Universität Rostock

Traditio et Innovatio

Practical expressiveness of internal and external domain-specific modeling languages

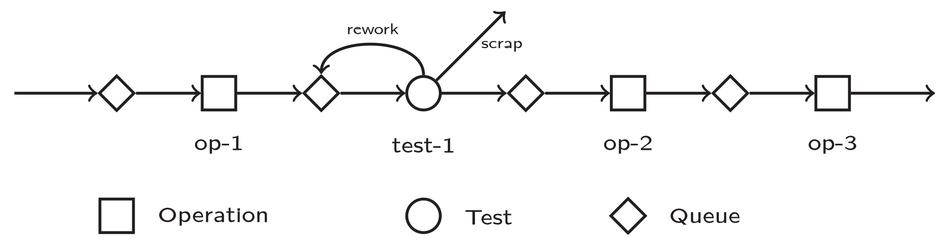
Tom Warnke & Adelinde M. Uhrmacher

Motivation

During the long history of modeling and simulation, many answers have been given to the question of how to specify simulation models. Many of these approaches can be perceived as domain-specific modeling languages (DSML) offering a syntax and a semantics. However, the individual languages are often vastly different. A central distinguishing aspect is the classification as external or internal domain-specific language. We illustrate the influence of using an external or internal language on different aspects of language performance, in particular the practical expressiveness, one of the central properties of modeling languages.

Example Supply Chain Model (after Persson and Olhager 2002)

- Items flow through operations and tests, ending up as a finished product
- Each operation takes log-normally distributed time to complete
- Each test can succeed or fail
- After a test failure, the item is either scrapped or reworked and tested again, which takes log-normally distributed time to complete
- Items are buffered in a queue in front of every operation and test



Developing and using an external DSML

Define a grammar

```
operation: O COLON name operationDuration;
test: T COLON name failureProb SCRAP scrapProb REWORK reworkDuration;
```

↓

Generate a parser from the grammar in some programming language (Tools: XText/XTend, ANTLR, etc.)

↓

Implement generation of code that represents a parsed model **or** Implement the creation of structures that represent a parsed model

↓

Let the modeler define the model according to the grammar

```
O: op1 (0.4, 0.1)
T: op1-test (0.05) scrap (0.5) rework (0.1, 0.05)
O: op2 (0.4, 0.1)
O: op3 (0.4, 0.1)
```

↓

Feed generated code to the simulator / Feed created objects to the simulator

Developing and using an internal DSML

Program an interface in a host language (e.g., Scala)

```
def operation(name: String)(location: Double, shape: Double): Step = {
  val o = new Operation(name, location, shape)
  addStep(o)
}
```

↓

Implement classes to represent a model

↓

Let the modeler define the model using the interface

```
val m = new Model {
  operation("op1")(0.4, 0.1)
  test("op1-test")(0.05) scrap (0.5) rework(0.1, 0.05)
  operation("op2")(0.4, 0.1)
  operation("op3")(0.4, 0.1)
}
```

↓

Feed the model to the simulator

```
Simulator.execute(m)
```

External DSMLs are practically expressive because

- they do not impose constraints when designing the language
- they define and document limits of their syntax and semantics
- they facialitate defining formal semantics

Internal DSMLs are practically expressive because

- they allow reusing the host language's ecosystem (editors, libraries, compilers)
- they allow modelers to implement model components in the host language
- they reuse existing syntax and semantics

Conclusion

- Domain-specific modeling languages imply strict separation of model and simulation
- The simulation algorithm to use is completely independent from the choice of language
- DSMLs speed up modeling (= human time) rather than simulation (= computer time)
- Internal as well as external domain-specific languages can be interpreted and compiled
- Which approaches is more suitable depends heavily on the application domain

References

Persson, F., and J. Olhager. 2002. “Performance simulation of supply chain designs”. International Journal of Production Economics 77 (3): 231 – 245.

Van Deursen, A., P. Klint, and J. Visser. 2000. “Domain-specific languages: An annotated bibliography”. Sigplan Notices 35 (6): 26–36.

Felleisen, M. 1991. “On the expressive power of programming languages”. Science of Computer Programming 17 (1): 35 – 75.

Tom Warnke | tom.warnke@uni-rostock.de

Modeling and Simulation Group

University of Rostock

mosi.informatik.uni-rostock.de