Army Research Laboratory

Performance Requirements of Tools and Methods for Specifying Network Communication Scenarios Using the Real-Time Application Representative Version 1.0

by Rommie Hardy and Binh Nguyen

ARL-TR-4614 September 2008

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-4614 **September 2008**

Performance Requirements of Tools and Methods for Specifying Network Communication Scenarios Using the Real-Time Application Representative Version 1.0

Rommie Hardy and Binh Nguyen
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* September 2008	**2. REPORT TYPE** Final
3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE Performance Requirements of Tools and Methods for Specifying Network Communication Scenarios Using the Real-Time Application Representative Version 1.0	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Rommie Hardy and Binh Nguyen	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory ATTN: AMSRD-ARL-CI-NT 2800 Powder Mill Road Adelphi MD 20783-1197	**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-4614
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	
13. SUPPLEMENTARY NOTES	

14. ABSTRACT

This report identifies the performance needs of tools and methods that will be potentially capable of specifying data communication scenarios. These communication scenarios will be automatically generated using the Real-time Application Representative (RAPR) tool. The specifications define a specific behavior of a communication scenario for the RAPR tool being deployed in the wireless emulation laboratory (WEL) at the U.S. Army Research Laboratory (ARL). The report includes example descriptions of the RAPR input files and scripts used to specify network communication scenarios.

15. SUBJECT TERMS

Real-time application representative (RAPR), performance, communication

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Rommie Hardy
U	U	U	UU	30	**19b. TELEPHONE NUMBER** (*Include area code*) (301) 394-1189

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Acknowledgments

Richard Gopaul provided much direction in the early phase of this project. His constructive comments have been incorporated into this report. Mr. Gopaul has left the U.S. Army Research Laboratory (ARL), but continues serving the Nation in a different role at a different place.

INTENTIONALLY LEFT BLANK.

Summary

The Real-time Application Representative (RAPR) tool is a traffic generation tool that was developed by the U.S. Naval Research Laboratory (NRL). The run-time behavior of the RAPR tool is defined by the textual specifications embedded in three different file types: dictionary, logic-table, and script files. The manual creation of these files is time-consuming, tedious, and error-prone. Therefore, this report is focused on how to specify the performance needs of a tool and method potentially capable of minimizing syntax errors, speeding up the creation of required files, and providing a graphical user interface tool for creating a communication scenario.

INTENTIONALLY LEFT BLANK.

1. Background

Emulating a dynamic wireless mobile ad hoc network (MANET) calls for executable specifications of mobility patterns and data communication scenarios. The U.S. Army Research Laboratory (ARL) has developed an innovative solution meeting the first requirement (*1*). This report identifies the performance needs of tools and methods that will be potentially capable of meeting the latter requirement: specifications of data communication scenarios. Because ARL has decided to use the Real-time Application Representative (RAPR) tool (*2*) to generate and respond to actual network traffic in the emulation test bed, the specifications define a specific behavior of a communication scenario for the RAPR tool being deployed in the wireless emulation laboratory (WEL) at ARL (*3*). The RAPR tool requires the following file types: (1) dictionary files, (2) logic-table files, and input script files (3).

1.1 Dictionary

The dictionary is used to translate name-value pairs used in the script and the logic table. Each dictionary has one or more **namespace** fields. Each **namespace** has a **label** field and one or more **item** fields. Each field is given a name and assigned one or more values. Each value is used to generate an event. An example of the dictionary is shown in figure 1.

```
<RaprDictionary>
  <namespace>
    <label>DEFAULT</label>
    <item>
      <field>SA_UNICAST_GROUP</field>
        <value>192.168.1.2</value>
    </item>
    <item>
      <field>VIDEO_SERVER_SRCPORT</field>
        <value>2000</value>
    </item>
    <item>
      <field>VIDEO_SERVER_DSTPORT</field>
        <value>2001</value>
        <value>2002</value>
    </item>
    ...
  </namespace>
  ...
  <namespace>
    <label>WEL</label>
    <item>
      <field>DEFENDER</field>
        <value>192.168.1.20</value>
    </item>
    ...
    <item>
      <field>AUDIO_SERVER_DSTPORT</field>
        <value>3001</value>
        <value>3002</value>
        <value>3003</value>
    </item>
    ...
  </namespace>
</RaprDictionary>
```

Figure 1. An example of the dictionary.

2.1 Behavior-Event/Logic Table

The behavior-event table is also known as the logic table. The table consists of one or more states. Each state has one or more logicids. Each logicid has an **id** field, a **percent** field, and one or more **entry** fields. Each **entry** field defines the behavior associated with a given logicid. The **percent** field defines the probability of the triggering event and ranges from 0.0 to 1.0 (default).

If the **percent** field is unassigned or its current value is removed, then it shall be assigned the default value (1.0). An example of a logic-table file is described in figure 2.

```
<RaprLogicTable>
  <state>
    <value>1.1</value>
    <logicid>
      <id>1</id>
      <percent>0.1</percent>
      <entry>DECLARATIVE UDP SRC … </entry>
    </logicid>
    <logicid>
      <id>2</id>
      <percent>0.2</percent>
      <entry>DECLARATIVE UDP SRC … </entry>
    </logicid>
  </state>
  …
  <state>
    <value>20.0</value>
    <logicid>
      <id>1</id>
      <percent>1.0</percent>
      <entry>0.0 STOP 5.0 … PAYLOAD 9</entry>
    </logicid>
  </state>
</RaprLogicTable>
```

Figure 2. An example of a logic-table file.

1.3 Input Script

Each input script file describes a sequence of textual commands and scheduled events that define traffic patterns in a scenario. Each event is specified in a single line in the script file. The script has three types of events: (1) RAPR Event, (2) Reception Event, and (3) Behavior Event:

- Each line specifying a RAPR Event is essentially a directive designed to control the behavior of the RAPR tool, such as specifying the use of the logic-table file or stopping it.
- A Reception Event causes the RAPR to monitor a certain type of traffic or to join a multicast group.
- A Behavior Event schedules and defines network traffic and consists of four types: (1) declarative, (2) interrogative, (3) stream, and (4) periodic event types.
    - A declarative event starts a traffic flow for a given duration.

- An interrogative event sends a message and waits for a response.
- A stream event sends traffic and waits for and reacts to responses from other nodes.
- A periodic event spawns declarative and interrogative event types at regular intervals.

An example of a script is included in figure 3.

```
LOAD_DICTIONARY /usr/src/dictionary-48nodes.xml
HOSTID 24
VERBOSE
...
TXLOG
FLUSH
CHECK
0.0 STOP 600.0 LISTEN UDP %SA_LISTEN_PORT%
0.0 STOP 600.0 DECLARATIVE UDP SRC %SA_UNICAST_SRC_PORT% ...
```

Figure 3. An example of a RAPR input script.

The above sections summarize the background of the RAPR tool and the definitions of its file types, and present some examples of the files. More detailed descriptions of the tool can be found on the Web site of its owner and developer (*2*). The rest of this report specifies operational concepts, and the technical performance of graphical tools and methods for specifying network communications scenarios that will be executed in RAPR environments.

2. Method

This report is the result of a coordinated elicitation research to identify and understand the requirements for building a graphical tool capable of providing convenient methods for specifying a communication scenario. During the elicitation period, the requirements were implemented in a software prototype using the Python programming language, which is a scripting language that is highly suitable for building prototype and graphical user interfaces (GUIs). The users of the tool were able to experiment with the prototype and provide constructive feedback. The developer then used the feedback to add a new functionality, correct a behavior, or improve its performance and usability.

3. Operational Concept

The software tool set implementing methods for creating RAPR files will consist of three main software tools: (1) a dictionary editor, (2) a logic-table editor, and (3) a script editor. To create a communication scenario, the user would perform the following steps:

- *Step 1.* Create a new dictionary or load an existing dictionary to define a set of name-value pairs using the RAPR dictionary editor. During the editing session, the user can view and correct the contents of the dictionary. At the end of the session, the editor provides the user an option to save the dictionary into a text file that will be formatted using the extensible markup language (XML). However, the editor will not require the user know the idiosyncratic syntax of the language or the structure of the file in order to format and save the data in an XML file.
- *Step 2.* Create a new logic table or load an existing logic table to define a set of discrete states in which a RAPR would behave in a special way using the RAPR logic-table editor. Similar to the process of creating a RAPR dictionary file, at the end of an editing session, the editor provides the user an option to save the contents of the logic table in an XML file without requiring the user to be concerned with the idiosyncrasy of the language.
- *Step 3.* Create a sequence of commands and a list of scheduled events defining traffic patterns for a host computer acting as a client using the RAPR script editor. At the end of an editing session, the editor provides the user an option to save the scripts into a text file.
- *Step 4.* Create a sequence of commands in response to a specific request from a client for a host computer acting as a server using the script editor. At the end of an editing session, the editor provides the user an option to save the scripts into a text file.

4. Performance Requirements

4.1 The RAPR Dictionary Editor

The RAPR dictionary editor will be equipped with a GUI providing a convenient means for editing and creating a dictionary. The editor will provide the user a familiar way to open a file menu to select a menu item that will open an existing dictionary, create a new one, or save the current dictionary data to a file. The editor will also provide the user a way to learn about the syntax of the dictionary or about using the editor to create or modify a dictionary. The editor will consist of two areas: (1) editing area and (2) viewing area.

4.1.1 The Editing Area

The editing area will display a list of names corresponding to the XML tags that have been built into the RAPR tool. Each name is associated with an entry box into which a new value is entered by the user. Each name-value pair shall have two actions buttons: one to add an item to the dictionary and the other to remove an item from the dictionary.

The editor will interact with the user to obtain the values of the **label**, **field**, and **value** tags. The user-defined values will not include any character in the character set "[]<>". The editor will automatically handle the **RaprDictionary**, **namespace**, and **item** tags, effectively enabling the tags to be transparent to the user. Therefore, the editing area will display the labels of the three tags, their corresponding input entry boxes, and two buttons for each tag as depicted in table 1.

Table 1. User defined input for the RAPR dictionary editor.

label	<user-defined value>	Add/Update button	Remove button
field	<user-defined value >	Add/Update button	Remove button
value	<user-defined value >	Add/Edit button	Remove button

- *The value tag.* The user-defined field value of the **value** tag is a list of one or more value items. Multiple value items can be specified by using a comma (,) to separate them or by using the notation low-high if they are numerical values; using a prefix [low-high] if a prefix is desired; or combining all these options in a single command. For example, if the value of the **field** tag is defined as "alpha" and the expression, `A, B, 1-3, host[11-13]`, is entered into the **entry** field of the **value** tag and the **Add** button is pressed, then the tool will automatically generate the following tags and values (figure 4).

```
<field>alpha</field>
    <value>A</value>
    <value>B</value>
    <value>1</value>
    <value>2</value>
    <value>3</value>
    <value>host11</value>
    <value>host12</value>
    <value>host13</value>
```

Figure 4. A **value** tag example.

- Adding or removing a value affects the sub-element(s) of the currently displayed **field** item. If a user-defined **field** value is defined, then pressing the **Add** button will add a new value sub-element to the field element. If a user-defined **field** value is not defined, then pressing the **Add** button shall open an editing window for the user to enter one or more value items.

  Similarly, if the **Remove** button is pressed, then the tool will remove the specified value sub-element from the **field** element. If the removal of a value sub-element causes the field element to have no sub-elements, then the field element itself shall also be removed from the dictionary.

- *The field tag.* The user-defined field value of the **field** tag will be single or multiple. The format of “field names = values” will be used for *adding* or *updating* multiple values. The left side and the right side of the character “=” will follow the rule described in the previous paragraph specifying how to enter multiple values in the entry field of the **value** tag. For example, if the expression, `hostA, host[1-3]=192.168.1.64, 192.168.2.[100-102]`, is typed into the **entry** field of the **field** tag and the **Add/Update** button is pressed, the tool will generate four **item** tags, each has a **field** tag and a **value** tag as shown in figure 5.

```
<item>
    <field>hostA</field>
        <value>192.168.1.64</value>
</item>
<item>
    <field>host1</field>
        <value>192.168.2.100</value>
</item>
<item>
    <field>host2</field>
        <value>192.168.2.101</value>
</item>
<item>
    <field>host3</field>
        <value>192.168.2.102</value>
</item>
```

Figure 5. A field tag example.

  Adding a field creates a new field item and its corresponding value. Updating a field replaces the content of an existing field with the corresponding value that is being displayed on the screen. Updating thusly can cause the loss of other existing value data; therefore, the dictionary editor will confirm with the user before updating an existing field.

  The comma-separated values will also be used for removing multiple values. If the format of “field names = values” is accidentally used, then the tool will process only the left side of the equal character (=). For example, if the expression, `hostA, host[1-3]`, is entered into the **entry** field of the **field** tag and the **Remove** button is pressed, the tool will remove four elements from the dictionary, consisting of four item elements and their sub-elements.

- *The label tag.* Each label defines a namespace. Each namespace can have one or more fields. Each field has at least one value. Adding or removing a label (namespace) affects the label element and its sub-elements.

  Adding a new label creates an additional namespace in the dictionary and requires that all other fields contain valid entries. If the requirement is not met, the tool will pop up an ephemeral window showing the field that has no value or invalid value. Removing an existing label purges all fields defined within the namespace and the **label** field itself. Updating an existing label replaces the existing contents with values that are being displayed. Updating thusly can cause the loss of existing data; therefore, the editor will confirm with the user before performing the requested action.

4.1.2 The Viewing Area

This area is designed to display the entire contents of the internal dictionary. The screen will be updated after the following user actions:

- Loading an existing dictionary into the tool.
- Creating a new dictionary.
- Adding an item to the dictionary.
- Updating an item in the dictionary.
- Removing an item from the dictionary.

The dictionary can be large; therefore, the viewing area of the dictionary editor will provide the user a vertical scrollbar to ease the navigation and viewing of the dictionary.

4.1.3 The RAPR Dictionary XML Tags

The RAPR dictionary editor will handle the XML tags that follow this paragraph. A well-formed dictionary will contain at least one namespace element. Each namespace element will have one label sub-element and at least one item sub-elements. Each item element shall have one or more field sub-elements. Each field element shall have at least one value sub-elements. See figure 6.

```
<RaprDictionary>
    <namespace>
        <label></label>
        <item>
            <field></field>
                <value></value>
        </item>
    </namespace>
</RaprDictionary>
```

Figure 6. Example of RAPR dictionary XML tag format.

4.2 The RAPR Logic-Table Editor

The RAPR logic-table editor will behave similarly to the dictionary editor to minimize the learning curve of the user. The editor will provide the user a convenient way to open a **File** menu to select a menu item that will open an existing logic-table file, create a new one, or save the current logic-table data to a file. The editor will also provide the user a way to learn about the syntax of the logic table or about using the editor to create or modify a logic table. The editor will consist of two areas: (1) editing area and (2) viewing area.

4.2.1 The Editing Area

The editing area will display a list of names corresponding to the XML tags that have been built into the RAPR tool. Each name associates with an entry box into which a new value is defined and entered by the user. The user-defined values will not include any character in the character set “[]<>”. Each name-value pair will have two actions buttons: one to add a new item to the logic table or to update an existing item, and the other to remove an item from the logic table.

Although the RAPR program can handles seven XML tags defining a logic table, the RAPR logic-table editor will interact with the user to obtain four values of the four tags: **value**, **id**, **percent**, and **entry**. The editor will automatically handle the **RaprLogicTable**, **state**, and **logicid** tags, effectively enabling the tags to be hidden from the user. Therefore, the editing area will display the labels of the four tags, their corresponding input entry boxes, and two buttons for each tag according to table 2.

Table 2. User defined input for the RAPR logic-table editor.

value	<user-defined>	Add/Update button	Remove button
id	<user-defined>	Add/Update button	Remove button
percent	<user-defined>	Add/Update button	Remove button
entry	<user-defined>	Add/Edit button	Remove button

The value is associated with a state of the logic table. The table can have more than one state. Each state has one or more **logicid** fields. Each logicid has a **percent** field and one or more **entry** fields. Thus, adding or removing a value affects the entire set of three other fields: **id**, **percent**, and **entry**. Adding a new value creates an additional state in the logic table or replaces the entire contents of an existing state having the same value. The latter action can cause the loss of existing data; therefore, the logic-table editor will ask the user for confirmation before replacing the present contents with the values that are currently displayed on the screen.

Adding or removing an id affects the **percent** and the **entry** fields. Adding a new id creates a new id item and its corresponding **percent** and the **entry** fields if it does not exist, or replaces an existing **id** field and its entire sub tags (**percent** and **entry**) if the **id** field already exists in the same state whose label is being displayed. The latter action can cause the loss of existing data; therefore, the logic-table editor will ask the user for confirmation before replacing the current id with the one that is being displayed.

The value of the **percent** field is an optional parameter. Although its tag is named **percent**, its value is assigned from 0.0 to 1.0*. If the field is not assigned a value, it is assigned a default value of 1.0, indicating 100%. Adding the **percent** field changes its current value; removing it sets the default value.

Adding the **entry** field creates a new **entry** field that is specified under the logicid being displayed if a user-defined value is present; otherwise, the editor will pop up a window to provide the user a way for defining an entry.

4.2.2 The Viewing Area

This area is designed to display the entire contents of the internal logic table. The editor will update the screen after the following user actions:

- Loading an existing logic table into the tool.
- Adding an item to the logic table.
- Changing an item in the logic table.
- Removing an item from the logic table.

The logic table can be large; therefore, the viewing area of the logic-table editor will provide the user a vertical scrollbar to ease the navigation and viewing the dictionary.

4.2.3 The RAPR Logic-Table XML Tags

The RAPR logic-table editor will handle the XML tags that follow this paragraph. A well-formed logic table will have the **RaprLogicTable** tag as its root node. The root node shall have one or more state elements. Each state element has at most a value element. Each value element shall have at least one logicid element. Each logicid element shall have at most one id element, one percent element, and at least one entry element. See figure 7.

*Entering a value between 0.0 and 100.0 is inconsistent with the current RAPR specification.

```
<RaprLogicTable>
    <state>
        <value></value>
            <logicid>
                <id></id>
                    <percent></percent>
                    <entry></entry>
            </logicid>
    </state>
</RaprLogicTable>
```

Figure 7. Example of RAPR logic-table XML tag format.

4.3 The RAPR Script Editor

The script editor is a graphical tool designed for describing a sequence of textual commands and scheduled events that define traffic patterns in a scenario. Each line command or scheduled event is essentially a directive defining a specific behavior of the RAPR tool. The editor will behave similarly to the other two editors to minimize the learning curve of the user. The editor will provide the user a convenient way to open a **File** menu to select a menu item that will open an existing script file, create a new one, or save the current script to a file. The editor will also provide the user a way to learn about the syntax of the script or about using the editor to create or modify a script file. The editor will consist of two areas: (1) editing area and (2) viewing area.

4.3.1 The Editing Area

The editing area will display two rows of editing interfaces. The first row will be used for entering a comment, the second row for entering a command. The action buttons located on the right side of the **entry** fields will behave the same way as they do in the dictionary and the logic-table editors. The editing area will display text labels distinguishing the comment and the command fields, their corresponding entry boxes, and two buttons for each entry box as shown in table 3.

Table 3. Editing area input for the RAPR script editor.

comment	<non-executable statements>	Add/Edit button	Remove button

The **Add/Edit** button will serve two functions. The first function will be to add an input comment into the script, the second will be to pop up a window showing the syntax of a command and a way for the user to compose a command depending on whether the contents of the entry box at the time the button is pressed. If the entry box has a textual command, then the button shall do the first function; otherwise, the editor will perform the second function. The **Remove** button will remove a comment or a command from the script. The editor will also treat the comma character "," as a line separator in both **entry** fields and insert the comment character "#" at the first position of each comment line. Table 4 provides an example of text being typed into the **entry** fields.

Table 4. An example of text being typed into the entry fields.

	Entry Box	**Results**
Comment	abc	#abc
	abc, ijk, xyz	#abc #ijk #xyz
Command	abc def	abc def
	abc, xyz	abc Xyz

4.3.2 The Viewing Area

This area is designed to display the script contents. The editor will update the viewing area after the following user actions:

- Loading an existing script file into the tool.
- Adding a command or a comment into the script.
- Removing a command or a comment from the script.

The script may have a long sequence of comments and commands; therefore, the viewing area of the script editor will provide the user a vertical scrollbar to view any section of the script.

4.3.3 RAPR Script – An Example

Figure 8 shows an example of RAPR script.

```
# log files …
MGENLOG /var/log/mgen-SA.log
RAPRLOG /var/log/rapr-SA.log
#
0.0 STOP 600.0 LISTEN UDP %SA_LISTEN_PORT%
0.0 STOP 600.0 DECLARATIVE UDP SRC %SA_UNICAST_SRC_PORT% DST
%SA_UNICAST_GROUP%/%SA_LISTEN_PORT% %SA_UNICAST_PATTERN%
```

Figure 8. An example of RAPR script.

5. Conclusion

The specifications documented in this report describe not only the functional behaviors, but also the GUIs of a software tool. The process of creating the requirements was iterative and provided an effective way for communicating the needs of the user to the developer. The building and usage of the software prototype facilitated the interaction between the developer and the user, expediting the understanding and agreement of functional requirements. From this, we are able to develop a prototype tool that is capable of generating the files required to create a communications scenario that can be deployed in the WEL MANET environment.

References

1. Nguyen, B. The ARL TOPODEF Tool for Designing Mobile Ad-Hoc Network Topologies to Support Emulation, *Proceedings of the IEEE Military Communications (MILCOM 2007) Conference*, Orlando, Florida, October 2007.

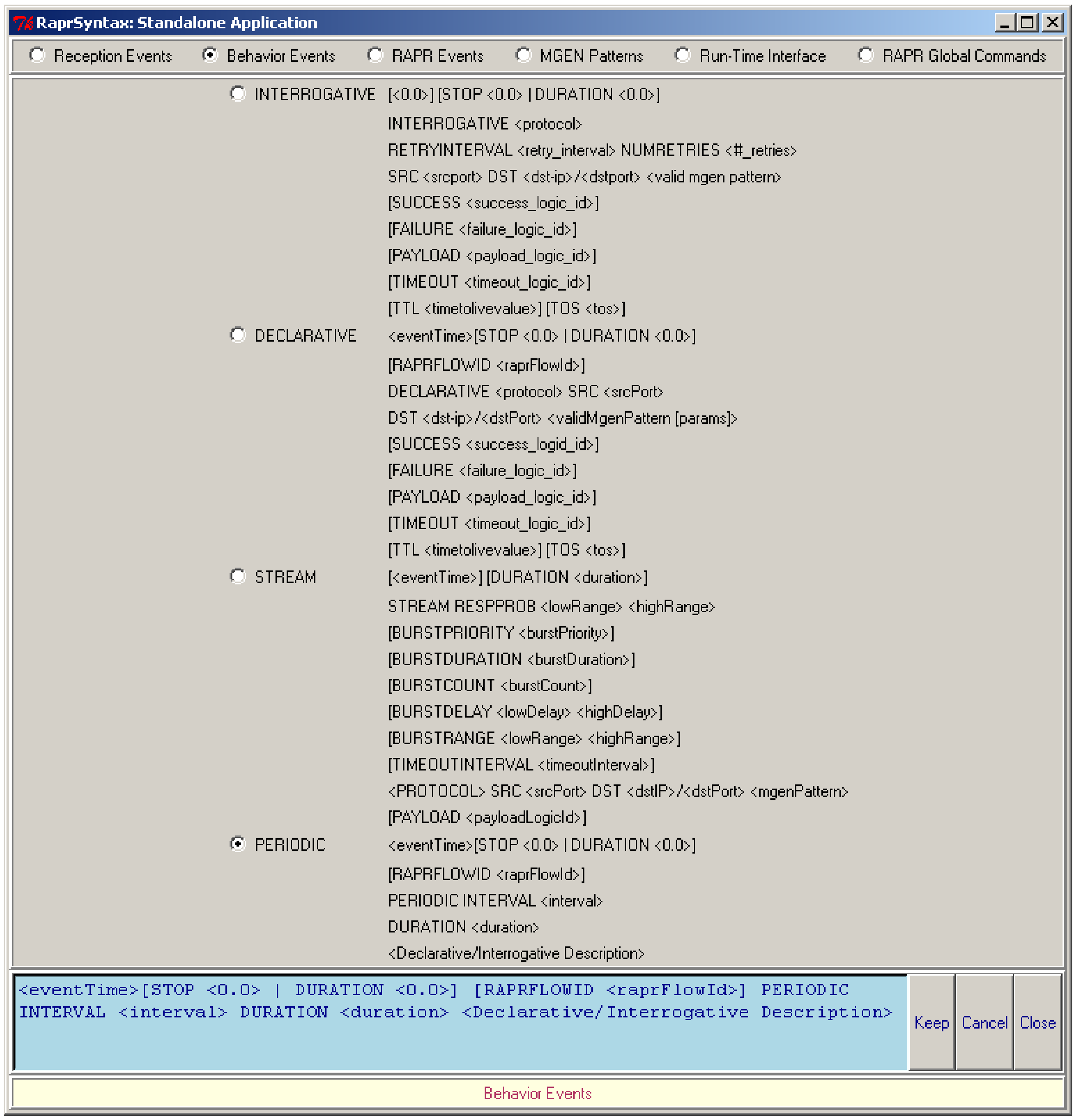
2. The Protocol Engineering Advanced Networking Group, "The Real-time Application Representative (RAPR)," U.S. Naval Research Laboratory (NRL). http://pf.itd.nrl.navy.mil/rapr/rapr.html (accessed 05 November 2007).

3. Ivanic, N; Rivera, B.; Gopaul, R.; Luu, B.; Gwyn, D. Gwyn; R; Hardy; K.; Scott; L.; Tran; G.; Nguyen; B. *A Scalable Test Bed for Emulating Wireless Mobile Ad-Hoc Networks*, submitted to MILCOM 2008, San Diego, CA, 17 November 2008.

4. The Protocol Engineering Advanced Networking Group, "The Multi-Generator (MGEN)," Version 4.2, U.S. Naval Research Laboratory (NRL). http://pf.itd.nrl.navy.mil/mgen/mgen.html (accessed 06 December 2007).

Appendix. Script Syntax

The appendix describes the RAPR syntax that is expressed in prototypical GUIs to complement textual descriptions of requirement, serving as unambiguous specifications.



INTERROGATIVE
[<0.0>] [STOP <0.0> | DURATION <0.0>]
INTERROGATIVE <protocol>
RETRYINTERVAL <retry_interval> NUMRETRIES <#_retries>
SRC <srcport> DST <dst-ip>/<dstport> <valid mgen pattern>
[SUCCESS <success_logic_id>]
[FAILURE <failure_logic_id>]
[PAYLOAD <payload_logic_id>]
[TIMEOUT <timeout_logic_id>]
[TTL <timetolivevalue>] [TOS <tos>]

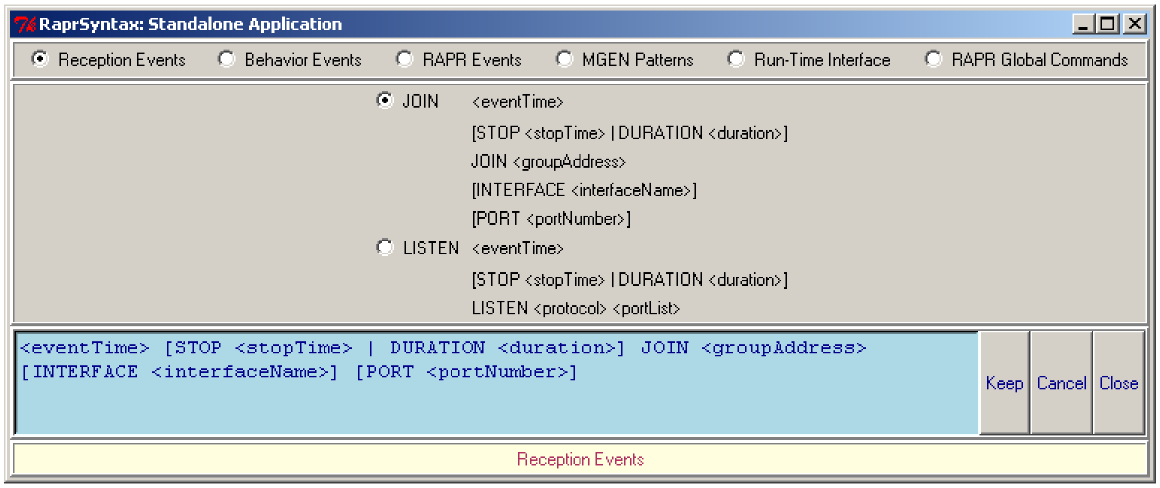
DECLARATIVE
<eventTime>[STOP <0.0> | DURATION <0.0>]
[RAPRFLOWID <raprFlowId>]
DECLARATIVE <protocol> SRC <srcPort>
DST <dst-ip>/<dstPort> <validMgenPattern [params]>
[SUCCESS <success_logid_id>]
[FAILURE <failure_logic_id>]
[PAYLOAD <payload_logic_id>]
[TIMEOUT <timeout_logic_id>]
[TTL <timetolivevalue>] [TOS <tos>]

STREAM
[<eventTime>] [DURATION <duration>]
STREAM RESPPROB <lowRange> <highRange>
[BURSTPRIORITY <burstPriority>]
[BURSTDURATION <burstDuration>]
[BURSTCOUNT <burstCount>]
[BURSTDELAY <lowDelay> <highDelay>]
[BURSTRANGE <lowRange> <highRange>]
[TIMEOUTINTERVAL <timeoutInterval>]
<PROTOCOL> SRC <srcPort> DST <dstIP>/<dstPort> <mgenPattern>
[PAYLOAD <payloadLogicId>]

[✓ PERIODIC]
<eventTime>[STOP <0.0> | DURATION <0.0>]
[RAPRFLOWID <raprFlowId>]
PERIODIC INTERVAL <interval>
DURATION <duration>
<Declarative/Interrogative Description>

```
<eventTime>[STOP <0.0> | DURATION <0.0>] [RAPRFLOWID <raprFlowId>] PERIODIC
INTERVAL <interval> DURATION <duration> <Declarative/Interrogative Description>
```

Figure A-1. Behavior events.

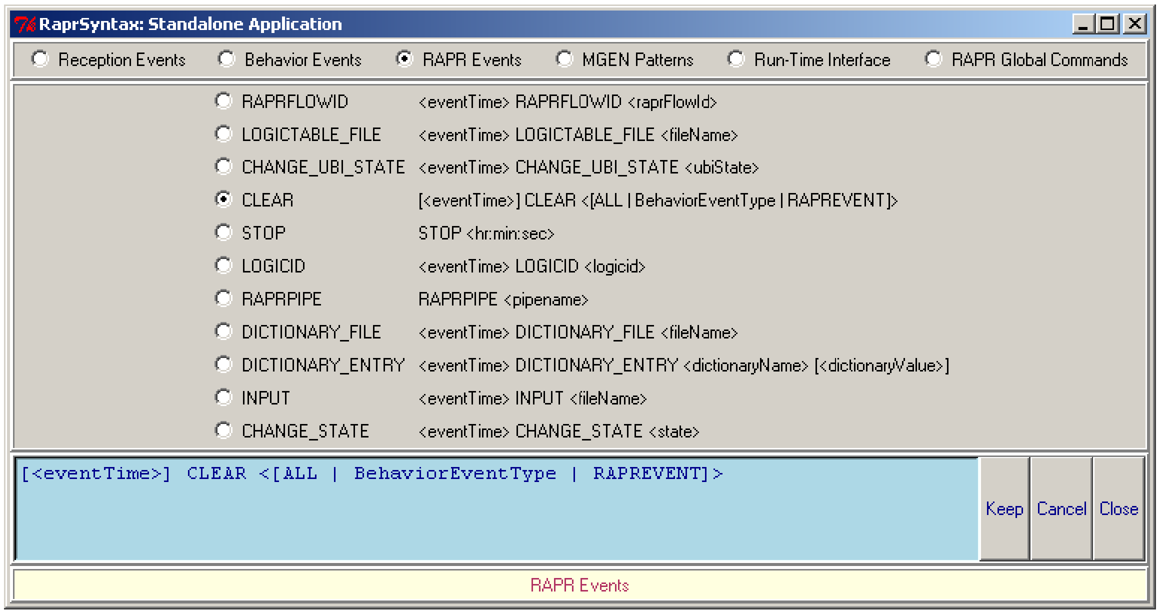

RaprSyntax: Standalone Application

[✓ Reception Events] Behavior Events RAPR Events MGEN Patterns Run-Time Interface RAPR Global Commands

[✓ JOIN]	<eventTime> [STOP <stopTime> | DURATION <duration>] JOIN <groupAddress> [INTERFACE <interfaceName>] [PORT <portNumber>]
LISTEN	<eventTime> [STOP <stopTime> | DURATION <duration>] LISTEN <protocol> <portList>

```
<eventTime> [STOP <stopTime> | DURATION <duration>] JOIN <groupAddress>
[INTERFACE <interfaceName>] [PORT <portNumber>]
```

Figure A-2. Reception events.

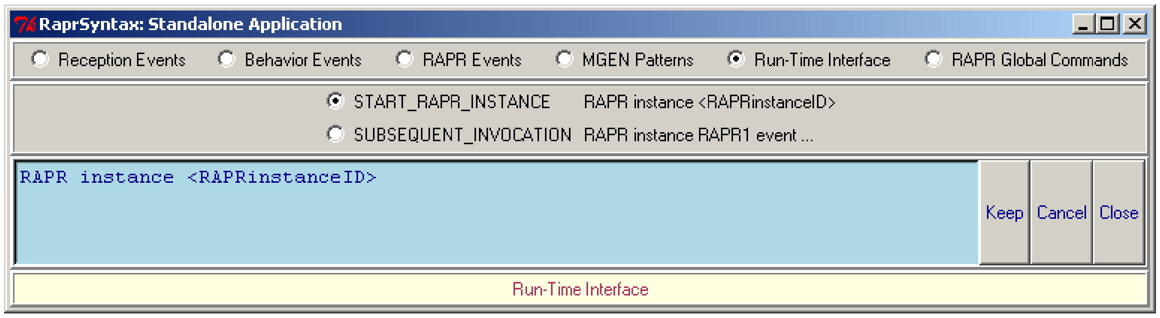

RaprSyntax: Standalone Application

Reception Events Behavior Events [✓ RAPR Events] MGEN Patterns Run-Time Interface RAPR Global Commands

RAPRFLOWID	<eventTime> RAPRFLOWID <raprFlowId>
LOGICTABLE_FILE	<eventTime> LOGICTABLE_FILE <fileName>
CHANGE_UBI_STATE	<eventTime> CHANGE_UBI_STATE <ubiState>
[✓ CLEAR]	[<eventTime>] CLEAR <[ALL | BehaviorEventType | RAPREVENT]>
STOP	STOP <hr:min:sec>
LOGICID	<eventTime> LOGICID <logicid>
RAPRPIPE	RAPRPIPE <pipename>
DICTIONARY_FILE	<eventTime> DICTIONARY_FILE <fileName>
DICTIONARY_ENTRY	<eventTime> DICTIONARY_ENTRY <dictionaryName> [<dictionaryValue>]
INPUT	<eventTime> INPUT <fileName>
CHANGE_STATE	<eventTime> CHANGE_STATE <state>

```
[<eventTime>] CLEAR <[ALL | BehaviorEventType | RAPREVENT]>
```

Figure A-3. RAPR events.

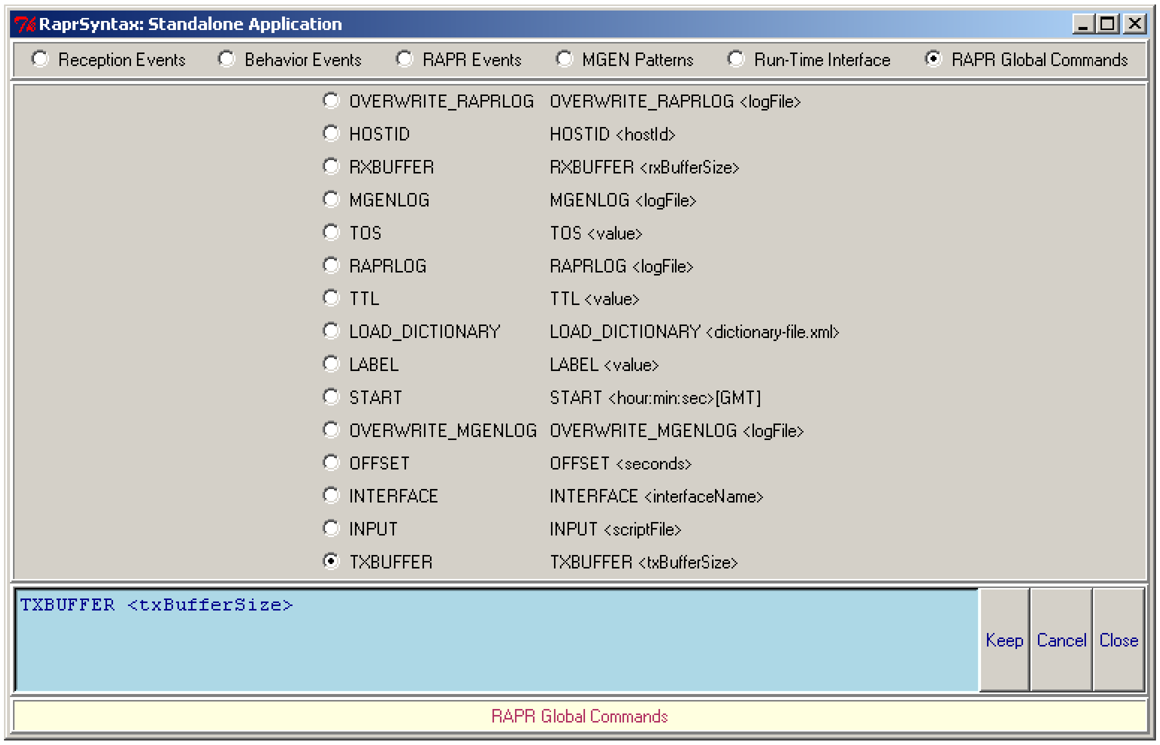

RaprSyntax: Standalone Application

Reception Events Behavior Events RAPR Events MGEN Patterns [✓ Run-Time Interface] RAPR Global Commands

[✓ START_RAPR_INSTANCE]	RAPR instance <RAPRinstanceID>
SUBSEQUENT_INVOCATION	RAPR instance RAPR1 event ...

```
RAPR instance <RAPRinstanceID>
```

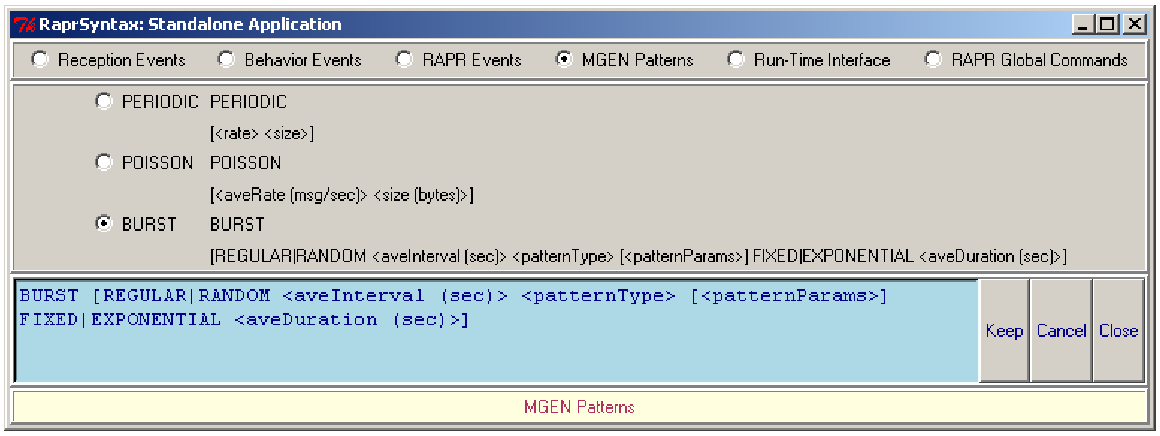
Figure A-4. Run-time interface.



OVERWRITE_RAPRLOG	OVERWRITE_RAPRLOG <logFile>
HOSTID	HOSTID <hostId>
RXBUFFER	RXBUFFER <rxBufferSize>
MGENLOG	MGENLOG <logFile>
TOS	TOS <value>
RAPRLOG	RAPRLOG <logFile>
TTL	TTL <value>
LOAD_DICTIONARY	LOAD_DICTIONARY <dictionary-file.xml>
LABEL	LABEL <value>
START	START <hour:min:sec>[GMT]
OVERWRITE_MGENLOG	OVERWRITE_MGENLOG <logFile>
OFFSET	OFFSET <seconds>
INTERFACE	INTERFACE <interfaceName>
INPUT	INPUT <scriptFile>
[✓ TXBUFFER]	TXBUFFER <txBufferSize>

```
TXBUFFER <txBufferSize>
```

Figure A-5. RAPR global commands.



PERIODIC	PERIODIC [<rate> <size>]
POISSON	POISSON [<aveRate (msg/sec)> <size (bytes)>]
[✓ BURST]	BURST [REGULAR|RANDOM <aveInterval (sec)> <patternType> [<patternParams>] FIXED|EXPONENTIAL <aveDuration (sec)>]

```
BURST [REGULAR|RANDOM <aveInterval (sec)> <patternType> [<patternParams>]
FIXED|EXPONENTIAL <aveDuration (sec)>]
```

Figure A-6. Multi-Generator (MGEN) patterns.

Acronyms

ARL	U.S. Army Research Laboratory
GUIs	graphical user interfaces
NRL	U.S. Naval Research Laboratory
MANET	mobile ad hoc network
MGEN	Multi-Generator
RAPR	Real-time Application Representative
WEL	wireless emulation laboratory
XML	extensible markup language

No. of Copies	Organization
1 (PDF ONLY)	ADMNSTR DEFNS TECHL INFO CTR ATTN DTIC OCP (ELECTRONIC COPY) 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1 CD	US ARMY RSRCH LAB ATTN AMSRD ARL CI OK TP TECHL LIB T LANDFRIED BLDG 4600 APG MD 21005-5066
3 CDs	US ARMY RSRCH LAB ATTN AMSRD ARL CI OK T TECHL PUB ATTN AMSRD ARL CI OK TL TECHL LIB ATTN IMNE ALC IMS MAIL & RECORDS MGMT ADELPHI MD 20783-1197
9HCs 1 CD	US ARMY RSRCH LAB ATTN AMSRD ARL CI NT R HARDY (2 HCs, 1 CD) B NGUYEN L SCOTT K MARCUS N IVANIC R PRESSLEY B RIVERA ATTN AMSRD ARL CI N G RACINE ADELPHI MD 20783-1197
Total:	15 (1 PDF, 9 HCs, 5 CDs)

INTENTIONALLY LEFT BLANK.