Abstract—In this paper, elements of network simulations are identified and defined to provide a clear terminology for discourse on network simulators. Then we identify the two stakeholders involved in conducting network simulation experiments: experimenters and network simulation system or library developers. We also discuss the goals and interests of these stakeholders to conclude that their role should not be mixed when realizing simulation experiments. We continue by detailing the experiment process of the experimenter. Building on the elements and stakeholders discussed, we present some hypotheses about network simulation system and library design and development. We believe satisfaction of these hypotheses would lead to network simulation systems and libraries that are easier to use, to a process that is easier to follow for the experimenter, and thereby to better quality experiments.

I. INTRODUCTION

Simulations have been one of the main methods for understanding, tuning, and therefore designing network protocols for a long time. As network technology changes, the models used in simulations also change. Therefore there is a constant demand in the networking community for creation of new models of protocols or systems, to be used along with particular simulation systems that are being used.

Network simulations also have attracted more and more researchers from the simulation community in the last decade. 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 network the 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. By that means, the wireless technologies have changed the processing power needs of the models employed in network simulations and effected the architectures of the simulators.

There are two characteristics of network simulation experiments that are not normally found in any other simulation domain. One is the mixed role of the network researcher as developer of the model, the implementer of the model in the simulation system, and the experimenter, whereas in most other simulation domains the user strictly has the role of the experimenter. This has lead to network simulations being built by collaborative efforts of network researchers, which lead to particular problems in the simulation systems developed.

The other distinguishing property of the network simulations is that a part of the models used, are models of software entities situated in computing platforms. This results in blurred borders between the actual software used in the network systems being modeled and their models used in simulations. This similarity between the model and what is being modeled greatly increases how feasible using network simulations in emulation settings is perceived from the network researchers’ perspective.

In this paper, we start by identifying a terminology for discourse on network simulation, by analyzing the network simulators and the experimentation process.

In Section III, the stakeholders involved in conducting network simulation experiments are identified, in order to clarify their roles and provide the necessary grounds for discussing the degree of compatibility between the goals of these stakeholders. Furthermore, this also prepares the background for discussion of two points: the expectations of the stakeholders from each other, and assumptions made when designing and building simulation systems and libraries, about the process and roles of the stakeholders.

We present the experimentation process from the experimenter’s point of view in Section IV. This serves two purposes: the first is to demonstrate the plausible set of actions and their sequence based on the goals of the experimenter, in order to provide the framework for identifying non-realizable assumptions made by the developers when designing network simulation systems and libraries. The second purpose is to describe the place network simulation systems and libraries fit, from the experimenter’s perspective, taking the experimenter’s goals and interests into account.

Building on the analysis of the elements and stakeholders, we give in Section V a list of hypotheses aimed at decreasing the chance of a conflict between the goals of the stakeholders. We expect that realization of these hypotheses would lead to an increase in usability of network simulation systems and libraries, and increased experiment quality.

It must be noted that our main focus is the development of network simulation systems and libraries, and their integration into the network researcher’s experimentation process. We are interested in realizing these in a way that maximally supports the goals of the experimenters, by correctly identifying the role that must be assumed by the system developers and characteristics that network simulation systems and libraries should have. This paper serves this interest by providing the foundations for a discourse on network simulation, for our ongoing investigation into network simulator...
Giving precise definitions about what a model is and properties of models involves complex philosophical discussions. Hardware and software components are not always easily distinguished, since a considerable amount of hardware uses software internally. The definitions provided in this section are meant to refer to some statistical adequacy of the results for the phenomena modeled by a particular experiment. A simulation library (SimLib) is a library of helper routines for implementing SMF in TNSims. The simulation libraries usually come with an executable model library, and impose certain software specifications onto both the SMF and MoTN parts of the TNSim. Since SimLibs usually come with a model library, they are adequate to build a TNSim except for the executable submodels that are to be custom built for a particular experiment. A simulation library can also be used to build a simulation system.

A. Target network (TN)

The phenomena under study in a network simulation experiment concern a particular network, which exists as real or as a design. Such a network we call a target network (TN). The TN is characterized by its hardware and software components, and external influences which form the workload specification. It should be further noted that hardware and software components are not always easily distinguished, since a considerable amount of hardware uses software internally.

B. Model of target network (MoTN)

The TN is represented in the network simulation experiment as a model. In the context of a simulation, a model can be defined as a description of the target phenomena. Giving precise definitions about what a model is and properties of models involves complex philosophical discussions. Therefore the definitions provided in this section are meant to provide an idea about our perspective, to establish some important terminology, and not meant to present a complete theory of modeling.

The models of target networks (MoTNs) that are subject of interest, are generally composite models. A composite model is a model which is constructed by putting together some models by allowing them to interact. With respect to a composite model, we will refer to the models composed in it as submodels of that composite model. The composite MoTNs of interest are generally not amenable to analytical analysis, therefore stochastic analysis, or simulations, are used to investigate their behavior.

Models can be attributed different characteristics, such as detailedness, precision, and accuracy. Precision and accuracy can be defined with respect to the difference between the results produced by models and actual measurements taken about the target phenomena, whereas providing a definition for detailedness is harder. Usually detailedness is regarded the same as complexity, however the relationship between complexity and detailedness, precision or accuracy is not as straightforward as to be easily expressed as a simple function. We can also talk about adequacy of precision, or adequacy of accuracy of a model with respect to a particular experiment to refer to some statistical adequacy of the results for the particular analyses planned for the experiment.

One can also define comparative relationships between models. 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 $M_{C2}$. Decreasing the granularity of a composite model is one way to manage the complexity introduced as more details are added to the model, for hopefully increasing precision and accuracy.

It should also be noted that we use metamodels or paradigms to prepare models, and therefore some entities that do not seem to have a direct relationship with the target phenomena, find their way into the models we use. One such type of entity that provides a good example, is the scheduler used in discrete event simulations. While a scheduler is usually regarded as part of a discrete event simulation system, it actually is part of the model which is prepared using the discrete event paradigm in mind, and it helps us to model causal relationships.

C. Executable model

Not all models are prepared as computational machinery. Those that are prepared as software entities we will refer to as executable models. When an executable model is to be used as part of an executable composite model, it should be designed and implemented taking into account the software specifications of the target network simulator (see below), which are in turn derived from the specifications of the simulation system or library being used.

D. Target network simulator (TNSim)

Target network simulator (TNSim) is the simulator whose software specifications, which consist of architecture and functionality, are derived from a given simulation system or library and a given MoTN. The TNSim is composed of an executable, generally composite, MoTN, and necessary software parts for running and managing the simulation, which will be referred to as simulation management functionality (SMF). If the MoTN to be used in a specific TNSim was not executable to start with, it has to be transformed into an executable MoTN during the experiment process.

E. Model library (MoLib)

A model library (MoLib) is a library of ready-made models, and an executable model library is a library of ready-made executable models. Executable MoLibs are designed and implemented in accordance with the software specifications, especially the architecture, of a particular simulation system or library.

F. Simulation library (SimLib)

A simulation library (SimLib) is a library of helper routines for implementing SMF in TNSims. The simulation libraries usually come with an executable model library, and imposes certain software specifications onto both the SMF and MoTN parts of the TNSim. Since SimLibs usually come with a model library, they are adequate to build a TNSim except for the executable submodels that are to be custom built for a particular experiment. A simulation library can also be used to build a simulation system.

G. Simulation system (SimSys)

A simulation system (SimSys) provides ready-made SMF functionality to be extended with an executable MoTN for building a TNSim. Like SimLibs, the SimSyses also usually come with an executable model library, and also imposes certain software specifications onto MoTN used in the TNSim.
H. Metamodels and specifications

There exists both a distinction and a parallelism between the metamodel used by the model builder to build the MoTN and the software specifications, especially architectural specifications, imposed by the simulation library or system onto the TNSim, and therefore MoTN. Although the MoTN does not necessarily have to be prepared as an executable MoTN to start with, it has to be converted to an executable MoTN at some point in the development of the TNSim. Therefore, from the perspective of the simulation library or system developer, it falsely appears as a plausible strategy to expect the model developers, whose main concern is the realization of a specific experiment, to use relevant parts of the software specifications imposed onto the TNSim as a metamodel for the MoTN. There is reason to partly support this approach as a coherence between the metamodel used for MoTN and the software specifications imposed by the SimLib or SimSys is ultimately useful for the experimenter to be able to describe the model in a way that he or she generally describes a model in his own day-to-day research. However, the author believes that it should not be the software specifications that the metamodels are built from, but the software specifications of the SimSyses and SimLibs should be derived from an easily understood and adequate metamodel for preparing MoTNs. In our current ongoing research, we are evaluating the actor model ([1], [2]) as a candidate for such a metamodel, and attempting to build an architectural basis for the software specifications of a SimSys or SimLib that is in agreement with the metamodel provided by the actor model.

III. Parties Involved

Two different parties are involved in realizing a network simulation experiment: the experimenters, and the simulation system or library developers. Although these two parties have different interests and goals, and follow different steps of actions before and during an experiment, today it is general practice, especially for open source systems like Ns-2 [3], for researchers to assume both roles without paying attention to the differences. As discussed in this section, the differences are subtle and goals of different parties seem to induce resource conflicts. One immediately apparent resource subject to such a conflict, is the amount of available time.

A. Experimenters

The experimenters are the researchers who are interested in doing experiments that are aimed at understanding behavior of networks under particular situations and workloads.

An experimenter has two major goals with respect to simulation. The first is the goal of minimum effort, which can be described as the wish to design, implement, and run a suitable experiment with minimum resources, especially time. The other goal is to build and realize the experiment in a manner that will produce meaningful data for the analyses planned to be carried out. This second goal will be referred to as the goal of adequately meaningful data.

The sequence of actions the experimenters take when conducting a network simulation experiment, is roughly described below, and will be further discussed in section IV:

1) Decide on a network simulation system or library to use.
2) Build the SMF if a simulation library is being used.
3) Build the MoTN. If the model library that comes with the chosen simulation system or library is not adequate to compose the MoTN, the experimenters have to implement additional submodels in accordance with the software specifications of the chosen simulation system or library.
4) Put together the SMF and MoTN to build the TNSim for the experiment. Decide on data to be collected and design and implement logging mechanism. The data logging mechanism is one of the major connections between the SMF and the executable MoTN in the TNSim.
5) Prepare initial conditions for the model, and workload to be injected into the simulation at decided virtual times.
6) Run the constructed TNSim.
7) Interpret and analyze the collected data in accordance with the statistical properties of the TNSim, initial conditions, and injected workload in order to derive results.

B. System and Library Developers

The other party to network simulation experimentation is the developers of the simulation systems and libraries. These developers have the following as goals:

- Provide a system that impose specifications on the executable MoTNs that are in fine accordance with the metamodel the experimenters use to prepare the MoTNs.
- Provide a model library with basic models of conventional network hardware and software that the experimenters can use when building the MoTN in order to minimize the efforts required to implementing a limited set of submodels directly related to their research.
- Design the software specifications of the simulation library or system for good performance, and optimize implementation and interactions of basic models provided in the model library.
- Keep it simple but adequate, and document the system well.

The system and library developers do the tasks described below for attaining these goals:

- Prepare base software specifications for the TNSims to be built using the system or library. For satisfying the first goal mentioned above, the software specifications should be in accordance with a metamodel that is sufficiently similar to tasks done by the networking experts for building networked systems.
- Decide on how the specifications can be mapped to programming constructs. This mapping should conserve the architectural style of the metamodel that is to be reflected in the software specifications in the first step.
- Design for clean interfaces to be used by experimenters when building and composing submodels.
- Implement with clean and understandable code, with enough comments. It is preferable to use an open-source approach, since the code is the most precise description
of what the system does. Therefore opening the source code supports the fourth goal.

C. Current state

When we look at today’s two mainstream open-source network simulation systems, Ns-2 [3] and GloMoSim ([4], [5]), we see that the model granularity is roughly fixed by the system. Both systems provide some amount of architectural specifications of how the submodels should be composed, and a MoLib, and both depend on being extended by the experimenters providing implementation of reusable submodels, such as of protocols. However, as can be observed for Ns-2, due to a lack of adequate restrictions on how entities will communicate, and a non-uniform mechanism to compose the submodels, the code of the SimSys and MoLib can become at the very least confusing, difficult to understand, and hard to extend.

The commercial systems provide a much clearer separation between the two stakeholders, since they depend on this double stakeholder structure to create a network simulation systems and libraries market. The downside is that academic studies is hindered because the code and the algorithms are closed and protected under copyright. In addition, the experimenters usually have to live with the granularity and structure provided by the commercial product.

IV. EXPERIMENTER’S PROCESS

In this section, the steps carried out by the experimenters described in section III will be further discussed.

The seven steps previously given can be divided into three periods: pre-run, run-time, and post-run. The pre-run period covers steps 1-5, the run-time period is step 6, and the post-run period is corresponds to step 7.

A. Pre-run period

In the pre-run period, the experimenter designs the experiment and prepares the TNSim. The design of the experiment involves drawing boundaries of the TN, deciding on the metamodel to be used to prepare the MoTN, and constructing parameter domains according to statistical properties of the MoTN and the analyses planned to be carried out. A scenario is developed describing a particular situation of interest involving the TN, initial conditions, and workload injections to be made at specific virtual time instances during the simulation of the MoTN. In addition, what data is to be collected on what events are planned. To prepare the TNSim, a SimSys or SimLib is decided upon, and SMF is implemented if a SimLib is to be used. Note that the choice of the metamodel in preparation of the MoTN and the choice of the SimSys or SimLib is not completely independent, as discussed in previous sections. Having decided on a SimSys or a SimLib, the MoTN is to be converted to an executable MoTN, which should conform to the software specifications imposed by SimSys or SimLib, using submodels provided in MoLibs and implementing new submodels when necessary. Then the executable MoTN is put together with the SMF provided by chosen SimSys or implemented using chosen SimLib, to construct the TNSim. Two particularly important considerations when putting together the SMF and executable MoTN, is flow of data to be logged from MoTN to static storage through generic functionality in SMF in an efficient manner, and mechanism to construct, transform, and destroy the run-time representation of the executable MoTN.

B. Run-time period

The run-time period covers the time the TNSim is being executed. Like all software, the TNSim’s stored form and run-time representation are most probably different. Particularly, this implies that the executable MoTN exists in stored form as some number of submodels and a description of how instances of these submodels are composed at runtime to arrive at the actual executable MoTN. Therefore before anything, a run-time representation of the SMF and executable MoTN should be constructed using such stored forms. Dual programming language systems such as Ns-2 provide scripting language interfaces to allow the stored representation of the executable MoTN and how the run-time representation will be built to exist as scripts, drawing upon the hypothesis that scripting languages are easier to understand and use for such purposes.

When the run-time representation is set up, the executable MoTN is run, and events related to the analyses planned to be carried out after or during the execution are monitored and/or logged. The analyses that are carried out during the execution are usually computationally light and for the purpose of summarizing the data for efficient logging or monitoring the experiment for interrupting timely if there is a problem in the MoTN or TNSim.

The execution of the MoTN is stopped when some predefined event happens. Normally, at this point the run-time period ends and the process continues to the post-run period. However, it is also possible, and not so unusual, to make changes to or to transform the executable MoTN by introducing, deleting, or reconfiguring some submodel instances while keeping the internal states of others intact, and execute this partly new executable MoTN until some other predefined event happens. Note that what is being changed is the structure and configuration of the original executable MoTN, and the TN or MoTN does not necessarily change as a result of the changes made in the executable MoTN. In particular, the TN should be kept constant throughout the experiment since the experiment is defined for evaluating a particular TN, and changing the TN changes the experiment. Also, changing the MoTN might have effects on how the experiment is organized and the validity of the results. Moreover, it is doubtful that allowing for changes in the MoTN or executable MoTN during the run-time period would really extend the expressive power of any metamodel used for MoTNs, or SimSys or SimLib specifications used for executable MoTNs.

C. Post-run period

The post-run period is when the analyses are carried out on the data gathered in run-time period to derive conclusions on the characteristics and behavior of the TN under conditions simulated in the experiment. Depending on the results obtained, it may be necessary to re-conduct the experiment with a modified scenario, model, or TNSim. In any case, the activities described above should be re-done or the results of activities that are not re-done should be checked for validity in the new experiment.
D. Separation of periods and activities

Clear separation of the periods and the activities in each period should provide better separation of concerns. The separation in the pre-run period can be done in terms of design-time, that is, the times the activities in pre-run period are done should not overlap. Separation in the run-time period can be accomplished in terms of execution-time, that is, the activities of setting up run-time representation, running executable MoTN, and optionally transforming or modifying the executable MoTN are executed non-concurrently. Another mode of separation for these activities in the run-time period should be done in terms of location of code, by ensuring that these activities are accomplished in separate modules. Separating the post-run period from the rest of the experiment ensures that the data collected is consistent. It should be noted that the main concern in pre-run period activities and optional transformation or modification of executable MoTN activity in the run-time period is arriving at a correct—in some sense depending on the experiment—executable MoTN, whereas the main concern in the rest of the activities in run-time is efficient and correct execution of the MoTN, and the concern in the post-run period is deriving results and conclusions correctly.

Ns-2 provides us with an example. The Ns-2 system does not separate the activities in the run-time period in terms of location of code: Although initiated from a script-based description of the executable MoTN, the code for constructing the run-time representation is distributed to the submodels provided as the MoLib, which sometimes even use global variables to communicate during set-up, and the executable MoTN can be modified by any submodel code. Furthermore, separation in terms of execution time is only due to the fact that Ns-2 uses single-thread execution, therefore no other code may be executing while the model is being transformed or modified.

V. Hypotheses

In the previous sections, an analysis of elements and stakeholders in network simulation is presented. Drawing upon this analysis, we will set forth in this section some hypotheses about network simulation that should hold for better quality network simulation experiments. It should be noted that the stakeholders that should mainly care for all these hypotheses are the system and library developers, and not the experimenters.

A. Different stakeholders (H1)

The first hypothesis states that the experimenters and system and library developers, are two different stakeholders in network simulation, with sufficiently different interests and resources that necessitate differentiating the two. Two particular derivations of this hypothesis is that no network simulation experiment should depend on experimenters understanding the SimSys or SimLib as a whole, and no SimSys and SimLib should be organized around the assumption that the experimenters will produce reusable extensions. Both have adverse effects on the learning curve of the experimenter, the former by increasing the amount of information to be learned, and the latter by creating code that is badly structured, hard to manage, and difficult to understand. Ns-2 provides an example of the adverse effects produced when the different stakeholders hypothesis is not honored.

B. Metamodel and expert’s knowledge in agreement (H2)

From the perspective of the experimenter, the metamodel is a tool to be used when constructing a model, which in turn is a description of how the experimenter perceives a certain phenomenon. In the case of network simulation, this phenomenon is a target network. Although it is a hard-to-test cognitive hypothesis, the better the agreement between how the experimenter perceives the networks of interest through his/her personal or expert knowledge and the metamodel he/she uses to document this perception, the easier it should become for him/her to document, and the less error-prone the process of modeling should become. Depending on how good this agreement is, the metamodel incurs an additional learning load on the experimenter, who will try to avoid high learning loads that are outside his/her research area. This means that the metamodel should be as clear and as simple as possible. Due to these reasons, one important task of SimSys or SimLib developers is to determine a suitable metamodel and base their system or library on it. These ideas parallel those presented in [6] and [7] on modeling, from a software engineering point of view, and the role of “conceptualization” or “paradigm” in building executable models.

C. Metamodel and detailedness of submodels (H3)

Sometimes, it happens that details of some parts of the TN are not directly relevant to the experiment, but they have to be present in the MoTN. Modeling such parts with basic models that produce statistical approximations, is a useful and usual technique employed in network simulations. From the terminology introduced in this paper, this technique leads to composite MoTNs whose submodels are of different detailedness. Therefore, the metamodel should be capable of expressing such composite MoTNs adequately and without unnecessary difficulty.

D. Minimal requirements on executable models (H4)

The metamodel should impose only a minimal set of requirements on the MoTNs if the MoTNs it describes are executable MoTNs. This is in order to minimize the risk of a conflict between the software specifications imposed by a SimSys or SimLib onto the TNSim, and the resulting executable MoTN. If the metamodel does not describe executable MoTNs, a metamodel that does so should be derived from taking both software specifications imposed by SimSys or SimLib of choice, and the requirements imposed by the metamodel into account. Since such a derived metamodel is expected to be reused in subsequent experiments, it should be thoroughly documented.

E. Efficiency of implementation (H5)

The executable MoTNs constructed using the metamodel or a derivation of it (see H4), should be implementable with adequate efficiency. Although adequate efficiency is hard to define and might change from experiment to experiment, the system and library developers have to assess and consider this property when choosing a base metamodel for their systems.
The executable MoTNs constructed using the metamodel or a derivation of it (see H4), should preferably be transparently distributable over some number of communicating computational platforms, and should be able to run concurrently. The transparency here is from the perspective of the experimenter: the experimenter need not bother with the details of how the resulting model will be distributed or run concurrently. Note that the concurrency here is about how the executable MoTN is executed, not about any concurrency or distributed computation in the TN.

This hypothesis follows from two observations: The first one is that distribution or concurrency is necessary for interfacing emulated entities with the simulations, and some amount of transparency is needed for reusing submodels developed for simulations in an emulation context [8]. The second observation is that there is a significant trend towards parallel systems and multi-core processors, therefore the network simulation systems will have to take advantage of concurrency and distribution to increase the performance of TNSims in order to support experiments in a scalable way, while the experimenter, who is not necessarily an expert on concurrent systems, should be spared the complexities of designing MoTNs that take advantage of concurrency.

G. Adequacy of homogeneous simple elements (H7)

This relatively more speculative hypothesis draws upon the adequacy of the actor model and object oriented approaches for expressing any practical computation. It states that 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 mentioned above. Furthermore, since most network simulators in use today are written using an object-oriented approach, the object-oriented approach presents one metamodel that can be used to refactor these network simulators in a way such that they can be used in conjunction, and our current research involves searching for models better than the pure object oriented approach that would also be used for such refactoring of existing simulators.

VI. Conclusion and Further Work

In this paper, we identified elements of network simulation by defining a terminology, and identified the stakeholders involved. Then we looked more closely into the experimenters’ experimentation process, so as to show where the simulation system or library must fit and provide background for analyzing the feasibility of assumptions made about stakeholders when designing simulation systems or libraries.

We have also listed some hypotheses that should be considered as part of our conclusion. Although aimed for the developers, these hypotheses should lead to simulation processes that are easier to follow for the experimenters. As a whole, this paper presents our observations and assumptions about the context our research on network simulator architectures is being conducted in.

As further work, we are currently investigating the suitability of the actor model like models as metamodels. We are also investigating into architectural specifications that might be imposed on the target network simulators by network simulation systems and libraries when such metamodels are used. Testing the validity of the hypotheses presented is another research direction this paper points to. Such a research would be mostly empirical, and, at some points, would heavily depend on a sound philosophical theory of modeling.

ACKNOWLEDGMENT

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

REFERENCES