1 context and problem
2 our approach
3 examples
4 conclusion

modular synthesis of mobile device applications from domain-specific models
context and problem

our approach

domains

examples

conclusion

modular synthesis of mobile device applications from domain-specific models
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 (dsms).

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 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

dsm is built on artifact generation

**dsm should not be built on complex ad-hoc black boxes**
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 dsms
1. context and problem

2. our approach

3. examples

4. conclusion
we illustrate our approach by describing how google android applications can be synthesized from dsms of mobile device applications

mobile applications = behaviour + layout + device features
we illustrate our approach by describing how Google Android applications can be synthesized from DSMS of mobile device applications

mobile applications = behaviour + layout + device features

→ phoneapps
modular synthesis of mobile device applications from domain-specific models
a **Google Android** application consists in

- a collection of XML files (**layout**)
- Java code (**behaviour + device features**)

---

**Modular synthesis of mobile device applications from domain-specific models**
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

PhoneApps

StateCharts  AndroidAppScreens

AndroidAppFiles

Actual files (AndroidManifest.xml, PhoneApp.java, PhoneAppStateChart.java, screen_*.xml)
isolating behaviour

what formalism can we map the behavioural aspects of a phone apps 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 phone apps 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

modular synthesis of mobile device applications from domain-specific models
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 app screens

modular synthesis of mobile device applications from domain-specific models
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 \textit{model transformation}
isolating layout and device features...

one rule of the phoneapps-to-androidappsscreens 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 DSM 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 dsm 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...

context and problem
our approach
elements
conclusion
module synthesis of mobile device applications from domain-specific models
example 1: conference registration

synthesizing a mobile conference registration application from a phoneapps DSM

modular synthesis of mobile device applications from domain-specific models
4 perspectives

modular synthesis of mobile device applications from domain-specific models
a web of correspondence links

modular synthesis of mobile device applications from domain-specific models
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”?
existing code generators could be extended to output performance metrics

this would reduce their modularity and increase their complexity
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
modular synthesis of mobile device applications from domain-specific models
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 dsms**

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 activities including **debugging**, **educating** and **performance measuring**
thank you!