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

outline

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

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)?

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 modelling language expressiveness to a narrow domain is the key to giving models unambiguous semantics

DSM for DSM?

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**?

DSM for DSM?

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?

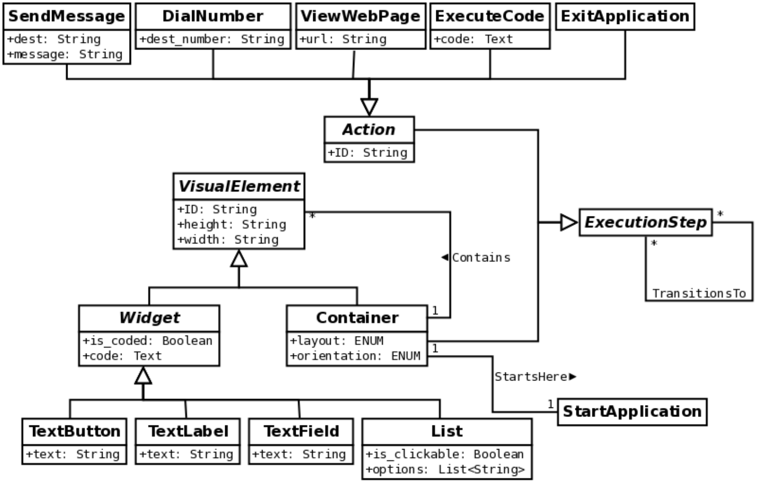
so couldn't it be used to deliver complete artifacts to DSL designers as well?

our solution, in a nutshell

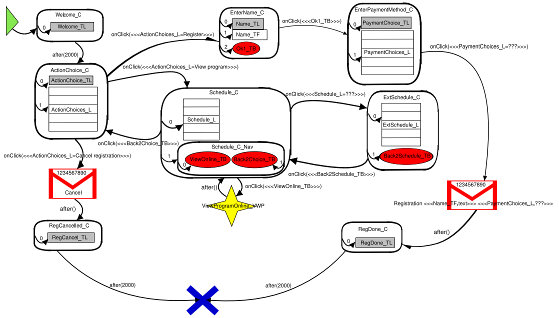
we propose an approach to domain-specific language (DSL) design that builds on DSM principles by providing dsl designers with constructs specific to their domain (i.e., the domain of all DSLs) which enables the automatic generation of DSm-to-artifact semantic mappings

outline

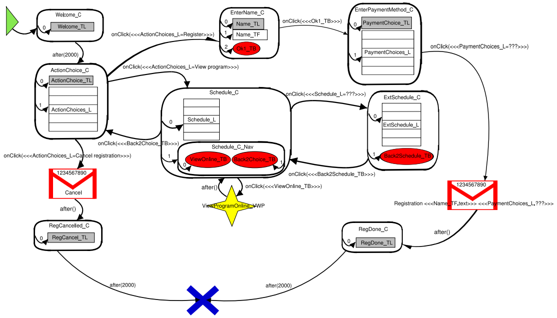
phoneapps, a DSL for mobile applications

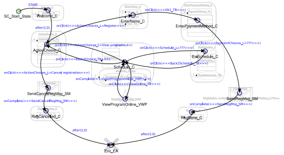


from DSm to mobile application

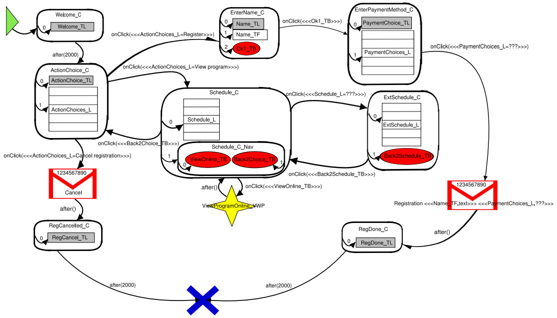


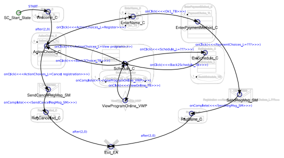
from DSm to mobile application

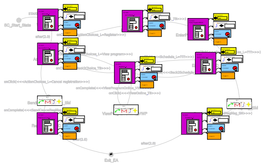




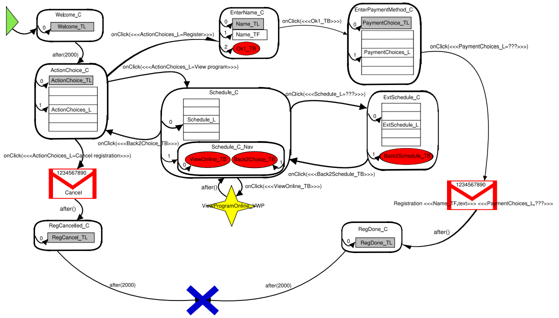
from DSm to mobile application

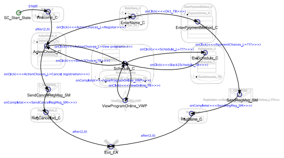
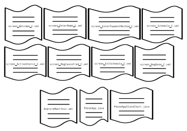
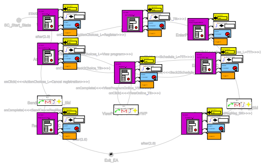




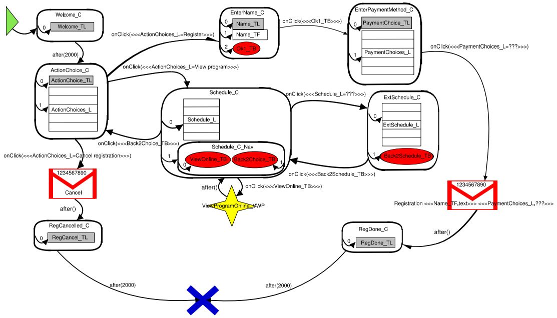


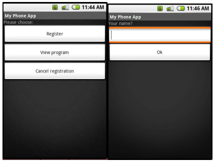
from DSm to mobile application

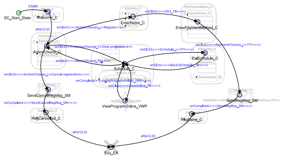


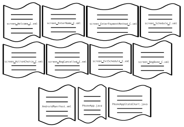




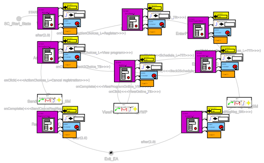
from DSm to mobile application











outline

DSLs 101

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

DSLs 101

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

claims and questions

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*?

claims and questions...

knowing how base formalisms are combined to form a given DSL is sufficient to construct its full semantics

claims and questions...

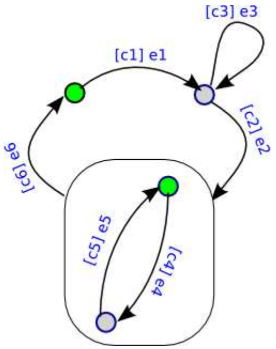
knowing how base formalisms are combined to form a given DSL is sufficient to construct its full semantics

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?

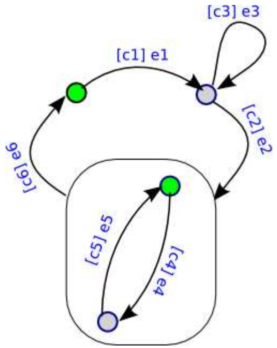
base formalisms

statecharts

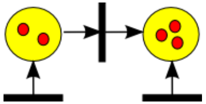


base formalisms

statecharts

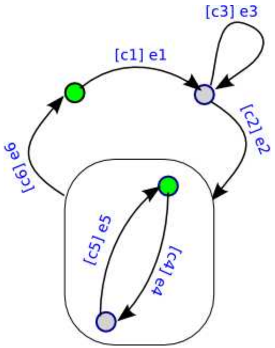


petri nets

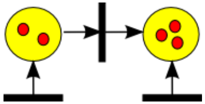


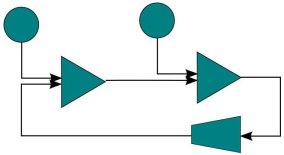
base formalisms

statecharts



petri nets

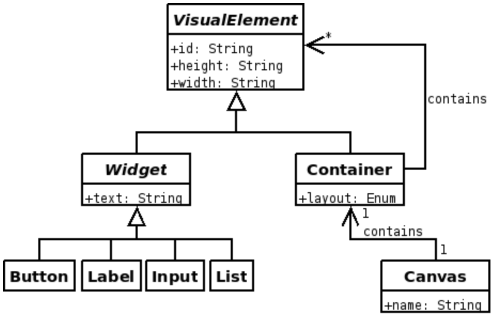




causal block diagrams

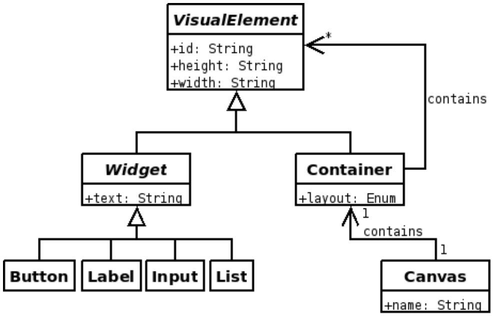
base formalisms...

layout

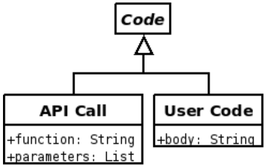


base formalisms...

layout



action code



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

outline

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.

a statechart ST

`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

a generic ST

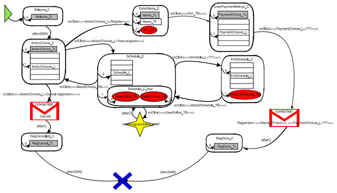
Actions and Screens are types of Steps

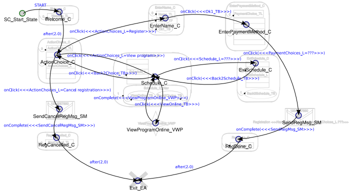
- Actions and Screens become children of Steps
- they can now also be connected via Statechart.Transition edges

a generic ST

Actions and Screens are types of Steps

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




START
SC_Start_State
Welcome_C
EnterName_C
EnterPaymentMethod_C
ActionChoice_C
Schedule_C
ExtSchedule_C
SendCancelRegMsg_SM
ViewProgramOnline_VWP
SendRegMsg_SM
RegCancelled_C
RegDone_C
Exit_EA
after(2.0)
onClick(<<<Ok1_TB>>>)
onClick(<<<ActionChoices_L=Register>>>)
onClick(<<<ActionChoices_L=View program>>>)
onClick(<<<Schedule_L=???>>>)
onClick(<<<PaymentChoices_L=???>>>)
onClick(<<<Back2Choice_TB>>>)
onClick(<<<Back2Schedule_TB>>>)
onClick(<<<ActionChoices_L=Cancel registration>>>)
onComplete(<<<ViewProgramOnline_VWP>>>)
onClick(<<<ViewOnline_TB>>>)
onComplete(<<<SendCancelRegMsg_SM>>>)
onComplete(<<<SendRegMsg_SM>>>)

semantic templates...

so what exactly can we infer from a set of STs?

1. how to construct an <u>internal</u> UML class diagram of a DSL (abstract syntax)

semantic templates...

so what exactly can we infer from a set of STs?

1. how to construct an <u>internal</u> UML class diagram of a DSL (abstract syntax)
2. how to map a DSm onto base formalism instance models (semantics)

outline

divided and conquered

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

divided and conquered

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

divided, conquered, and reunited

artifacts a_i might need to interact

divided, conquered, and reunited

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

divided, conquered, and reunited...

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...

divided, conquered, and reunited...

remember that `Screens` and `Actions` are `Statechart.States`

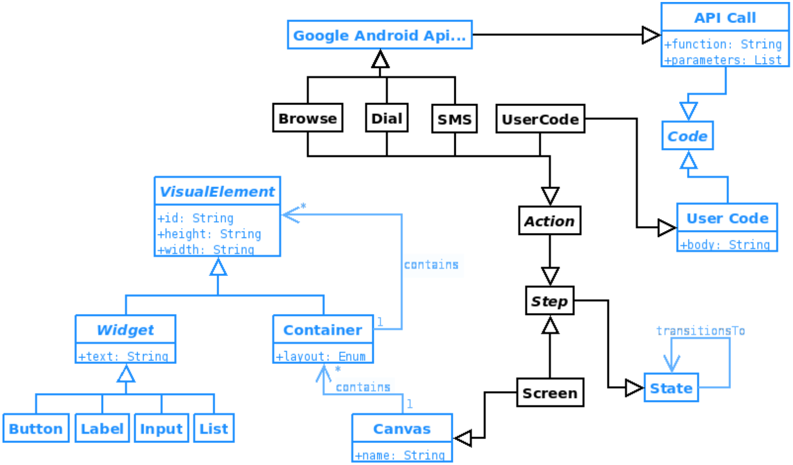
entering a `Screen` should display it, entering an `Action` should launch it

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

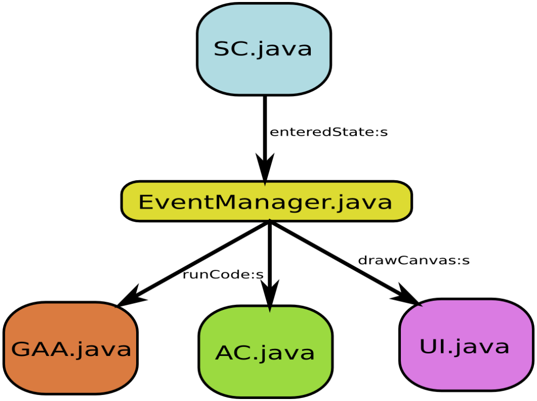
on event `enteredState:s`, produce event `drawCanvas:s`
on event `enteredState:s`, produce event `runCode:s`

outline

beneath the *phoneapps* DSL



beneath a *phoneapps* DSm



- generated artifact(s) for each base formalism
- interaction via events

outline

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!