Adding a HenshinEngine to GEMOC Studio

An experience report
Steffen Zschaler, with thanks to Erwan Bousse and Tanja Meyerhöfer
October 15th, 2018
Executable Domain-Specific Modelling Languages

- Parser/Unparser (Xtext)
- Graphical Editor (Sirius)
- Projection (MPS)

Concrete Syntax

Abstract Syntax

Semantic Domain

Language Semantics → debuggers, simulator, analysers, ...

15/10/2018 (c) Steffen Zschaler, King’s College London
Language workbenches

Language workbenches generate tool support from (declarative) descriptions of software languages

- Editors, analysers, simulators, debuggers, ...
- Many exist for syntax part, much fewer for semantics
- Here I build on **GEMOC Studio**
  - Built in Rennes/Toulouse
  - Generic description of language using:
    - *Abstract syntax*: EMF
    - *Graphical concrete syntax*: Sirius
    - *Operational Semantics*: a range of imperative options: KerMeta, fUML, ...
Goal

Add support for operational semantics specified using graph-transformation system (GTS)

• **Why?**
  • Declarative description of semantics
    • Enables reasoning (e.g., about concurrency)
    • Enables robust composition and weaving to extend the language semantics and combine different languages
  • Not an interpreter, but a semantics
    • Implicit scheduling

• **Why GEMOC Studio?**
  • Claims to be easily extensible with new semantics formalisms
  • Wanted to evaluate this claim
Core Idea

Standard GEMOC

Henshin rules for operational semantics
So what’s involved?

**Execution Engine**
- Implement interpretation of semantics
- Currently based on sequential execution engine
  - At each step randomly pick an available match
- Need to “fake” semantics operation names for GEMOC
  - Using rule names
  - “Flat” semantics

**ModelExecutionContext**
- To remove dependency on Melange

**Launch Configuration**
- To wire all of the above for execution
- Needed to copy existing code because not accessible for reuse
Demonstration: The Production Line Model
Concurrency

Concurrency is currently badly supported

• Using sequential execution engine
  • Need to make a choice of the next step at each point rather than giving the choice to the user

• Should really build on concurrent execution engine
  • BUT: could not figure out how to extend this; it seems much less modular
  • HELP!

GTSs have the potential of making concurrent semantics much easier

• No need for explicit concurrency model → could potentially be inferred from semantics and model
Conclusions

Two contributions:

1. Support for GTS-based semantics in GEMOC Studio
2. Initial evaluation of extensibility of GEMOC Studio

Future work:

• Proper support for concurrency
• Support for time
• Connection to our previous composition work
Questions?

szschaler@acm.org
www.steffen-zschaler.de
With thanks to Erwan Bousse and Tanja Meyerhöfer