DMLEditor

Andreas Hartl
Lehrstuhl Informatik VIII
(Prof. Feldmann)
Technische Universität München
80290 München

hartla@in.tum.de
Contents

1 Introduction 3
  1.1 Motivation .................................................. 3
  1.2 Background — What is DML? .............................. 3

2 Requirements 4

3 Design 5
  3.1 Overview ..................................................... 5
    3.1.1 Main Class ............................................ 5
    3.1.2 Representation of the DML data ...................... 5
    3.1.3 Reading and writing DML files ....................... 5
  3.2 Flexible Representation of DML files ................... 6
  3.3 Elements of a DML file .................................... 6
    3.3.1 Net ..................................................... 6
    3.3.2 Host, Router .......................................... 7
    3.3.3 Link ................................................... 7
    3.3.4 Interface .............................................. 8
    3.3.5 Route .................................................. 9
    3.3.6 Dictionary ............................................ 9
  3.4 Example — modeling a small network using DML ....... 9

4 Using the Editor 11
  4.1 Installation ................................................ 11
  4.2 Main Window ............................................... 11
    4.2.1 The File Menu ........................................ 12
    4.2.2 The Toolbars ......................................... 12
    4.2.3 The Painting Section ................................. 13
  4.3 The network dialog ....................................... 13
  4.4 The peer dialogs ......................................... 13
    4.4.1 The peer dialog ...................................... 13
    4.4.2 The multiple peers dialog .......................... 15
    4.4.3 The peer dialog for dictionary .................... 15
  4.5 The interface dialogs .................................... 16
    4.5.1 The interface dialog ................................ 16
    4.5.2 The multiple interfaces dialog .................... 17
    4.5.3 The interface dialog for dictionary ................ 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6</td>
<td>The link dialogs</td>
<td>18</td>
</tr>
<tr>
<td>4.6.1</td>
<td>The link dialog</td>
<td>18</td>
</tr>
<tr>
<td>4.6.2</td>
<td>The multiple links dialog</td>
<td>18</td>
</tr>
<tr>
<td>4.7</td>
<td>The route dialogs</td>
<td>19</td>
</tr>
<tr>
<td>4.7.1</td>
<td>The route dialog</td>
<td>19</td>
</tr>
<tr>
<td>4.7.2</td>
<td>The route dialog for listing routes</td>
<td>19</td>
</tr>
<tr>
<td>4.7.3</td>
<td>The route dialog for changing/removing routes</td>
<td>19</td>
</tr>
<tr>
<td>4.8</td>
<td>Example — modeling a small network using the editor</td>
<td>20</td>
</tr>
</tbody>
</table>

5 Conclusion and Future Work | 21   |

A Licensing | 22   |
Chapter 1

Introduction

1.1 Motivation

The internet today has become a large place for data exchange. Because the internet is very huge and very complex it is hard to understand how it works.

For researchers it is important to be able to test new features like data transfer or routing to understand more about the internet. Therefore a simulation environment is essential to be able to simulate the structure of large networks. One such simulation environment is SSFNet [1], the Scalable Simulation Framework Network Simulator.

To model a network, SSFNet reads in the model description from a file, a so called DML File (DML = domain modeling language). For a description of DML refer to 1.2. When simulating large network structures, these DML Files grow large and become unintelligible very quickly. So it takes a lot of time to write these files or even to understand what a given file written by other persons simulates or what network structure is simulated.

For relief in this case the SSFNet community needs a graphical editor to build DML Files in a short time and to edit already existing files to analyse their structure in a short time.

1.2 Background — What is DML?

The Domain Modeling Language (DML) [1] is used to describe network models for the SSFNet simulation environment. Each DML file follows a well-defined structure and describes the network(s) and network entities to be simulated.

Each DML file consists of (keyword, value) pairs. Keywords and values are separated by whitespace. The same holds for the separation of the pairs themselves. The keywords are strings and define the meaning of the values. A value can be either a non-whitespace string, or it can be a list of (keyword, value) pairs itself. In the latter case this list is enclosed in brackets. In this way, nesting of lists is also possible.
Chapter 2

Requirements

Before the start of the project, the requirements for a graphical DML editor were defined to be as follows:

**Easy handling** The editor should be easy to handle. For new users, it should not be too difficult to get accustomed to the program. Thus identifiers, names and terms should be the same as in SSFNet and DML.

**Efficient handling** However, the ability to operate the editor swiftly and efficiently should be given greater priority than the needs of new users. The editor should be able to create a large and complex network with few mouse clicks.

**Saving networks as DML files** Of course, the editor should be able to save created networks as DML files.

**Reading existing DML files** The editor should be able to read existing DML files and display them graphically. This also should be the case for DML files that were not created with the editor itself. Parts of the DML file that the editor does not understand should be kept and written again when saving the modified file.

**Extensibility** It should be easy to enhance and extend the editor, e.g. in a follow-up project.

**Portability** The editor should run on all platforms and therefore be written in Java.
Chapter 3

Design

3.1 Overview

The most important design decisions are briefly explained in this section.

3.1.1 Main Class

The central class of the editor is called \textit{DMLEditor}. It contains the \textit{main}-method which creates a new instance of the editor. This class also builds the whole graphical representation of the main window (see \ref{fig:main_window}).

The painting section is represented by an instance of the \textit{DMLNetPainter} class. This class graphically displays the representation of the DML data the editor contains.

3.1.2 Representation of the DML data

The different DML data types (like Hosts, Links, \ldots) are represented by one class for each type, e.g. the class \textit{Link}. An exception is the representation of Hosts and Routers which are summarized into one single class called \textit{Peer}. The fields of the classes represent the different attributes of the DML data types. Which attributes are supported by the editor and which are not will be explained in \ref{subsec:attributes}.

3.1.3 Reading and writing DML files

The writing of a DML file is very easy as each DML data type class includes a method \textit{toDMLString}. This method returns the DML string representation of the object. To write the file, only the \textit{toDMLString}-method of the DML file is called which recursively calls the \textit{toDMLString}-methods of all included DML types.

Reading from an existing DML file is much more complex. The reading is done by the class \textit{DMLFileReader}. It contains a list of keywords and a “main” method \textit{read}. This method reads the file token by token and tests if keywords, unknown strings or numbers are read, thereupon sets corresponding variables and calls helper methods where needed.
3.2 Flexible Representation of DML files

In 1.2 the structure of a DML file was explained. The editor of course has to recognize some important keywords and to interpret their values. However, it would be beyond the scope of a Systementwicklungsprojekt if the editor had to recognize all keywords. Furthermore such a design would make the editor rather unflexible towards future extensions of SSFNet.

Instead, when reading files the editor “keeps in mind” those keyword-value-pairs that it does not understand. These are stored as so-called addOn attributes which can store values as strings without parsing them. The user can edit the contents of these attributes; see section 4.

3.3 Elements of a DML file

Since the editor needs to analyse and interpret existing DML files, this chapter gives a rough overview of DML. The most important elements of a DML file are briefly described below.

3.3.1 Net

A Net is a collection of Hosts, Routers, Links and included Nets (the latter is not yet supported by the editor). A Net contains the following fields the editor understands:

**frequency** Net.frequency is used only in toplevel Net and specifies the time resolution for the simulation. It is automatically computed and set by the editor.

**id/idrange** A Net either must contain a single integer ID or a range of contiguous, ascending integer IDs but not both.

**Host** Net.Host is used to construct a new Host in the current Net. The single fields of a Host are explained in 3.3.2.

**Router** Net.Router is used to construct a new Router in the current Net. The single fields of a Router are the same as in Net.Host. They are explained in 3.3.2.

**Link** Net.Link specifies a physical connection of attached interfaces. The single fields of a Link are explained in 3.3.3.

The following fields also can belong to a Net but they are not understood by the editor. These fields thus will be written to the addOn-section of the Net.

**Net** Net.Net is used to recursively construct subnets.

**alignment** Each Net component may be aligned to a symbolic alignment partition.

**cidr** The optional attribute Net.cidr characterizes a hierarchical partition into CIDR IP address blocks within the enclosing Net context.
The optional attribute `Net.ip` is a variable length subnet mask IP address in a.b.c.d/p notation.

The attribute `Net.randomstream` is used to create and share random numbers to consumers.

The attribute `Net.traffic` is used only in toplevel `Net` to specify the global traffic scenarios for workload generation.

### 3.3.2 Host, Router

The attributes for a `Host` are the same for a `Router`. The following fields are understood by the editor.

**id/idrange** A `Host` must either contain a single integer ID or a range of contiguous ascending integer IDs but not both. Each host or router must have a unique id, i.e. no id value may be used more than once.

**interface** `Host.interface` is used to construct a new `interface` for the current `Host`. The single fields of an `interface` are explained in [3.3.4](#).

**route** `Host.route` is used to construct a new `route` for the current `Host`. The single fields of a `route` are explained in [3.3.5](#).

**nhi_route** `Host.nhi_route` is used to construct a new `nhi_route` for the current `Host`. The single fields of a `nhi_route` are the same as for a `route`. They are explained in [3.3.5](#).

The following fields also can belong to a `Host` but they are not understood by the editor. These fields can be written to the addOn-section of the `Host`.

**alignment** Each `Host` component may be aligned to a symbolic alignment partition.

**graph** `Host.graph` is an internal attribute specifying a list of protocols to be configured.

### 3.3.3 Link

The following fields of a `Link` are understood by the editor.

**attach** A `Link` must have two or more attached network interfaces. Currently, the editor only supports two interfaces to be attached, since SSFNet does not fully support multi-access links. Interfaces are described in [3.3.4](#).

**delay** `Link.delay`, with value in seconds, represents the contribution of the `Link` itself to the total transmission delay of an IP packet.

The following fields also can belong to a `Link` but they are not understood by the editor. These fields can be written to the addOn section of the `Link`. 

---

7
alignment  The attribute Link.alignment causes different synchronization behavior for point-to-point links.

cidr  This attribute is recommended not to be used and let the Net.cidr do the work instead.

ip  This attribute is recommended not to be used and let the Net.ip automatically allocate IP addresses instead.

3.3.4  Interface

The following attributes of an Interface are understood by the editor.

id/idrange  An Interface either must contain a single integer ID or a range of contiguous, ascending integer IDs but not both.

bitrate  Interface.bitrate describes the timing behavior of a stream of packets leaving the Interface. It is given as an integer value (bits per second). The default value is 10 Mbps.

latency  Interface.latency describes the timing behavior of a stream of packets leaving the Interface. It is given as an double value (on-card latency per second) with default value 0.0001 sec. (Note that this latency is different from the link latency.)

            The following fields also can belong to an Interface but they are not understood by the editor. These fields can be written to the addOn-section of the Interface.

ip  Interface.ip is recommended not to be set manually and should be let be created automatically.

virtual  Interface.virtual can be set if the Interface is virtual. This value is optional and false by default.

queue  The optional attribute Interface.queue sets the queue manager class to use.

buffer  The optional attribute Interface.buffer specifies the buffer size in bytes.

monitor  The optional attribute Interface.monitor specifies to use class for the queue monitor.

tcpdump  The optional attribute Interface.tcpdump instructs SSFNet to capture all traffic passing this interface into a tcpdump file.

flaky  The optional attribute Interface.flaky with a float value between 0 and 1 will cause the Interface to drop packets with the given probability value.
3.3.5 Route

The attributes of a Route are the same as of a NHL.Route. The difference is that Route.dest and Route.next_hop must be IP addresses in a.b.c.d/p notation while NHL.Route.dest and NHL.Route.next_hop are an NHI address. All existing fields are understood by the editor.

dest  Route.dest specifies the destination of the route.

interface  Route.interface specifies the outgoing interface to be used for this Route. This attribute is an integer representing the ID of the Host the Route belongs to.

next_hop  Route.next_hop specifies the next hop of the link. It needs not to be provided if Route.interface is given and the interface is a point-to-point-link.

3.3.6 Dictionary

A Dictionary can contain Hosts, Routers and Interfaces. It is useful to write elements that are used very often, e.g. an interface with bitrate 100 MBit, into the Dictionary. Elements in the Dictionary can be accessed by using the keyword extends followed by the specified name of the element in the declaration.

3.4 Example — modeling a small network using DML

To illustrate how DML is used to model networks, I will give a short and very simple example for a DML file here.

Net[
  frequency 100000000
  id 1

  Host[
    id 1
    interface[
      id 1
      bitrate 1000000
      latency 0.1
    ]
  ]

  Host[
    id 2
    interface[
      id 1
      bitrate 10000000
      latency 0.1
    ]
  ]
]
Router[
  id 10
  interface[
    id 1
    bitrate 1000000
    latency 0.1
  ]
  interface[
    id 2
    _extends 100MBit
  ]
]

Link[attach 1(1) attach 10(1) delay 0.5]
Link[attach 2(1) attach 10(2) delay 0.3]

Dictionary[
  100MBit[
    bitrate 100000000
    latency 0.01
  ]
]

This DML file first creates a new network with id 1. The frequency is equivalent to the highest bitrate of an interface used in this network, here 100 Mbps. The network contains two hosts with ids 1 and 2, each host containing one interface with id 1.

Then a router with id 10 and two interfaces is created with the second interface extending the 100 Mbps defined in the Dictionary.

Finally, two links are created. The first connects host 1’s interface 1 with router 10’s interface 1. The second connects host 2’s interface 1 with router 10’s interface 2. Both links are delayed by the specified time.
Chapter 4

Using the Editor

4.1 Installation

The editor is shipped in an archive file that contains the following components:

- The source code and class files of the de.tum.in.net.dmlEditor package.
- The javadocs of this package.
- The executable archive DMLEditor.jar.
- The Makefile to recompile, archive, remake javadocs and start the editor.

The following making options are available.

**make all** / **make**  This includes recompiling, making new javadocs and building a new executable archive but **not** starting the editor.

Please note that for compiling and execution, the editor needs at least Java Version 1.4.

**make start**  This target starts the editor using the archive file DMLEditor.jar. Make sure that this target is updated to run newer versions of the editor.

**make jar**  This target recompiles the sources in the de.tum.in.net.dmlEditor package and updates the executable archive file DMLEditor.jar.

**make docs**  This target is used to update the javadocs. The docs are written into the docs directory.

4.2 Main Window

After starting the editor with **make start** or java de.tum.in.net.dmlEditor.DMLEditor, the main window is displayed.

The main window is divided into three parts: the toolbar, the painting section and a menu bar at the top.
4.2.1 The File Menu

The File Menu offers the following commands.

**New**  A new simulation scenario is created. As it does not contain any network components yet, the Painting Area (see 4.2.3) will be empty at first.

**Open...**  An existing DML File will be read, its content will be interpreted and displayed graphically in the painting area.

**Save/Save as...**  *Save* saves the file under its current name. If the name is *default*, a new name must be chosen. *Save as* first asks the user for a file name under which it then will save the network scenario as a DML file.

**Exit**  Terminates the program.

4.2.2 The Toolbars

The Toolbars contain several buttons described here.

**Create new Network**  Opens a new network dialog (4.3).

**Create new Peer**  Opens a new peer dialog (4.4).

**Create multiple Peers**  Opens a new multiple peers dialog (4.4).
Remove Peer(s)  Removes those peer(s) that have been marked by the user (see 4.2.3).

Create new Link  Before using this option, the two peers that are to be connected must be marked with the mouse. A new link dialog (4.6) will open.

Remove Link  Removes a marked link.

Edit Dictionary Peer  Opens a new peer dialog for a peer contained in the dictionary (4.4).

Edit Dictionary Interface  Opens a new interface dialog for an interface contained in the dictionary (4.5).

4.2.3 The Painting Section

In the Painting Section the different components of the network are displayed graphically. Hosts are painted in blue, routers are painted in red and links are painted in black. Components can be marked/unmarked by a single mouse click and edited by a double click. Marked components are painted in green. Currently, at most two components can be marked simultaneously. To move a Host or Router, click and hold the mouse on the component to move, and drag it to the desired position. Links between components automatically adjust to new component positions.

![Network Dialog](image)

Figure 4.2: Network Dialog

4.3 The network dialog

The network dialog only contains a field to enter the ID of the network.

4.4 The peer dialogs

4.4.1 The peer dialog

The peer dialog contains various fields to enter the desired properties of the peer.

Network ID  The id of the network that the peer shall belong to can be chosen using the popup menu. The menu contains all networks that exist at this time. If no network has been created yet, the value None available will be shown and a new network will be created automatically on submitting the settings of the peer.

ID of Peer  In this text field, the id of the peer can be defined. If no id is entered, a valid default value will be used.
Type of Peer  The type of the peer can be chosen between the values Host and Router with this menu selection.

Extends  If desired, the user can specify a peer from the dictionary that should be used as a prototype for this peer. The menu contains all existing peer entries of the dictionary. If a value other than None is chosen, then most fields of the peer cannot be changed any more because they are defined by the extended peer.

Number of interfaces  The number or interfaces desired for this peer can be specified here. If no value is defined, a default value of 1 will be used.

Modify all interfaces  For assigning the same parameters to several interfaces (apart from the IDs), click this button. A new multiple interfaces dialog (4.5) will be opened.

Modify interface  To activate this button change the value of the neighboring menu to the interface-id that shall be modified. If there is no value to change, make sure that the Number of Interfaces-Field is not empty. A new interface dialog (4.5) will be opened.

Define Route  To define a new Route, click this button and a new route dialog (4.7) will be opened.

List all Routes  To look at a list of all routes, click this button and a new route dialog for listing routes (4.7) will be opened. From there routes can be changed or removed.

Default route interface  This menu list contains the IDs of all interfaces the peer has. To set the default route interface, select the desired interface id.
**AddOn**  Any additional configuration data can be entered here as raw DML code. It will be written in the DML file into the section of the peer.

![Create multiple peers](image)

4.4.2 The multiple peers dialog

The multiple peers dialog is similar to the peer dialog. Fields not described here bear the same functionality as those described in the peer dialog.

**Number of peers**  The number of peers to be created simultaneously can be entered here.

**ID to start**  The lowest of the ascending IDs can be entered here.

**Link to peer**  It is possible to create multiple links to one common peer. By selecting the ID of the desired peer from this list, a new multiple links dialog (4.6) will be opened.

4.4.3 The peer dialog for dictionary

The peer dialog for a dictionary is similar to the peer dialog. Fields not described here bear the same functionality as those described in the peer dialog.

**Entry to edit**  This menu contains all names of peer entries of the dictionary that are defined at this time. Select a name to edit an entry or select *New Peer* to define a new peer entry.

**Name of peer**  The new peer to be created can be given a name using this text field. If no name is entered, a default name is used.
4.5 The interface dialogs

4.5.1 The interface dialog

The interface dialog contains various fields to enter the desired properties of the interface. The ID(s) is (are) set automatically when submitting.

**Bitrate**  In this text field the bitrate of the interface can be entered. The unit is the same as in a DML file.

**Latency**  In this text field the latency of the interface can be entered. The unit is the same as in a DML file.

**Extends**  Here, an interface from the dictionary can be chosen as parent for this interface. The menu lists all interfaces currently existing in the dictionary. If a value other than *None* is chosen, most fields of the interface cannot be changed any more because they are provided by the parent interface from the dictionary.
**AddOn** Any additional configuration data can be entered here as raw DML code. It will be written in the DML file into the section of the interface.

![Figure 4.7: Multiple Interfaces Dialog](image)

### 4.5.2 The multiple interfaces dialog

The multiple interfaces dialog contains the same fields as the interface dialog described above.

![Figure 4.8: Interface Dialog for Dictionary](image)

### 4.5.3 The interface dialog for dictionary

The interface dialog for dictionary is similar to the interface dialog. Fields not described here bear the same functionality as those described in the interface dialog.

**Entry to edit** The name of the interface entry to be edited can be selected here. Choosing *New Interface* will create a new interface entry in the dictionary.

**Name of interface** In this text field the desired name of the interface entry to be created can be entered. If no name is entered, a default name will be used.
4.6 The link dialogs

4.6.1 The link dialog

The link dialog contains various fields to enter the properties of a link.

Interface of Peer  This menu is used to select the interface number of the peer that the link shall be attached to.

Delay  In this text field the delay of the link can be specified. The unit is the same as in a DML file.

AddOn  Any additional configuration data can be entered here as raw DML code. It will be written in the DML file into the section of the link.

4.6.2 The multiple links dialog

This dialog only can be reached from the multiple peers dialog (4.4). The multiple links dialog contains the same fields as the link dialog. However, instead of choosing the two peers’ interfaces in the link dialog, here the multiple peers’ interface and the single peer’s interface to be attached are selected.
4.7 The route dialogs

4.7.1 The route dialog

This dialog contains various fields to enter the properties of a route.

Route for Interface  With this menu, the outgoing interface to be used for this route can be selected.

Destination  In this text field the destination host of the route can be entered.

Next Hop  In this text field the next hop address of the route can be entered.

AddOn  Any additional configuration data can be entered here as raw DML code. It will be written in the DML file into the section of the route.

4.7.2 The route dialog for listing routes

This dialog presents a list that shows all interfaces and the routes defined on them. By pressing on a button the according route can be edited or removed.

4.7.3 The route dialog for changing/removing routes

This dialog is similar to the route dialog. The fields have the same function as with the route dialog. The only difference is the Remove this route button. Clicking on it effects in deleting the route.
4.8 Example — modeling a small network using the editor

To show a short example of creating a small network using the editor, we want to reproduce the DML File of 3.4. Take the following steps to create this DML File:

1. (optional) Create a new network with id 1. Click on the Create new Network button, enter the id and submit the input. If this step is not performed, a new network with id 1 will be created in step 2 automatically.

2. It is recommended to fill the dictionary with elements first. In this example only an interface is to be defined. Click on the Edit Dictionary Interface button, fill the fields with the according values and submit the input.

3. Create a new host with id 1. Click on the Create new Peer button. Set the id to 1 and set the number of interfaces to 1.

4. Create one interface for host 1. Click either on the Modify all interfaces button or set the menu at interface 1 and click the Modify interface button. In the dialog that will appear, enter the according values of the interface as described in 3.4 and submit the input.

5. For the second Host with id 2, repeat steps 3 and 4 with modified values.

6. Create a new router with id 10. Click on the Create new Peer button. The difference to hosts only lies in the change of the list label Type of Peer to the label Router. The rest is just like with the hosts, except for the definition of a second interface. To define this second interface, set the menu Extends to the value 100MBit.

7. To define a link, first mark the two elements which it shall attach to. In our example, these are host 1 and router 10. Click on the Create new link button and select from the lists the according interfaces to be linked. After the selection, a line (=link) between Host 1 and Router 10 should appear.

8. For the second link repeat the previous step with modified values.

9. Save the file.

After saving, the file should look like the file given in 3.4. In the graphical presentation, the peers can be moved by dragging the mouse to the desired place. The links can be distinguished better in this example if e.g. Host 2 is moved a bit to the right.
Chapter 5

Conclusion and Future Work

Concluding it is to say that the editor is able to model large networks in a fraction of the time that you would need to write the file manually. It is easy to handle and it should be able to read most existing DML Files. But of course it is not perfect and can be extended, e.g. to make accessible more features of SSFNet via dialogs. Here is a short list of possible future work:

- Native support of more SSFNet/DML features, e.g. protocols, nets that are nested within other nets, and other components and properties that currently only are supported as DML fragments entered by the user (i.e., as "addOn" sections).
- The design of the graphical interface could be enhanced to make it look more appealing, e.g. using icon images etc.
- Duplicate ID numbers of networks, peers or interfaces should be found and reported to the user.
- Reading files could be eased by using a lexical analyser, e.g. JLex, JFlex. In particular, this should be considered if extensions require the recognition of many more keywords.
- The arrangement of graphically displayed peers and links could be improved. However, displaying graphs in a clear and concise fashion is not an easy problem; it still is subject to research. Thus the complexity of the task to include such a feature in the editor should not be underestimated.
Appendix A

Licensing

The editor stands under the GNU General Public License (GPL). Basically, this means that everyone is allowed to copy the source code and use it how he wants, but the editor always has to stand under the GPL-License when re-published. The full license can be found under http://www.opensource.org/licenses/gpl-license.php.
Bibliography


[2] DML-Editor javadocs (see docs/index.html).