Möbius Shell: A Command-Line Interface for Möbius

Ken Keefe and William H. Sanders

University of Illinois, Urbana, IL 61801, USA
kjkeefe@illinois.edu whs@illinois.edu
https://www.mobius.illinois.edu

Abstract. The Möbius modeling environment is a mature, multi-formalism modeling and solution tool. Möbius provides a user-friendly graphical interface for creating discrete-event models, defining metrics, and solving for the metrics using a variety of solution techniques. For certain research needs, the graphical interface can become a limiting use pattern. This paper describes recent work that adds a comprehensive text-based interface for interacting with the Möbius tool, called the Möbius Shell. The Möbius Shell provides an interactive command shell and scriptable command language that can leverage all the existing and future features of Möbius.

Keywords: text-based interface, multi-formalism modeling, simulation, analytical solution, discrete-event systems.

1 The Möbius Modeling Environment

The Möbius Modeling Environment is an extensible modeling and solution tool. It offers a variety of existing modeling formalisms, including compositional modeling formalisms, a metric specification formalism that allows for time- and event-based rewards, global model parameterization with several means of defining experiments, and a set of analytical and simulation solution methods [1][2]. Until recently, Möbius has provided only a graphical user interface for working with each of its components. With the addition of the Möbius Shell, Möbius now offers a text-based user interface that enables an interactive or scriptable method of performing actions in the top-level tool or within one of the Möbius components.

2 Using the Möbius Shell

In the Möbius installation directory, the `mobius` executable launches the graphical version of the tool. The text-based, interactive version of Möbius can be launched using the `mobius-shell` executable:

```
$ mobius-shell
Welcome to Mobius 2.4.1!
  Enter "help" for a list of commands.
Mobius>
```

K. Joshi et al. (Eds.): QEST 2013, LNCS 8054, pp. 282–285, 2013.

Commands can then be executed at the `Mobius>` prompt. Each time a command is executed, text feedback is provided. Long-running jobs, such as the execution of a simulation, will provide continuous feedback and can be interrupted by hitting `Ctrl+C`.

Alternatively, the Möbius Shell can execute a script either by using a command pipe or by passing the script file path using a command switch:

```
$ cat myScript.txt | mobius-shell
$ mobius-shell -s myScript.txt
```

3 Key Möbius Shell Commands

The Möbius Shell is intended to be a full-fledged alternative to the traditional graphical user interface. Because of space limitations, here we detail only a few important commands. For a full treatment of the Möbius Shell command language, see the 2.4.1 (or later) version of the Möbius Manual [3].

3.1 Help

The Möbius Shell provides a comprehensive, integrated help system. Users can obtain a list of all available commands by executing the *help* command.

```
Mobius> help
Mobius Shell Command Help
Further help can be found for each command by executing:

help <command>
  archive    - Archive a project
  clean      - Clean a project or model component
  ...
```

Detailed help for each command can be accessed by including the command as an argument to the `help` command:

```
Mobius> help save
Generate and compile a model component command:

save <project name> (a|c|r|y|t|s) <component name>
    a  - Atomic model type
    c  - Composed model type
    r  - Reward model type
    y  - Study type
    t  - Transformer type
    s  - Solver type
```

3.2 Generate, Compile, and Save

When a user saves a model component in Möbius, the first step that Möbius performs is generation of a C++ representation of the model component. That typically consists of a set of classes that derive from base classes in the Möbius code library[1]. Next, Möbius compiles those classes and links them to code library archives that come with Möbius.

In the Möbius Shell, those steps can be performed individually or combined, as in the graphical tool. To generate the C++ representation of a model component, use the *generate* command. To compile the C++ representation, use the *compile* command. To do both, use the *save* command. For example, the below commands generate and compile the reward model called "perfEx" in the "satRelay" model.

```
Mobius> generate satRelay r perfEx
Generating code..............Done!
Mobius> compile satRelay r perfEx
make: Entering directory '/home/kjkeefe/MobiusProject/satRelay/Reward/perfEx'
make lib TARGET=libperfExPV_debug.a OBJS="perfExPVNodes.o perfExPVModel.o "
...
make: Leaving directory '/home/kjkeefe/MobiusProject/satRelay/Reward/perfEx'
Compile completed: SUCCESS
```

3.3 Run

The *run* command begins the execution of a transformer, analytical solver, or simulator. When a transformer or analytical solver is run, the feedback in the Möbius Shell is a summary, and the results are stored in a file in the component's directory. However, when a simulator runs, an aggregation of reward variable statistics is reported on the fly until all variables have converged within their defined confidence intervals, or until some other ending condition has been met (e.g., max number of iterations simulated). Those behaviors mirror those of the graphical version of the tool.

```
Mobius> run satRelay t AvNumSSG
Building State Space Generator for Linux architecture
Building for Linux systems on darboux
...
Generated: 8190 states
Computation Time (user + system): 2.160100e-01 seconds
State Generation of Experiment_3 on model AvNumSSG finished at Wed Mar 06 21:13:11
CST 2013.
```

3.4 Edit

The *edit* command allows the user to step into project components. For model components that have nested child elements (e.g., a Stochastic Activity Network (SAN) [4] model containing input gates and activities), the *edit* command can further step into those elements to make changes to their attributes (e.g., input predicate, firing distribution).

In the following example, we start by editing the "cpu_module" SAN model in the "Multi-Proc" project (which is included in the standard set of examples that come with Möbius). On line 3 we execute the *show* command to get a brief summary of this atomic model. Next, on line 10, we ask for further details on the activities in this SAN model. We could get a complete description of the "cpu_failure" activity by using the *show activity* command. On line 13 we begin editing the "cpu_failure" activity. We start by showing the details of the timing distribution. Next, on lines 19, 21, and 23, we alter the timing distribution type, mean, and variance, respectively. Having made the desired changes, we close

that activity on line 25. Finally, we close the "cpu_module" SAN model. Möbius Shell then asks us if we would like to save our changes, which we do, and the model compiles successfully.

```
1  Mobius> edit Multi-Proc a cpu_module
2  Now editing the cpu_module SAN Atomic Model (enter the "close" command when finished)...
3  Multi-Proc/Atomic/cpu_module> show
4  Model contains 10 elements and 18 connections:
5    1 activity (0 instantaneous, 1 timed)
6    0 extended places
7    1 input gate
8    3 output gates
9    5 places
10 Multi-Proc/Atomic/cpu_module> show activities
11 Model contains 1 activity:
12   cpu_failure (timed, incoming: Input_Gate1, outgoing: Case 1: OG1, Case 2: OG2, Case 3: OG3)
13 Multi-Proc/Atomic/cpu_module> edit cpu_failure
14 Now editing the cpu_failure Timed Activity (enter the "close" command when finished)...
15 Multi-Proc/Atomic/cpu_module/cpu_failure> show timing
16 Timing distribution: Exponential
17 Parameters:
18   Rate: 6.0 * failure_rate * cpus->Mark()
19 Multi-Proc/Atomic/cpu_module/cpu_failure> set timing distribution Normal
20 Timing distribution set to Normal (Mean: , Variance: )
21 Multi-Proc/Atomic/cpu_module/cpu_failure> set timing Mean "failure_rate * cpus->Mark()"
22 Timing distribution set to Normal (Mean: failure_rate * cpus->Mark(), Variance: )
23 Multi-Proc/Atomic/cpu_module/cpu_failure> set timing Variance "0.1"
24 Timing distribution set to Normal (Mean: failure_rate * cpus->Mark(), Variance: 0.1)
25 Multi-Proc/Atomic/cpu_module/cpu_failure> close
26 Closing the cpu_failure Timed Activity (changes will not be saved until atomic model is saved)...
27 Multi-Proc/Atomic/cpu_module> close
28 You have unsaved changes, would you like to save? (Y|n) y
29 Generating code..............Done!
30 make: Entering directory '/home/kjkeefe/MobiusProject/Multi-Proc/Atomic/cpu_module'
31 ...
32 Compile completed: SUCCESS
33 Mobius>
```

Acknowledgments. The authors would like to acknowledge the current and former members of the Möbius team and the outside contributors to the Möbius project. The authors would also like to thank Jenny Applequist for her editorial work.

References

1. Deavours, D.D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J.M., Sanders, W.H., Webster, P.G.: The Möbius framework and its implementation. IEEE Transactions on Software Engineering 28(10), 956–969 (2002)
2. Doyle, J.M.: Abstract model specification using the Möbius modeling tool. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois (January 2000)
3. Möbius Team: The Möbius Manual. University of Illinois at Urbana-Champaign, Urbana, IL (2013), `http://www.mobius.illinois.edu`
4. Sanders, W.H., Meyer, J.F.: Stochastic activity networks: Formal definitions and concepts. In: Brinksma, E., Hermanns, H., Katoen, J.-P. (eds.) FMPA 2000. LNCS, vol. 2090, pp. 315–343. Springer, Heidelberg (2001)