Domain-Specific Engineering of Domain-Specific Languages

†Raphaël Mannadiar and †,‡Hans Vangheluwe

†McGill University, Canada and ‡University of Antwerp, Belgium

October 18th @ DSM 2010
Outline

1. context and problem
2. our example
3. DSL foundations
4. constructing DSLs
5. from model to artifact
6. our example...
7. conclusion

domain-specific engineering of domain-specific languages
why do domain-specific modelling (DSM)?

problem and solution domains are often far apart

mapping problems to solutions manually is difficult, slow and error-prone

DSM enables the modelling of problems instead of solutions and automates the mapping between them
past **UML-to-code** efforts only succeeded in generating **partial** applications

**How can complete artifacts be generated from domain-specific models (DSMs)?**
how does it work?

Past UML-to-code efforts only succeeded in generating **partial** applications.

**How can complete artifacts be generated from domain-specific models (DSMs)?**

Restricting modeling language **expressiveness** to a narrow domain is the key to giving models **unambiguous semantics**.
DSL designers (i.e., DSM experts) design models of languages (e.g., using UML) and their mappings to the solution domain (e.g., code generators)

using UML?! but you just said... and doesn’t DSM automate that mapping and deliver complete artifacts to modellers?
DSL designers (i.e., DSM experts) design models of languages (e.g., using UML) and their mappings to the solution domain (e.g., code generators)

using UML?! but you just said... and doesn’t DSM automate that mapping and deliver complete artifacts to modellers?

so couldn’t it be used to deliver complete artifacts to DSL designers as well?
we propose an approach to **domain-specific language (DSL) design** that builds on **DSM principles** by providing DSL designers with **constructs specific to their domain** (i.e., the domain of all DSLs) which enables the **automatic generation of DSm-to-artifact semantic mappings**
1. context and problem

2. our example

3. DSL foundations

4. constructing DSLs

5. from model to artifact

6. our example...

7. conclusion
phoneapps, a DSL for mobile applications
domain-specific engineering of domain-specific languages
from DSm to mobile application
domain-specific engineering of domain-specific languages
from DSm to mobile application
from DSm to mobile application
context and problem

our example

DSL foundations

constructing DSLs

from model to artifact

our example...

conclusion
a DSL has three components

- abstract syntax
  language concepts and relationships + constraints that encode domain rules

- concrete syntax(es)
  graphical and/or textual representations of abstract syntax elements

- semantics
  compilers and/or interpreters that define the meaning of instance models in the language
a **DSL has three components**

- **abstract syntax**
  language concepts and relationships + constraints that encode domain rules

- **concrete syntax(es)**
  graphical and/or textual representations of abstract syntax elements

- **semantics**
  compilers and/or interpreters that define the meaning of instance models in the language

**DSL abstract syntax** is commonly specified and communicated via UML class diagrams

**DSL semantics** are commonly specified as code generators or model transformations
our approach is based on two claims

any conceivable DSL is a combination of a finite set of lower level formalisms
claims and questions

our approach is based on two claims

any conceivable DSL is a combination of a finite set of lower level formalisms

which formalisms form this basis for DSL design?

how can DSLs be defined in terms of these base formalisms?
knowing how base formalisms are combined to form a given DSL is sufficient to construct its full semantics
knowing how base formalisms are combined to form a given DSL is sufficient to construct its full semantics

how can artifacts be generated from instance models of these DSLs without manually defined DSL semantics?

how can semantic transformations be generated from a DSL definition?
base formalisms

statecharts
base formalisms

statecharts

petri nets
base formalisms

- statecharts
- petri nets
- causal block diagrams
base formalisms...

layout

```
VisualElement
+id: String
+height: String
+width: String

Widget
+text: String

Container
+layout: Enum

Button
Label
Input
List

Canvas
+name: String
```
base formalisms...

**layout**

```
VisualElement
+id: String
+height: String
+width: String

Widget
+text: String

Container
+layout: Enum

Canvas
+name: String

Button
Label
Input
List
```

**action code**

```
Code

API Call
+function: String
+parameters: List

User Code
+body: String
```
base formalisms...

our basis is a work in progress

so far, we can use it to model DSLs that **arbitrarily combine**

- determinism and non-determinism
- states and transitions
- discrete and continuous flow
- user interfaces
- api calls and code-based escape semantics
<table>
<thead>
<tr>
<th>#</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>context and problem</td>
</tr>
<tr>
<td>2</td>
<td>our example</td>
</tr>
<tr>
<td>3</td>
<td>DSL foundations</td>
</tr>
<tr>
<td>4</td>
<td>constructing DSLs</td>
</tr>
<tr>
<td>5</td>
<td>from model to artifact</td>
</tr>
<tr>
<td>6</td>
<td>our example...</td>
</tr>
<tr>
<td>7</td>
<td>conclusion</td>
</tr>
</tbody>
</table>
semantic templates

we need **domain-specific concepts** to be related to **base formalisms** tightly enough to enable **automatic generation** of DSL-to-base-formalism transformations
semantic templates

we need domain-specific concepts to be related to base formalisms tightly enough to enable automatic generation of DSL-to-base-formalism transformations

we introduce semantic templates (STs) as “interfaces” to base formalisms

each base formalism “exposes” a set of STs that encode the unambiguous mapping of arbitrary domain-specific concepts onto concepts from the given base formalism.
Steps transition to Steps

- Steps become children of Statechart\_State
- Steps can be connected via Statechart\_Transition edges
- a set of such connected Steps can be projected onto a statechart
Actions and Screens are types of Steps

- Actions and Screens become children of Steps
- they can now also be connected via Statechart.Transition edges
Actions and Screens are types of Steps

- Actions and Screens become children of Steps
- they can now also be connected via Statechart.Transition edges
so what exactly can we infer from a set of STs?

1 how to construct an **internal** UML class diagram of a DSL (abstract syntax)
so what exactly can we infer from a set of STs?

1. How to construct an internal UML class diagram of a DSL (abstract syntax)

2. How to map a DSm onto base formalism instance models (semantics)
domain-specific engineering of domain-specific languages
divided and conquered

artifact synthesis now consists in automatically generating

1. base formalism instances $f_i$ from the DS

2. desired artifacts $a_i$ (e.g., Java code) for each $f_i$
artifact synthesis now consists in automatically generating

1. base formalism instances \( f_i \) from the DSm
2. desired artifacts \( a_i \) (e.g., Java code) for each \( f_i \)

it no longer falls upon the DSL designer to manually define the mapping of hand-picked portions of DSms onto lower level formalisms
artifacts $a_i$ might need to interact
artifacts $a_i$ might need to interact

how?

- we push the idea of "interfaces to base formalisms"
- base formalisms are now described by STs and i/o events
- a new generic ST enables inter-artifact event mapping

on event e1, produce event e2
remember that Screens and Actions are Statechart. States

entering a Screen should display it, entering an Action should launch it

we want to map statechart events to Action Code and Layout events...
remember that Screens and Actions are Statechart.States

entering a Screen should display it, entering an Action should launch it

we want to map statechart events to *Action Code* and *Layout* events...

on event enteredState:s, produce event drawCanvas:s
on event enteredState:s, produce event runCode:s
domain-specific engineering of domain-specific languages
beneath the *phoneapps* DSL
beneath a *phoneapps* DSm

- generated artifact(s) for each base formalism
- interaction via events
1. context and problem
2. our example
3. DSL foundations
4. constructing DSLs
5. from model to artifact
6. our example...
7. conclusion
our solution, in a nutshell

we proposed a novel approach to defining DSLs based on the combination of base formalisms that capture commonly recurring DSL features

domain-specific concepts are unambiguously mapped onto base formalisms via semantic templates

given “base formalism to target artifact” transformations, DSms can be transformed to the said target artifacts without the DSL designer having to manually define the semantic mapping

our approach brings DSM to DSM
questions?

thank you!