CUI would be created in the `main()` function, as depicted in Figure 2. Since this initialization code is not a proper instance in the AMINES-HLA, it can not send the `CONSTRUCT`, `INITIALIZE`, or `RUN` messages that are to be sent to the SIM-CUI. Therefore the initialization code must invoke some behavior in the SIM-

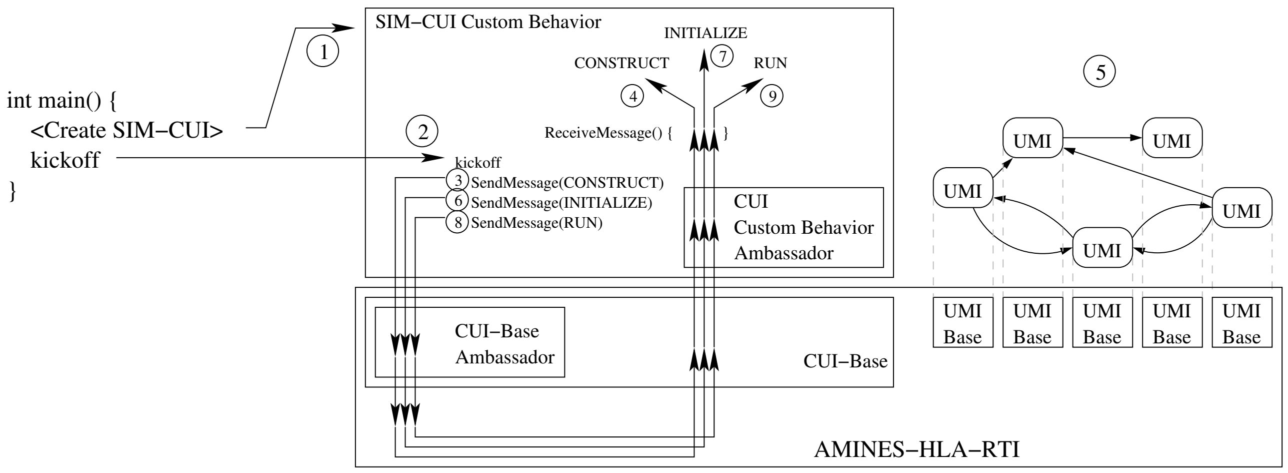

Figure 2: Normal Run of Simulator SIM

CUI through some means other than the message passing mechanism provided by AMINES-HLA, for example by calling a function in the implementation of the customized behavior of the SIM-CUI (step 2). This kickoff function may then send the `CONSTRUCT` (step 3), `INITIALIZE` (step 6), and `RUN` (step 8) messages to the SIM-CUI through the `SendMessageToCui` service provided by the CUI-base associated with the SIM-CUI. In response to these messages, the SIM-CUI receives three `ReceiveMessage` callbacks (steps 4, 7, and 9), and constructs (step 5), initializes, and runs the RTR of the simulation model.

We can now describe how the virtualized program and protocol stack can be made to interoperate with the simulator SIM. Up to the initialization of the simulation model and the virtualized program and protocol stack, the steps described here are outlined in Figure 3.

First and foremost, the virtualized program and protocol stack must have been encapsulated in, or abstracted behind, one or more UMI custom behavior implementations. Therefore the virtualized program and protocol stack should appear in the AMINES-HLA as one or more UMIs. For the sake of simplicity, we will assume in this scenario that the virtualized program and protocol stack will be represented by a single UMI in the AMINES-HLA, which we will refer to as VPPS-UMI. When the virtualized program and protocol stack is represented by more than one UMI, there will presumably be a CUI for overseeing the construction of these UMIs and links between them, and the situation will resemble the second scenario we will describe later in this paper.

In order to integrate the VPPS-UMI into the RTR of the simulation model built by SIM, we need to have a new main CUI for the combined SIM-VPPS system. To accomplish this, we will have to be able to either change or bypass the main initialization code, the main function in Figure 2. If we choose to change the main function, we need to recompile the new initialization code, and re-link the SIM simulator. When re-linking is not an option, bypassing the main function through library preloading techniques presents an alternative method in UNIX environments.

Once the new main CUI for the combined SIM-VPPS system is created (step 1 in Figure 3), the new initialization code will hand over control to the new main CUI (step 2 in Figure 3). As discussed before, the initialization code has to accomplish this hand-over by some other means than AMINES-HLA supported message exchange mechanism, for example by calling a kickoff function in the implementation of the custom behavior for the new main CUI. The main CUI then follows the send-to-itself mechanism outlined when we were describing the normal operation of the simulator SIM. The main CUI following the same send-to-itself mechanism, ensures that even the combined SIM-VPPS system can later be modified using the technique outlined in scenarios we describe in this paper.

When the main CUI receives the message `CONSTRUCT` that it has sent to itself (steps 3 and 4 in Figure 3), it has a list of tasks to do:

1. construct the simulation model using the simulator SIM,
2. construct the virtualized program and protocol stack through VPPS-UMI,
3. find and replace the customized behavior of certain UMIs,
4. and construct some UMIs and their links that will transform messages to be passed between the simulation model and VPPS-UMI.

To accomplish the first task, the main CUI will first create a SIM-CUI (step 5 in Figure 3), using the `CreateCui` service request. Then it will send the SIM-

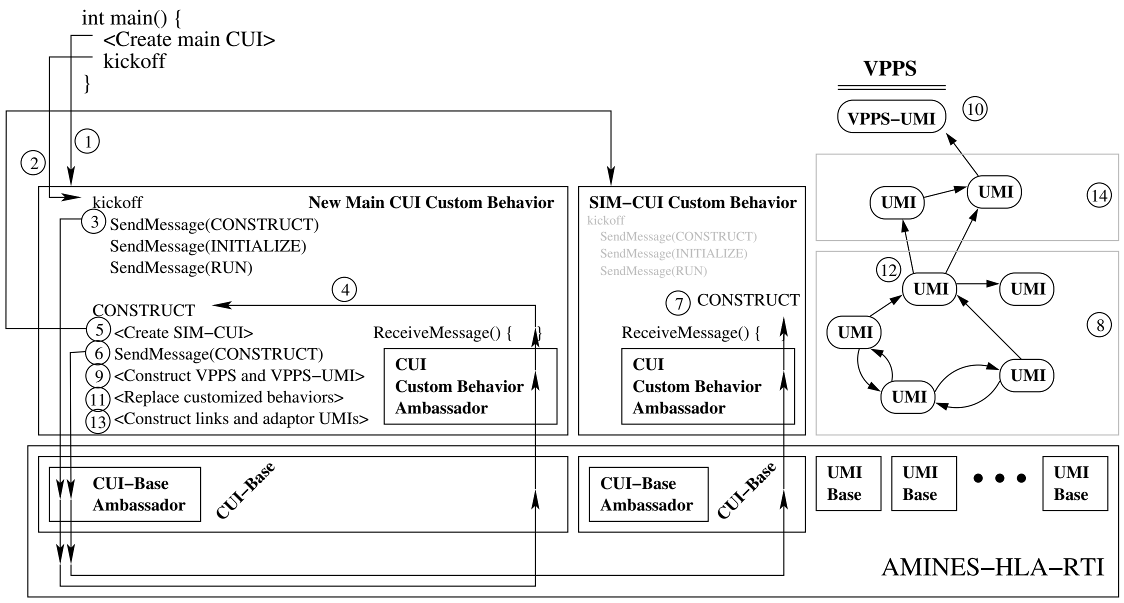

Figure 3: Running Simulator SIM with Virtualized Program and Protocol Stack, up to Initialization

CUI a `CONSTRUCT` message (step 6 and 7 in Figure 3). The SIM-CUI knows the `CONSTRUCT` message since we assumed that it satisfies our third preliminary condition. In response to the `CONSTRUCT` message, the SIM-CUI will construct the RTR of the simulation model (step 8 in Figure 3).

The main CUI will then create the VPPS-UMI, using a `CreateUmi` service request (steps 9 and 10 in Figure 3). The VPPS-UMI may construct the necessary environment related to virtualization of the program and protocol stack at this point in execution, or it may defer such initialization until the simulation model is initialized by the SIM-CUI. The reason why it is possible to defer the initialization of the virtualization environment until initialization of the simulation model, is that it is possible for the VPPS-UMI to get information and notification about initialization from the UMIs through which the VPPS-UMI is connected to the RTR of the simulation model. The decision about when to initialize the virtualization environment would not affect our scenario. Even if the VPPS were represented with more than one UMIs, we would require a version of the second preliminary condition. Therefore we would have required that their topological configuration may not change once constructed, and the decision about initialization time would still not affect our scenario.

The third task that the main CUI must do, is to find and modify the UMIs that represent the part of the simulation model whose role will be assumed by VPPS (steps 11 and 12 in Figure 3). In order to find the ids of the relevant UMIs, and how they are connected to the other UMIs in the RTR of the simulation model, the main CUI must use the queries we described as the fourth preliminary condition. Just to remind the reader, one of these queries was about obtaining the topology of how the UMIs are connected, and the other was about mapping a name that has semantic connection to some part of the simulation model to the UMI or UMIs that represent that part of the model in the RTR. Once the main CUI identifies the relevant UMIs, it issues `ReplaceUmiCallbackAmbassador` service requests to replace the custom behavior on these UMIs, with behaviors that will take role in channeling messages back and forth between the RTR of the simulation model built by SIM, and the VPPS. The resources used by the implementations of the replaced customized behaviors may be reclaimed at this point if necessary.

An apparent alternative to using the service request `ReplaceUmiCallbackAmbassador` is to delete the relevant UMIs in the RTR of the simulation model, creating new UMIs with necessary behavior, and then linking these new UMIs to the RTR of the simulation model in exactly the same way as the deleted ones. The reason why this "apparent" alternative does not present a real alternative, stems from the fact that information about UMIs' identifiers can be used in two places in the AMINES-HLA: in describing links between the UMIs, and in CUI custom behaviors. Although we update the information about the deleted UMIs when we link the newly created ones in exactly the same way the deleted ones were linked, we did not require that the SIM-CUI be the only CUI in the simulator SIM, and we have no control over the data structures of any CUIs except for the main CUI. AMINES-HLA mandates that a UMI is to be identified with nothing else but its UMI identifier. Therefore, unless we impose additional conditions on the SIM-CUI for updating its and any other

CUIs data structures with such UMI replacements, the replacement strategy results in invalidation of UMI identifier data kept in CUIs. To prevent that, the main CUI must use the `ReplaceUmiCallbackAmbassador` service, as it does not change the UMI-base of the UMI it is invoked on. This, in effect, means that the UMI identifier of the UMI, which we have updated to have new behavior, does not change, and no data structure kept by CUIs is invalidated. Furthermore, since the links are defined via UMI identifiers as well, in a proper AMINES-HLA-RTI implementation, the `ReplaceUmiCallbackAmbassador` should not necessitate recreation of the links connected to the UMI the behavior of which is being replaced.

The fourth and last task of the main CUI must do in response to the `CONSTRUCT` message, is to establish the connection between the VPPS-UMI and the UMIs whose behavior were replaced (steps 13 and 14 in Figure 3). Depending on the design of the VPPS-UMI and the RTR of the simulation model, this may involve simply linking VPPS-UMI and the UMIs whose behaviors were replaced, or it may require creation of some number of UMIs working to adapt the behavior of the VPPS-UMI to the behavior expected from the UMIs whose behaviors were replaced.

At this point the RTR of the simulation model extended with the VPPS is set up and ready to be run. The main CUI sends the `INITIALIZE` and `RUN` messages to the SIM-CUI, and the simulation runs. When the simulation is finished, the main CUI will have to do necessary clean-up tasks for the UMIs that the SIM-CUI is not aware of, such as the VPPS-UMI, and the UMIs used for adapting the VPPS-UMI to the simulation model. This concludes our first scenario.

7 SCENARIO FOR SIMULATOR INTEROPERATION

In our second scenario, we will discuss how two simulators, each of which works on AMINES-HLA-RTI, can be interoperated. We will not give a detailed step-by-step description as was done in the previous section, but we will focus on differences from the previous scenario.

Let us name the two simulators we have in this scenario as SIM-1 and SIM-2. In a stand-alone normal run, these simulators follow the sequence of steps described in the previous section when we described the stand-alone normal run of the simulator SIM. Therefore, each of these simulators have their own CUIs similar to the SIM-CUI in Figure 2. A new main CUI for the combined SIM-1 SIM-2 system is also needed. This main CUI will create the SIM-1-CUI and SIM-2-CUI, in the same was as the main CUI in our previous scenario creates the SIM-CUI. Then the main CUI will find and modify certain UMIs in each of the RTRs of models built by the simulator SIM-1 and SIM-2, by replacing their customized behavior, again as discussed in the previous section. Then the main CUI will create the necessary adapter UMIs, set up their links, and send the `INITIALIZE` and `RUN` messages to the SIM-1-CUI and SIM-2-CUI.

Although the mechanism for interoperating two simulators can be described with relative ease at the architectural level where the AMINES-HLA lies, architectural interoperability is not the only issue. How the adapter UMIs and their connection to the simulators should be implemented, is a problem whose solution depends on various design details of the simulators involved. These design details can be categorized under two main groups. The first category of design details relates to the specifics of the meta-model of the target domain, and the mapping between the simulation model and the RTR. The other category relates to the structure of the RTR.

Modeling causal relationships, or in other words time, is one of the most important meta-model related issues, which we can cite as an example to the first category of design details. The AMINES-HLA regards time management as a normal part of the simulation model. Therefore, unlike other simulation architectures and systems, AMINES-HLA does not define a simulation executive which provides time management services. Instead, the time management services have to be designed as part of the simulation meta-model assumed by the simulator built on AMINES-HLA, and implemented using model instances on AMINES-HLA-RTI. This means that the mechanics of making the schedulers of two different simulators interoperate, would involve the same mechanics described before with respect to virtual program and protocol stack example in the previous section. The solution to the problem of designing and employing the suitable adapter UMIs will have to be tackled case by case. This is because although general solutions can be provided to the problem of how to make two simulators with different time models can be formulated at the meta-model level, there is more than one way to implement RTRs of various time models.

As an example to the second category of design details, we can mention the specifics of how the control flow is established in the RTR of the simulation model, and the relationship between this control flow and the semantics of interactions between the models instances in the simulation model. By control flow here, we refer to the sequence of execution of the various code fragments representing the model instances in the simulation model at run-time. One solution to the control flow problem, might be running the simulators in their own execution units, therefore running them concurrently. Then each simulator would run exactly in the way it would if run in a stand-alone fashion. Another alternative would be to transfer control flow from one simulator to the other when messages are carried through the adapter UMIs. The solution to the problem would again be dependent on specifics of the RTR of the simulators.

8 CONCLUSIONS

In this paper, we have shortly presented the high level architecture we have developed for network simulators, and described two scenarios for model replacement and simulator interoperability. Our main contribution in this paper is the demonstration, through the detailed description of the scenarios, of the feasibility to realize just-before-run modifications in and across network simulators, by employing a suitable high level architecture in simulator design.

We argued that for simulator interoperability and model interreplaceability, the simulators must conform to additional behavioral restrictions in addition to working on AMINES-HLA. One such restriction is that the customized behavior running on the AMINES-HLA fundamental entities, the unit model instances and constructor unit instances, must be confined to locally available information in their processing, and communicate only through AMINES-HLA if it is necessary. This restriction causes a considerable challenge in re-factoring network simulators such as OMNeT++ and NS-2, which allow for arbitrary communication between modules, and objects respectively. Our currently ongoing work covers this problem.

Another considerable limitation which was required by our scenarios, is that the topology of the unit model instances must not change once the simulation model starts to execute. Most, if not all, network simulators also allow such topological reconfigurations of the base elements of the run-time representation of the model, to be made anytime. Although such reconfigurations can be incorporated in the simulator design by using constructor unit instances actively in the run-time representation of the simulation model, AMINES-HLA is designed with focus toward simulators that keep the topology of the unit model instances in the simulation model constant after initial construction. Assessment of the impact of having to support dynamic run-time representations due to requirements of the re-factored simulator, remains as one of possible future research directions.

Subjects for future work are many. The literature on network simulators is large, and mainstream network simulators are complex programs that take time to master. Comparing AMINES-HLA to SSF and SSFNet, and to Parallel/Distributed NS (PDNS) is possible. We are looking into re-factoring OMNeT++ and NS-2 on AMINES-HLA. We currently have a single-threaded RTI for AMINES-HLA, and we are in the process of implementing an RTI that will support the concurrency extensions, with a focus towards use of multi-core processor architectures. Implementing small scale special-purpose network simulators is also considered, but not preferred, as there already is an inflation of such small scale special-purpose network simulation projects. Analyzing performace currently remains as future work, as the performance of the RTI would be based mainly on the optimizations in the RTI implementation and structure of the models simulated on this architecture.

REFERENCES

Agha, G. A. 1986. *ACTORS: A model of concurrent computation in distributed systems*. Series in Artificial Intelligence. Cambridge, Massachusetts: The MIT Press.

Agha, G. A. 1996. Modeling concurrent systems: Actors, nets, and the problem of abstraction and composition. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets, ICATPN'96 (Osaka, Japan, June 24-28, 1996)*, ed. J. Billington and W. Reisig, Volume 1091 of *LNCS*, 1–10. Springer-Verlag.

Breslau, L., D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. 2000, May. Advances in network simulation. *IEEE Computer* 33 (5): 59–67. Expanded version available as USC TR 99-702b at `<http://www.isi.edu/~johnh/PAPERS/Bajaj99a.html>`.

Chow, A. C. H., and B. P. Zeigler. 1994. Parallel DEVS: A parallel, hierarchical, modular, modeling formalism. In *Proceedings of the 26th Winter Simulation Conference (WSC'94)*, 716–722. San Diego, CA, USA: SCS.

Fujimoto, R. M. 2000. *Parallel and distributed simulation systems*. John Wiley & Sons.

Gerla, M., L. Bajaj, M. Takai, R. Ahuja, and R. Bagrodia. 1999, May. GloMoSim: A scalable network simulation environment. Technical Report 990027, University of California, Los Angeles, Computer Science Department.

Göktürk, E. 2005, October. Emulating ad hoc networks: differences from simulations and emulation specific problems. In *New Trends in Computer Networks*, Volume 1 of *Advances in Computer Science and Engineering: Reports*. Imperial College Press.

Göktürk, E. 2006a, May. Design of a higher level architecture for network simulators. In *Proceedings of the 20th European Conference on Modelling and Simulation (ECMS 2006)*.

Göktürk, E. 2006b. Elements and stakeholders of network simulation. Technical Report 338, University of Oslo, Department of Informatics.

IEEE 2000. IEEE standard 1516 for modeling and simulation (M&S) high level architecture (HLA).

Nicol, D. M., and J. Liu. 2001, November. The scalable simulation framework. In *Proceedings of the ACM/IEEE conference on Supercomputing 2001 (CD-ROM)*: ACM SIGARCH/IEEE. Poster.

Riley, G. F. 2003. The Georgia Tech network simulator. In *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, 5–12. New York, NY, USA: ACM Press.

Varga, A. 2001, June. The OMNET++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*.

Wutzler, T., and H. S. Sarjoughian. 2006, April. Simulation interoperability across parallel devs models expressed in multiple programming languages. In *Proceedings of DEVS Integrative M&S Symposium (DEVS'06)*: SCS.

Xu, D., G. F. Riley, M. H. Ammar, and R. Fujimoto. 2001. Split protocol stack network simulations using the dynamic simulation backplane. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*, 158–165. Washington, DC, USA: IEEE Computer Society.

Zeigler, B. P. 1976. *Theory of modeling and simulation*. Wiley.

Zeng, X., R. Bagrodia, and M. Gerla. 1998. GloMoSim: a library for parallel simulation of large-scale wireless networks. In *PADS '98: Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation*, 154–161. Washington, DC, USA: IEEE Computer Society.

AUTHOR BIOGRAPHY

EREK GÖKTÜRK is a doctoral research fellow in the Department of Informatics, University of Oslo, since 2004. He graduated with BS degree in 2000 and MS degree in 2003 from Dept. of Computer Engineering (CEng), Middle East Technical University (METU), Ankara, Turkey. He worked as an assistant in CEng@METU between 2000-2004, and participated part-time in a multi-agent simulation project in the Modeling and Simulation Laboratory, a METU-TSK (Turkish Armed Forces) joint research facility. He also has instructed in METU-CES (Continuing Education Center) Regional Cisco Networking Academy during 2002-2004. His research interests include emulation and simulation, ad hoc networks, multi-agent systems, and philosophy of computing and information. His e-mail address is `<erek@ifi.uio.no>`, and his web page is `<www.ifi.uio.no/~erek>`.