The following paper was originally published in the
Proceedings of the Fifth Annual Tcl/Tk Workshop
Boston, Massachusetts, July 1997

GeNMSim - The Agent Simulator
Tcl Based Agent Simulation Software

Ilana Gani-Naor, Ehud (Udi) Margolin, Raz Rafaeli
Milestone Software & Systems
Yoqneam Ilit, Israel

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

GeNMSim - The Agent Simulator

Tcl Based Agent Simulation Software

Ilana Gani-Naor (ilana@milestone.co.il)
Ehud (Udi) Margolin (udi@milestone.co.il)
Raz Rafaeli (raz@milestone.co.il)
Milestone Software & Systems, P.O.B 152, Yoqneam Ilit, Israel

Abstract

Network Management (NMS) application vendors, often encounter a situation where the device being managed or tested (which includes an SNMP agent) is not available at the time of the NMS application development, whereupon this becomes the critical path in the development cycle of the new device. To shorten this critical path, we've developed GeNMSim, which is a Tcl/Tk based Multi Platform SNMP agent simulator. The main features of GeNMSim are user customisation using Tcl callbacks, Portability across Unix and Window95/NT platforms and automatic creation of the simulator database by a set of Tcl based tools. GeNMSim is a commercial product targeted at Data Communication and Telecommunication companies involved in SNMP development.

Introduction

GeNMSim is a Tcl based SNMP agent simulation software. This article gives a short background on the need for this product and looks at the incentive to build this product with Tcl. Then we'll dive a little deeper into the technical aspects of GeNMSim, give some examples of the database structure and cover the process of creating this database, see how GeNMSim runtime is built (with a very short glimpse into the GeNMS technology). The Tcl based callback mechanism follows and the conclusions section discusses the performance and portability problems encountered and the need for better development tools that arised in the process of developing this product.

What is an Agent Simulator

Network Management Systems (NMS)

Network Management Systems (NMS) are software platforms which provide the functionality and tools to centrally manage communication networks. An NMS platform may be from a small scale Windows based application to a large scale, Unix based distributed application. A new equipment added to the network needs a management application that hooks up to the existing NMS platform currently managing the network. This management application is usually supplied by the network equipment vendor and is used to monitor and control the equipment.

SNMP, Management Information and MIB Files

An important part in the design of a new networking device is the design of the management information that will be available for this new device. SNMP (Simple Network Management Protocol) which is the most prevalant management protocol, defines a syntax for this management information called MIB (Management Information Base). The MIB includes the variables and tables that can be read from (or written to) this device. It also defines other important attributes of this managed information. SNMP Supports GET, SET, GET_NEXT, RESPONSE and TRAP messages to and from the agent. [1],[3]

Following is an example of the MIB file syntax:

```
sysName OBJECT-TYPE
SYNTAX  DisplayString (SIZE (0..255))
ACCESS  read-write
STATUS  mandatory
DESCRIPTION

       "An administratively-assigned name for
       this managed node.  By convention, this is
       the node's fully-qualified domain name."

::= { system 5 }
```

MIB File syntax example

NMS Applications Development Cycle

A networking equipment, includes a software agent which communicates with the management application managing this equipment. In order to develop a management application, one needs the device's MIB defi-

nition and a working agent. Since a networking device cannot be shipped without a management application, starting the application development only after the agent is already functional, causes a substantial delay in the availability of the equipment with its management application.

Using a Simulator instead of the Real Agent

Using an agent simulator can help to reduce this delay, by turning the development process of the agent and the management software to concurrent processes, and reducing the overall development cycle. Using a simulation can also improve the final quality of the agent by causing the design problems in the MIB to arise at an earlier stage, when its easier to make changes in the agent software. The simulator can also be used for testing the management software since it is much easier to configure a simulation then to build a real working language for testing. Developing an agent prototype in a high level scripting environment instead of the regular embedded software environment, helps to get better results from the agent in a shorter time.

GeNMSim Main Features

The main features supported by GeNMSim are:

- Automatic creation of a working database from MIB files
- Multiple agents in one GeNMSim process
- User customisation with Tcl callbacks
- Online traffic statistics with GeNMSim 'Probe'
- Multi Platform - runs on Unix and Windows95/NT

Why Tcl ?

Portability

GeNMSim is designed as a portable product for Unix and MS/Windows platforms. Using Tcl as the engine behind GeNMSim avoids many of the portability issues that are encountered in such a product. The main part in GeNMSim for which Tcl does not provide portability is the network interface which is implemented in the Unix version using the SNMP library provided by Carnegie Mellon University and in the Windows version using an agent enabled WinSNMP package. [5]

User Customisation

GeNMSim provides many hooks for user customisation of the agent. Using Tcl as the scripting language for user customisation gives the user all the power of Tcl/Tk without the need to compile or link the program. User scripts written for one operating systems are ported automatically. The user gets all the GeNMSim added Tcl commands of which some are implemented as C functions and others as Tcl procedures.

Ease of Development

Developing a project using Tcl saves substantial R&D time. Making changes to the code does not require compilation and thus much of the programmers idle time is reduced. The user interface part is very simple to build and does not require learning a Motif or an MS/Windows GUI package. The GeNMSim database is also implemented as a set of Tcl scripts which avoids the need to write and maintain a separate parser and allows using the functionality of the Tcl script loading mechanism while loading the data base.

Alternatives

The most important requirement from GeNMSim is its ability to be customised by the user. This requirement can be achieved either by using a script language (like Tcl) or by having the user work in a Compile/Link programming environment. Other scripting languages (such as Visual Basic or Perl) are not portable between Unix and Windows and are hard to customise. Using a C/C++ programming model, forces the user to have a full development environment and also takes us back to the portability issue. [2]

GeNMSim Tcl Based Data Base

General

An SNMP agent simulation requires 3 types of information:

1. The SNMP MIB files which are designed by the networking equipment vendor as part of the overall design, and are shared by both the agent and the manager.
2. Current agent MIB values - These values represent the current state of the simulated agent with current values of the data held in the agent.
3. Agent behaviour - A real agent is characterised by both the information it can provide and the actions it can take when something happens. These actions have to be defined when creating a simulated agent.

Defining a new agent for simulation is done in 2 phases:

1. Define the ***Agent Type***. This can be viewed as defining a new class in object oriented programming. An ***Agent Type*** defines the information and behaviour of a certain agent type.
2. Define the ***Agent Instance.*** This can be viewed as defining an object of a given class in object ori - ented programming. The ***Agent Instance*** includes the current state and values of a specific agent being simulated.

Agent Type Data Base

The process of creating a new ***Agent Type*** is divided into two parts:

1. Automatically extract the information contained in the MIB files which define the management scope of the simulated agent. This process is carried out via the ***Create Agent*** tool which is described later.
2. Manually configure the new agent type by adding the agent behaviour via callback functions. Callback functions are Tcl procedures that are called at certain points of the simulation. Callbacks are described in further detail later on.

Lets assume we're defining an agent type named **MyAgentType.** The ***Agent Type*** is defined by 4 ASCII files which are actually Tcl scripts.

The first and most important file is **MyAgentType.def** file which includes the definition of MIB information as extracted from the MIB files.

Figure 1 shows an example of the .def file:

The keyword **Sim_Db_loadMibDef** is actually a Tcl procedure which adds an attribute for the specified MIB object to the data base.

The second file is **MyAgentType.oid** which defines translation from ascii names to ASN.1 decimal notation which is required for the simulation process. This file is also a Tcl script in which each line is a call to a procedure to add a <name,oid> pair in the data base.

An important part of the agent behaviour is to send SNMP traps which are asynchronous messages sent to the manager with accordance to the agent behaviour. The third file named **MyAgentType.traps** defines the traps supported by this agent and are also automatically extracted from the MIB definition files.

The fourth and last file is **MyAgentType.user_def** . This file contains all the callback registration commands for this agent type. This file has the same syntax as .def files. The reason for seperating the user additions from the automaticlly created file is in order to allow easier migration in subsquent creation of the AgentType database and in new GeNMSim versions, since this file is not erased when the database is recreated.

Agent Instance Data Base

The ***Agent Instance*** is also built in two phases:

1. As part of the automatic process, a template of the ***Agent Instance*** is created to allow quick start for the agent simulator user.

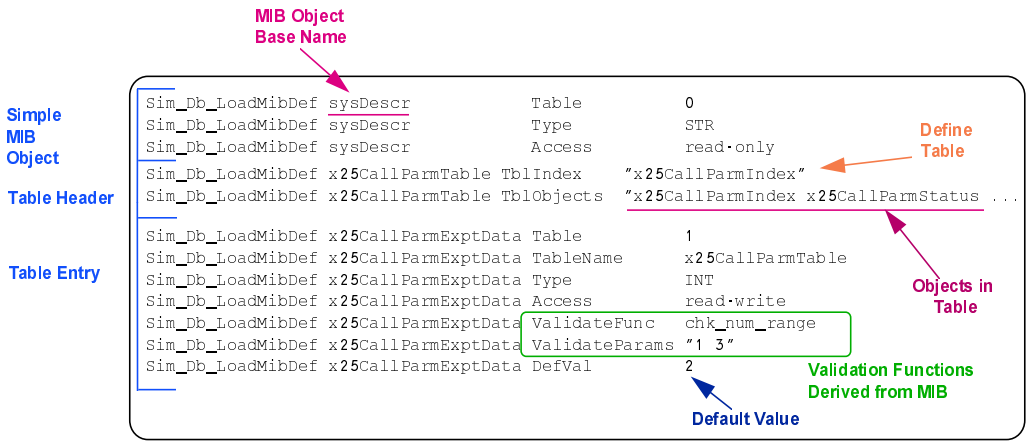

Figure 1: AgentType .def file example

2. This template is modified by the user (currently via a text editor) to reflect the real (initial) values of the simulated agent and are updated by the GeNMSim runtime as values in the database are modified.

Lets assume we're defining an agent instance named **MyAgent.** This instance is of type **MyAgentType.** The ***Agent Instance*** is defined by 2 more Tcl scripts.

The first and most important file is **MyAgent.rt** file which includes the current values of the database tables.

Figure 2 is an example of the .rt file.

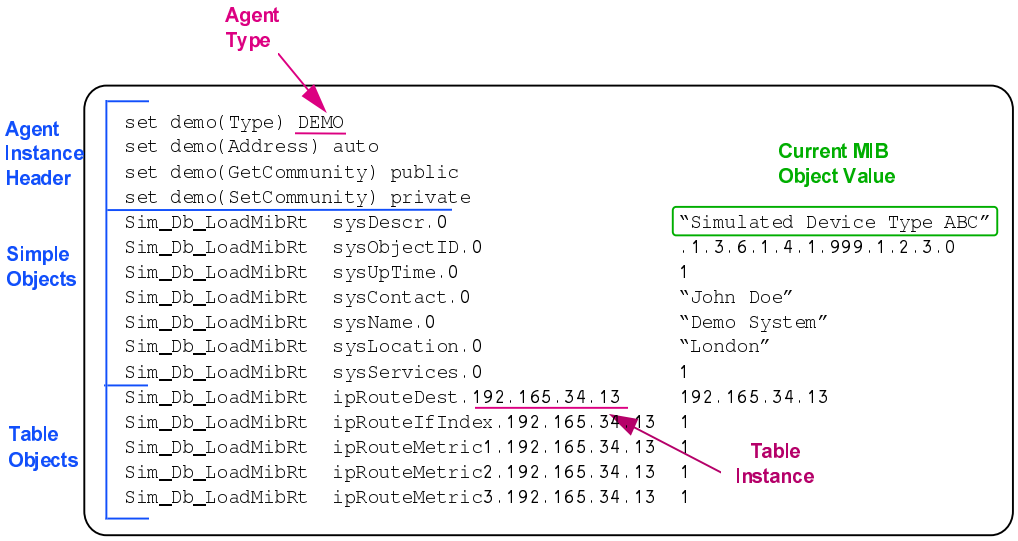

Figure 2: Agent Instance .rt file example

The keyword **Sim_Db_loadMibRt** is actually a Tcl procedure which adds an entry for the given MIB object to the data base.

The second file is **MyAgent.managers** which defines the addresses of the managers which will receive traps generated by this agent.

GeNMSim Offline Tools

There are several offline tools in GeNMSim which help in preparing the simulation. We'll concentrate on the **CreateAgent** tool which plays an important role in GeNMSim and is the most interesting in terms of Tcl. This tool is a Tcl program which parses MIB files and generates the automatic parts of the GeNMSim database.

Parsing the MIB Files

A MIB file is a text file in a standard format defining **Attributes** (Types, Access, Syntax etc.). MIB files may import definitions from other MIB files (such as include files). Each MIB object defined in the MIB file has a unique location in the global **MIB tree.**

The following is an example of the MIB file syntax:

```
iso      OBJECT IDENTIFIER ::= { 1 }
org      OBJECT IDENTIFIER ::= { iso 3 }
dod      OBJECT IDENTIFIER ::= { org 6 }

ObjectSyntax  ::= INTEGER
ObjectName    ::= INTEGER
internet    OBJECT IDENTIFIER ::= { iso org(3)
                                    dod(6) 1 }
directory    OBJECT IDENTIFIER ::={ internet 1 }
mgmt         OBJECT IDENTIFIER ::={ internet 2 }
experimental OBJECT IDENTIFIER ::={ internet 3 }
private      OBJECT IDENTIFIER ::={ internet 4 }
enterprises  OBJECT IDENTIFIER ::={ private 1 }

sysLocation OBJECT-TYPE
SYNTAX  DisplayString (SIZE (0..255))
ACCESS  read-write
STATUS  mandatory
DESCRIPTION
"The physical location of this node (e.g.,
`telephone closet, 3rd floor')."
::= { system 6 }
```

The ***CreateAgent*** program parses this text, extracting from it the MIB **Name**, **Syntax**, **Access** and its **Location** in the MIB tree. If the MIB object is part of a **Table**, the table information is also resolved. In order to parse these files, the Tcl **string** and **regexp** commands are used extensively. In the example above, we can see the SYNTAX of the MIB object is `'DisplayString (SIZE (0..255))'` meaning it is a string with maximal length of 255 characters. In this case, a validation callback function will automatically be registered to check that the values set to this object do not exceed the limits in the definition. The same scenario applies to other conditions that may appear for MIB object values.

The ***CreateAgent*** is actually comprised of several Tcl scripts each responsible for one part of the agent database. These scripts are called as separate tasks using the Tcl exec command in order to allow an independent global variable scope for each of the scripts. See also the section on portability later on.

Figure 3 shows the input and output of the ***CreateAgent*** program.

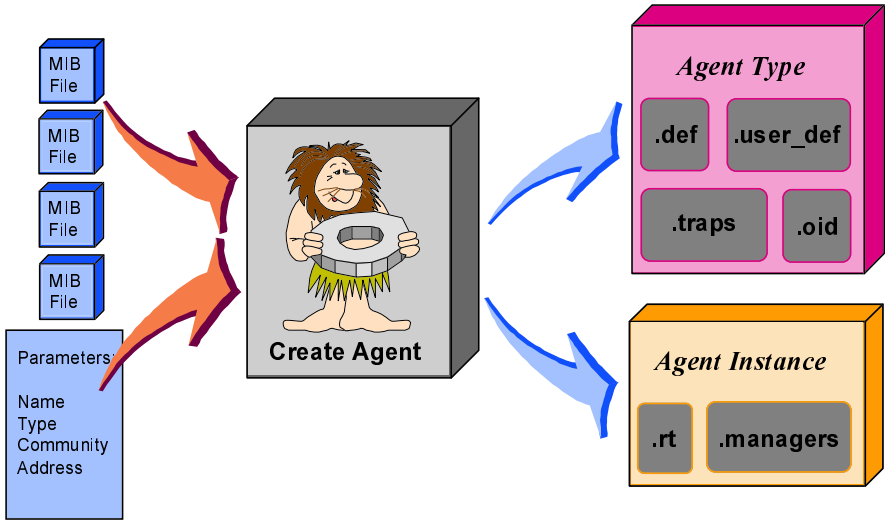

Figure 3: Create Agent - Input and Output files

The GeNMS Technology

Overview

Since GeNMSim is based on the GeNMS technology, it is important at this point to take a brief glance on the GeNMS technology. GeNMS is a Tcl/Tk based technology which provides a framework for creating network management products, applications and tools. GeNMS includes many mechanisms intended to allow easy development of these products. Tcl/Tk gives it the GUI portability and allows for most of the OS independence. GeNMS also supports portability over NMS platforms and management protocols.

Message & Trap handling

Messages, Which are one of the GeNMS mechanisms, are implemented as a C structure and function library which has a Tcl interface. A message is represented in Tcl as a string handle. There are API functions to create new messages, add or fetch variables, read or update header parameters and forward the message to the network. A message received from the manager via the network interface, is passed to a message dispatching mechanism which then registers the message (in a Tcl

hash table) and forwards the message to the simulator handling function (in the case of GeNMSim) which is implemented fully in Tcl. The simulator processes this message by extracting the request code and variables and accessing the data base as requested. The response message is then built and send via the GeNMS scheduling mechanism (Tcl TimerHandler based scheduling mechanism) back to the manager via the network and platform layers.

GeNMSim Runtime

The GeNMSim runtime is a GeNMS/Tcl/Tk based executable enriched with numerous application specific commands.

The process starts with an initialisation phase which loads the data base files, starts the network interface, creates the GeNMSim windows and prepares for message processing. This includes creating a new socket and file handler and adding this file handler via the Tcl FileHandler mechanism to the Tcl select loop. Since SNMP is based on UDP, there's usually a well known UDP port to which an SNMP agent listens to.

When initialisation is complete, GeNMSim is ready to receive SNMP and to 'act' as a real agent.

GeNMSim Traps

Traps are generated in GeNMSim callback functions or as a result of a delayed action which is activated by the scheduling mechanism. The trap is a message which originates at the simulator level and is sent to the network.

Using Tcl Data Structures for holding data

Most of the agent's internal data is held in Tcl two dimensional arrays. Since GeNMSim can support more the one concurrent agent, one of the array dimensions is the agent type or agent name and the other index is the attribute. Each MIB name has its own attribute array and there are several general purpose arrays to hold the name to oid translation and a linked list of the objects currently held in the agent.

Callback Type	Description
PreInit	Called when GeNMSim is started before loading the data base
PostInit	Called after data base is loaded
PreGet	Called upon a get request before the GET is done
Get	Called upon a get request instead of the standard GET processing
PostGet	Called upon a get request after the GET is done
PreSet	Called upon a set request before the SET is done
Set	Called upon a set request instead of the standard SET processing
PostSet	Called upon a set request after the SET is done
Validate	Called upon a set request to check if the value to be set meets field validation criteria
ValidateGroup	Called upon a set request after each one of the object validation callbacks are processed to validate mutual dependencies

Figure 4: GeNMSim Callback Types

Customising GeNMSim with Tcl based Callback Functions

What is a Callback Function

Customising an application can be done is several methods. One of the most common ways is to give the application user, the hooks to add procedures that will determine the behaviour of the application at that point. In order to add a new callback to GeNMSim, the user has to write this callback in a designated directory (in Tcl) and then register the callback in the database and associate it with the desired MIB object and callback type.

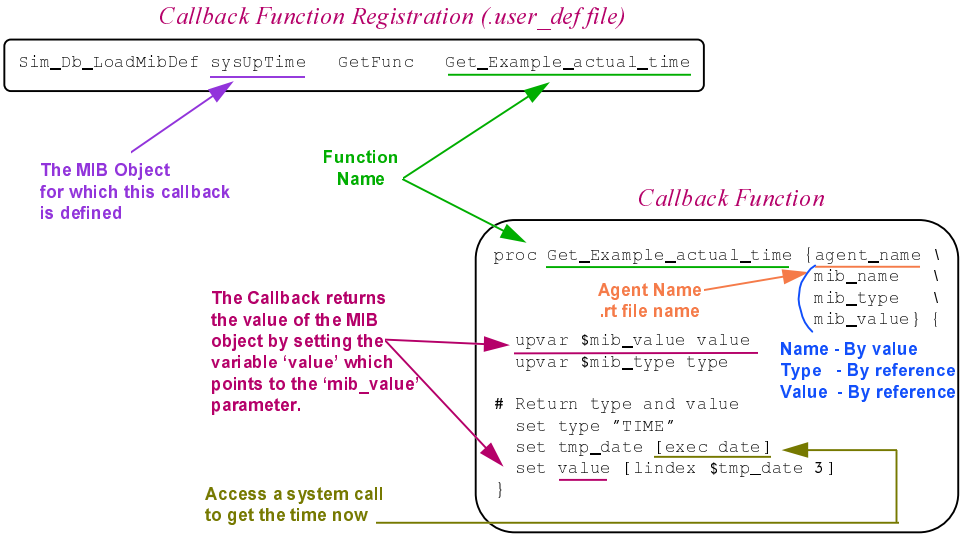

Figure 5: Callback registration and implementation

Types of Callback Functions

GeNMSim supports 10 types of Callbacks. Figure 4 lists the supported callback types.

Callback Registration and Usage Example

The following illustration shows an example of a callback function. This example is of a Get callback for the MIB object ***sysUpTime*** which generally means 'how much time has this systems been running'. In our case, the callback function returns the current time from the system. 1st The callback is registered in the **.user_def** file. The callback itself uses the Unix '**date'** command to get the current time. Another way to implement this callback in a more portable way is to use the Tcl '**clock'** command.

Using GeNMSim Database API from Callbacks

It often occurs that callbacks need to have access to the GeNMSim database. In such a case, the callback has access via the database API functions which is a set of Tcl procedures or commands. An example of a callback function that makes use of some of these API functions is listed below. In this example, the **PostGet** callback increments a counter in the MIB. The name of the counter is passed to the callback by the calling mechanism. This information is available from the callback registration.

```
proc Ex_PostGet_Increment_Counter {agent_name \
                mib_name mib_type \
                mib_value args} {
# Extract counter name from the 'args' variable
  set cnt_name [lindex $args 0]

# Get the current value of the counter.
  if {[Sim_Db_GetMibVal $agent_name $cnt_name \
```

```
      cnt_value] != 0} {
    puts "--> Error while getting value of  \
         object $cnt_name"
    return
  }

# Get the type of the incremented object
  Sim_Db_GetMibType $agent_name $cnt_name \
          cnt_type

# Set the new value (old value + 1) to cnt_name
  Sim_Db_SetMibVal $agent_name $cnt_name \
      $cnt_type [incr cnt_value]

# Save the run-time values to the rt file
  Sim_Db_SaveRtValues $agent_name
}
```

A Callback using GeNMSim database API functions

GeNMSim Windows

GeNMSim has several windows implemented in Tk to display statistics of the traffic which passes through the agent. The windows include a general statistics window, a message list window and a message details window. Since these are pretty simple Tk windows, There is no need to discuss them in detail.

Conclusions and Future Directions

Performance

An SNMP agent can in some cases be a computation intensive application. These cases may be when a GET_NEXT command is applied for a MIB object which is currently not present in the database. (GET_NEXT means, fetch the object which follows the given object in the MIB tree). Since implementing a tree data structure in Tcl is far from optimum, GeNMSim holds the objects in Tcl arrays with linked lists giving the order of the variables. In a test case, made on a simulated agent with about 200 entries, a GeNMSim agent responds after 3 seconds, while a real agent responds in only a fraction of a second. In a larger agent the performance penalty is linear since the search is performed sequetially. If we'll consider a table with 5 columns (a typical case) which is located in an agent of 500 lines, then a GET_NEXT to this table when it is empty, using GeNMSim may take up to 25 seconds (5 columns X 5 seconds each). This kind of a response from an agent cannot be considered acceptable. In most cases though, fetching or updating a variable from the database is done in reasonable time. To improve performance, we're now considering to add a C implemented tree structure for the MIB tree.

Portability

GeNMSim was initially developed for SunOS with Tcl 7.5 and Tk 4.1 . Most of the GeNMSim Tcl code has been ported with no changes to Windows95/NT with several exceptions:

1. **Unix file access systems calls** - Commands like **chmod** and **cat**, which have no Tcl commands to 'cover' them, were used in the Unix version but could not be ported to the windows version.
2. **Unix environment variables** - The **env** array which contains the system environment variables was extensively used in the original Unix version. This feature exists on the Windows/NT system but in Windows/95, which is based on DOS, environment variable names are case insensitive and there is a problem with allocation of environment space for the environment variables. To overcome these problems, most of the environment variables where switched with configuration variables in a Tcl global array.
3. **Tcl exec command** - The **exec** command was used in the Unix version in the CreateAgent tool to invoke the sub-tools in separate shells. Since the Windows version does not support exec (Tcl version 7.5), we used **multiple interpreters** in which we started the sub-tools. Slave interpreters, in combination with the **alias** command, allowed invocation of sub-tools in their own scope of global variables.

Other then the Tcl code portability issues, there was a need to rewrite the networking layer to support WinSNMP. Since GeNMS is a modular technology, this task was simple (about weeks of programming).

Development Tools

GeNMSim is a pretty large project. It contains over 10,000 lines of Tcl code and a large number of C code lines. A good, source level debugger for Tcl (plain vanilla Tcl/Tk) would have been much help during the process of debugging GeNMSim.

Performance analysis is also an important issue when creating a computation intensive Tcl program.

GeNMSim makes use of the debugging aids that are part of the GeNMS technology. These include a sophisticated debug printings mechanism, a Tk widget

configuration dump and the GeNMS Classes & Objects mechanism which is beyond the scope of this paper.

The Future of GeNMSim

GeNMSim is one of a family of products that Milestone is creating for the NMS market. The next version of GeNMSim will include an offline graphic configuration tool to edit and configure the database. There is also a thought about supporting other management protocols besides SNMP.

Availability

GeNMSim is a commecial product. For more information contact (via email): GeNMS@milestone.co.il.

Evaluation copies available upon request.

Refrences

[1] Marshall T. Rose. *The Simple Book: An Introduction to Internet Management* (2nd edition). Prentice Hall, 1993. ISBN 0-13-177254-6

[2] Marshall T. Rose, Keith McCloghrie. *How to Manage Your Network Using SNMP*. Prentice Hall, 1995. ISBN 0-13-141517-4

[3] William Stallings. *SNMP, SNMPv2, and CMIP. Practical Guide to Network-Management Standards.* Addison-Wesley, 1993, ISBN 0-201-63331-0

[4] John Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994/ ISBN 0-201-63337-X

[5] Eric F. Johnson. *Graphical Applications with Tcl & Tk.* M&T Books, 1996. ISBN 1-55851-471-6