Tcl Me EMA

Ted Nolan

SRI International
Information, Telecommunications, and Automation Division
4210 Columbia Road, Suite 5A, Augusta, GA 30907
email: ted@erg.sri.com

ABSTRACT

The MIL 3, Inc. External Model Access (EMA) library provides a powerful interface to OPNET models. At times this power entails a level of complexity that can make small tasks daunting.

Tcl/Tk is an extensible graphical user interface (GUI) and scripting language available for UNIX, Macintosh and Windows NT platforms. This paper presents a package, TkEMA, that wraps the EMA library into Tcl commands. TkEMA makes complex EMA tasks more approachable, and simple EMA tasks trivial. We will consider some of the design issues involved in implementing TkEMA, give command-line and GUI examples of its use, and suggest some future directions for TkEMA development.

EMA

MIL 3, Inc. (MIL3) provides a C-based External Model Access (EMA) library with OPNET.* This library provides a programmatic interface for reading and writing OPNET models. EMA enables users to create or customize instances of the various OPNET model types without having to laboriously enter each parameter or draw every link from within the OPNET graphical user (GUI) interface. On the other hand, the power that EMA provides entails a considerable degree of complexity, so that creating a C program for relatively simple tasks may be daunting. In addition, a graphical representation of a networking or modeling problem is often easier to comprehend than a text-based one, and EMA programs written in C have no inherent support for graphics.

Tcl/Tk

Tool Control Language (Tcl, pronounced "tickle") is a high-level scripting language developed by Dr. John Ousterhout. It is freely available, and supports UNIX, Windows NT and Macintosh platforms. Tcl's inherent extensibility makes it especially well suited for use with specialized C application program interfaces (APIs). Tcl commands built on top of these APIs can be linked statically into the Tcl interpreter or loaded dynamically at run-time. ToolKit (Tk) is the most commonly used extension package for Tcl (indeed, the combined package Tcl/Tk is usually assumed whenever Tcl is mentioned). Tk provides cross-platform graphical support, which makes it possible to use Tcl/Tk to write GUIs that operate without change across different operating systems.

TkEMA

Given EMA's well-defined API and Tcl's extensibility, Tcl and EMA are a natural match. TkEMA, the result of that match, provides all of EMA's OPNET model support coupled with Tcl's ease of use and Tk's graphical capabilities.

The design goal for TkEMA was to keep the familiar EMA interface as unchanged as possible while still providing complete integration with Tcl. Although as implemented TkEMA largely achieves this goal, it does make compromises when necessary to achieve a result more natural to a Tcl programmer.

In particular, the use of the TkEMA command which corresponds to an EMA function has been slightly changed in some cases. These changes can be stated roughly as follows:

- *Get* commands return a value rather than setting a variable.
- *Get* and *set* commands work on one attribute at a time—Attribute Component Value (ACV) lists are not supported. As a consequence, it is never necessary to use EMAC_EOL in a TkEMA command.
- It is necessary to supply the data types of values being stored or retrieved.

In addition to commands implementing direct one-to-one mappings from the EMA and Textlist APIs, TkEMA also provides commands to store and retrieve EMA Object IDs by string keys, and a special command to handle the `ATTR_EMA_NUM_OBJECTS` attribute.

Unlike C programs, where return values often double as exception indicators, Tcl provides an integrated exception handling mechanism. Any Tcl command

*All product or company names mentioned in this document are the trademarks of their respective holders.

can return a value or throw an exception. Uncaught exceptions terminate the execution of a script with an error message, but exceptions may also be caught and handled intelligently.

TkEMA uses the `EMAC_MODE_NONE` flag to *Ema_Init()*, rather than `EMAC_MODE_ERR_HALT`, or `EMAC_MODE_ERR_PRINT`, so as to retain control in the face of errors. After the execution of each call to EMA, TkEMA commands check the `EmaS_Oper_Status variable`. If it indicates `EMAC_COMP_CODE_FAILURE`, then the `EmaS_Error_Condition` variable (which holds an error message from the EMA library) is placed in the TkEMA command result, and the command throws a Tcl exception.

To illustrate the flavor of TkEMA, consider a simple request to retrieve the "name" attribute of a subnetwork. In C the request might look like this.

```
char subnetname[80];
Ema_Object_Attr_Get(model_id,
 subnetid, "name",
 COMP_CONTENTS, subnetname,
 EMAC_EOL);
```

In TkEMA, it would look like this.

```
set subnetname  \
  [Ema_Object_Attr_Get \
  $model_id $subnetid \
  "name"  EMAC_STRING \
  COMP_CONTENTS ]
```

Creating TkEMA commands is simple. First, a command is registered with the Tcl interpreter, along with a pointer to a function that handles that command. When the command is used in a Tcl script, the registered function is called. A Tcl command function is similar in form to a C *main()* routine, but instead of `argc` and `argv` parameters, it receives `objc` and `objv` parameters. The function must extract from its arguments the necessary data to pass to the EMA call, make the EMA call, check for errors, place a Tcl return value in the Tcl result object and return a status indication as the function value.

As a simple example, consider the function that implements the TkEMA command `Ema_Model_Attr_Nth` by wrapping the EMA *Ema_Model_Attr_Nth()* function.

```
/*
 * Create the Ema_Model_Attr_Nth
 * command, bound to the
 * Ema_Model_Attr_Nth() EMA call
 */
static int
tkEma_Model_Attr_Nth(
  ClientData clientData,
  Tcl_Interp *interp, int objc,
  Tcl_Obj *CONST objv[])
{

  EmaT_Model_Id model_id;
  int index;
  int error;
  char buf[8192];
  Tcl_Obj *resultPtr;

  resultPtr =
    Tcl_GetObjResult(interp);

  if(objc != 3){
    Tcl_WrongNumArgs(interp, 1,
          objv,
          "Usage: "
          "Ema_Model_Attr_Nth "
          "model_id index");
    return TCL_ERROR;
  }

  error =
    Tcl_GetIntFromObj(interp,
    objv[1], &model_id);
  if(error != TCL_OK)
    return error;

  error =
   Tcl_GetIntFromObj(interp,
    objv[1], &index);
  if(error != TCL_OK)
    return error;

  Ema_Model_Attr_Nth(model_id,
    index, buf);
  if(tkEma_errchk(resultPtr) < 0)
    return TCL_ERROR;

  Tcl_SetStringObj(resultPtr,
    buf, -1);

  return TCL_OK;

}
```

EXAMPLE APPLICATIONS OF TKEMA

NETWARS:
Subnets as Primitive Types

Under the aegis of Lieutenant General Douglas Buchholz and the day-to-day direction of LTC Pat Vye, the J6 NETWARS initiative is an ambitious project designed to simulate U.S. military battlefield communications in the joint arena. OPNET is being used as the simulation engine for this effort, but the number of devices and networks involved makes it

impractical for NETWARS networks to be constructed in the standard OPNET Network Editor. The anticipated project architecture includes a graphical, Java-based front end designed by Science Applications International Corporation (SAIC), communicating with a simulation engine back end through files or (in the case of a front end running in a browser) sockets.

The front end designs a simulation scenario based on user interaction and creates a simulation description file (SDF) that is passed to the back end.

The back end invokes a TkEMA script, `netwars_ema`, that dynamically loads a parser library defining new Tcl commands for dealing with SDFs. These commands are used to iterate through all the specified communications platforms and instantiate them as mobile subnets.

As different types of point-to-point links and broadcast networks are encountered, the script dynamically loads Tcl packages that understand how to set the OPNET attributes required to support them.

One novel feature of the `netwars_ema` script is its ability to implement populated subnets as primitive types. EMA is often used to automate the creation of complex networks, based on a well defined set of rules. Unfortunately, not every complex network can easily be reduced to such a set of rules: although OPNET provides process models that can be assigned to CPUs and node models that can be assigned to nodes, it does not provide subnet models that can be assigned to subnets.

NETWARS involves the creation of large numbers of mobile subnets with an internal structure and connectivity dictated by the communications resources inherent in the deployment of a particular type of military unit. In previous stages of the project, the NETWARS development team attempted to characterize the internal communications structure of each subnet with a collection of configuration files that a C-based EMA program would parse and use to programmatically construct each subnet.

This implementation proved to be extremely CPU intensive, and could not represent configuration types that had not been envisioned during the project's first stages. With TkEMA a much simpler approach is possible. An instance of each type of subnet is created and saved from the familiar OPNET Network Editor. Then `extract_ema`, a TkEMA script, is used to extract the EMA code associated with that network via the `Ema_Model_Code_Gen` command.

```
#!/usr/local/bin/wish -f
# Extract the ema code from a
network
# model
package require Tkema

Ema_Init

foreach arg $argv {

    set model_id  [Ema_Model_Read \
        MOD_NETWORK $arg]
    Ema_Model_Code_Gen $model_id
$arg
}
exit
```

Thus, for example, running the command `extract_ema A0160` with an *A0160.nt.m* network file present would create an *A0160.em.c* file. This extracted code is automatically massaged by an edit script that invokes the UNIX *ed* editor to make global variables static and rename the C *main()* routine to *pdef_A0160()*. Finally, the new subroutine is compiled and stored in a shared library. This shared library also contains a hash table that maps subnetwork type names to the subroutines that implement them. When the `netwars_ema` script is run, this library is loaded as a TkEMA extension, and one Tcl command is now sufficient to create a complex subnet. For example,

```
set subnet [make_platform $model_id
\
    A0160 $topnet "my net" ].
```

Trying to compare programs by counting lines of code is always a dubious endeavor; nonetheless it is interesting to observe that the previous `netwars_ema` C program had over four thousand lines of code, while the current TkEMA `netwars_ema` script has less than four hundred lines of code. Granted, many lines of C code exist in the subroutines that make up the shared library, but this code is automatically generated, and does not generally have to be understood or debugged.

There is an issue of object code size here. Keeping all subnetwork types in one shared library makes that library quite large, though one large file is easier to keep track of than many smaller files. In the future, NETWARS may place each network type in its own shared library, and dynamically load only those that are needed for a particular configuration.

Process Model Documentation

In order to produce documentation on process models, it may be necessary to include the model source code, and a picture of the finite state machine that it implements. This process can be quite laborious because the normal OPNET procedures for performing it are somewhat crude. In particular,

saving the various C code blocks and executives of a process model to separate files requires bringing up the Process Editor and repeatedly invoking the text editor. Capturing a process state diagram is even more problematic, because the supplied tools produce a printer page that is not directly accessible and contains bitmap data that scales poorly. With TkEMA, it is relatively easy to construct tools to remedy these shortcomings.

For instance, the TkEMA tool *proc_dump* takes only 103 lines of code, including comments, and extracts all the C code from a process model. Consider the standard MIL 3, Inc. process model *pc_fifo*. In response to the command line

```
proc_dump pc_fifo .
```

`proc_dump` produces a directory named *pc_fifo* containing the following files

pc_fifo_BRANCH_enter.c
pc_fifo_BRANCH_exit.c
pc_fifo_INS_TAIL_enter.c
pc_fifo_INS_TAIL_exit.c
pc_fifo_SEND_HEAD_enter.c
pc_fifo_SEND_HEAD_exit.c
pc_fifo_child_processes
pc_fifo_diag_block
pc_fifo_function_block
pc_fifo_header_block
pc_fifo_state_vars
pc_fifo_temp_vars
pc_fifo_termination_block.

Each file contains the C code associated with the particular block or executive suggested by its name. These files can then be included in documentation or word processing tools, as desired.

Because of its inherent scalability and the support for it in documentation and presentation tools ranging from Microsoft Corporation's PowerPoint to LaTeX and Adobe's FrameMaker, Encapsulated PostScript (EPS) is a very desirable format for graphical output. Creating an EPS representation of a process state model is somewhat more complex than extracting the C code, but the built-in graphics support provided by Tk facilitates the task, and a 500-line TkEMA script called *pview* accomplishes it. A process loading in *pview* is shown in Figure 1.

Tk provides a canvas widget that implements as primitives the arc, rectangle, and oval operations needed to draw process states, the spline operations needed to draw smooth transitions, and the text operations needed to position and draw labels. The canvas widget also understands how to create an EPS representation of its contents, so that after drawing a process state model, it is trivial to export it.

Pview is invoked as

```
pview modelname
```

and displays the chosen process model in its main window. It also provides a file menu with the options:

```
Explode Model..
Print Model..
Save Model EPS..
Quit.
```

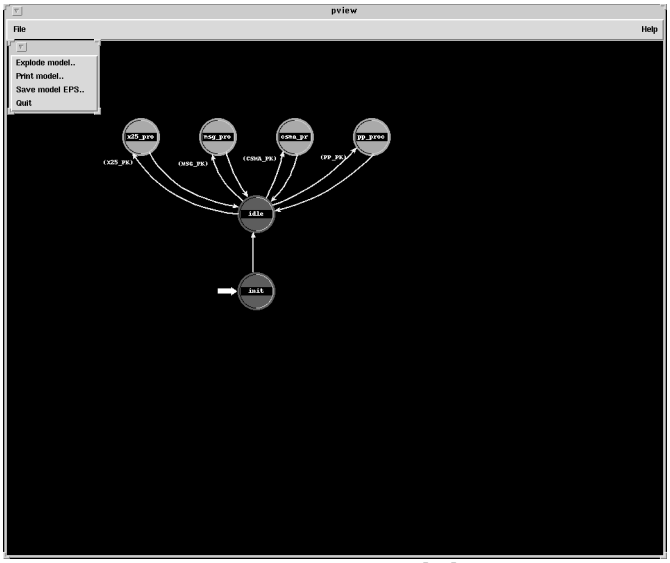

Figure 1 - A Process Loaded in *pview*

The `Explode Model` option invokes `proc_dump` on the chosen model. The `Print Model` option sends a picture of the state diagram to a PostScript printer. The `Save Model EPS` option saves an EPS representation of the state diagram to a file called *modelname.eps.* An example is show in Figure 2.

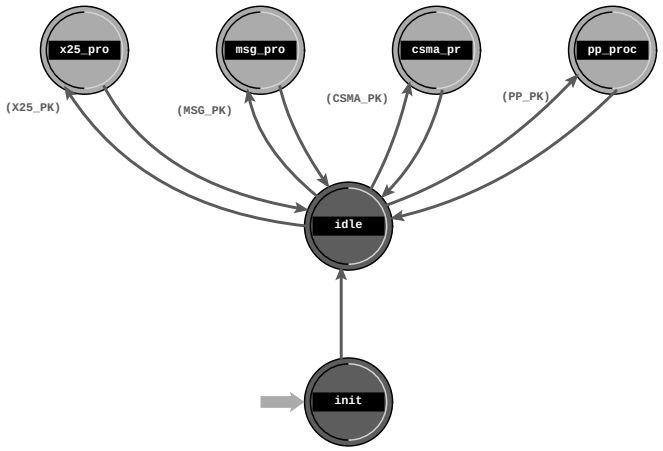

Figure 2 - EPS FSM Representation From *pview*

TkEMA Directions

TkEMA is a young package, parts of which have yet to be exercised. Doubtless, doing so will expose bugs that must be fixed. Some OPNET constants still

remain to be entered into the TkEMA mapping table, and while there should be, in principle, no difficulty in moving the package to Windows NT, this has not yet been done. Future experience will identify common operations more complex than single EMA calls: these should be implemented as additional TkEMA commands.

Availability

Tcl and Tk are available on the World Wide Web at

`http://sunscript.sun.com`.

The core TkEMA package is available at

`ftp://ftp.erg.sri.com/pub/people/ted/tkema.tar.gz`.

REFERENCES

MIL 3 Inc., 1998. *Chapter EMA: External Model Access*, MIL 3 Inc., Washington, D.C.

Ousterhout, J. 1994. *Tcl and the Tk Toolkit*, ISBN 020163337X, Addison-Wesley Professional Computing.

Welch, B. 1997. *Practical Programming in Tcl and Tk*, 2nd ed., ISBN 0-13-616830-2, Prentice Hall PTR.