DESIGN OF A HIGHER LEVEL ARCHITECTURE FOR NETWORK SIMULATORS

Erek Göktürk
Department of Informatics
University of Oslo
Blindern, N-0371 Oslo, Norway
Email: erek@ifi.uio.no

Keywords— **Network simulation, high level architecture, architecture, terminology, stakeholders**

Abstract— **In this paper, we present the motivation for designing a high level architecture for network simulators. We discuss the elements and stakeholders of network simulation, in order to set the terminology for our discourse on network simulator architectures. We also present a list of hypotheses about network simulators, which we believe would lead to better network simulators and network simulation experiments, if realized as design requirements for network simulators. Then we describe the design of the AMINES high level architecture for network simulators. AMINES-HLA have been developed to provide a common architectural basis for network simulators that aim to support the hypotheses presented.**

I. Introduction

Simulations have been one of the main methods for understanding, tuning, and therefore designing networks and network protocols. Interest in network simulation in the simulation community also have been increasing. This continuing increase in interest is caused by challenges introduced by two developments at the networking research side. The first development is the ever-growing Internet. Understanding how existing and newly proposed protocols would behave in this massive network, has created a demand for highly scalable network simulation systems. The other development is the development and deployment of new wireless technologies. These wireless technologies have introduced mobility, dynamic and complex physical channel characteristics, and resource limited network nodes. The wireless technologies have thereby changed the processing power needs of the models employed in network simulations and affected the architectures of the simulators.

The number of network simulators is growing by the day. Among the mostly used network simulators, Ns-2 [1], GloMoSim [2], SSFNet [3], OPNET Modeler, and OMNeT++ [4] can be mentioned. Beside these major systems, papers reporting new simulators are being published in almost every major conference on networking or simulation.

What is described in this paper, is not a new network simulator, but a high level architecture for network simulators. We believe that the lack of a common higher level architecture increases the cost for reusing models of networks across the network simulation systems. Interestingly this happens despite an apparent architectural similarity between the simulators available today. This last comment naturally does not cover commercial systems like OPNET Modeler where the source is not available.

The idea behind having a high level architecture for network simulators is twofold. The first idea is to standardize channels of communication between architectural elements of simulators, so that different simulators can be connected and run together by tapping into these standardized channels, and moving back and forth data between the simulators. This is also one of the ideas behind the High Level Architecture (HLA), the IEEE standard 1516 for modeling and simulation [5]. However, the high level architectures for network simulators should aim for a lower granularity mapping between the architectural elements and the simulation model elements. This is because it appears beneficial to compose the models of networks from submodels of varying detail in network simulation experiments. Therefore we favor imposing a one-to-one mapping between elements in the metamodel, which is assumed and used when creating the simulation model, and the elements of the simulator's architecture. IEEE standard HLA leaves such issues of multiplexing model elements onto federates to the designers of the simulators.

The second reason for having a high level architecture for network simulators is to ensure that the run-time simulation management functionality in the simulators is sufficiently abstracted from the executable models of the networks being studied. Although part of such functionality is also included in the IEEE HLA's Run-Time Infrastructure, IEEE HLA does not explicitly specify where and by which entities or programs the construction of the run-time representation of the model will be made. We believe this is an integral part of the network simulation systems, as evidenced by various script based or graphical tools for designing experiments on current network simulation systems. Abstraction of this functionality would ease simulation model element reuse at the level of executable models. It would become possible that the executable models of two network simulators constructed using different simulation libraries or systems can be run together by stripping their management functionality and customizing it. Naturally, necessary model entities that will act as data bridges between the two models should also be added.

This paper reports our ongoing work on the AMINES high level architecture (AMINES-HLA) for network simulators. We have finished our initial design of the system, implemented a sequential (single-thread) run-time infrastructure, formulated some extensions to the architecture for concurrency support, and we are currently in the process of implementing a run-time infrastructure for AMINES-HLA with concurrency extensions. The concurrency extensions involve managing execution units, which are similar to log-

Proceedings 20th European Conference on Modelling and Simulation
Wolfgang Borutzky, Alessandra Orsoni, Richard Zobel © ECMS, 2006
ISBN 0-9553018-0-7 / ISBN 0-9553018-1-5 (CD)

ical processes [6], and will not be discussed in this paper due to size limitations. Design of a typical simulator is exemplified in [7], along with comparison of AMINES-HLA to architecture of various other simulation systems.

The paper is organized as follows: in Section II, we analyze the role a higher level architecture would play by discussing the elements and stakeholders of network simulation, and propose a set of hypotheses about building network simulators. Then, in Section III, we describe the design of AMINES-HLA. We present our conclusions, and point to further research topics and directions in the last section.

II. Elements and Stakeholders of Network Simulation

In this section, we will lay the grounds for the rest of the paper by identifying the elements of network simulations by defining terminology, identifying the stakeholders involved, and discussing their goals and interests. The section is concluded with some hypotheses intended to serve as guidelines for the network simulation system and library developers. An extended presentation of the material in this section, can be found in [8].

A. Elements

The goal of a simulation experiment is to investigate some target phenomena through modeling. The central element in the target phenomenon in a network simulation experiment (NSE) is a network which exists for real or as a design. Such a network we will call a *target network* (TN).

The TN is represented in an NSE as a model. Since we don't intend to attempt at giving a complete definition and a theory of what constitutes a model, if will suffice to say that a model can be roughly defined as a description of a target phenomenon. In the case of networks, we will refer to such a description of a TN used in a NSE, as the *model of the target network* (MoTN). We would like to stress that although usually regarded as part of the simulation system, schedulers are in fact parts of models.

Certain characteristics of models appear in a discourse on simulation. Detailedness, precision, accuracy, and granularity are such characteristics that appear frequently in the discourse on network simulations. Precision and accuracy can be defined relatively easily with respect to actual measurements taken from the TN, or realization of the TN if the TN exists as a design. Providing a definition for detailedness is more difficult, so we will refer to the intuitive meaning of being detailed. Granularity is a comparative relationship between two models, and can be defined using specificity, which itself is a comparative relationship. For two models M_a and M_b, M_a can be said to be more specific than model M_b if M_a models a subset of the phenomena modeled by M_b. A composite model M_{C1} can be said to have a smaller granularity than M_{C2} if they model the same phenomena, and M_{C1} is composed of more specific models than those composed into M_{C2}.

We define a *target network simulator* (TNSim) as the simulator whose software specifications are derived from a chosen simulation system or library and a MoTN. A TNSim is composed of parts that provide simulation management functionality (SMF), and an executable MoTN. By an executable MoTN we refer to an MoTN prepared as a computational machinery (software). For effectively building a TNSim, the metamodel the MoTN is derived from, should be in reasonable agreement with the architectural and functional specifications inherited from the use of a particular simulation system or library.

A simulation library (SimLib) is a library of helper routines for implementing SMF in TNSim. Similarly, a simulation system is an extensible software that provides ready-made SMF functionality to be extended with an executable MoTN to build a TNSim. For connecting the so implemented SMF functionality to the executable MoTNs, SimSys and SimLibs imposes a set of interfaces and architectural requirements on the executable MoTN they are used in conjunction to. SimSys and SimLibs usually come with a library of ready-made models, which we will refer to as the model library (MoLib), or executable MoLib to stress that the models in the MoLib are executable models, where necessary.

B. Stakeholders

Two different parties are involved in NSEs: the experimenters, and the network simulation system or library developers. The experimenters are the researchers who conduct NSEs. They have two major goals or interests: to design, implement, and conduct the experiment with minimum effort and resources, and to carry out the experiment in such a manner so as to obtain adequately meaningful data. The experimenter's process of realizing the experiment can be divided into three stages: pre-run, run-time, and post-run. In the pre-run stage, a network SimSys or SimLib is chosen. It is important at this point that the specifications imposed by the chosen system or library should be sufficiently compatible with the metamodel to be chosen to prepare the MoTN with. After choosing the metamodel and the SimSys or SimLib, the experimenter prepares the executable MoTN, initial conditions, and the workload. Then the MoTN is converted into a TNSim using the SMF functionality of the chosen SimSys, or using the SMF functionality the experimenter implements using the chosen SimLib. In the run-time period, the TNSim is executed. During the execution of the TNSim, a run-time representation of the executable MoTN is constructed, and run. The results are gathered, and subjected to limited online analysis if necessary. In the post-run period, the data gathered are analyzed and results derived.

The other stakeholder in NSEs is the developers of the SimSys and SimLibs. The developers design, implement, and maintain a system or a library. They don't conduct NSEs themselves. They aim to build systems or libraries that are simple but adequate, and well documented. Furthermore, they aim to make the specifications that their systems would impose on the TNSim, compatible with some assumptions about metamodels that will be used by the experimenters when building MoTNs. They also attempt to design for good performance. Usually, to provide a model library with basic models of conventional hardware

and software, is also regarded as a goal for the developers, though this goal would definitely involve experts from the networking domain. The tasks undertaken by the developers involve choice of target metamodel, preparing compatible software specifications, determining how the specifications are to be mapped to programing language structures, designing the interfaces for the experimenters, and implementing clean and well documented code.

The goals, interests, and tasks of the two stakeholders are sufficiently different so as to grant a clear separation of their roles when realizing NSEs. The Ns-2 simulator provides a counterexample, where the experimenters are expected to extend the core architecture in a reusable way if they need so when implementing their MoTNs. This being done over the last decade (since 1995), the result is that the Ns-2 code is hard to understand and difficult to follow. Such a state is also reinforced by the fact that the SMF and executable MoTNs, and therefore the SMF and the provided MoLib, are not sufficiently separated in Ns-2's architecture.

C. Hypotheses

Building on the elements and stakeholders described, we will provide a list of hypotheses for the network SimSys and SimLib developers. We believe the satisfaction of these hypotheses in design requirements would lead to network SimSys and SimLibs that are easier to use, to an experimentation process that is easier to follow for the experimenter, and thereby to better quality experiments.

H1: The experimenters and SimSys and SimLib developers are two different stakeholders whose roles should not be confused when conducting NSEs and building network SimSys, SimLibs, or MoLibs.

H2: Assumptions made by developers about the metamodel used by the experimenter, should be made explicitly. The metamodel assumed should be in good agreement with the expert knowledge that the network researchers use when they model networks in their day-to-day research. Such an agreement is aimed at decreasing the learning and cognitive load on the experimenter. The software specifications that will be imposed on TNSims by the developed SimSys and SimLibs, should be compatible with the metamodel assumed that the experimenters would use.

H3: Modeling parts of TNs with submodels of lesser detail, is common practice. Therefore any metamodel assumed while developing SimSys and SimLibs, and the software specifications imposed by them, should allow for submodels of different detail and complexity in the MoTN.

H4: The metamodel may describe executable MoTNs directly, or some transformation might be needed to convert the MoTN described by the metamodel to an executable MoTN. In any case, the resulting executable MoTNs should be efficiently implementable.

H5: The executable MoTNs constructed conforming to the software specifications imposed by the SimSys or SimLib, should preferably be transparently distributable, or transparently transformable to run in a concurrent manner. The transparency as used here, is from the perspective of the experimenter. This hypothesis follows both from emulation related needs, and current processor trends towards multi-core architectures.

H6: A metamodel for producing executable MoTNs that uses compositions and connections of homogeneous simple model elements with customizable behavior, should be of sufficient expressive power. Along with restrictions on customized behavior, a clean simple interface for customization, and a transparently distributable architecture for simple model elements, such a metamodel should be in good agreement with the hypotheses H1–H5.

III. Design of AMINES-HLA

AMINES-HLA is a higher level architecture for network simulators. It is designed to be used as basis for network simulator architectures.

The AMINES-HLA assumes a metamodel that is somewhat similar to the actor model ([9], [10]). The reason for making this selection on behalf of the developers, is to impose specifications upon the architecture of the network simulators that are built on AMINES-HLA, so as to influence their design to satisfy hypothesis H2.

AMINES-HLA defines two types of units, the unit models and constructor units, and also specify how they communicate.

A. Unit Models

A *unit model* (UM) in AMINES-HLA represents an indivisible (non-composite) model provided in the network simulator for building the MoTNs. The detailedness of these UMs is not specified by AMINES-HLA, therefore a UM might be representing anything from something as simple as a FIFO queue, up to something as complex as a backbone router or a whole network.

Since the models represented by UMs can be at any detailedness level, their limitations might lead to multiple instantiations of these models in the run-time representation of the MoTN. This is in fact what is generally being done in network simulators in use today. Each of these instantiations are represented by *unit model instance*s (UMIs) in AMINES-HLA. In a way, the relationship between UMs and UMIs resemble class-object relationship in object oriented modeling. Therefore, although not strictly necessary, it appears favorable to organize a library of unit models to take advantage of inheritance in object oriented programming.

A.1 Connecting UMIs

In the run-time representation of the MoTN, the UMIs are connected to other UMIs with uni-directional data paths called *links*. Given a UMI u, a link connected to u can be either an *inlink* if its direction is towards u, or an *outlink*, if its direction is away from u. Inlinks of an UMI carry messages to it, whereas outlinks carry messages away from it. Every link in the run composite model is an inlink for exactly one UMI u_1, and an outlink for exactly one UMI u_2, where u_1 is not necessarily different than u_2. Therefore whether a link is an inlink or an outlink can only be meaningfully distinguished with respect to a UMI. The inlinks and outlinks of a UMI u are defined to be logically

distinguishable by u through locally meaningful identifiers. Therefore, locally to a specific UMI u, no two inlinks or two outlinks of u can have the same identifier.

When sending messages, a UMI uses the outlink's locally meaningful identifier to specify the message destination. Therefore the address of any message sent by a UMI does not include any specification about the receiver.

Upon reception of a message, the receiving UMI can distinguish which inlink the message was received from, as it is passed the locally meaningful identifier of that inlink.

The contents of the message is completely left out from AMINES-HLA, to be determined in MoTN. Therefore in a particular MoTN, it can be decided to include some form of identifiers for the source and target UMIs of the message. Although this is allowed, using global identifiers as part of the general messaging schema in a simulator, would decrease reusability of the UMs considerably. Therefore, it should be avoided unless it appears necessary.

A.2 Behavior of UMs

The behavior of a UM is left out to be defined by the MoTN builder. The UMs with customized behavior serve as the set of types the UMIs that will be used in the MoTNs, will be instantiated from.

There are some restrictions on the behavior of the UMs. UMIs cannot create new UMIs, nor new links between any two UMIs. This amounts up to saying that the UMIs are not allowed to change the structure of the run-time representation of the MoTN while TNSim is being executed. Furthermore, as discussed above with respect to connections between UMIs, it is highly undesirable for one UMI to operate on direct references to other UMIs.

Other than these restrictions, a UMI can do any computation on the local information available to the UMI.

The logical structure of a UMI is shown in Figure 1. The ambassador entities provide points of contact for the base of the UMI to receive service requests from the UMI customized behavior (UMI-CB), and UMI-CB to receive callbacks from the base.

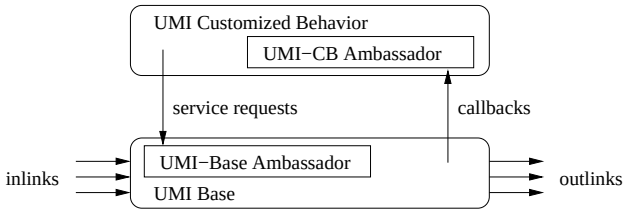

Fig. 1. Logical structure of a UMI.

A.3 Services Provided to UM Custom Behavior

Three types are involved in description of the services given to UMI-CB:

link-id: The *link-id* type is used to label the elements of the sets of inlinks and outlinks. This type should support smaller-than and equality operations, and given a set of link-ids S, generation of a new link-id l where $l \notin S$.

message: The *message* type represents the messages exchanged between the units in the AMINES-HLA.

UMI-Base-Ambassador: This type represents a UMI-Base's ambassador.

The following services are to be provided by the UMI-Base to the customized behavior of the UM:

sendMessage: Request to send a message through an outlink. The link-id of the outlink, and the message should be provided. There is no return value. As a result, a receiveMessage callback will happen at the UMI the given outlink is connected to.

receiveMessage: Callback indicating that a new message has been received. The link-id of the inlink that the message was received from, and the message is provided. No return value is expected.

umiBaseCreated: Callback indicating that the UMI, now has an associated UMI-Base. The UMI-Base's ambassador is provided as a parameter of type UMI-Base-Ambassador. No return value is expected. A UMI-CB receives this callback only once in its lifetime.

B. Constructor Units

The constructor units (CUs) are somewhat like the UMs described in the previous section, but the restrictions on their behavior are less strict, and they have an extended messaging mechanism. Therefore, in a way, the CUs are more powerful UMs.

As discussed with regard to UMs and UMIs, the CU can also serve as types for multiple instantiations. Each of these instantiations of a CU is referred to as a constructor unit instance (CUI).

The CUIs are the only units in the AMINES-HLA that are capable of being used for construction and initialization, and transformation of the run-time representation of the MoTN. The indication is that CUs are not really meant to be used in the run-time representation of the MoTN, but for constructing or doing modifications on it. CUs are the entities on top of which to implement the mechanics of setting up the initial state, and to provide the initial spark to start the execution of the run-time representation of the MoTN. Therefore the CUs are specifically targeted for implementing the SMF in TNSims.

One of the CUIs in any TNSim architecture is to be designated as the starter CUI, and created by an initialization code such as the `main()` function for C or C++ environments. It is the unit responsible for starting the creation of the other CUIs, UMIs, and links, which form the run-time representation of the SMF and MoTN.

B.1 Connecting CUIs

CUIs use instance ids (I-ids) assigned to each one of the UMIs and CUIs in the TNSim in order to describe destination of the messages they send. When receiving, the CUIs also receive the sender's I-id along with the message. AMINES-HLA does not specify a repository of all such ids, therefore the CUIs can only communicate with the UMIs or CUIs they know the I-id of. This restriction is in accordance with localization of information to CU and UM instances.

Whereas the use of I-ids explain how the CUIs are connected among themselves, it does not fully explain the com-

munication between the CUIs and UMIs, since the UMIs have a simpler connection mechanism. The CUIs can send messages to the UMIs using their I-ids, and decide on which inlink the UMI's customized behavior will receive the message from. In order for a UMI to be able to send a message to a CUI c, one outlink of the UMI must previously been setup (by a CUI) to send to c. Thereafter the UMI can communicate to c by directing messages to this particular outlink.

B.2 Behavior of CUs

Like UMs, the behavior of CUs are also left blank to be customized by the simulator model builder. The CUs with customized behavior serve as the set of types the CUIs will be instantiated from.

The only restriction on the behavior of the CUs is that CUIs work with the local information they have, and use only the set of services provided by AMINES-HLA to interact with other CUIs and UMIs. The CUIs can create or delete UMIs and CUIs. They can also create or delete links between two UMIs or UMIs and CUIs. Therefore CUs can construct and transform the run-time representation of the TNSim, and hence the SMF and MoTN.

The logical structure of a CUI is shown in Figure 2, is similar to the structure of UMIs. The points of service and callback are again provided through ambassador entities. The main difference between the logical structure of the CUIs and UMIs is the message flow paths into and out of the instances.

B.3 Services Provided to CU Custom Behavior

There are additional types defined in describing the services and callbacks related to the CUIs. Every instance of a UM or a CU in a given simulation executable model has an id that is unique in that model. The type of these ids is defined as the *I-id* (instance-id). There are two subtypes of I-id: *UMI-id* is the type of the ids of UMIs, and *CUI-id* is the type of the ids of CUIs. Given an I-id, it should be possible to deduce whether it is a CUI-id or a UMI-id.

In addition to these id types, three other types are needed to be able to refer to customized behavior ambassadors: UMI-CB-Ambassador, CUI-CB-Ambassador, and CUI-Base-Ambassador. They represent the types of UMI custom behavior, CUI custom behavior, and the CUI-Base ambassadors respectively.

The following services are provided by the CUI-Base to the CUI-CB:

sendMessageToUmi: Request to send a message to an UMI. The UMI-id of the target UMI, the link-id of the inlink that the message will be received from at the target UMI, and the message should be provided. There is no return value. As a result, a receiveMessage callback will happen at the target UMI.

sendMessageToCui: Request to send a message to a CUI. The CUI-id of the target CUI and the message should be provided. There is no return value. As a result, a receiveMessage callback will happen at the target CUI.

createUmi: Request to create an UMI. An UMI-CB should be provided as a parameter of type UMI-CB-Ambassador. The return value is the UMI-id of the created UMI. As a result, a UmiBaseCreated callback will happen at the created UMI.

createCui: Request to create a CUI. A CUI-CB should be provided as a parameter of type CUI-CB-Ambassador. The return value is the CUI-id of the created CUI. As a result, the CuiBaseCreated callback will happen at the created CUI.

linkUmiToUmi: Request to create a link between to UMIs. The link to be created is described by four parameters provided: a UMI-id and outlink's link-id at the initiating side of the link, and a UMI-id and inlink's link-id at the ending side. There is no return value.

linkUmiToCui: Request to create a link initiating from a UMI and ending up in a CUI. The UMI-id and the outlink's link-id for the initiating side of the link, and the CUI-id for the ending side should be provided. There is no return value.

unlinkUmiFromUmi: Request to delete a link between two UMIs. The link is described as in linkUmiToUmi service. There is no return value.

unlinkUmiFromCui: Request to delete a link between a UMI and a CUI. The link is described as in linkUmiToCui service. There is no return value.

deleteUmi: Request to delete a UMI. The UMI-id of the UMI to be deleted is provided. There is no return value. As a result, the links of the given UMI are deleted, and the resources used by the UMI-Base are released.

deleteCui: Request to delete a CUI. Similar to deleteUmi, but works for CUIs.

replaceUmiCbAmbassador: Request to replace the UMI-CB ambassador of a UMI. The UMI-id of the UMI whose UMI-CB will be replaced, and a new UMI-CB as a parameter of type UMI-CB-Ambassador should be provided. This service is intended for use in model replacement and simulator interoperability [7].

receiveMessage: Callback indicating that a message has been received. The I-id of the UMI or CUI the message was received from and the message is provided. No return value is expected.

CuiBaseCreated: Callback indicating that the CUI has now an associated CUI-Base. The CUI-Base is provided as a parameter of type CUI-Base-Ambassador. No return value is expected. A CUI-CB receives this callback only once during its lifetime.

IV. Conclusion and Future Work

In this paper, we have discussed the elements and stakeholders of network simulation, presented some hypothesis on network simulator design, and described our design of the AMINES high level architecture for network simulators.

We believe the benefits of having a high level architecture for network simulators justifies the cost of developing one and conforming to it. A high level architecture for network simulators would enable model reuse across different network simulators at the executable model level. A high level architecture that supports transparent distribution of basic architectural elements, would help us to harness the

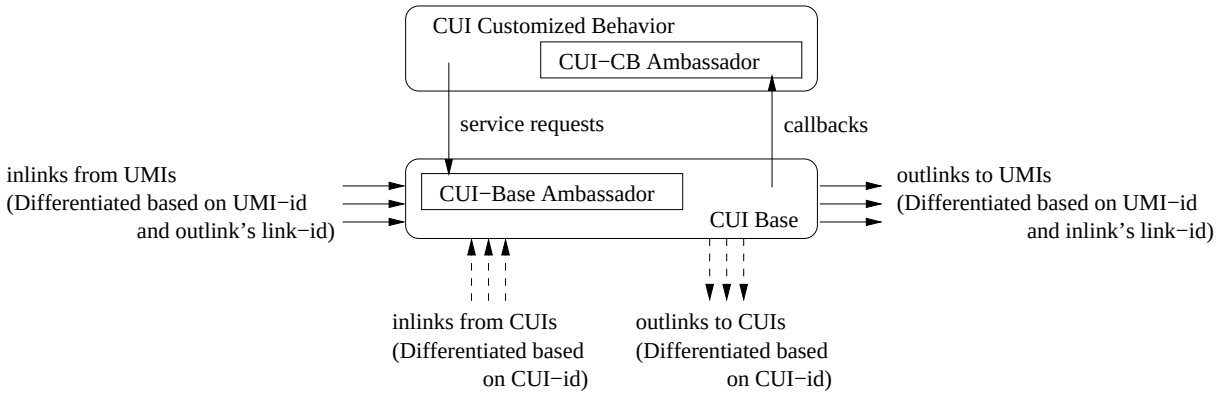

Fig. 2. Logical structure of a CUI.

power of the coming multi-core processors. In addition, a clear high level architecture that is sufficiently similar to the experimenters' own perception of networking systems, would provide an easy to learn and easy to use base architecture that is shared across different network simulators.

The discussion we have presented in this paper, on elements and stakeholders, contributes by providing the terminology for the discourse on network simulator architectures. By defining and clarifying borders between the elements, and by identifying the stakeholders and their goals and interests, it provides grounds for our argument in favor of having a high level architecture for network simulators. The AMINES-HLA demonstrates one possible design of a high level architecture for network simulators.

One conclusion from our work on the design of AMINES-HLA, is that it is a feasible attempt to design a high level architecture for network simulators. Providing help to the developers to satisfy the hypotheses in Section II-C, is one of the major design goals of the AMINES-HLA.

The work described in this paper is ongoing work, and further research directions and problems are many. We are currently in the process of implementing a concurrent run-time infrastructure for the AMINES-HLA. One possible research subject is working out the details of the metamodel assumptions of the AMINES-HLA, in preferably a formal manner. We also would like to explore transparently distributing the CUs and UMs in a simulator built on AMINES-HLA. Another direction we have already started to work on, is to implement existing simulators on AMINES-HLA. For this purpose, we have initiated a project for re-factoring the Ns-2 simulator.

Another question involves an assumption we have made in the design of the AMINES-HLA. The CUs as the only entities in AMINES-HLA that are capable of transforming the run-time representation of the MoTN. Yet, at the same time, we have suggested that they should not be used in MoTNs. The reason why we have separated the CUs and UMs in this manner, is that we doubt the necessity of supporting self-transforming MoTNs, where number and connection patterns of submodels change during simulation. We believe that what actually is beneficial, is to have the ability to dynamically allocate resources to sufficiently self-contained or autonomous elements in the run-time representation of the MoTN.Formulating this assumption in a formal framework is a topic for further research.

ACKNOWLEDGMENT

The author wishes to thank Dr. Ellen Munthe-Kaas and Dr. M. Naci Akkøk for their valuable feedback.

References

[1] *The ns Manual*, Online: http://www.isi.edu/nsnam/ns/ns-documentation.html.

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

[3] D. M. Nicol and J. Liu, "The scalable simulation framework," in *Proceedings of the ACM/IEEE conference on Supercomputing 2001 (CD-ROM)*. ACM SIGARCH/IEEE, November 2001, poster.

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

[5] IEEE, "IEEE standard 1516 for modeling and simulation (M&S) high level architecture (HLA)," 2000.

[6] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*. John Wiley & Sons, 2000.

[7] E. Göktürk, "Towards simulator interoperability and model replacability in network simulation and emulation through AMINES-HLA," Submitted to the Winter Simulation Conference '06, 2006.

[8] ——, "Elements and stakeholders of network simulation," Submitted to the 7^{th} International Symposium on Computer Networks (ISCN'06), 2006.

[9] G. A. Agha, *ACTORS: A Model of Concurrent Computation in Distributed Systems*, ser. Series in Artificial Intelligence. Cambridge, Massachusetts: The MIT Press, 1986.

[10] ——, "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)*, ser. LNCS, J. Billington and W. Reisig, Eds. Springer-Verlag, 1996, vol. 1091, pp. 1–10.

EREK GÖKTÜRK 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 project in the Modeling and Simulation Laboratory, a METU-TSK (Turkish Armed Forces) joint research facility. He has instructed in METU-CES (Continuing Education Center) Cisco Networking Regional Academy during 2002-2004. Since 2004 he works in Dept. of Informatics in University of Oslo, Norway, as a Doctoral Research Fellow. His research interests include emulation and simulation, ad hoc networks, multi-agent systems, and philosophy of computing and information.