Coping with Semantic Variation Points in Domain-Specific Modeling Languages

1st International Workshop on Executable Modeling (EXE 2015)
MODELS 2015 Workshops

Florent Latombe[1] Xavier Crégut[1]
Julien De Antoni[2] Marc Pantel[1] Benoît Combemale[3]

[1]University of Toulouse, IRIT, Toulouse, France
first.last@irit.fr
[2]Univ. Nice Sophia Antipolis, CNRS, I3S, Inria, Sophia Antipolis, France
first.last@polytech.unice.fr
[3]University of Rennes I, IRISA, Inria, Rennes, France
first.last@irisa.fr

2015/09/27

Table of contents

Overview

Introduction
- Context: xDSMLs
- Semantic Variation Points

Concurrency and SVPs

Implementing SVPs

Conclusion

eXecutable Domain-Specific Modeling Languages (xDSMLs)

- Modern systems: Too big to be addressed only as a whole.
- Domain-Specific Languages (DSLs) capitalize domain knowledge (security, fault tolerance, etc.) as language constructs.
- Modeling Languages (MLs) provide user-friendly abstractions for domain experts.
- eXecutable DSMLs ease the design, verification and validation of modern systems.

Overview

Semantic Variation Points

Semantic Variation Point (SVP) $\triangleq$ language specification part left intentionally under-specified to allow further language adaptation. Usually dealt with:

- Further refinement of the specification (*e.g.*, stereotypes or profiles in UML).
- Arbitrary choices in the implementation (*e.g.*, multithreaded programs in CPython or Jython, fUML, etc.).
- Tool vendors responsible for specifying and documenting the implemented solution.

Example: priorities of conflicting transitions in Statecharts

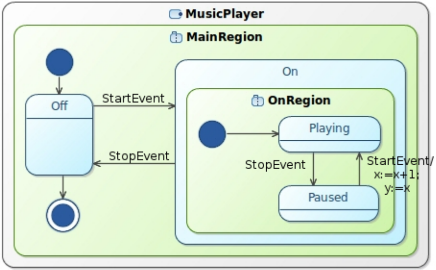


Figure 1: Simple music player statechart.

When *Event* "StopEvent" occurs in *States* "On" and "Playing".

- Original formalism: *Transition* from "On" to "Off" is fired.
- UML/Rhapsody: *Transition* from "Playing" to "Paused" is fired.

Courtesy of the comparative study of the different Statecharts dialects and their SVPs by M. Crane and J. Dingel [3].

Vocabulary Clarification

- *Language*: syntax and semantics specification that may contain SVPs.
- *Dialect*: language implementation, making choices about some – possibly all – its SVPs.

Problem

- SVPs usually identified informally in the syntax and semantics specification documents.
- Tools usually only provide one dialect, constraining the end-user.
- Complicates cooperation between tools (providing different dialects) and users (who may assume different meanings for the same syntax).
- Hinders cooperation in larger projects using different variants of the same language that may be better fit for some aspects.

Summary of our Contribution

- A concurrent executable metamodeling approach enabling the specification of **Concurrency-aware eXecutable Domain-Specific Modeling Languages**.
- Specification of operational semantics that makes explicit the language concurrency concerns in an adapted formalism based on concurrency theory.
- **That allows to explicitly specify and implement xDSML's SVPs.**

Overview

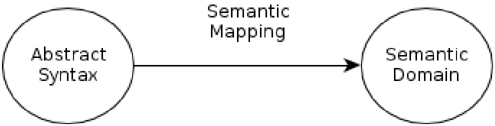
Designing xDSMLs

Language design

$Language \triangleq AS + SD + \mathcal{M}(AS, SD)$ Where:

- Abstract Syntax (AS): concepts and relations between concepts
- Semantic Domain (SD): meaningfull existing language
- Semantic Mapping ($\mathcal{M}(AS, SD)$): maps concepts from the AS to their meaning in the SD.

Three main approaches to the Semantic Mapping: Axiomatic, Translational (incl. Denotational) and Operational.

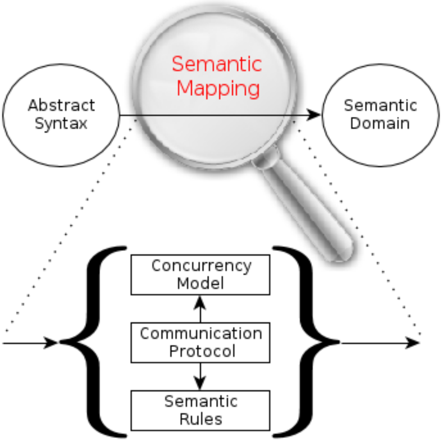


Overview of the GEMOC Approach

Separation of Concerns
(SLE 2013 [2])

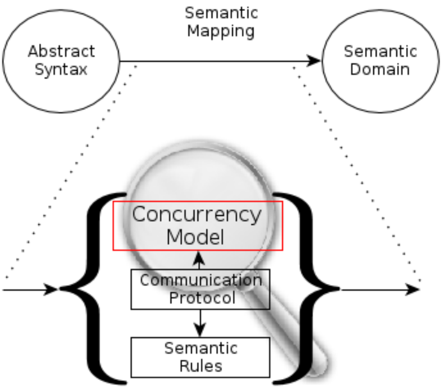
Split the Semantic Mapping in:

- *Semantic Rules:* Operational specification of the model runtime state evolution.
- *Concurrency Model:* Partial ordering of abstract actions in a formalism inspired by concurrency theory.
- *Communication Protocol:* Relates abstract actions and Semantic Rules.



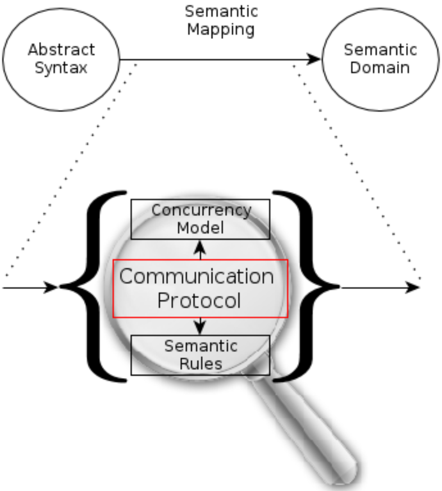
Concurrency Model

- Called Model of Concurrency and Communication (MoCC) in GEMOC.
- Focuses on concurrency, synchronization and the, possibly timed, causalities between actions.
- Actions are opaque (data manipulations are abstracted).
- Given as an *EventType Structure* which builds the Event Structure [10] for each model concurrent control flow.

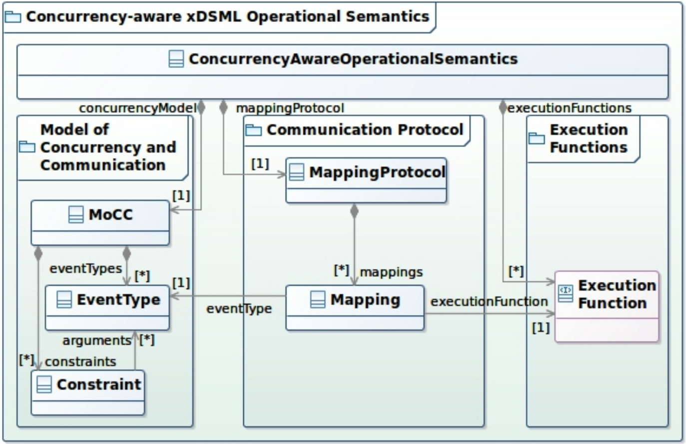


Communication Protocol

- Relates the Concurrency Model and the Execution Functions
- See *Weaving Concurrency in eXecutable Domain-Specific Modeling Languages* (SLE 2015) [8].



Class Diagram of the concurrent executable metamodeling approach



Execution

Translation

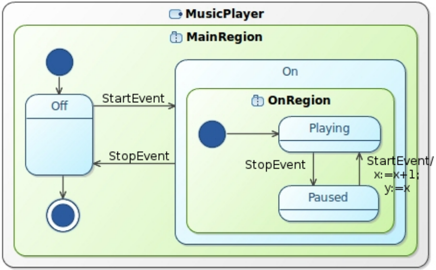
Model-specific specifications are generated (*i.e.*, Semantic Rules, Communication Protocol and MoCC all specific to the given model).

Runtime

- The Event Structure (MoCC at the model level) gives a partial ordering over abstract events.
- Abstracts all the possible model executions paths (including all interleavings of concurrent events).
- Event occurrences are mapped to model runtime state changes by the Communication Protocol.
- Nondeterministic situations are resolved by runtime heuristics.

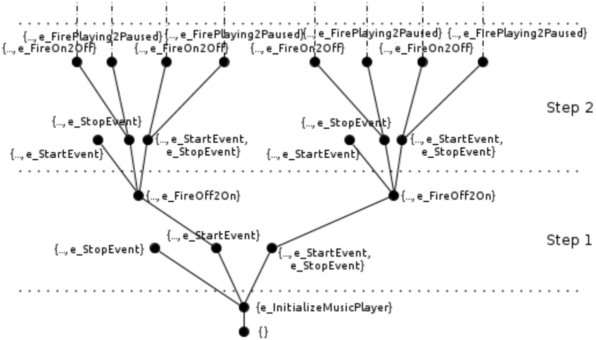
Overview

Event Structure for our example model



At the Language Level

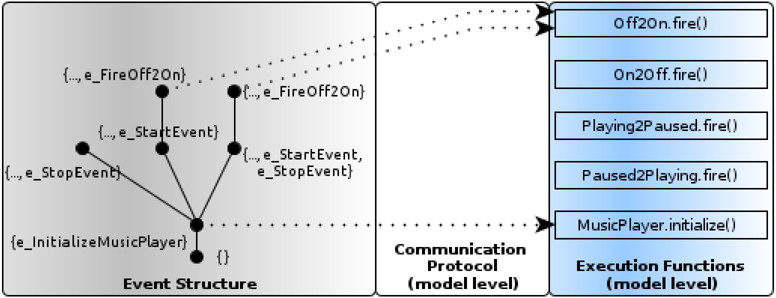
- Main EventTypes: `Event.et_occur`, `Transition.et_fire`
- Main constraints: When an*Event* occurs, one of the *Transitions* it triggers will be fired.



Event Structure

Nodes are*configurations*: Unsorted set of event occurrences which occured at this execution point.

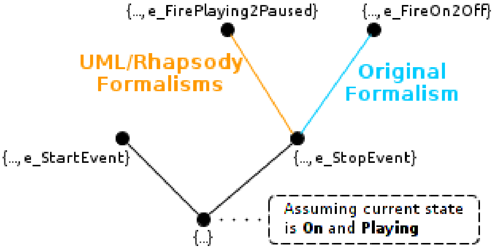
Illustration of the Separation of Concerns at the model level



Overview

Event Structures and SVPs

- Nondeterminism in Event Structures gives potential SVPs.



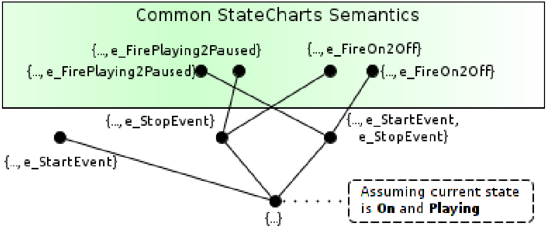
- SVPs can be implemented by constraining the *Event Structure* partial ordering.
- Done at the language level in the *EventType Structure*.

Overview

Example: Conflicting Transitions SVP

Main Idea

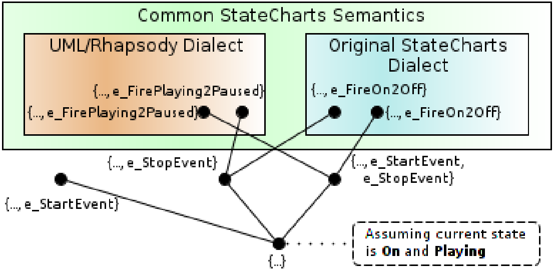
- Most of the concurrency concerns are shared by Statecharts dialects.
- Statecharts MoCC specifies the superset of possible partial orderings.



Example: Conflicting Transitions SVP (2)

SVP Implementations

- Dialects extend the common MoCC to restrict partial orderings.



- Removes `On2Of` for UML/Rhapsody dialects, and `Playing2Paused` for Original dialect.

Overview

Introduction

Concurrency and SVPs

Implementing SVPs

Conclusion

- Implementation
- Overview
- Perspectives
- Questions

Implementation

- *GEMOC Studio*[1] implementation based EMF [7].
- Abstract Syntax: Ecore (EMF implementation of EMOF [9]).
- Semantic Rules: Kermeta 3 [6] (based on Xtend [1]).
- Concurrency Model: MoCCML [4] and ECL [5].
- Communication Protocol: Gemoc Events Language (GEL) [8].

[1] `http://www.gemoc.org/studio`

Overview

Overview of our contribution

- SVPs are usually poorly identified in language specifications.
- GEMOC concurrent executable metamodeling approach, based on *Event Structure* for the *Concurrency Model* of concurrency-aware xDSMLs provides potential SVPs as nondeterministic situations.
- Restricting the partial ordering defined in the *Concurrency Model* implements SVPs.
- SVP implementations are weaved in the language definition, allowing the execution tool to remain independent from any arbitrary choice.

Overview

Perspectives

- Distinguish wanted nondeterminism from potential SVPs (forbid restrictions).
- Other kind of SVPs using Semantic Rules and/or Communication Protocol.
- Integration with SVPs at the syntax level, both abstract and concrete.
- Experiment the use of variability management techniques to handle dialects.

Acknowledgement

This work is partially supported by the ANR INS Project GEMOC (ANR-12-INSE-0011).

Overview

Thank you for your attention.
Questions?

References I

L. Bettini.
Implementing Domain-Specific Languages with Xtext and Xtend.
Packt Publishing Ltd, 2013.

B. Combemale, J. De Antoni, M. Vara Larsen, F. Mallet, O. Barais, B. Baudry, and R. France.
Reifying Concurrency for Executable Metamodeling.
In *SLE 2013*, LNCS. Springer-Verlag.

M. L. Crane and J. Dingel.
UML vs. Classical vs. Rhapsody statecharts: not all models are created equal.
SoSyM, 2007.

J. De Antoni, P. Issa Diallo, C. Teodorov, J. Champeau, and B. Combemale.
Towards a Meta-Language for the Concurrency Concern in DSLs.
In *DATE'15*, Mar. 2015.

References II

J. De Antoni and F. Mallet.
ECL: the event constraint language, an extension of OCL with events.
Technical report, Inria, 2012.

DIVERSE-team.
Github for K3AL, 2015.

Eclipse Foundation.
EMF homepage, 2015.

F. Latombe, X. Crégut, B. Combemale, J. Deantoni, and M. Pantel.
Weaving Concurrency in eXecutable Domain-Specific Modeling Languages.
In *8th ACM SIGPLAN International Conference on Software Language Engineering (SLE 2015)*, Pittsburg, United States, Oct. 2015. ACM.

References III

OMG.
MOF specification v2.5, 2015.

G. Winskel.
Event structures.
Springer, 1987.