A language-independent methodology for compiling declarations into open platform frameworks

Paul van der Walt

http://phoenix.inria.fr

Advisor: Charles Consel

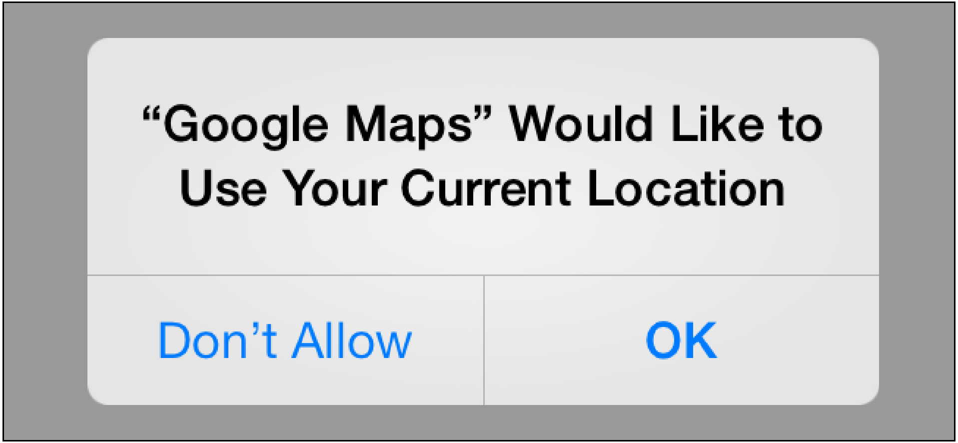
14th of December, 2015

Problem statement

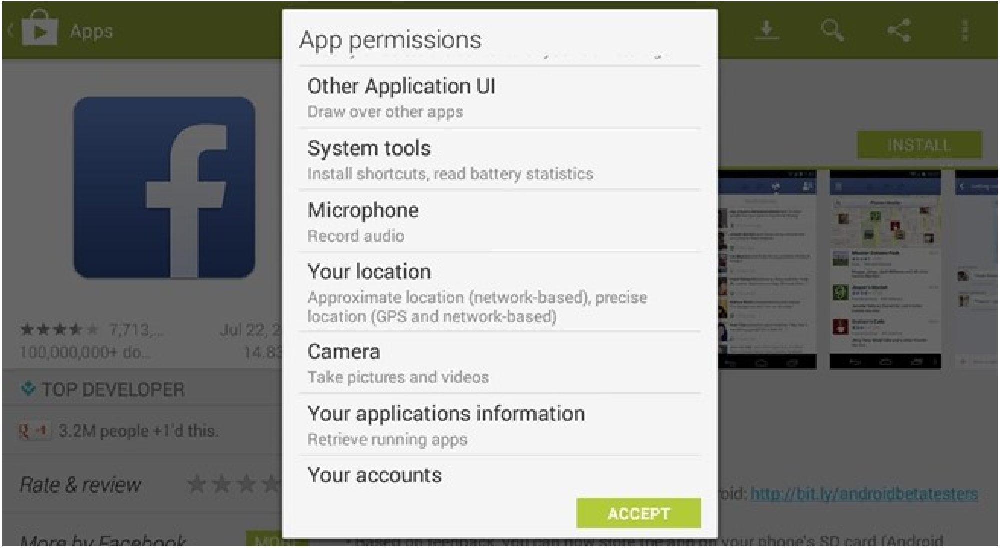
- Mobile devices extremely widespread
- . . . containing ever more personal data
- Untrusted applications have access

Some perspective

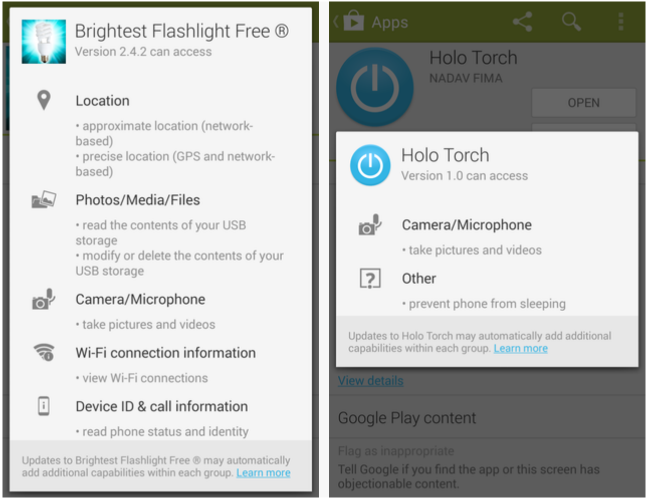
An improvement!



What does this application do?



And these ones?



[Wei et al., 2012]

Remark: why focus on privacy?

- Methodology is not limited to privacy preservation
- Previously shown to work for QoS, simulation, *etc.* [Gatti, 2014, Bruneau and Consel, 2013]
- Privacy is a relatable motivation, highlighting consequences of design decisions

Running example: EvilCam!

Running example application.

Supposedly:

- Takes a picture
- Applies sepia filter
- Displays it to user

Running example: EvilCam!

Running example application.

Supposedly:

- Takes a picture

- Applies sepia filter
- Displays it to user
- . . . and shows an advert

Running example: EvilCam!

Running example application.

Supposedly:

- Takes a picture
  $\rightarrow$ $\boxed{\text{camera}}$ permission
- Applies sepia filter
- Displays it to user
- . . . and shows an advert

Running example: EvilCam!

Running example application.

Supposedly:

- Takes a picture
  $\rightarrow$ $\boxed{\text{camera}}$ permission
- Applies sepia filter
- Displays it to user
- . . . and shows an advert
  $\rightarrow$ $\boxed{\text{network}}$ permission

Potential data flow

What one hopes:

- camera $\rightarrow$ screen
- internet $\rightarrow$ fetch advert
- nothing more.

Potential data flow

What one hopes:

- camera → screen
- internet → fetch advert
- ~~nothing more.~~

Reality:

- image → stalker.net and nsa.gov [Do et al., 2015, Stevens et al., 2012, Felt et al., 2012]

Challenges

Guarantees:

- Transparency, empowering the end-user
- Containment of data flow
- Conformance of behaviour to specification

Guidance:

- Support for the developer with framework

[Balland and Consel, 2010]

Related Work

Static program analysis [Liu and Milanova, 2008, Elish et al., 2013, Xiao et al., 2012]

- Prefer to avoid inspecting source code (invasive, copyright)
- Frequently inaccurate, difficult problem [Rountev et al., 2004]
- Limited user transparency

Related Work

Real-time (remote) taint analysis [Enck et al., 2014]

- Not desirable on mobile devices (limited computational power)
- Lack of developer support
- Privacy concerns!
- Will not scale

Related Work

Operating system security (capability-based systems)
[Watson et al., 2010, Shapiro et al., 1999, Shapiro et al., 2004]

- Data-flow capabilities only enforced at run-time
- Major changes to existing infrastructure
- Potentially not fine-grained enough (per-app, *e.g.*, Android)

Related Work

Language-level restrictions

ELib, W7 [Rees, 1995, Miller, 2006]

- Powerful approach, permissions per component baked into language
- Again, low adoption,
- major changes required

DiaSuite [Cassou et al., 2012], created in research team

- Specify app → generate framework
- Minimal infrastructure modification
- Previously mainly for assisted living / home automation
- Only in the context of Java!

Improving on DiaSuite

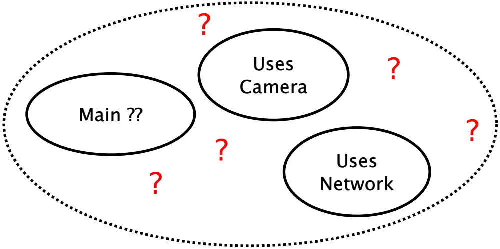
- Work builds upon DiaSuite methodology
  - No infrastructure changes required
  - Promising tailored framework approach
- Rethink the approach, without assumptions
- Delineate then explore the design space

Major thesis contributions

- Formalisation of key phases of existing DiaSuite methodology
  - To reveal design choices
  - . . . and design decisions influence behaviour (example is privacy: consequences)
  - Identify key concepts. How do they map into PL concepts?
- Generalisation to language-independent methodology
  - Explore spectrum of programming languages
- Application to mobile computing domain
- Prototype implementations

The problem with current declaration approaches

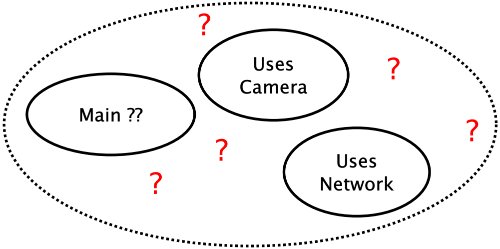
The Android model: permissions



Even with conservative permissions,
behaviour is unpredictable.

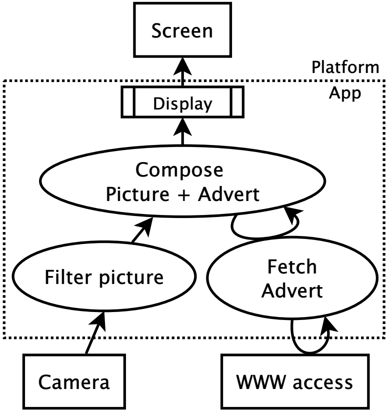
The problem with current declaration approaches

The Android model: permissions



Even with conservative permissions, behaviour is unpredictable.

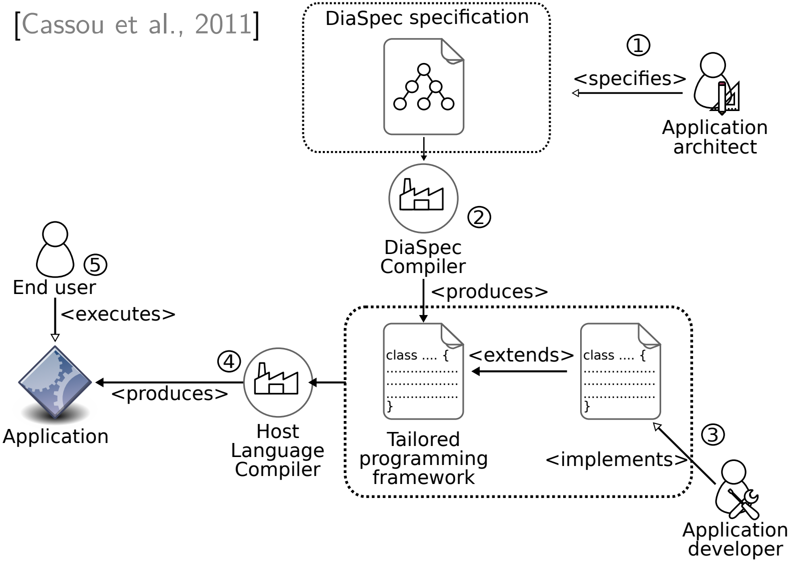
The DiaSuite approach: decomposition+permissions



(SCC) [Taylor et al., 2009]

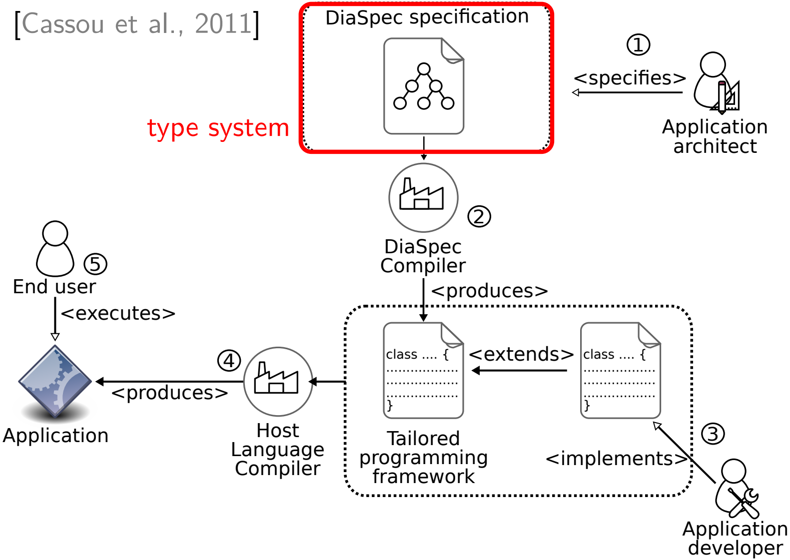
How (existing) DiaSuite methodology works

[Cassou et al., 2011]



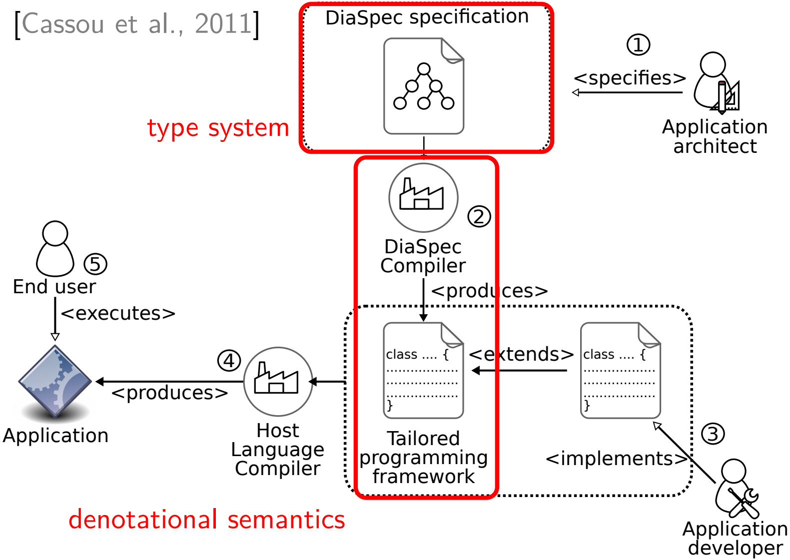
How (existing) DiaSuite methodology works

[Cassou et al., 2011]



How (existing) DiaSuite methodology works

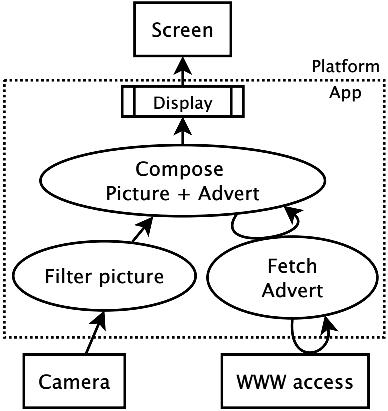
[Cassou et al., 2011]



Example of types

Example:

```
(source  Camera as Pic)
(context Filter as Pic
  [when provided Camera
    (get nothing)
    always-publish])
```



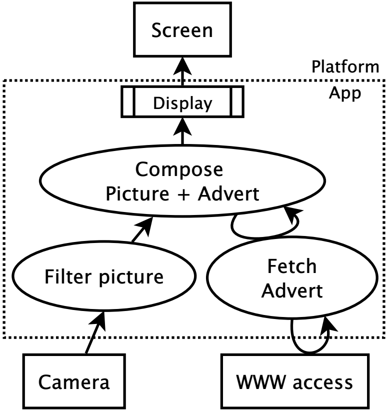
Example of types

Example:

```
(source  Camera as Pic)
(context Filter as Pic
  [when provided Camera
    (get nothing)
    always-publish])
```

should result in:

```
Camera :: Pic
Filter :: Pic -> () -> Pic
```



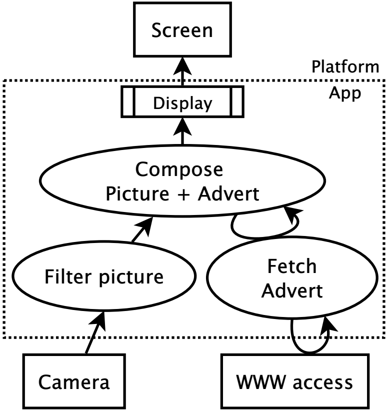
Example of types

Example:

```
(source  Camera as Pic)
(context Filter as Pic
  [when provided Camera
    (get nothing)
    always-publish])
```

should result in:

```
Camera :: Pic
Filter :: Pic -> () -> Pic
```



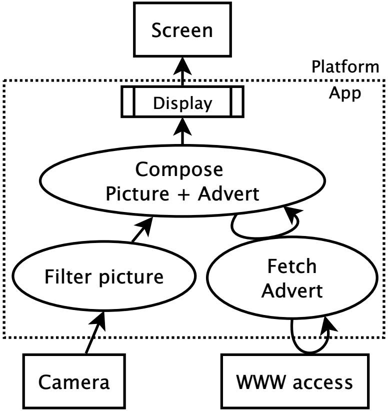
Example of types

Example:

```
(source  Camera as Pic)
(context Filter as Pic
  [when provided Camera
    (get nothing)
    always-publish])
```

should result in:

```
Camera :: Pic
Filter :: Pic -> () -> Pic
```



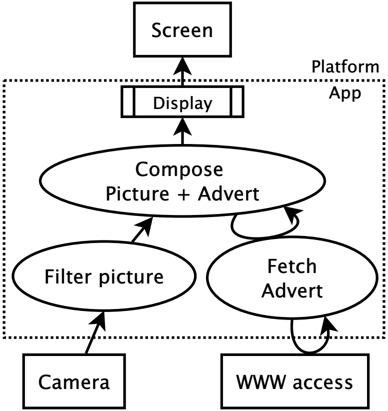
Example of types

Example:

```
(source  Camera as Pic)
(context Filter as Pic
  [when provided Camera
    (get nothing)
    always-publish])
```

should result in:

```
Camera :: Pic
Filter :: Pic -> () -> Pic
```



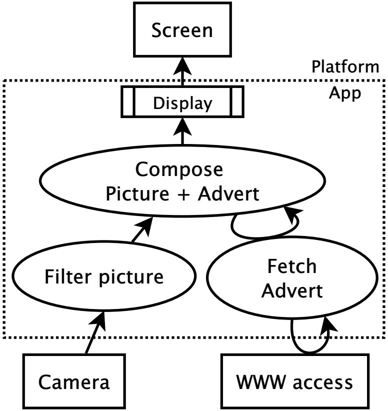
Example of types

Example:

```
(source  Camera as Pic)
(context Filter as Pic
  [when provided Camera
    (get nothing)
    always-publish])
```

should result in:

```
Camera :: Pic
Filter :: Pic -> () -> Pic
```



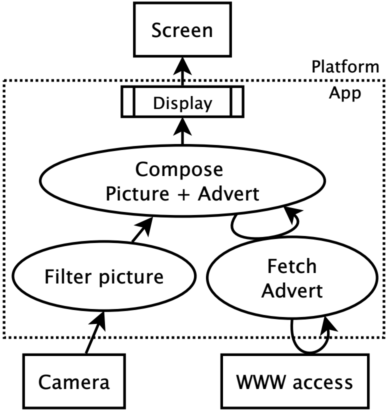
Example of types

Example:

```
(source  Camera as Pic)
(context Filter as Pic
  [when provided Camera
    (get nothing)
    always-publish])
```

should result in:

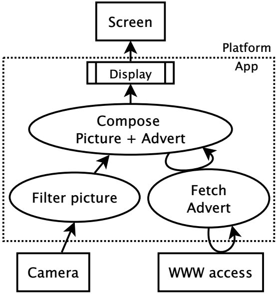
```
Camera :: Pic
Filter :: Pic -> () -> Pic
```



Done using PLT Redex [Felleisen et al., 2009]

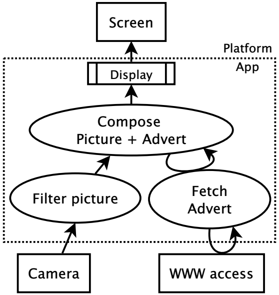
DiaSpec recap, types

$$
\begin{array}{rl}
\mathit{specification} ::= & (\mathit{declaration} \ \ldots) \\
\mathit{declaration} ::= & (\text{source} \quad X \text{ as } \tau) \\
 | & (\text{action} \quad X \text{ as } \tau) \\
 | & (\text{context} \quad X \text{ as } \tau \ \mathit{ctxt\text{-}interact}) \\
 | & (\text{controller } X \quad \mathit{ctrl\text{-}interact}) \\
\tau ::= & \text{Bool} \\
 | & \text{Int} \\
 | & \text{String} \\
 | & \text{Picture} \\
\mathit{ctxt\text{-}interact} ::= & [\text{when provided } Y \ \mathit{getresource} \ \mathit{pub}] \\
 | & [\text{when required} \quad \mathit{getresource}] \\
\mathit{ctrl\text{-}interact} ::= & [\text{when provided } Y \text{ do } Z] \\
\mathit{getresource} ::= & (\text{get nothing}) \\
 | & (\text{get } Z) \\
\mathit{pub} ::= & \text{always-publish} \\
 | & \text{maybe-publish} \\
X, Y, Z ::= & \mathit{variable\text{-}not\text{-}otherwise\text{-}mentioned}
\end{array}
$$



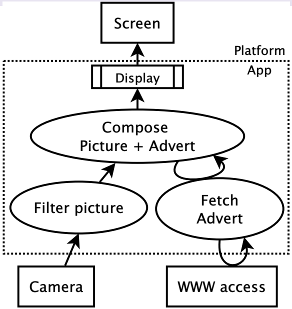
DiaSpec recap, types

$$
\begin{aligned}
\mathit{specification} &::= (\mathit{declaration}\ \ldots)\\
\mathit{declaration} &::= (\textsf{source}\quad X\ \textsf{as}\ \tau)\\
&\mid (\textsf{action}\quad X\ \textsf{as}\ \tau)\\
&\mid (\textsf{context}\quad X\ \textsf{as}\ \tau\ \mathit{ctxt\text{-}interact})\\
&\mid (\textsf{controller}\ X\quad \mathit{ctrl\text{-}interact})\\
\tau &::= \textsf{Bool}\\
&\mid \textsf{Int}\\
&\mid \textsf{String}\\
&\mid \textsf{Picture}\\
\mathit{ctxt\text{-}interact} &::= [\textsf{when provided}\ Y\ \mathit{getresource}\ \mathit{pub}]\\
&\mid [\textsf{when required}\quad \mathit{getresource}]\\
\mathit{ctrl\text{-}interact} &::= [\textsf{when provided}\ Y\ \textsf{do}\ Z]\\
\mathit{getresource} &::= (\textsf{get nothing})\\
&\mid (\textsf{get}\ Z)\\
\mathit{pub} &::= \textsf{always-publish}\\
&\mid \textsf{maybe-publish}\\
X, Y, Z &::= \mathit{variable\text{-}not\text{-}otherwise\text{-}mentioned}
\end{aligned}
$$



$$
\begin{aligned}
\Gamma &::= ((X : t)\ \ldots)\\
t &::= (\textsf{ACT}\ \tau)\\
&\mid (\textsf{SRC}\ \tau)\\
&\mid (\textsf{CTX-req}\ \tau)\\
&\mid (\textsf{CTX-prov}\ \tau)\\
&\mid (\textsf{CTRL})
\end{aligned}
$$

Type system

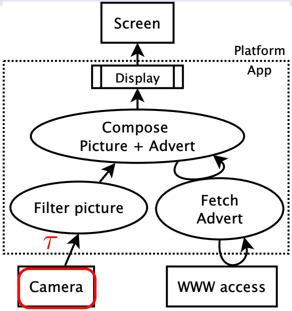


- Type system encodes constraints of SCC

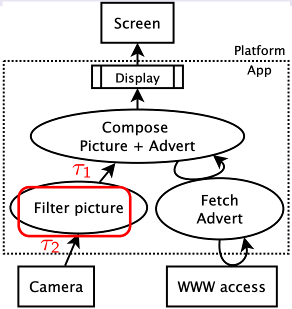
Type system

- Type system encodes constraints of SCC

$$\frac{\text{unique?}[\![X, \Gamma]\!]}{\vdash [\![\Gamma, (\text{source } X \text{ as } \tau), (\text{SRC } \tau)]\!]}\ \text{[intro-src]}$$



Type system

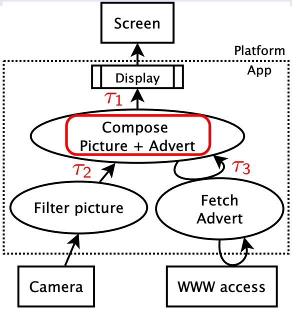


- Type system encodes constraints of SCC

$$\frac{\text{unique?}[\![X, \Gamma]\!]}{\vdash [\![\Gamma, (\text{source } X \text{ as } \tau), (\text{SRC } \tau)]\!]}\ [\text{intro-src}]$$

$$\frac{\begin{array}{c}(\text{SRC } \tau_2) = \text{lookup}[\![\Gamma, X_2]\!] \\ \text{unique?}[\![X_1, \Gamma]\!]\end{array}}{\vdash [\![\Gamma, (\text{context } X_1 \text{ as } \tau_1 \text{ [when provided } X_2 \text{ (get nothing) \_]}), (\text{CTX-prov } \tau_1)]\!]}\ [\text{ctx-onSrc-get-ø}]$$

Type system



- Type system encodes constraints of SCC

$$\frac{\text{unique?}[\![X, \Gamma]\!]}{\vdash [\![\Gamma, (\text{source } X \text{ as } \tau), (\text{SRC } \tau)]\!]}\ \text{[intro-src]}$$

$$\frac{\begin{array}{c}(\text{SRC } \tau_2) = \text{lookup}[\![\Gamma, X_2]\!] \\ \text{unique?}[\![X_1, \Gamma]\!]\end{array}}{\vdash [\![\Gamma, (\text{context } X_1 \text{ as } \tau_1 \text{ [when provided } X_2 \text{ (get nothing) \_]}), (\text{CTX-prov } \tau_1)]\!]}\ \text{[ctx-onSrc-get-ø]}$$

$$\frac{\begin{array}{c}(\text{CTX-prov } \tau_2) = \text{lookup}[\![\Gamma, X_2]\!] \\ (\text{CTX-req } \tau_3) = \text{lookup}[\![\Gamma, X_3]\!] \\ \text{unique?}[\![X_1, \Gamma]\!]\end{array}}{\vdash [\![\Gamma, (\text{context } X_1 \text{ as } \tau_1 \text{ [when provided } X_2 \text{ (get } X_3\text{) \_]}), (\text{CTX-prov } \tau_1)]\!]}\ \text{[ctx-onCtx-get-ctx]}$$

A note on stages

Practical question: when do we implement checks?

A note on stages

Practical question: when do we implement checks?

	Publication and types	Resource access
Compile-time	static fn types	no invalid-access crash (no examples!)
Run-time	contracts, guards	more accuracy: *e.g.*, address book entries (Android, iOS, ...)
	both feasible	depends!

Note: choice need not be global.

Especially resource access is an important decision. See Ch. 7.2.

Semantics

- Requirement: Decouple approach from Java implementation
- Requirement: Clarify where choice can be made for static/dynamic checks
- Translation from DiaSpec $\rightarrow$ simply-typed lambda calculus
- Using STLC, encode the shape of the framework (intermediate language for compiler back-end)

Semantics

$$[\![(\texttt{source}\ X\ \texttt{as}\ \tau)]\!]_{eval} \quad \rightsquigarrow \quad (\lambda() \quad \boxed{\{\ \}_?} \quad) \ :: \ [\![\tau]\!]_{type}$$

Semantics

$$[\![(\texttt{\textbf{source}}\ X\ \texttt{\textbf{as}}\ \tau)]\!]_{eval} \quad \rightsquigarrow \quad (\lambda() \quad \boxed{\{\ \}_?} \quad) :: [\![\tau]\!]_{type}$$

$$[\![(\texttt{\textbf{context}}\ X\ \texttt{\textbf{as}}\ \tau\ [\texttt{\textbf{when provided}}\ X_2\ get\ pub])]\!]_{eval}$$

$$\rightsquigarrow$$

$$(\lambda(x_2 :: [\![X_2]\!]_{type},\ x_3 :: [\![get]\!]_{get}) \quad \boxed{\{\ \}_?} \quad) :: [\![pub, \tau]\!]_{pub}$$

Semantics

$$[\![(\texttt{\textbf{source}}\ X\ \texttt{\textbf{as}}\ \tau)]\!]_{eval} \quad \rightsquigarrow \quad (\lambda() \quad \boxed{\{\ \}_?} \quad) :: [\![\tau]\!]_{type}$$

$$[\![(\texttt{\textbf{context}}\ X\ \texttt{\textbf{as}}\ \tau\ [\texttt{\textbf{when provided}}\ X_2\ get\ pub])]\!]_{eval}$$

$$\rightsquigarrow$$

$$(\lambda(x_2 :: [\![X_2]\!]_{type},\ x_3 :: [\![get]\!]_{get}) \quad \boxed{\{\ \}_?} \quad) :: [\![pub, \tau]\!]_{pub}$$

$$[\![(\texttt{\textbf{get nothing}})]\!]_{get} \quad \rightsquigarrow \quad \texttt{NULL}$$

$$[\![(\texttt{\textbf{get}}\ Y)]\!]_{get} \quad \rightsquigarrow \quad (\texttt{NULL} \rightarrow [\![Y]\!]_{type})$$

Example for Camera and Filter

$$[\![(\texttt{\textbf{source}}\ \mathit{Camera}\ \texttt{\textbf{as}}\ \mathit{Pic})]\!]_{\mathit{eval}} \quad \rightsquigarrow \quad (\lambda() \quad \boxed{\{\ \}_?} \quad) :: [\![\mathit{Pic}]\!]_{\mathit{type}}$$

Example for Camera and Filter

$$[\![(\textbf{source}\ \mathit{Camera}\ \textbf{as}\ \mathit{Pic})]\!]_{\mathit{eval}} \quad \rightsquigarrow \quad (\lambda() \quad \boxed{\{\ \}_?} \quad) :: [\![\mathit{Pic}]\!]_{\mathit{type}}$$

$$\begin{aligned}[\![(\textbf{context}\ \mathit{Filter}\ \textbf{as}\ \mathit{Pic}\\ &[\textbf{when provided}\ \mathit{Camera}\\ &(\texttt{get nothing})\ \texttt{always-publish}])]\!]_{\mathit{eval}}\end{aligned}$$

$$\rightsquigarrow$$

$$(\lambda(x_2 :: [\![\mathit{Pic}]\!]_{\mathit{type}},\ x_3 :: ()) \quad \boxed{\{\ \}_?} \quad) :: [\![\mathit{Pic}]\!]_{\mathit{type}}$$

Note: important choice here regarding static/dynamic enforcing!

Implementation

- We want to explore the spectrum of programming paradigms
- Investigate checks at different stages (compile-time, run-time, . . .)
- Statically typed, dynamically typed

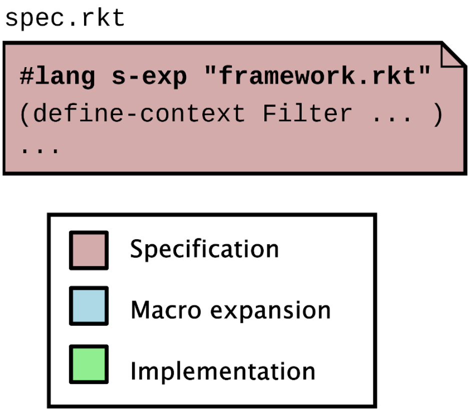
Implementation

- We want to explore the spectrum of programming paradigms
- Investigate checks at different stages (compile-time, run-time, . . .)
- Statically typed, dynamically typed
- Racket is a good language-experimentation tool
  - DSL experimentation
  - contract library
  - advanced module system
  - versatile: static/dynamic typing, OO, FP, . . .

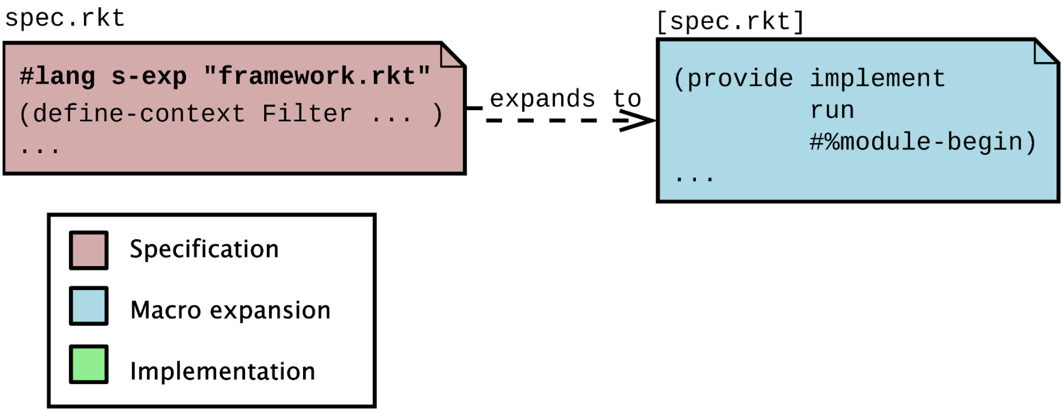
Contributions in this section

- Showing that methodology generalises; discovering design possibilities
- Framework design as language generation (**`#lang`**)
  - An aside: frameworks need not only be an OO phenomenon

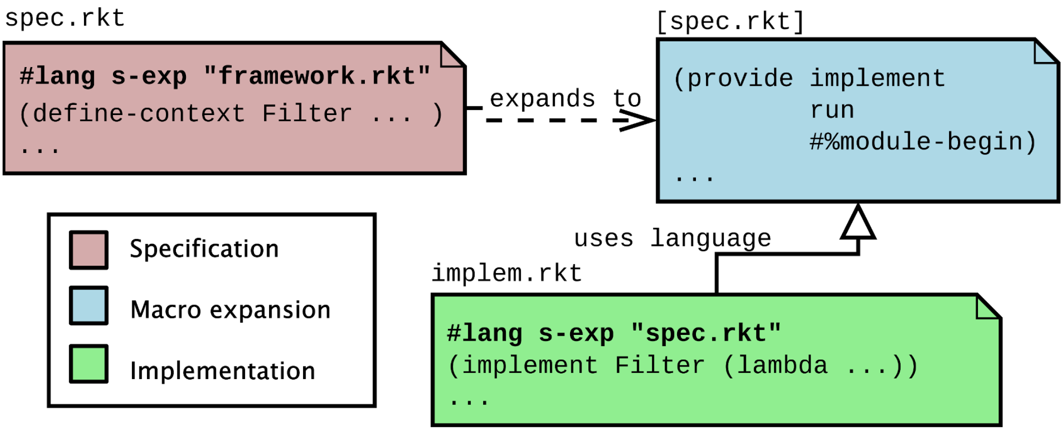
Racket prototype architecture



Racket prototype architecture



Racket prototype architecture



Application specification

- Example from the point of view of the application developer

Application specification

- Example from the point of view of the application developer

```
#lang s-exp "framework.rkt"
;;; Specifications file, webcamspec.rkt
```

Application specification

- Example from the point of view of the application developer

```
#lang s-exp "framework.rkt"
;;; Specifications file, webcamspec.rkt

(define-source  Camera Picture) ; built-in

(define-context Filter                      ; name
                Picture                     ; return type
                [when-provided Camera])     ; subscribed to
;; ...
```

Application implementation

The developer does the following:

```
;;; Implementation file, webcamimpl.rkt
#lang s-exp "webcamspec.rkt"
```

Application implementation

The developer does the following:

```
;;; Implementation file, webcamimpl.rkt
#lang s-exp "webcamspec.rkt"
(implement Filter
```

Application implementation

The developer does the following:

```
;;; Implementation file, webcamimpl.rkt
#lang s-exp "webcamspec.rkt"
(implement Filter
  (lambda (pic)
```

Application implementation

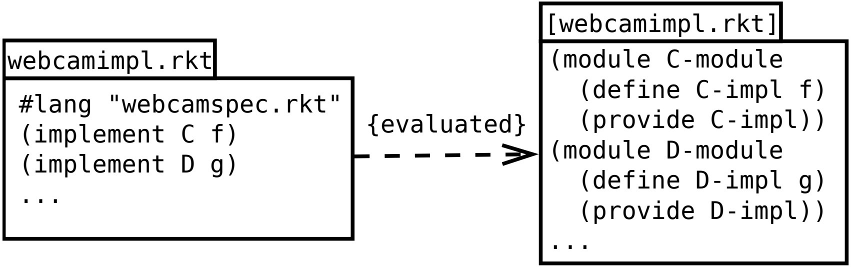
The developer does the following:

```
;;; Implementation file, webcamimpl.rkt
#lang s-exp "webcamspec.rkt"
(implement Filter
  (lambda (pic)
    (let* ([canvas  (make-bitmap pic ..)])
      ; ... process the picture
      canvas)))
;; ...
```

But what about conformance? Are other components in scope? Are the types correct?
When should we actually check?

Separation into submodules

Compartmentalise with lexical scoping: C and D cannot communicate.



Implementation

So, the `implement` transformer expands to:

```
(module webcamimpl "webcamspec.rkt"
```

Implementation

So, the `implement` transformer expands to:

```
(module webcamimpl "webcamspec.rkt"
  (module Filter-module racket/gui
```

Implementation

So, the implement transformer expands to:

```
(module webcamimpl "webcamspec.rkt"
  (module Filter-module racket/gui
    (define/contract Filter-impl
      (-> bitmap%? bitmap%?)
```

Implementation

So, the `implement` transformer expands to:

```
(module webcamimpl "webcamspec.rkt"
  (module Filter-module racket/gui
    (define/contract Filter-impl
      (-> bitmap%? bitmap%?)
      ;; lambda-term from previous step
    )
    (provide Filter-impl))
  ...)
```

Implementation

So, the `implement` transformer expands to:

```
(module webcamimpl "webcamspec.rkt"
  (module Filter-module racket/gui
    (define/contract Filter-impl
      (-> bitmap%? bitmap%?)
      ;; lambda-term from previous step
    )
    (provide Filter-impl))
  ...)
```

Note: Semantics and decisions

Implementation

So, the `implement` transformer expands to:

```
(module webcamimpl "webcamspec.rkt"
  (module Filter-module racket/gui
    (define/contract Filter-impl
      (-> bitmap%? bitmap%?)
      ;; lambda-term from previous step
    )
    (provide Filter-impl))
  ...)
```

```
(module webcamimpl "webcamspec.rkt"
  (module Filter-module typed/racket
    (: Filter-impl (-> Bitmap Bitmap))
    (define Filter-impl
      ;; lambda-term from previous step
    )
    (provide Filter-impl))
  ...)
```

Implementation

So, the `implement` transformer expands to:

```
(module webcamimpl "webcamspec.rkt"
  (module Filter-module racket/gui
    (define/contract Filter-impl
      (-> bitmap%? bitmap%?)
      ;; lambda-term from previous step
    )
    (provide Filter-impl))
  ...)
```

```
(module webcamimpl "webcamspec.rkt"
  (module Filter-module typed/racket
    (: Filter-impl (-> Bitmap Bitmap))
    (define Filter-impl
      ;; lambda-term from previous step
    )
    (provide Filter-impl))
  ...)
```

The generated `webcamspec` language also

- checks that all `define`s have `implement`s
- and provides `run`

Evaluation

- Transparency: allow end-user to make an informed decision
  - Finer-grained specifications

Evaluation

- Transparency: allow end-user to make an informed decision
  - Finer-grained specifications
- Containment: predict where data can end up, what it will be used for
  - Framework controls data flow and separates into submodules

Evaluation

- Transparency: allow end-user to make an informed decision
  - Finer-grained specifications
- Containment: predict where data can end up, what it will be used for
  - Framework controls data flow and separates into submodules
- Conformance: ensure that the behaviour of the application corresponds to the specification (Ch. 4.4)
  - Developer can only provide a valid snippet of code (contract or type checking)

Evaluation

- Transparency: allow end-user to make an informed decision
  - Finer-grained specifications
- Containment: predict where data can end up, what it will be used for
  - Framework controls data flow and separates into submodules
- Conformance: ensure that the behaviour of the application corresponds to the specification (Ch. 4.4)
  - Developer can only provide a valid snippet of code (contract or type checking)
- Support: help the developer as much as possible
  - Warnings given if application does not conform

Limitations: Reflection

- Reflection (and `eval` in Racket) would allow circumventing access control

Limitations: Reflection

- Reflection (and `eval` in Racket) would allow circumventing access control
- Example:

```
(eval '(begin (require net/http-client)
              (define-values (status header response)
                (http-sendrecv "www.google.com" "/" #:ssl? 'tls))
              ...))
```

- Luckily, easy to disable

Limitations: safe module import

Lack of safe module importing

- Importing common module would allow communication
- *E.g.*, context A and B import M, then write to $M.var1$
- Must be solved by run-time / OS (see ELib [Miller, 2006])

Lessons learnt

- Static types are unnecessary [Cassou, 2011]
  - *E.g.*, compile-time resource management in dynamic language is feasible
- In fact, methodology is paradigm-independent [van der Walt et al., 2015]
- Only requirement is pre-run-time stage (Ch. 7.2 §3)
  - Examples include type system, macro stage, external compiler, . . .
- Choosing the right stage to implement a check is crucial (Ch. 7.2 §2)

Summary

- Open platforms are in widespread use
- Concerning privacy, current approaches fall short
  - Require major infrastructure changes
  - Do not provide insight to end-user
- Methodology is applicable to wide spectrum of programming languages
- Rich specifications enable improved guarantees and guidance (illustrated with privacy)
- Methodology is applicable to diverse application domains (not only home automation w/ sensors)

Major thesis contributions

- Formalisation of key phases of existing DiaSuite methodology
  - Requirements for open platforms
  - Type system for specifications
  - Denotational semantics for specification terms
- Generalisation to wide spectrum of languages
  - Only pre-run-time stage necessary [van der Walt et al., 2015]
- Prototype implementations [van der Walt, 2015]
  - Qualitative evaluation according to Requirements
- Application to mobile computing domain
  - Addressing major, widespread privacy concern

Perspectives

- User acceptability study [Felt et al., 2012]
- Improved run-time support (borrow from capability-based systems)
- Specifications drive static analysis [Hallett and Aspinall, 2014]
- Fully formally verified implementation (Coq, Agda, . . .)

References I

Balland, E. and Consel, C. (2010).
Open platforms: New challenges for software engineering.
In *Programming Support Innovations for Emerging Distributed Applications*, PSI EtA '10, pages 3:1–3:4, New York, NY, USA. ACM.

Bruneau, J. and Consel, C. (2013).
Diasim: a simulator for pervasive computing applications.
Software: Practice and Experience, 43(8):885–909.

Cassou, D. (2011).
Développement logiciel orienté paradigme de conception : la programmation dirigée par la spécification.
PhD thesis, Université Sciences et Technologies–Bordeaux I.

Cassou, D., Balland, E., Consel, C., and Lawall, J. (2011).
Leveraging software architectures to guide and verify the development of Sense/Compute/Control applications.
In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 431–440, New York, NY, USA. ACM.

Cassou, D., Bruneau, J., Consel, C., and Balland, E. (2012).
Toward a tool-based development methodology for pervasive computing applications.
IEEE Trans. Software Eng., 38(6):1445–1463.

Do, Q., Martini, B., and Choo, K.-K. R. (2015).
Exfiltrating data from Android devices.
Computers & Security, 48:74–91.

References II

Elish, K. O., Yao, D. D., Ryder, B. G., and Jiang, X. (2013).
A static assurance analysis of Android applications.
Virginia Polytechnic Institute and State University, Tech. Rep.

Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. (2014).
TaintDroid: an information flow tracking system for real-time privacy monitoring on smartphones.
Communications of the ACM, 57(3):99–106.

Felleisen, M., Findler, R. B., and Flatt, M. (2009).
Semantics Engineering with PLT Redex.
The MIT Press, 1st edition edition.

Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., and Wagner, D. (2012).
Android permissions: User attention, comprehension, and behavior.
In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 3. ACM.

Gatti, S. (2014).
A step-wise approach for integrating QoS throughout software development process.
PhD thesis, Université de Bordeaux.

Hallett, J. and Aspinall, D. (2014).
Towards an authorization framework for app security checking.
In *Engineering Secure Software and Systems (ESSoS) PhD Symposium*, Munich, Germany.

References III

Liu, Y. and Milanova, A. (2008).
Static analysis for inference of explicit information flow.
In *Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '08, pages 50–56, New York, NY, USA. ACM.

Miller, M. S. (2006).
Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control.
PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA.

Rees, J. A. (1995).
A security kernel based on the lambda-calculus.
PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Rountev, A., Kagan, S., and Gibas, M. (2004).
Evaluating the imprecision of static analysis.
In *Proceedings of the 5th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '04, pages 14–16, New York, NY, USA. ACM.

Shapiro, J. S., Doerrie, M. S., Northup, E., Sridhar, S., and Miller, M. (2004).
Towards a verified, general-purpose operating system kernel.
In Klein, G., editor, *Proceedings of the NICTA Formal Methods Workshop on Operating Systems Verification*, NICTA Technical Report 0401005T-1, Sydney, Australia. National ICT Australia.

Shapiro, J. S., Smith, J. M., and Farber, D. J. (1999).
EROS: A Fast Capability System.
SIGOPS Oper. Syst. Rev., 33(5):170–185.

References IV

Stevens, R., Gibler, C., Crussell, J., Erickson, J., and Chen, H. (2012).
Investigating user privacy in Android ad libraries.
In *Workshop on Mobile Security Technologies (MoST)*.

Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2009).
Software architecture: foundations, theory, and practice.
Wiley Publishing.

van der Walt, P. (2015).
Constraining application behaviour by generating languages.
In *8th European Lisp Symposium*, London, United Kingdom.

van der Walt, P., Consel, C., and Balland, E. (2015).
Frameworks compiled from declarations: a language-independent approach.
Technical Report hal-01236352, INRIA Bordeaux, France.
https://hal.inria.fr/view/index/docid/1236352, submitted to S:PE.

Watson, R. N. M., Anderson, J., Laurie, B., and Kennaway, K. (2010).
Capsicum: Practical capabilities for UNIX.
In *Proceedings of the USENIX Security Symposium*. USENIX.

Wei, X., Gomez, L., Neamtiu, I., and Faloutsos, M. (2012).
Permission evolution in the android ecosystem.
In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 31–40, New York, NY, USA. ACM.

References V

Xiao, X., Tillmann, N., Fähndrich, M., de Halleux, J., and Moskal, M. (2012).
User-aware privacy control via extended static information-flow analysis.
In Goedicke, M., Menzies, T., and Saeki, M., editors, *ASE*, pages 80–89. ACM.