# Integrating ns-3 Model Construction, Description, Preprocessing, Execution, and Visualization

Peter D. Barnes, Jr. (pdbarnes@llnl.gov), Betty B. Abelev (abelev@llnl.gov),
Eddy Banks (lebanks@llnl.gov), James M. Brase (brase1@llnl.gov),
David R. Jefferson (drjefferson@llnl.gov), Sergei Nikolaev (nikolaev2@llnl.gov),
Steven G. Smith (smith84@llnl.gov), Ron A. Soltz (soltz@llnl.gov)
Lawrence Livermore National Laboratory
7000 East Ave.
Livermore CA 94550 USA
1-925-422-3384

## ABSTRACT

There are many examples of meta-languages to describe computer network models, experiment designs, data formats, and visualization parameters. It is clear that the field recognizes the importance and benefits of a concise, portable and precise problem description, abstracted from the native representation used by any specific modeling or visualization framework. Yet none of the prior efforts have achieved widespread adoption. In this paper we discuss some of the issues in developing a model abstraction of wide utility. We survey the various communities that would benefit from such an abstraction, a range of use cases, and some of the semantic difficulties encountered. We propose a path forward to address the needs of various parts of the ns3 ecosystem, in a way that can be generalized to other network simulation frameworks and communities.

## Keywords

Network simulation, ns-3, modeling and simulation, System modeling languages.

## 1. INTRODUCTION

One of the first difficulties encountered by any serious modeler is describing the model being simulated, in an accurate yet concise way. This is important, not only for the modeler to keep track of which variants have been run, and the results, but also for communicating the nature and details of the model to others. Precise descriptions of the simulation model, the measurement points, and the analysis algorithms are essential to enable others to reproduce and extend the claimed results.

In the case of computer network simulations, this can be particularly challenging because of the many conceptual levels involved, each of which has its own terminology and parameters. Starting at perhaps the most abstract level, one has to specify the topology, that is, the pattern of connections between computer nodes in the network, and the assigned addresses. Second, one has to specify the parameters of the connecting media (link bandwidth, latency, error rates, access control, *etc.*). Even restricting ourselves to well-developed and stable wired standards, we have to specify the signaling protocol, *i.e.*, that the link is an Ethernet-type channel, as opposed to, *e.g.*, token ring. Wireless channels have even more variety, in signaling protocols and physical channel description, perhaps including propagation effects such as noise, interference, multi-path due to the local environment, *etc.* Third, at each connected node, one has to specify the parameters of the network interface to be simulated (they need not be the same as the physical medium!).

At the protocol level, we have to specify that the nodes will be using TCP/IP protocols (not, *e.g*, DecNet), and which variant of TCP congestion control. For forwarding nodes we have to specify the queuing discipline, and size of internal buffers.

Moving further up the OSI stack, we have to specify the routing tables and routing behavior at each forwarding node. Each routing protocol and implementation has its own configuration that has to be provided. Finally we have to specify the behavior and performance of the applications that will actually generate and consume the simulated network traffic.

After all that, we have only specified the model to be simulated. What is to be measured? We have to specify each parameter to be monitored, as well as any data treatment/summarization to be applied, such as averaging, windows, recording intervals, bounds and bins for histograms, *etc*.

For parallel simulations we may need to specify the number of compute nodes to be used, the mapping of model elements to the compute nodes, and perhaps a checkpoint/restart or dynamic load balancing strategy.

Finally, since no single simulation run is of much use, we need to describe an *ensemble* of runs that constitute a simulation study. How many identical runs are needed for statistical analysis in an ensemble? How are the parameters varied between ensembles? Where in the file system are the inputs and outputs stored? What random seeds are used? What are the simulation times of termination? What resource estimates are to be applied? How are failed runs handled? What ensemble data is collected, and how is it to be summarized or visualized?

Clearly, creating a precise and comprehensive specification for a specific model and measurement is a challenging task. The benefits of a portable specification would be widespread. It would be trivial to exchange models with others with confidence the models would run. It would enable comparisons between simulation frameworks on the same problem, as a benchmark. It would make possible precise studies of the impact of new protocols and technology, even when the required model component is only implemented in a single simulator. One would use the existing model of interest in the new simulator, instead of


WNS3 2013, March 05-07

DOI 10.4108/icst.simutools.2013.251756

porting (or waiting for a port) of the required component protocol to the original simulator.

What's somewhat surprising is that every new simulation framework appears to have tackled this problem anew. Why has no specification spread beyond its framework of birth? In this paper we discuss some of the reasons and possible solutions to this conundrum. In Section 2 we discuss prior work in this area, particularly calling out features that subjectively appear to have widespread utility. Section 3 discusses some of the conceptual and semantic challenges in developing portable solutions. Section 4 discusses the communities and use cases for portable model descriptions, and specifically the implications for requirements. Based on Sections 2-4, we develop a list of desirable features in Section 5, and outline a possible architecture in Section 6.

## 2. PRIOR WORK

Prior work in this area is abundant. Practically every network simulation framework has some form of external model representation. Here (in no particular order) we give a subjective survey of some of the more prominent examples, highlighting particular noteworthy features.

OpNet models are primarily constructed using the built-in graphical network editor.[1] While the simulation engine itself is proprietary, the published API has enabled third parties, particularly vendors in the defense space, to build comprehensive libraries of model components, particularly for various proprietary radio communications standards. Models can incorporate predefined subnetworks. OpNet appears to use solely a proprietary file format.

SSFNet is a notable example in several respects.[2, 3] It used plain text files, in an INI-like format, to define models and measurement points.[4] These text files are written in a fully defined grammar, "domain modeling language." The grammar is extensible, by use of "schema dictionaries," which facilitates natural expression of attributes for model elements. The distribution includes a schema checking program, to verify that new schemas conform to the underlying grammar. Allowed values for attributes can be restricted to specific forms using a regular-expression-like syntax. Attributes can be referenced by a path-like expression, and reused in multiple places in a model description. Model elements can be defined in an inheritance hierarchy, allowing elements to be expressed as specializations of previously defined more generic representations. Finally, SSFNet allowed composability and reusability, by defining a subnetwork in one file, and including it by reference into a larger model defined in a second file. A simple and direct addressing scheme allows the same subnetwork topology to be included several times, with different base addresses, in a larger model.

OMNeT++ is simulation framework for network models in the most general sense, with specific problem domains and protocols, *e.g*, TCP/IP, supported by extension libraries.[5, 6] While OMNeT++ has a graphical editor (*via* an Eclipse plug-in), the primary model description mechanism is via the NED (NEtwork Description) language, used to represent models in text files. NED has a complete grammar specification, and tools to transform to/from XML. The OMNeT++ build system effectively compiles NED into C++, and ultimately into executable code. Notable features include modularity and composability through defined interfaces, and inheritance. The use of defined interfaces enables the choice of implementation to be deferred to run-time configuration. The language also supports non-simulation metadata annotations, such as icons and measurement units.

The REAL, ns, ns-2, and ns-3 simulators represent probably the oldest lineage in network simulators. REAL used a loosely specified hierarchical (name, value) grammar in text files to define the model.[7] ns, probably the first general purpose network simulator, used Tcl for scripting and defining models, and C++ for model elements.[8] It's immediate successor, ns-2, is also written in C++, but uses the OTcl dialect for defining models.[9]

ns-3 is a complete rewrite, merging ideas from ns-2, GTNets, and YANS.[10-13] ns-3 can represent models either in C++ source code or Python. There have been two XML projects for defining ns-3 models. NEDL/NSTL was a project to define templated models, as well as parameter surveys using those templates.[14] Network simulation template language (NSTL) is a mashup of C++ model snippets encapsulated in XML. NSTL files are used to create a web form to facilitate model building. A server-side script processes the form to generate a network description language (NEDL) XML file, which describes the model options and parameter sets chosen by the user. The NEDL file is then used to build and execute the simulation on a simulation server, returning results to the user. ns3xml provides a moderately detailed grammar for defining models.[15] It is somewhat limited in requiring handlers for each XML element, a fairly rigid ordering of elements, and mixing visualization and model elements in the same file. At this time, XML is used in a limited capacity in ns-3, to store and read back global configuration options, and as an internal path to communicate topology to the netanim visualizer.

The Extendable Mobile Ad-hoc Network Emulator (EMANE), primarily intended for realtime wireless network emulation, uses several well-defined XML document types to control various aspects of an emulation and/or modeling scenario, including the configuration of the emulation computing platform, the definition of network layers, and monitoring points at each node.[16] The basic XML schema uses (name, value) pairs, with values accessible to model element code via a configuration system. This simplifies the configuration API, at the expense of somewhat verbose XML. The overall schema supports hierarchical inclusion of sub-elements by referencing additional XML files. Parameters defined by included elements can be overridden at the point of inclusion, giving the effect of inheritance. In addition, the EmulationScript schema defines several document types, primarily to describe time-dependent simulation events, such as node mobility, in a modular, composable way.[17] EmulationScript also has some support for modifying node properties, such as interface addresses, with timed events.

NetDMF is a network description language built on top of the eXtensible Data Model and Format (XDMF).[18, 19] XDMF dynamically separates so-called light data and heavy data. Light data is represented directly in XML, while heavy data is represented in external HDF5 files or MySQL database.[20, 21] The format of the data itself is described redundantly in both the XML and the HDF5 file (if it exists). With this approach tools need not be linked with the HDF5 libraries, but can still make use of the format information, while tools generating or consuming large amounts of data can use a high performance data representation and organization.

NetDMF itself extends the XNDF model to describe terrain, physical communication devices and conditions, network protocols and topologies, discrete simulation events, node location and mobility, and packet data. These are generally described with (name, value) pairs. As it is built on XNDF, itself reliant on HDF5, NetDMF is a rather heavy weight in implementation. In addition, it

requires tailored translators to construct models for each target simulator framework.

QualNet appears to use a text-based model configuration file.[22]

ROSSNet is notable as the only optimistic network simulator.[23] ROSSNet models are represented directly in C source code.

RocketFuel, while not a simulator *per se*, is a widely used (though somewhat dated) source of measured network topologies.[24] It is notable for including link weights in its text file format.

From this survey of prior work we can draw some higher level lessons. Existing approaches to describing simulation models have spanned a range of technologies. Proprietary simulators, particularly those primarily based on graphical user interfaces, often use closed binary file formats, which makes describing models to others very difficult. About the only way to precisely communicate a model of any significant size is to share the binary file directly. While this offers specificity, the closed nature of the file format makes it very difficult, if not impossible, to use a model in another simulator, or even another computing platform (unless the vendor also ports their simulator).

Several simulators, notably the ns series (ns, ns-2, ns-3), embed the model description, and the measurement points, directly in source code. While this approach has the benefit of specificity (it's clearly enough specified for the model to execute), it tends to obfuscate the model structure (topology, for instance) within the programmatic idioms used to instantiate the model, using the particular application-programming interface (API) supplied by the simulation framework.

This approach has another insidious effect when it comes to parameterized studies. Typical usage is to configure the model with run-time parameter values supplied on the command line. This creates an external (to the simulation code) burden of documenting the actual values in use, either by the model programmer, in echoing the actual parameters values in the model output, or the user in associating the output files from a specific run with the associated parameter set used to launch the run.

Perhaps the most accessible format type in use has been some type of machine- *and* human-readable text file. For example, SSF's INI-like text format, or the NED files used by OMNeT++. The advantages of this approach are that the files are immediately transportable, and they are human readable (at least for modest sized models), which facilitates understanding, verification, and validation.

The specifics of a text file format impose a grammar. One drawback of this approach (except for XML-based formats) is that the grammar is rarely documented adequately. Often the only authoritative reference is the actual implementation of the parsing code. This is a significant disadvantage in using models in different simulators, since the parsing code often has to be written, and validated, for the new simulator.

XML would appear to offer several advantages over other text file formats.[25] While *ad hoc* and INI-like formats only support a limited hierarchy, the possibilities in XML are almost unbounded. XSD can be used to unambiguously define the supported grammar, separately from any parsing software.[26] Automatic tools exist to check a given XML instance against the XSD, which makes it very easy to validate that the XML model description is validly represented. The availability of XML-parsing libraries in a variety of languages also facilitates adding XML as an input option to many simulators.

# 3. CONCEPTUAL CHALLENGES

There are a number of conceptual challenges that make it difficult to define models in a portable way. First is simple the choice of grammar. In developing a grammar for a particular simulator, it is natural to adopt the terminology specific to that API. This has the advantage of making the grammar immediately familiar to the users of the target simulator, facilitating adoption by that specific community. A second advantage is that it is conceptually simpler to write the parsing code, when the model description has a close mapping to the API.

In tension with the tendency to adopt familiar, if local, terms, is the goal to make the grammar portable to other simulators, and a broader community of users. The adoption of terms or concepts too specific to or limited to a particular simulator becomes a practical impediment to users more familiar with other simulators, since they now have to learn a new vocabulary, and even new concepts for features without direct analogs.

A second axis of difficulty is the tension between generic concepts and terms and specific realizations. For example, ns-3 has the notion of a "node container," whereas other simulators might define this implicitly as "the nodes on this CSMA channel." In mapping from generic concepts to specific realizations, there is plenty of room for (mis)interpretation. If a generic model specification says "shortest path routing," there are many implementations that could be adopted in realizing a specific instance of the model, for example: OpenFlow, OSPF, BGP, nix-vector, or GOD-routing. Some of these even have multiple alternative implementations within a single simulation framework, further complicating the interpretation.

Many of the higher-level protocols have their own *defacto* standard configuration schemes, often by text files with protocol- or tool-specific grammars. To ease interoperation with real deployments, one would like to represent or reference these external configurations from within a portable model description format. A more dramatic example would be the specification of virtual machines that are to be executed as part of a model.

A third axis is the degree of specificity in describing a model. For example, in some models, it may be sufficient to assign node names and IP numbers to a set of hosts in a block fashion; this can be done by giving the starting number and letting the simulation automatically assign sequential numbers to each node. In other cases, it may be important to specify the exact parameters for each node, in order to correspond with physical systems, or protocol configuration files taken from external devices. For some problems it may be sufficient to assign applications to nodes probabilistically, for example to generate representative background traffic; other nodes may have to run fully specified applications.

Another example would be the specification of node mobility by a parameterized generative model (a computationally intensive task) *vs.* replaying mobility traces (an I/O intensive task) from a prior execution (or real world data capture). In some simulators (*e.g.*, NetDMF) it is possible to give the parameters for a generative model, *and* a trace file for the output. The first run of the simulation uses the generative model and saves the result to the trace file. Subsequent runs use the trace file if it exists and matches the supplied parameters.

The description of measurements to be performed on the simulation model has even less commonality of concepts and terms across the various simulation frameworks. Portable

description of measurement points and data treatment is particularly problematic.

## 4. COMMUNITIES AND USE CASES

As if the diversity of concepts and terms between simulators were not enough complication, we must also address the diversity of uses that a portable model description will be adapted to, beyond just executing the model within a specific simulator.

We've already discussed the benefits of a portable specification in: facilitating exchange of models with others, enabling benchmark comparisons between simulation frameworks, and enabling cross-simulator protocol studies.

As an example of non-simulation usage, many frameworks have dynamic visualization tools, as adjuncts to the simulator itself. In addition to the specificity needed by the simulator, a visualizer may need additional parameters and data to render the model, such as node and link colors, node icons, hints for summarizing traffic, how to represent collapsed (visually suppressed) portions of a model, geographic baselines for representing 3D renderings of the model nodes in real space, *etc*.

A portable model description format would enable a new ecosystem of tools for generative models. These could range from standard graph models (Erdös-Rényi, Watts-Strogatz, Barabási-Albert), to network models (HOTNet, hierarchical), to observed topologies (RocketFuel, CAIDA). Investing in generators outputting standard model descriptions would make much more sense than writing topology readers for every simulator.

Another use of portable models is in coupling to external systems, for example in EMANE. One of the issues is mapping simulated traffic to/from the appropriate connection point in a real network.

One possible extension to the model description would be to support external representation of a snapshot of the internal simulation state at one point in simulation time. This would enable checkpoint and restart of simulation runs. The immediate use cases here are to restart a simulation to recover from a hardware crash, or to debug near a particularly interesting (or troublesome) point. Additional use cases would be to support initialization of models after a burn-in period, perhaps with altered element configuration. (For example, initialize a model with static nodes, run an ad-hoc routing algorithm to convergence, then launch with different mobility models.)

For executing a model in parallel, it's usually necessary to partition the model, that is, assign portions of the model to each physical processor. One generic feature of parallel simulators is that not all model elements can be split across partitions or processors. Ideally one would have a stand-alone tool that would parse the portable model description, determine the regions that are not amenable to partitioning for the target simulator, call one of the standard partitioning tools, and finally use the results to assign model nodes to processors.[27, 28] As an additional feature, such a partitioning tool could be directed to rewrite specific regions of the model to enhance its partitionability, for example by replacing two node CSMA channels with point-to-point channels (assuming point-to-point links are partitionable in the target simulator, but that CSMA channels are not). Of course, any such model rewriting implies a trade-off between execution time and model fidelity. Ultimately the user has to be in control of which rewrites are enabled, and where they should be applied in the model.

One (perhaps obvious) advantage of a portable model description is that simulation engines become compile-once, run many. There is no longer a need to modify source code and recompile, just to change the topology. At this point it becomes feasible to provision access to a shared computing cluster *via* a web frontend, but in a more general way than NEDL/NSTL. Users would submit simulation jobs as uploaded model description files, while the cluster would provide simulation-as-a-service, requiring much less overhead in terms of login accounts, user authentication, and compile and disk resources.

## 5. DESIRABLE FEATURES

Based on the previous discussions, we can identify a set of desirable features for a portable model description format:

1. Completely capture all information needed to execute a model in a single model description grammar, and optionally in a single file.
2. Capture all information except certain parameters, which can be completely described in a second instance of the same model description format. (This enables parametric ensemble studies with identical base models.)
3. Describe the measurement points and parameters, as well as basic data reductions.
4. Machine *and* human readable. (Substitution of a sufficiently powerful human interface application would reduce the need for a human-readable representation, but that's too ambitious.)
5. Automatic validation against a completely specified grammar.
6. Specified normal or canonical form. A specified canonical form facilitates comparison and differencing between models which may be very similar but expressed in alternative ways allowed by the model description format.
7. Broadly familiar grammar and terms for (most) network elements and concepts.
8. Generic, easily understood terms for useful modeling concepts such as collections of nodes, measurement points, probability distributions, *etc*.
9. Easy (or at least straightforward) creation of parsers for a variety of simulation frameworks.
10. Extensible, modular grammar, to facilitate incorporation of new model elements.
11. Support for generic stanzas (*e.g*, <parameter name=…, value=…> pairs), for cases where a specific model element hasn't defined the required new grammar (and the simulator can instantiate arbitrary elements from the generic grammar).
12. Composable: partial or full model descriptions can be incorporated as sub elements into larger models, (*cf.* SSFNet).
13. Support for hierarchical model descriptions or inheritance: composable sub-elements can be included with altered (specialized) parameters, (*cf.* SSFNet).
14. Compact, preferably by reference, to reduce duplication of repetitive stanzas.
15. Support for hierarchical model descriptions or inheritance.
16. Portable model API for programmatic model creation (from topology generators; from models expressed in the simulator native representation, just before simulation execution).
17. Controlled (and controllable) interpretation of generic terms. ("'Shortest path first' should be implemented as OSPF in this run.")

18. Incorporate directly (or by reference) external configurations for model elements.
19. Support both automatic (sequential or probabilistic) model element parameters, as well as fully specified parameters, within the same model description.
20. Support generative component models simultaneously with traces from prior simulation runs.
21. Support automatic mapping of other (non-simulation) metadata to model elements, to facilitate uses such as visualization or coupling to external systems.
22. Support parallel execution of models, using simulation-agnostic (but simulation-aware) partitioning tools. Allow simulation-specific model rewrites for partitioning.
23. Execute simulations directly from the portable model description, without code-generation or compilation.
24. Support parallelization constructs, such as mapping of model elements to compute nodes (partitioning), checkpoint/restart configuration, or load balancing. Support for partitioning should allow a single model to be specified as the union across a set of configuration files, to facilitate parallel I/O of the model definition by the compute nodes.
25. Support for describing statistical and parametric studies, referencing a base model, including parameter values and the number of runs for each data point, control of random number seeds, and data reduction.

We believe this feature set is comprehensive in meeting the needs and uses cases for a portable model description format.

## 6. PATH FORWARD

Needing a method to transfer network descriptions between an in-house graph topology tool and our simulator of choice (ns-3), we jumped in and developed a XML-based approach that met our immediate needs. While it provides many of the features on our list (1, 4, 5, 9, 10, 11, 13, 16, 17, 20, and 21), its limitations quickly became apparent. Most glaring is the heavy use of ns-3 terms and concepts (7), lack of simultaneous support for generative parameters and prior traces (18), and complete lack of support for non-simulation metadata (19).

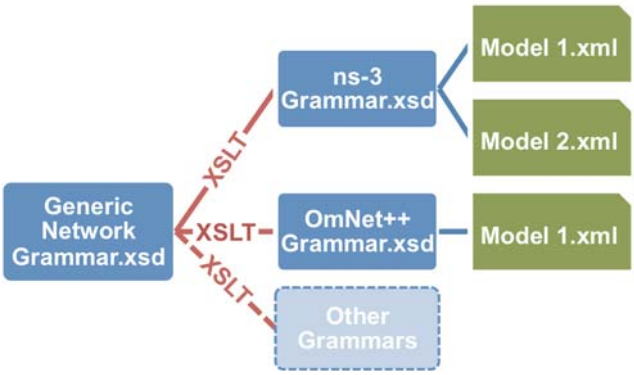


Figure 1. Grammar and instance hierarchy, schematically illustrating the use of XSLT to specialize from a generic network grammar to grammars specialized for different simulation frameworks, with ns3 and OmNet++ as examples. Each XSLT line represents the step of utilizing a specific transformation description (encoded in an XML file, not shown) to transform from one XSD to another. Model descriptions are represented in .xml files, which can follow any of the grammars.

To address these deficiencies, and support the other features listed above, we are developing an architecture based on XML, XSD, and XSLT, as illustrated in Figures 1 and 2.[29]

The use of XSLT enables a much cleaner separation of concerns at several levels, compared to XML alone. First, it allows successive refinement (or abstraction) from a generic network description grammar to simulation-specific grammars, as illustrated schematically in Figure 1. For example, the generic grammar might use the term "bitrate," whereas a particular simulation framework might have adopted the inverse term, "bandwidth." XSLT facilitates automatic conversion back and forth, by rewriting equivalent terms, converting values as necessary. In cases where multiple interpretations exist (such as the "shortest path routing" example given above), the specific interpretation used is encoded in the file driving the transformation (represented implicitly by the "XSTL" lines in the figure). Once the grammar is defined at any level, it becomes possible to declare, and validate, model instances in XML conforming to the grammar defined in the XSD. It also becomes possible to transform automatically a model description in XML in one grammar to an equivalent model description in another grammar.

Figure 2 illustrates two other uses of XSLT. The upper transformation is used to separate the invariant description of a parameterized model from the specific parameter values used in a particular model run. In this use case XSLT is used to substitute the desired parameter values in a coordinated way into the parameterized model description, generating one output .xml file per parameter set. The parameter sets could just as easily be supplied by a database.

One implementation of automatic partitioning can function in an analogous way. The partitioning tool effectively generates an XSLT transformation, indicating how the model elements are to be assigned to compute nodes. Separating the model description from the partitioning assignment facilitates comparison studies of alternative element weighting schemes, which affect the partitioning, as well as alternative partitioning algorithms or tools. Model rewriting, as discussed above, can also be implemented as an XSLT transformation, either in advance of or as part of the partitioning step. Again, this would facilitate comparison studies of the impact of various model rewriting rules, either to quantify the impact, or verify that it is negligible.

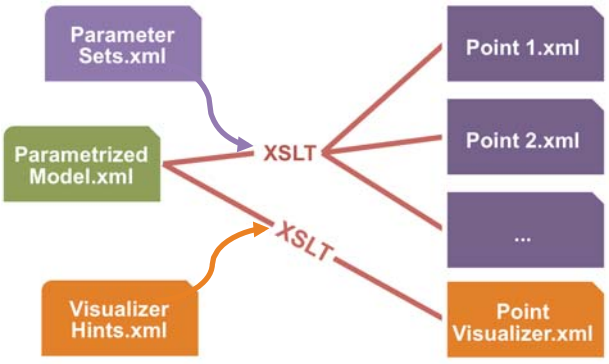


Figure 2. Schematic of use of .xml files to direct an XSLT transformation a) to insert specific parameter values into a parameterized model description, yielding fully specified model .xml files representing specific combinations of parameter values, and b) to rewrite the model description, using a visualizer hints file, into a form usable by a visualizer tool.

As a final illustration, the lower XSLT transformation in Figure 2 indicates how visualizer instructions and hints can be incorporated by an XSLT transformation, again providing separation of the basic model from the additional meta-data needed by a specific visualizer. Multiple visualizers can be supported just be creating the appropriate transformation rules. There is no requirement that XSLT transformations output XML files, so it is quite possible to generate visualize input file(s) directly in native format.

## 7. CONCLUSION

Creating a precise and comprehensive specification for a specific network model and measurement is a challenging task, made more difficult if the description is to be abstracted from the native representation of any specific modeling or visualization framework. Given that the field clearly recognizes the importance and benefits of such a description, as evidenced by the numerous prior efforts referenced above, it seems well worth the effort to tackle this problem anew. We discussed the advantages of an architecture based on XML, XSD, and XSLT In particular, it facilitates separation of the model and measurement description from ancillary metadata required for non-simulation purposes. This allows extensions and adaptations to multiple, parallel consumers, by appropriate transformation of the basic model description.

Our first goals are: to enhance the ns-3-specific grammar we have already defined, in order to deliver the other desirable features; to make the format and toolset directly usable by netanim; and to tie in to NEDL/NSTL-like functionality for managing parameter studies on general models.

Longer term, we aim to expand the supported simulator-specific grammars to NetDMF and OMNet++, which will enable direct model portability between all three simulation frameworks. These seem the easiest to extend to since they already have well-defined grammars and support their own XML model specifications.

For this program to be successful, we have to bear in mind a spanning feature set, which we discussed above. With a XML-XSD-XSLT architecture and the comprehensive feature set we believe a portable and comprehensive model description format for network simulation is within reach.

## 8. REFERENCES


[1] OpNet Modeler, 2012, http://www.opnet.com/solutions/network_rd/modeler.html.

[2] Nicol, D. M., et al., *Simulation of large scale networks using SSF*. 2003.

[3] SSF Research Network, SSFNet, 2005, http://www.ssfnet.org/homePage.html.

[4] Cloanto, *INI File Format*. 2000, http://www.cloanto.com/specs/ini/.

[5] Varga, A., "The OMNET++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference ESM'2001* (Prague, Czech Republic, June 6-9, 2001, 2001), http://http://www.OMNeTpp.org.

[6] Varga, A. and Hornig, R., "An overview of the OMNeT++ simulation environment," in *Proceedings of the the 1st international conference on Simulation tools and techniques for communications, networks and systems* (Marseille, France, 2008). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[7] Keshav, S., REAL Network Simulator, 1997, http://www.cs.cornell.edu/skeshav/real/overview.html.

[8] ns Network Simulator, 1999, http://ee.lbl.gov/ns/.

[9] ns-2 Network Simulator, 2000, http://nsnam.isi.edu/nsnam/index.php/Main_Page.

[10] ns-3 Collaboration, The ns-3 network simulator, Washington, 2011, http://www.nsnam.org/.

[11] Riley, G. F., "The Georgia Tech Network Simulator," in *Proceedings of the ACM SIGCOMM workshop on Models, Methods and Tools for Reproducible Network Research* (Karlsruhe, Germany, 2003). ACM.

[12] Riley, G. F., "Large-scale network simulations with GTNetS," in *Proceedings of the 2003 Winter Simulation Conference* (2003), http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/.

[13] Lacage, M. and Henderson, T., "Yet Another Network Simulator," in *Proceedings of the WNS2 '06 Workshop on ns-2, the IP Network Simulator* (Pisa, Italy, 2006), http://cutebugs.net/files/wns2-yans.pdf.

[14] *NEDL/NSTL*. B.S. Thesis, Bucknell University, Bucknell, PA, 2010, http://redmine.eg.bucknell.edu/safe/wiki/4/Web-based_interface_for_experiment_configuration http://digitalcommons.bucknell.edu/honors_theses/43/.

[15] Riley, G. F. and Pelkey, J., *ns3xml: an XML Experiment Description Language for ns-3*. Barcelona, Spain, 2011.

[16] eMANE, 2010, http://labs.cengen.com/emane/ http://cs.itd.nrl.navy.mil/work/emane/index.php.

[17] EmulationScript Schema Description, 2010, http://pf.itd.nrl.navy.mil/mnmtools/EmulationScriptSchemaDescription.pdf.

[18] NetDMF, 2010, http://netdmf.org/index.php/Main_Page.

[19] eXtensible Data Model and Format (XDMF), 2010, http://www.xdmf.org/index.php/Main_Page.

[20] HDF5, http://www.hdfgroup.org/HDF5/.

[21] MySQL, http://www.mysql.com/.

[22] QualNet. QuallCom, http://www.scalable-networks.com/content/products/qualnet.

[23] Carothers, C. D., ROSSNet. Rensalier Polytechnic Institute, Troy, NY, http://odin.cs.rpi.edu/ross/index.php/Main_Page.

[24] Spring, N., et al., "Measuring ISP topologies with Rocketfuel." *IEEE/ACM Transactions on Networking*, 12, 12004), 2-16, http://www.cs.washington.edu/research/networking/rocketfuel/.

[25] W3C, XML: Extensible Markup Language. W3C, http://www.w3.org/XML/.

[26] W3C, XML Schema. W3C, http://www.w3.org/XML/Schema.html.

[27] Karypis, G., METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering, Minneapolis, MN, version 5.0.2, 2011, http://glaros.dtc.umn.edu/gkhome/views/metis.

[28] Pellegrini, F., SCOTCH. Universit´e Bordeaux I, Talence, FRANCE, version 5.1.12, 2010, http://www.labri.fr/perso/pelegrin/scotch/.

[29] XSLT. http://www.w3.org/TR/xslt.



This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. LLNL-CONF-613593