XML Meets Simulation: Concepts and Architecture of the IBKSim Network Simulator

Lukas Wallentin, Marco Happenhofer, Christoph Egger, Joachim Fabini
Vienna University of Technology, Austria

SNE Simulation Notes Europe SNE 20(1), 2010, 16-20, doi: 10.11128/sne.20.tn.09962

One option to evaluate various aspects concerning the performance of large computer networks is simulation. In order to understand the influence of single parameters onto the overall performance of a network, large series of simulations must be processed. Therefore, automation of simulation series is an important issue in the development of network simulators. In this paper we present the second version of the discrete event based simulator IBKSim. This simulator incorporates the experience of IKNSim which has been developed in 2005 at the same institute. Using XML (Extensible Markup Language) for the configuration and logging function, the new version bridges the gap between usability and automation. In addition to the simulator description we outline our experience with XML as configuration and logging language.

Introduction

Performance analysis and identification of design problems in communication systems can be very challenging due to the size of communication networks, the inherent complexity of network protocols and the influence of the environment. Detailed simulation of communication systems offers an excellent opportunity to perform experiments to gain a deeper insight into their behavior under different conditions.

In this paper we present IBKSim, the second simulator which has been developed at the Institute of Broadband Communications of the Vienna University of Technology. Based on the experience with IKNSim and other simulation environments, IBKSim was developed to meet the requirements of a state of the art network simulator. The motivation was to produce not only a simulator which supports large simulation series but also to facilitate single experiments on networks. As a result XML [1] was introduced as format to describe both, simulation scenarios as well as the simulator's output. The usage of XML differentiates IBKSim from other network simulators i.e. NS2[2] and OMNET++[3]. Another important target was the extensibility of the simulator to allow other developers to use IBKSim as an development and testing platform for own network protocols and algorithms.

The remainder of this paper is structured as follows: Section 1 provides a closer description of the architecture of IBKSim. Section 2 explains the benefits of XML as configuration and logging language, whereas section 3 presents the usage of IBKSim to solve typical scenarios. The paper concludes with a summary.

1 Basic IBKSim Concept

IBKSim is a discrete event based simulator written in C++. The simulator kernel is based on the simulator described in [4] and is pictured in Figure 2. It consists of a basic `simEntity` class for the representation of simulation objects. Simulation objects exchange objects of the type `simEvent` which represents current or future events. Future events are stored in the order of their occurrence in an queue-object of the class `simQueue`. As long as there is at least one event in the simulation queue the simulator controller (`simControl`) removes the next event and executes it on the associated simulation object. The object reacts on this event and eventually submits new events to the simulation queue. The simulation ends if there is either no event left in the simulation queue or if the moment of the occurrence of the next event lies behind a preconfigured point of time.

By using this kernel it is possible to build own simulations by implementing new simulation objects which are derived from `simEntity`. Since the purpose of IBKSim is to simulate networks, many additional modules and functions have been implemented.

The IBKSim clear differentiates between user and object developer which is particularly important for the usability and automatic simulation series. In this context a user is a person or a program which generates a simulation scenario using the implemented simulation objects. The implementation of simulation object is the object developer's main task.

1.1 User perspective

From the user's point of view, IBKSim takes a configuration file as input and generates a logging file. If the simulator runs in the so called debugging mode it additionally generates output to the terminal. In the configuration file, which is written in XML, the user can describe the whole simulation scenario. Listing 1 depicts an example of such a XML file. After configuring the maximum simulation duration and the initial random function seed, the user models the network. A network consists of nodes which are containers for components. Every node and every component within this node has a unique name. Additionally each component has a specific type and an optional debugging parameter. The type describes to which simulation object class the component belongs to. The debugging parameter is just used in the debugging mode. Depending on the value of the parameter, the component generates trace information on the terminal.

Since components are the actual simulation objects, every node needs at least one component. The concept that a node is a group of simulation objects, as shown in Figure 1, allows to simulate different network layers as different objects. Each component can have different additional parameters which are specific to it. The component description follows the configuration of the connections.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<simulation>
  <paramlist>
    <duration value="100"/>
    <initialseed value="1"/>
  </paramlist>
  <network name="net1">
    <node name="source">
      <component name="source0" type="source" debug="0">
        <source prio="0" rate="10">
        </source>
      </component>
      <component name="wait" type="delay" debug="1">
      </component>
      <clink src="source0" dst="wait" direction="both"/>
    </node>
    <node name="sink">
      <component name="sink0" type="sink" debug="0">
      </component>
    </node>
  </network>
  <linklist>
    <link name="src-sink" direction="both">
      <src net="net1" node="source" cmp="wait"/>
      <dst net="net1" node="sink" cmp="sink0"/>
    </link>
  </linklist>
  <loglist>
    <logging type="file" interval="10">
     <config filename="logfile.xml"/>
     <log net="net1" node="source" cmp="source0" v="1"/>
     <log net="net1" node="sink" cmp="sink0" v="1"/>
    </logging>
  </loglist>
</simulation>
```

Listing 1. Example description of a simulation scenario.

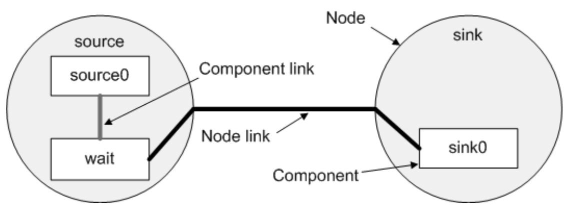

Figure 1. Visualization of the network described in Listing 1.

After the description of all network nodes, the nodes can be linked using their unique names. In contrast to the intra node links within nodes the inter node links represent physical links and therefore they support the additional parameters delay, jitter and loss-rate.

The last configuration concerns the logging function. The logging function collects cyclical logging information from the selected components. This allows to monitor certain user defined component parts over the time. The logging information which is again in XML format is saved in the specified log file as pictured in Figure 2.

The user must know which components are implemented in the simulator and which parameters they have, without possessing any knowledge about the code. The benefit of using XML for configuration and cyclical logging is described in Section 3.

1.2 Developer perspective

To implement a new simulation object the developer has to create a new class which is derived from `simEntity`. Furthermore he can use additional functions which have been implemented into the simulator.

Component factory Following start-up, the simulator parses the input file and generates a network of simulation objects which are commonly connected by links. This is realized using a component factory. For every component specified in the configuration file, the factory calls the static build function of the class which is associated with the type of the component. This function generates a simulation object and configures it depending on the additional parameters in the XML file. Afterwards it returns the new object to.

The usage of this so called factory pattern has some benefits for the developers. Since every class of simulation object has its own build function, and the build function gets the complete XML-content between the start and the end tag of the component, the developer can easily specify new XML-tags for the component configuration. Additionally he can use the build function to trigger init functions in the newly generated objects.

The developer does not need to use the component factory if it is not required. This is useful, e.g., when a simulation object is just started by another simulation object. One specific example is a server object which dynamically starts new simulation objects to handle requests. If the developer wants to use the component factory, he must implement an own build function and register the class with the factory.

Static connections As mentioned previously, it is possible to describe connections between components in the configuration file. To implement this functionality, the basic simEntity class implements the function connect. The component factory calls this function automatically to perform the linking between components which is basically an exchange of pointers. Using the pointers, a simulation object can transmit an event to another object. To simulate a physical link between two nodes the simulator places an additional simulation object between them which simulates the delay and the loss rate of the physical link.

Layered architecture IBKSim supports layered architectures by providing a class which has a special implementation of the already mentioned connect function. It automatically connects the components within a node derived from this class.

Messages and timer The reception of messages in the form of packets or frames are common events in communication networks. By including objects into events, it is possible to combine the transmission of events and messages between the simulation objects in an intuitive way. It is also an efficient way to implement preemptive timers. One option to implement a timer in an event based simulator is to send an event from a simulation object to itself. In the case that the timer must be deleted, the simulator searches for the event in the event list to remove it. To avoid the search, this is solved in a different way in IBKSim. Every timer includes a small object with a flag which states if the timer is active. To delete a timer the flag must be disabled. When the event occurs, the simulation object can ignore the event when the flag is not set. Using this concept, the search in the event list can be avoided.

Logging. The simulator supports two ways of logging. If the simulator runs in the debug mode and the

```
<ibkSimLog>
  <logentry time="10">
    <log net="net1" node="source" cmp="source0">
      <source sentPackets="102"/>
    </log>
    <log net="net1" node="sink" cmp="sink0">
      <sink receivedPackets="89" averageDelay="0.452"/>
    </log>
  </logentry>
  <logentry time="20">
    <log net="net1" node="source" cmp="source0">
      <source sentPackets="98"/>
    </log>
    <log net="net1" node="sink" cmp="sink0">
      <sink receivedPackets="92" averageDelay="0.483"/>
    </log>
  </logentry>
  <logentry time="30">
    <log net="net1" node="source" cmp="source0">
      <source  sentPackets="105"/>
    </log>
    <log net="net1" node="sink" cmp="sink0">
      <sink receivedPackets="102" averageDelay="0.464"/>
    </log>
  </logentry>
  ...
</ibkSimLog>
```

Listing 2. Example of a log file.

debug level for a component is set, the component can output traces to the terminal. This event triggered logging function is intended mainly for evaluating single simulations and for development. For simulation series the cyclic logging function has been developed.

For the cyclic logging function each simulation object implements a function called `getLogs()`.This function returns an XML string sharing statistical information. A special simulation object called `logger` calls this function on a regular basis, collects all the strings and writes them to a log file. This time triggered logging function is useful to monitor the change of different simulation object variables over time.

2 The benefit of XML as configuration and logging language

XML is a markup language to structure data in text files. This system and vendor independent language is a royalty free open standard [1]. IBKSim uses XML files for configuration and for logging due to the benefits it offers to the user of the simulator as well as for the development of additional simulator tools.

By using XML for configuration a user is not bound to a specific tool or editor to generate simulation scenarios. There are a number of XML editors, i.e. XPontus [5] and Altova XMLSpy [6], specifically designed to generate and validate xml files. Alterna-

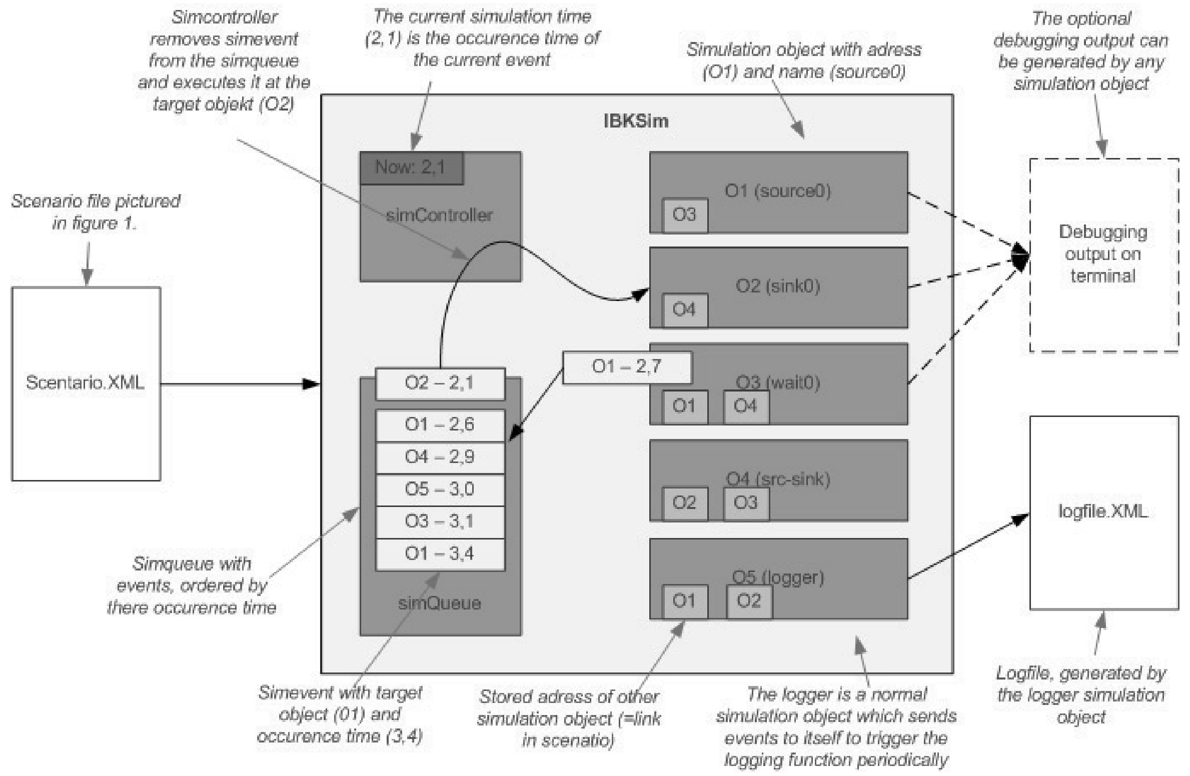
Simcontroller removes simevent from the simqueue and executes it at the target objekt (O2)
The current simulation time (2,1) is the occurence time of the current event
Simulation object with adress (O1) and name (source0)
The optional debugging output can be generated by any simulation object
IBKSim
Now: 2,1
simController
O1 (source0)
O3
O2 (sink0)
O4
O3 (wait0)
O1
O4
O4 (src-sink)
O2
O3
O5 (logger)
O1
O2
Debugging output on terminal
Scenario file pictured in figure 1.
Scentario.XML
O1 – 2,7
O2 – 2,1
O1 – 2,6
O4 – 2,9
O5 – 3,0
O3 – 3,1
O1 – 3,4
simQueue
logfile.XML
Simqueue with events, ordered by there occurence time
Simevent with target object (01) and occurence time (3,4)
Stored adress of other simulation object (=link in scenatio)
The logger is a normal simulation object which sends events to itself to trigger the logging function periodically
Logfile, generated by the logger simulation object

Figure 2. Overview over IBKSim.

tively, one can use a text editor to write a simulation scenario.

Since XML does not use a binary file format, simple scripts can be used to generate series of simulation scenarios. Additionally there exist a number of libraries for different programming languages to handle XML files in programs, i.e. Xerces [7] for C++, Java and Perl which simplifies the development of tools for the generation of complex simulation scenarios.

The usage of XML as log file format has some additional advantages. Microsoft Excel 2007 can directly handle XML files, which eases post-simulation processing and analysis.

Additionally XML processors can convert XML-files into other XML or human readable file types. One type of XML file is SVG [8] (Scalable Vector Graphics), a file format specifying vector graphics and animations using XML. By converting the log-file into a SVG file, a XML processor can generate an animation for presentation. With the same technique the log file can be converted into a comma separated values file (csv). This is useful if the user wants to analyze the data with statistic tools like *R* [9] which expects this data format as input.

Summarizing, we consider XML to be the most appropriate and flexible choice, both for scenario generation and for logging.

3 Work experience using IBKSim

The workflow for single simulations is straight forward. A user generates a simulation scenario in XML by using a text or XML editor. He starts the simulation and afterwards analyses the log file using e.g. Excel. Additionally he can analyze the simulator's debugging output if the components are configured accordingly.

For simulation series we are using a Beowulf cluster with 20 CPUs running Debian GNU/Linux 4.0 [10]. To manage the work load the sun grid engine [11] is used which accepts jobs in the form of shell scripts. To perform a simulation series the cluster uses three files: a template of the simulation scenario, a template of a job script which should be processed by the grid engine and a generation script.

A template is a XML configuration file where parameter values have been replaced by keywords. For instance the file in Listing 1 could be transformed to a template by replacing the initial seed value in line five from *1* to *--seed--*. The generation script can use this template to generate e.g. 100 simulation scenarios by copying the template and replacing the keyword *--seed--* with numbers from 1 to 100. The inserted value also becomes part of the name of the configuration and log file.

Additionally the generation script produces for each generated configuration file a job script using the job template. To perform solely the simulation a call of the simulator with the according configuration file in the job script is sufficient. Since in most cases a statistical analysis follows the simulation series, the job script is frequently used to pre-process logging data. A typical example is the calculation of the median and the quartiles of a logged parameter. For such operations *r* ("little R"), the command line version of *R,* is used. After generation of job scripts and configuration files, the generation script submits all jobs to the grid engine. When all simulations are processed, the log files or the files containing the preprocessed data are analyzed.

4 Summary

In this paper we presented the network simulator IBKSim. We outlined the architecture of this discrete event based simulator and described selected functions in detail. Particularly the advantages of XML as configuration and logging file format have been described in detail. Additionally typical examples of the simulation workflow using IBKSim were given. Concluding, the integration of XML has shown to be efficient and rewarding both for the user and the developers of IBKSim.

Acknowledgement

The authors would like to thank Jon Schuringa for the development of the IKNSim and all the other members of the Institute of Broadband Communications who were involved in the IKNSim and IBKSim projects.

References

[1] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau: *Extensible markup language (XML) 1.0.*, W3C recommendation, vol. 5, 2008.

[2] S. McCanne, et al.: *The Network Simulator - ns-2*, online: http://www.isi.edu/nsnam/ns/, visited: April 2009

[3] A. Varga, et al.: *The OMNeT++ discrete event simulation system*, Proceedings of the European Simulation Multiconference (ESM'2001), 319-324, 2001

[4] O. Spaniol, S. Hoff: *Ereignisorientierte Simulation*, Thomson Publishing, 37, 1995

[5] Y. Zoundi.: *XPontus XML Editor*, online: http://xpontus.sourceforge.net/, visited: April 2009

[6] Altova Inc.: *XMLSpy - XML Editor for Modeling, Editing, Transforming, & Debugging XML Technologies*, online: http://www.altova.com/xml-editor/, visited: April 2009

[7] The Apache Software Foundation.: *Xerces-C++ XML Parser*, online: http://xerces.apache.org/, visited: April 2009

[8] J. Ferraiolo, F.Jun, D. Jackson: *Scalable Vector Graphics (SVG) 1.1 Specification*, W3C recommendation, 2003

[9] R.Ihaka, R. Gentleman: *R: a language for data analysis and graphics*, Journal of Computational and graphical statistics, American Statistical Association, Institute of Mathematical Statistics, and Interface Foundation of North America,299-314, 1996

[10] Debian-Project*: Debian – The Universal Operating System*, online: http://www.debian.org/, visited: April 2009

[11] W. Gentzsch: *Sun grid engine: Towards creating a compute power grid*, First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001. Proceedings, 35-36, 2001

Corresponding author: Lukas Wallentin
Vienna University of Technology
Institute of Broadband Communications
Wiedner Hauptstraße 8 – 10, 1040 Vienna, Austria
Lukas.Wallentin@tuwien.ac.at

Received: March 10, 2009

Revised: January 20, 2010

Accepted: February 20, 2010