Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

Setting up Simulation Experiments with SESSL

Roland Ewald
Adelinde M. Uhrmacher

Albert Einstein Str. 22
University of Rostock
18059 Rostock, GERMANY

ABSTRACT

Setting up simulation experiments is hard, even more so as simulation systems usually offer only custom interfaces for this task (e.g., a graphical user interface or a programming interface). This steepens the learning curve for experimenters, who have to get accustomed with the idiosyncrasy of each simulation system they want to experiment with. It also makes cross-validation experiments between simulation systems cumbersome, since the same experiment needs to be set up for each system from scratch. In the following, we give a brief overview of SESSL, a domain-specific language for simulation experiments. SESSL addresses these issues by providing a common interface to set up simulation experiments in a more declarative manner, i.e., specifying *what* to do, not *how* to do it. Therefore, SESSL can also be used for documenting and reproducing simulation experiments.

1 SESSL: SIMULATION EXPERIMENT SPECIFICATION VIA A SCALA LAYER

SESSL is an embedded domain-specific language based on the SCALA programming language (Odersky, Spoon, and Venners 2011), i.e., it provides a SCALA-based programming interface that has *"'[...] the feel of a custom language [...]"'* (Fowler 2010). We tried to keep the interface of SESSL as general as possible, so that it can be used to configure various kinds of simulation experiments for various kinds of simulation systems. Additionally, we identified several separate experiment *aspects* that can be configured if need be (and the given simulation system supports them). These aspects include, for example, means to configure the parallel execution of an experiment, the simulation output that shall be observed, or the post-processing of the simulation output to generate result reports. Support for some of these aspects can also be used *across* simulation systems, e.g., SESSL allows to apply the result reporting mechanism of JAMES II to the output of any other simulation system supporting SESSL.

To use a simulation system through SESSL, a corresponding *binding* has to be developed. A binding has little more to do than to configure the simulation system at hand with the model and its parameters specified by the experimenter, and to notify SESSL after finishing the execution of a simulation run. As SCALA is fully compatible with Java, developing such bindings is straightforward for any Java-based simulator.

Figure 1 shows a sample experiment specification that illustrates some features of SESSL. From a technical viewpoint, the `execute` function (line 4) is invoked with an instance of an anonymous sub-class of type `Experiment` (l. 5), which is augmented to configure a parallel execution via mix-in composition (using the `with` keyword, line 5). The actual specification of the experiment (line 6–15), e.g., specifying the model to be executed or when to stop a simulation run, is the *constructor* of the anonymous sub-class, which in SCALA is simply written into the class body. As SCALA is statically typed, experimenters will also receive immediate feedback (in form of compiler errors) in case their experiment specification is invalid, e.g., because they try to configure the random number generator (line 12) with a simulation algorithm.

978-1-4673-4781-5/12/$31.00 ©2012 IEEE

```
1  import sessl._ // use SESSL language constructs
2  import sessl.james._ // use JAMES II to execute SESSL experiment
3
4  execute { // execute experiment
5    new Experiment with ParallelExecution { // create experiment
6      model = "sampleModel.file" // use model stored in this file
7      // complex stopping and replication conditions are supported:
8      stopCondition = AfterSimTime(0.6) and
9                      (AfterWallClockTime(seconds = 30) or AfterSimSteps(10000))
10     // however, simplified constructs are available for typical cases, e.g.:
11     replications = 2
12     rng = MersenneTwister(1234) // use random number generator with given seed
13     parallelThreads = -1 // exploit parallelism, but leave one core idle
14     set("answer" <~ 42) // set model parameter to fixed value
15     scan("x" <~ (1, 2), "y" <~ range(1, 1, 10)) // define factorial experiment
16   }
17 }
```

Figure 1: A sample SESSL experiment, using the binding for JAMES II. The meaning is described by comments (green, starting with `//`), which would usually be left out as the syntax is rather self-explanatory.

By putting the experiment specification (line 6–15) into a trait (Odersky, Spoon, and Venners 2011, p. 217 et sqq.) instead of a class, one can also specify experiment setups in a system-independent way, and re-use them for cross-validation. Currently, bindings for JAMES II (Himmelspach and Uhrmacher 2007), SBMLSIMULATOR (Dräger et al. 2012), and OMNET++ (Varga 2011) are available. SESSL is open source (Apache 2.0 license); see http://sessl.org for details.

ACKNOWLEDGMENTS

This research has been supported by the DFG (German Research Foundation), via research training group 1387 (*dIEM oSiRiS*) and research project `EW 127/1-1` (*ALeSiA*).

REFERENCES

Dräger et al. 2012, March. "SBMLsimulator: An efficient Java(tm) solver implementation for SBML". http://www.ra.cs.uni-tuebingen.de/software/SBMLsimulator, accessed 6/2012.

Fowler, M. 2010, October. *Domain-Specific Languages*. 1st ed. Addison-Wesley Professional.

Himmelspach, J., and A. M. Uhrmacher. 2007. "Plug'n simulate". In *Proceedings of the 40th Annual Simulation Symposium*, 137–143: IEEE Computer Society.

Odersky, M., L. Spoon, and B. Venners. 2011, January. *Programming in Scala*. 2nd ed. Artima.

Varga, A. 2011. *OMNeT++ User Manual Version 4.2.2*. OpenSim Ltd. Last accessed 6/2012, http://www.omnetpp.org/doc/omnetpp/Manual.pdf.