Concurrency-aware eXecutable Domain-Specific Modeling Languages as Models of Concurrency

2nd International Workshop on Executable Modeling (EXE 2016)
collocated with MODELS 2016 in Saint-Malo, France

@see: http://gemoc.org/exe16/

2016 - 10 - 03

Florent Latombe, Xavier Crégut, Marc Pantel

Université de Toulouse,
IRIT,
Toulouse, France
first.last@irit.fr
The GEMOC Concurrency-aware xDSML Approach

Overview of the Approach
Specification
Execution
Benefits

Conclusion and Perspectives
A Synergetic Language Design Approach

- Introduced by Benoit Combemale *et al.* in SLE 2012 and 2013.
- xDSML design approach with a separation of concerns in the operational semantics.
- Concurrency concerns expressed using a Model of Concurrency at the language level.
A Synergetic Language Design Approach

- Introduced by Benoit Combemale et al. in SLE 2012 and 2013.
- xDSML design approach with a separation of concerns in the operational semantics.
- Concurrency concerns expressed using a Model of Concurrency at the language level.
Overview of the GEMOC Concurrency-aware xDSML Approach

Overview of the Approach

Specification

Execution

Benefits
**Structural Elements**

Example xDSML:

- **Concrete Syntax(es)**
- **Abstract Syntax (metamodel)**

The concurrent aspects are *underspecified*. ⇒ Several possible executions.
**Structural Elements**

Example xDSML:

- Concrete Syntax(es)
- Abstract Syntax (metamodel)

The concurrent aspects are **underspecified**. ⇒ Several possible executions.
Structural Elements

Example xDSML:

- Concrete Syntax(es)
- Abstract Syntax (metamodel)

The concurrent aspects are underspecified. ⇒ Several possible executions.
**Semantic Rules**

**Definition**

Dynamic data (**Execution Data**) and their evolution (**Execution Functions**).

- **fUML Execution Data:** Tokens held by ActivityEdges.
- **fUML Execution Functions:** `ActivityNode.execute()`, `ActivityEdge.evaluateGuard()`.
Model of Concurrency Application & Mapping

- MoCAplication: Concurrent aspects of a model.
- Conforms to a Model of Concurrency (initially only Event Structure).
- Generated based on the MoCMapping (EventType Structure).
Communication Protocol

**Definition**

**Mappings** between the MoCTrigger (abstract actions of the MoCMapping) and the Execution Functions (concrete actions of the Semantic Rules).
The Concurrency-related Specifications

Abstract Syntax Language (e.g. EMOF/Ecore)

conforms to

Metalanguages (M3)

Abstract Syntax (e.g. fUML Abstract Syntax)

conforms to

Languages (M2)

Model (e.g. TalkAndDrinkActivity)

Models (M1)

Language Designer

Domain Expert
The Concurrency-related Specifications

- **Model of Concurrency** (e.g. Event Structures formalism)
- **Abstract Syntax Language** (e.g. EMOF/Ecore)
- **Metalanguages** (M3)
- **Languages** (M2)
- **Models** (M1)
- **MoCMMapping** (e.g. EventType Structures)
- **Abstract Syntax** (e.g. fUML Abstract Syntax)
- **MoCMApplication** (e.g. Event Structure for TalkAndDrinkActivity)
- **Language Designer**
- **Domain Expert**
- **Generator**

The diagram illustrates the relationship between models and languages, showing how the models conform to the languages and how the languages conform to the metalanguages.
Overview

The GEMOC Concurrency-aware xDSML Approach

Overview of the Approach
Specification
Execution
Benefits
Execution
The runtimes for each concern are coordinated by the Execution Engine.

Possible execution step(s):
\[ e_{\text{MyInitial}} \rightarrow \text{MyInitial}.execute() \]
Execution

The runtimes for each concern are coordinated by the Execution Engine.

Possible execution step(s):

\[ e_{MyFork} \rightarrow \| \rightarrow MyFork.execute() \]
Execution

The runtimes for each concern are coordinated by the Execution Engine.

Possible execution step(s):

\[ e_{Talk} \rightarrow \text{Talk.execute()} \]

OR

\[ e_{DrinkSomething} \rightarrow \text{DrinkSomething.execute()} \]

OR both
Execution

The runtimes for each concern are coordinated by the Execution Engine.

Possible execution step(s):

\[ e_{\text{Talk}} \rightarrow \| \rightarrow \text{Talk.execute()} \]

OR

\[ e_{\text{DrinkSomething}} \rightarrow \| \rightarrow \text{DrinkSomething.execute()} \]

OR both
Execution

The runtimes for each concern are coordinated by the **Execution Engine**.

Possible execution step(s):

\[
\begin{align*}
  e_{\text{Talk}} & \rightarrow [ ] \rightarrow \text{Talk.execute()} \\
  \text{OR } e_{\text{DrinkSomething}} & \rightarrow [ ] \rightarrow \text{DrinkSomething.execute()} \\
  \text{OR both}
\end{align*}
\]
The runtimes for each concern are coordinated by the **Execution Engine**.

Possible execution step(s):

\[ e_{MyJoin} \rightarrow \parallel \rightarrow MyJoin.execute() \]
Execution

The runtimes for each concern are coordinated by the **Execution Engine**.

Possible execution step(s):

\[ e_{\text{MyFinal}} \rightarrow \text{MyFinal}.execute() \]
Execution

The runtimes for each concern are coordinated by the **Execution Engine**.

Possible execution step(s):

∅
The GEMOC Concurrency-aware xDSML Approach

Overview of the Approach
Specification
Execution
Benefits
Systematic Use of a Model of Concurrency

- Concurrency concepts are not manipulated by the domain experts.
- By construction, the MoC is used correctly.

- The different concerns can be implemented and debugged separately.
- Depending on the Model of Concurrency used, behavioral properties may be assessed.
Systematic Use of a Model of Concurrency

- Concurrency concepts are not manipulated by the domain experts.
- By construction, the MoC is used correctly.
- The different concerns can be implemented and debugged separately.
- Depending on the Model of Concurrency used, behavioral properties may be assessed.
Overview

The GEMOC Concurrency-aware xDSML Approach

Tailoring Models of Concurrency to xDSMLs

Conclusion and Perspectives
OVERVIEW

TAILORING MODELS OF CONCURRENCY TO xDSMLs

Motivation
Illustration and Generalization
Validation and Implementation
Adequacy of a Model of Concurrency

- In “Why Do Scala Developers Mix the Actor Model with Other Concurrency Models?”, inadequacies of the actor model.
- In the GEMOC project, MoCCML was designed as a merge of CCSL and automata.
**Integration of new Models of Concurrency**

Defining and integrating a Model of Concurrency

\[ \text{MoC + MoCMapping + tools} \]

\[ \text{(rich editor, generator, runtime)} \]

---

**Motivation**

Illustration and Generalization

Validation and Implementation

---

**MoCMapping**

\[ \text{MoCMapping Metalanguage} \]

\[ \text{(e.g. EventType Structures formalism)} \]

---

**MoCApplication**

\[ \text{MoCApplication} \]

\[ \text{(e.g. Event Structure for TalkAndDrinkActivity)} \]

---

**Model of Concurrency**

\[ \text{Model of Concurrency} \]

\[ \text{(e.g. Event Structures formalism)} \]

---

**Metallanguage**

\[ \text{Metalanguages (M3)} \]

---

**Languages**

\[ \text{Languages (M2)} \]

---

**Models**

\[ \text{Models (M1)} \]
Integration of new Models of Concurrency

Defining and integrating a Model of Concurrency

\[
\text{Model of Concurrency} \quad \boxed{\text{MoC}} \quad \text{MoCMapping Metalanguage} \quad \boxed{\text{MoCM}} \quad \text{tools} \quad \boxed{\text{MoCApplication}}
\]

- MoCMapping metalanguage often not pre-existing.
- Identifying the MoCTriggers of the MoCMapping.

F. Latombe, X. Crégut, M. Pantel – IRIT, Université de Toulouse

Concurrency-aware xDSMLs as Models of Concurrency
A Recursive Definition of the Approach

For an xDSML $\mathcal{L}_{\text{DOMAIN}}$:
- MoC → Another concurrency-aware xDSML, hereafter $\mathcal{L}_{\text{MoC}}$.
- MoCMapping → Model transformation from $\mathcal{L}_{\text{DOMAIN}}$ to $\mathcal{L}_{\text{MoC}}$.
- MoCTriggers → Mappings of the Communication Protocol of $\mathcal{L}_{\text{MoC}}$.
- MoCAplication → Model conforming to $\mathcal{L}_{\text{MoC}}$.
- Runtime → Execution Engine of $\mathcal{L}_{\text{MoC}}$. 
A Recursive Definition of the Approach

For an xDSML $\mathcal{L}_\text{DOMAIN}$:
- MoC → Another concurrency-aware xDSML, hereafter $\mathcal{L}_\text{MoC}$.
- MoCMapping → Model transformation from $\mathcal{L}_\text{DOMAIN}$ to $\mathcal{L}_\text{MoC}$.
- MoCTriggers → Mappings of the Communication Protocol of $\mathcal{L}_\text{MoC}$.
- MoCApplication → Model conforming to $\mathcal{L}_\text{MoC}$.
- Runtime → Execution Engine of $\mathcal{L}_\text{MoC}$.
A Recursive Definition of the Approach

For an xDSML $\mathcal{L}_{\text{DOMAIN}}$:

- MoC $\rightarrow$ Another concurrency-aware xDSML, hereafter $\mathcal{L}_{\text{MoC}}$.
- MoCMapping $\rightarrow$ Model transformation from $\mathcal{L}_{\text{DOMAIN}}$ to $\mathcal{L}_{\text{MoC}}$.
- MoCTriggers $\rightarrow$ Mappings of the Communication Protocol of $\mathcal{L}_{\text{MoC}}$.
- MoCAplication $\rightarrow$ Model conforming to $\mathcal{L}_{\text{MoC}}$.
- Runtime $\rightarrow$ Execution Engine of $\mathcal{L}_{\text{MoC}}$. 

F. Latombe, X. Crégut, M. Pantel – IRIT, Université de Toulouse
A Recursive Definition of the Approach

For an xDSML $\mathcal{L}_{\text{Domain}}$:

- MoC → Another concurrency-aware xDSML, hereafter $\mathcal{L}_{\text{MoC}}$.
- MoCMapping → Model transformation from $\mathcal{L}_{\text{Domain}}$ to $\mathcal{L}_{\text{MoC}}$.
- MoCTriggers → Mappings of the Communication Protocol of $\mathcal{L}_{\text{MoC}}$.
- MoCAppliedS → Model conforming to $\mathcal{L}_{\text{MoC}}$.
- Runtime → Execution Engine of $\mathcal{L}_{\text{MoC}}$. 
Overview

Tailoring Models of Concurrency to xDSMLs

Motivation
Illustration and Generalization
Validation and Implementation
ILLUSTRATION ON fUML
Illustration on fUML

With the Event Structures MoC...

...obtained after manual “translation” of the fUML specification.
Illustration on fUML

Main Thread

SubThread1

SubThread2

TalkAndDrink

Talk

MyInitial

MyFork

MyDecision

MyMerge

MyJoin

MyFinal

CheckTableForDrinks

DrinkCoffee

DrinkTea

DrinkWater

Execute MyInitial

Execute MyFork

Start SubThread1 and SubThread2

Join SubThread1 and SubThread2

Execute MyJoin

Execute MyFinal

Execute CheckTableForDrinks

Execute MyOutputPin

Execute MyDecision

... 

Execute DrinkCoffee or DrinkTea or DrinkWater

Execute MyMerge

Execute Talk
**Language Specifications**

- **MoCMapping**: Model transformation from $\mathcal{L}_{\text{DOMA}IN}$ to $\mathcal{L}_{\text{MoC}}$.
  - Used to capture *exclusively* the concurrency aspects.
  - The source and the target are thus *not semantically equivalent*.

- **Projections**: part of the metamodel of the trace of the MoCMapping. Used to compensate the $1 \rightarrow n$ nature of the MoCMapping.

- **Communication Protocol**: Mappings (of $\mathcal{L}_{\text{DOMA}IN}$) connect an Execution Function (of $\mathcal{L}_{\text{DOMA}IN}$) to a Mapping of $\mathcal{L}_{\text{MoC}}$ through a Projection.

---

- **MoCMapping**: Model transformation from $\mathcal{L}_{\text{DOMA}IN}$ to $\mathcal{L}_{\text{MoC}}$.
  - Used to capture *exclusively* the concurrency aspects.
  - The source and the target are thus *not semantically equivalent*.

- **Projections**: part of the metamodel of the trace of the MoCMapping. Used to compensate the $1 \rightarrow n$ nature of the MoCMapping.

- **Communication Protocol**: Mappings (of $\mathcal{L}_{\text{DOMA}IN}$) connect an Execution Function (of $\mathcal{L}_{\text{DOMA}IN}$) to a Mapping of $\mathcal{L}_{\text{MoC}}$ through a Projection.

---

- **Language Specifications**
  - **MoCMapping**: Model transformation from $\mathcal{L}_{\text{DOMA}IN}$ to $\mathcal{L}_{\text{MoC}}$.
  - Used to capture *exclusively* the concurrency aspects.
  - The source and the target are thus *not semantically equivalent*.

- **Projections**: part of the metamodel of the trace of the MoCMapping. Used to compensate the $1 \rightarrow n$ nature of the MoCMapping.

- **Communication Protocol**: Mappings (of $\mathcal{L}_{\text{DOMA}IN}$) connect an Execution Function (of $\mathcal{L}_{\text{DOMA}IN}$) to a Mapping of $\mathcal{L}_{\text{MoC}}$ through a Projection.
Language Specifications

- **MoCMapping**: Model transformation from $\mathcal{L}_{\text{Domain}}$ to $\mathcal{L}_{\text{MoC}}$.
  - Used to capture *exclusively* the concurrency aspects.
  - The source and the target are thus *not semantically equivalent*.

- **Projections**: part of the metamodel of the trace of the MoCMapping.
  - Used to compensate the $1 \rightarrow n$ nature of the MoCMapping.
  - *e.g.*, ForkNode is transformed into many “start thread” instructions.

- Communication Protocol: Mappings (of $\mathcal{L}_{\text{Domain}}$) connect an Execution Function (of $\mathcal{L}_{\text{Domain}}$) to a Mapping of $\mathcal{L}_{\text{MoC}}$ through a Projection.
Validation on fUML
Implementation

The GEMOC Studio:
- Based on the Eclipse Modeling Framework (EMF).
- Dedicated metalanguage for the different aspects of a language specification.
- Generic execution, animation and debugging facilities.

Implementation of the contribution in the GEMOC Studio
- MoCMapping: any model transformation language can be used.
- Projections: small dedicated metalanguage (using Ecore + Xtext).
- Communication Protocol: dedicated metalanguage, the GEMOC Events Language (GEL) [5], extended with the use of the Projections.
Conclusion

Recursive definition of the concurrency-aware xDSML approach:

- Seamless integration into the approach.
- No MoC-specific MoCMapping metalanguage.
- Common interface for MoCs (i.e., as concurrency-aware xDSMLs).
- Verification of behavioral properties can be performed based on the selected MoC.

⚠️ Additional overhead to the runtime performance.
PERSPECTIVES

- Standard library of Models of Concurrency including a bootstrapping of Event Structures.
- Integration of existing verification tools for standard MoCs. e.g., for Petri nets, Actors, etc.
- Verification of domain properties through higher-order transformations. i.e., translating the verification results back into the original domain.
- Generating efficient implementations of the xDSMLs.

Acknowledgement

This work is partially supported by the ANR INS Project GEMOC (ANR-12-INSE-0011).
More information at: http://www.gemoc.org
Artefacts and Questions

@see: http://gemoc.org/exe16/

contains:

- Videos illustrating:
  - the Language Workbench (Concurrency-aware specification of the Threading xDSML and of fUML using Threading as its MoC);
  - the Modeling Workbench (Execution of the example fUML Activity).

- The GEMOC Studio with:
  - our Threading xDSML implementation;
  - our fUML implementation;
  - the example fUML Activity.

Thank you for your attention.

Questions?

F. Latombe, X. Crégut, M. Pantel – IRIT, Université de Toulouse
L. Bettini.  
*Implementing Domain-Specific Languages with Xtext and Xtend.*  

Reifying Concurrency for Executable Metamodeling.  
In *SLE'13*.

Bridging the Chasm between Executable Metamodeling and Models of Computation.  
In *SLE, 2012*.

Towards a Meta-Language for the Concurrency Concern in DSLs.  
In *DATE, 2015*.

Weaving Concurrency in eXecutable Domain-Specific Modeling Languages.  

F. Mallet.  
Clock constraint specification language: specifying clock constraints with UML/MARTE.  

S. Tasharofi, P. Dinges, and R. E. Johnson.  
Why do scala developers mix the actor model with other concurrency models?  

G. Winskel.  
Event structures.  
F. Zalila, X. Crégut, and M. Pantel.

A Transformation-Driven Approach to Automate Feedback Verification Results.