modular synthesis of mobile device applications from domain-specific models

raphaël mannadiar and hans vangheluwe

presented by bart meyers

MOMPES 2010

outline

1 context and problem

2 our approach

3 examples

4 conclusion

outline

1 context and problem

2 our approach

3 examples

4 conclusion

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

but the process can be automated!

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

but the process can be automated!

dsm allows domain experts to play active roles in the development process, even if they aren't solution domain experts

what's under the hood?

artifacts are generated from domain-specific models (ds*m*s)

artifacts may be configuration files, programs, performance models, etc.

traditionally, this is done via ad-hoc hand-coded generators that parse and "compile" models via modelling tool APIs

so what's wrong?

traceability between dsms and artifacts is necessary for debugging and reasoning about (and more!) dsms

unstructured artifact generation makes maintaining traceability more complex

maintaining traceability makes unstructured artifact generation more complex

so what's wrong?

traceability between ds*m*s and artifacts is necessary for debugging and reasoning about (and more!) ds*m*s

unstructured artifact generation makes maintaining traceability more complex

maintaining traceability makes unstructured artifact generation more complex

dsm is built on artifact generation
dsm should not be built on complex ad-hoc black boxes

our solution, in a nutshell

we propose a more structured approach to artifact generation where layered model transformations are used to modularly isolate, compile and re-combine various aspects of ds*ms*

outline

1 context and problem

2 our approach

3 examples

4 conclusion

phoneapps

we illustrate our approach by describing how google android applications can be synthesized from ds*m*s of mobile device applications

mobile applications = behaviour+layout+device features

phoneapps

we illustrate our approach by describing how google android applications can be synthesized from ds*ms* of mobile device applications

mobile applications = behaviour+layout+device features

→ phoneapps

behaviour

timed, conditional and user-prompted transitions control flow between `ExecutionSteps`

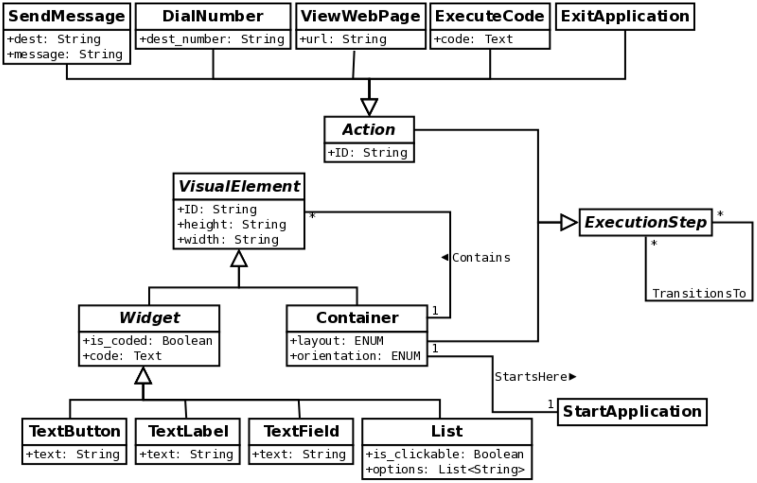
layout

`Containers` contain `Containers` and `Widgets`

device features

sending text messages, dialing numbers, opening browsers

phoneapps...


SendMessage
+dest: String
+message: String
DialNumber
+dest_number: String
ViewWebPage
+url: String
ExecuteCode
+code: Text
ExitApplication
Action
+ID: String
VisualElement
+ID: String
+height: String
+width: String
*
Contains
1
ExecutionStep
*
*
TransitionsTo
Widget
+is_coded: Boolean
+code: Text
Container
+layout: ENUM
+orientation: ENUM
1
StartsHere
1
StartApplication
TextButton
+text: String
TextLabel
+text: String
TextField
+text: String
List
+is_clickable: Boolean
+options: List<String>

phoneapps to google android

a google android application consists in

- a collection of xml files (layout)
- java code (behaviour+device features)

phoneapps to google android

a google android application consists in

- a collection of xml files (layout)
- java code (behaviour+device features)

traditional artifact synthesis approach

run through a phoneapps model with a hand-coded parser and generator and output xml and java files

phoneapps to google android

a **google android** application consists in

- a collection of xml files (layout)
- java code (behaviour+device features)

traditional artifact synthesis approach

run through a phoneapps model with a hand-coded parser and generator and output xml and java files

but we can do better!

phoneapps to ... to google android

$$\begin{tikzcd}
& \text{PhoneApps} \arrow[dl] \arrow[dr] & \\
\text{StateCharts} \arrow[dr] & & \text{AndroidAppScreens} \arrow[dl] \\
& \text{AndroidAppFiles} \arrow[d] & \\
& \text{Actual files (AndroidManifest.xml, PhoneApp.java, PhoneAppStateChart.java, screen\_*.xml)} &
\end{tikzcd}$$

isolating behaviour

what formalism can we map the behavioural aspects of a phoneapps ds*m* onto?

a formalism that **models behaviour** well, and whose semantics and **mapping to code** are well understood

like **statecharts**

isolating behaviour

what formalism can we map the behavioural aspects of a phoneapps ds*m* onto?

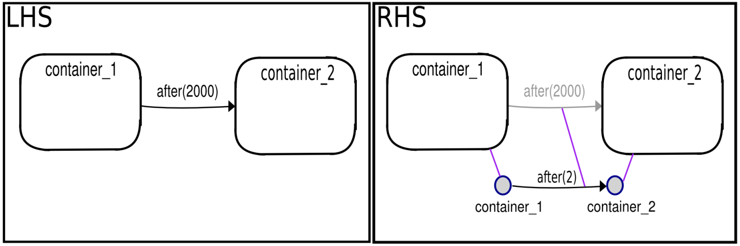
a formalism that **models behaviour** well, and whose semantics and **mapping to code** are well understood

like **statecharts**

ok, but **why** isolate behaviour?

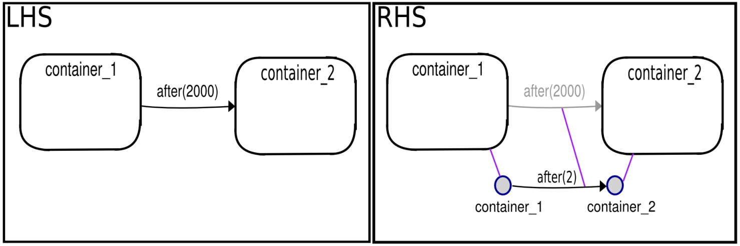
to **display**, **debug**, **understand**, **compile** (and more!) an application's behaviour without irrelevant distractions

isolating behaviour...



one rule of the phoneapps-to-statecharts **model transformation**

isolating behaviour...



one rule of the phoneapps-to-statecharts model transformation

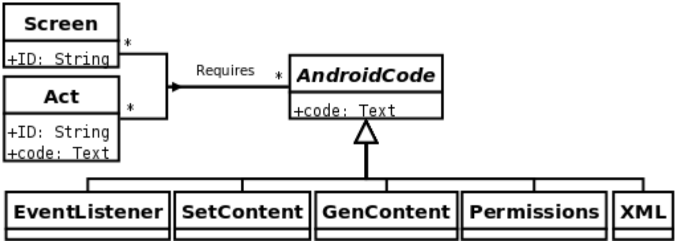
top-level `Containers` and `Actions` become statechart `States`

these `States`' entry actions are populated with callback code that to launch `Actions` and display `Containers`

the edges between them become statechart `Transitions`

isolating layout and device features

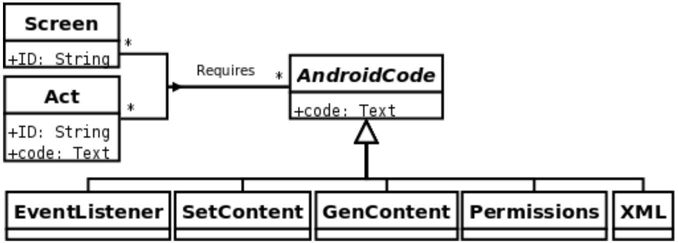
what formalism can describe disjoint screens and operations?



android*apps*creens

isolating layout and device features

what formalism can describe disjoint screens and operations?

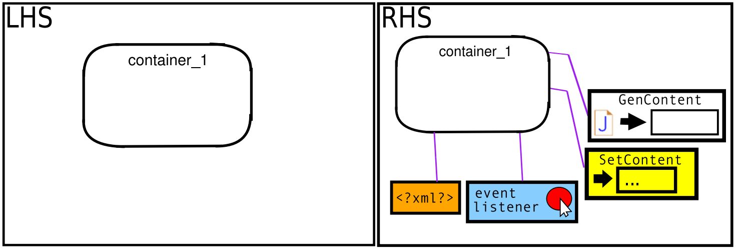


androidappscreens

mea culpa

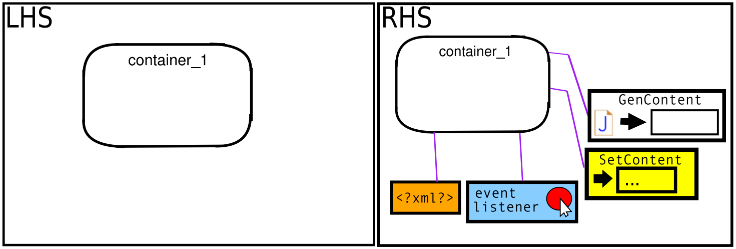
future work should isolate layout and device features separately

isolating layout and device features...



one rule of the phoneapps-to-androidappscreens model transformation

isolating layout and device features...



one rule of the phoneapps-to-androidappscreens model transformation

`Containers` and `Widgets` are broken down into code (xml, java) snippets

top-level `Containers` and `Actions` become `Screens` and `Acts`

benefits of isolation

artifact generation is more **structured** and **modular**

benefits of isolation

artifact generation is more structured and modular

simplified views of the system can be reviewed and studied

benefits of isolation

artifact generation is more structured and modular

simplified views of the system can be reviewed and studied

the purple edges between constructs from different formalisms establish correspondences that can be used to

- propagate data between artifact and ds*m* for animation and debugging
- easily observing what was generated from what
- relate higher and lower level constructs for educating

benefits of isolation

artifact generation is more **structured** and **modular**

simplified views of the system can be reviewed and studied

the purple edges between constructs from different formalisms establish **correspondences** that can be used to

- propagate data between artifact and ds*m* for **animation** and **debugging**
- easily **observing** what was generated from what
- relate higher and lower level constructs for **educating**

but most of all

the **level of abstraction** of artifact synthesis is **raised** from complex code and tool APIs to domain-specific constructs and simple rules

modelling and generating artifacts

androidappscreens and **statechart** models need to be **merged** into a google android application

this is achieved via 2 **model transformations** that incrementally generate

- xml code for each `Screen`
- java code for the compiled statechart
- java code for the compiled layout and device features

modelling and generating artifacts

androidappscreens and statechart models need to be merged into a google android application

this is achieved via 2 model transformations that incrementally generate

- xml code for each `Screen`
- java code for the compiled statechart
- java code for the compiled layout and device features

but this code is not written to disk!

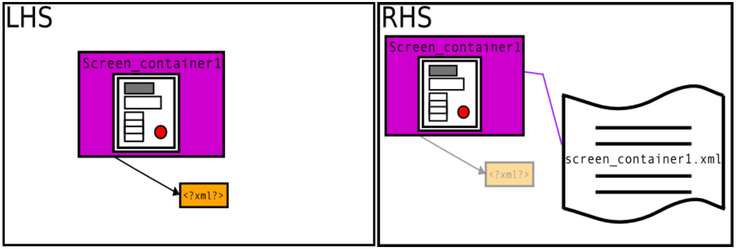
modelling and generating artifacts...

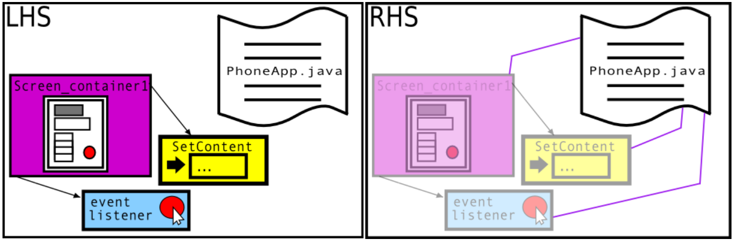
to facilitate the continued **linking** of artifacts and models, and

to **review** the generated code within the modelling tool

the final **files are modelled** before being written to disk

modelling and generating artifacts...





outline

1 context and problem

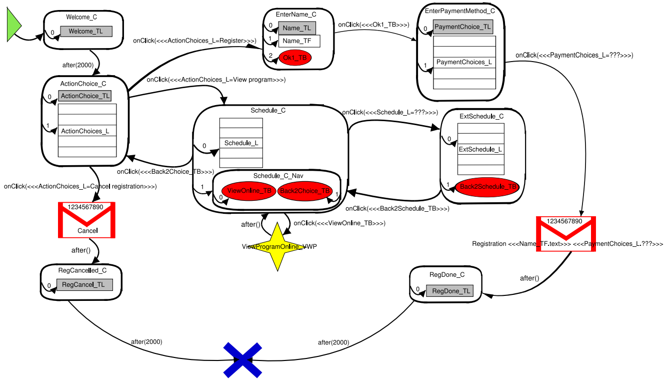
2 our approach

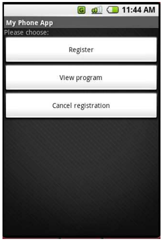
3 examples

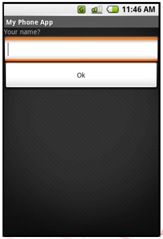
4 conclusion

example 1 : conference registration

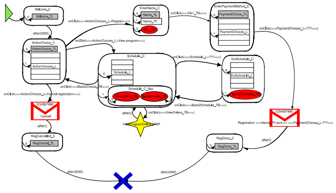
synthesizing a mobile conference registration application from a phoneapps dsm

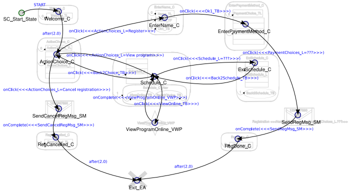


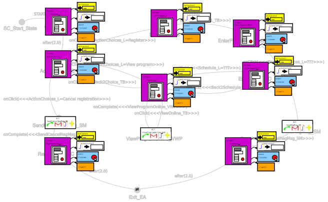


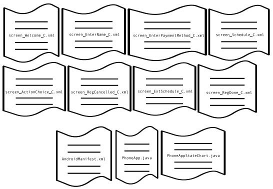


4 perspectives

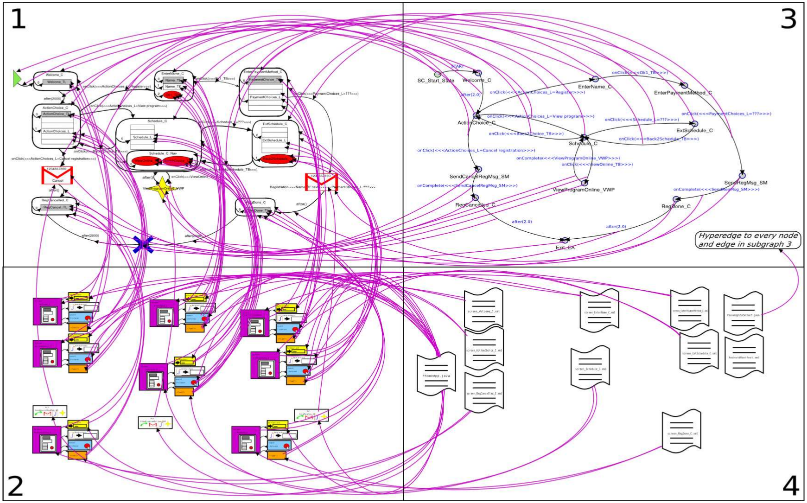



START
SC_Start_State
Welcome_C
EnterName_C
EnterPaymentMethod_C
ActionChoice_C
Schedule_C
ExtSchedule_C
SendCancelRegMsg_SM
ViewProgramOnline_VWP
SendRegMsg_SM
RegCancelled_C
RegDone_C
Exit_EA
after(2.0)
onClick(<<<ActionChoices_L=Register>>>)
onClick(<<<Ok1_TB>>>)
onClick(<<<ActionChoices_L=View program>>>)
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>>>)


SC_Start_State
after(2.0)
onClick(<<<ActionChoices_L=Cancel registration>>>)
onComplete(<<<ViewProgramOnline_VWP>>>)
onClick(<<<ViewOnline_TB>>>)
Exit_EA



a web of correspondence links



example 2 : performance metrics

how can we **measure** or **estimate** execution time, battery usage (and more) to satisfy **non-functional requirements**?

to **answer design questions** like "should data be fetched or preloaded"?

… old school

existing code generators could be extended to output performance metrics

this would reduce their modularity and increase their complexity

... old school

existing code generators could be extended to output performance metrics

this would reduce their modularity and increase their complexity

a new generator could be written to output performance metrics

this would probably result in considerable code duplication

… old school

existing code generators could be extended to output performance metrics

> this would reduce their modularity and increase their complexity

a new generator could be written to output performance metrics

> this would probably result in considerable code duplication

in both cases, advanced features (e.g., "tagging" domain-specific constructs with battery usage, after or during execution) that require communication between model and artifacts would add considerable complexity to the generator's code

… with our approach

existing model transformations could be extended to output performance metrics

this would reduce their modularity and increase their complexity

a new model transformation could be written to output performance metrics

this would probably result in considerable duplication

… with our approach

existing model transformations could be extended to output performance metrics

this would reduce their modularity and increase their complexity

a new model transformation could be written to output performance metrics

this would probably result in considerable duplication

but, correspondence links between model and artifacts would be left behind to facilitate implementing advanced features

outline

1 context and problem

2 our approach

3 examples

4 conclusion

our solution, in a nutshell

we proposed a more structured approach to artifact generation where layered model transformations are used to modularly isolate, compile and re-combine various aspects of ds*ms*

our approach raises the level of abstraction of artifact synthesis and leaves behind a web of correspondence links that enables a wide range of advanced activites including debugging, educating and performance measuring

questions?

thank you!