Inferring Required Permissions

for Statically Composed Programs

Tero Hasu Anya Helene Bagge Magne Haveraaen
{tero,anya,magne}@ii.uib.no

Bergen Language Design Laboratory
University of Bergen


travel
NOKIA
Carl Zeiss
41MP sensor
BlackBerry
23:59
Contacts
Browser
Calendar
Text Messages
BlackBerry World
File Manager
Remember
Pictures
Settings
Search
Calculator
Clock
NOKIA
0:00
Weather forecast for Bergen
Rich Hickey and core.async - Po...
Duck Duck Go
productivity
experience

smartphones—a security risk for users

- privacy and usage cost concerns


14 // NYHETER

Frykter at bankene snoker på kundene

- natively third-party programmable
  - "app stores" have programs in large numbers
    - including malware and "grayware"

permission-based security models

- similar to VAX/VMS "privileges" introduced in late 70's
- popularized by smartphone OSes
- primarily: access control for sensitive APIs
- user approval of permissions $\rightarrow$ security and usability implications?

permissions—a concern for app developers

declaring permissions

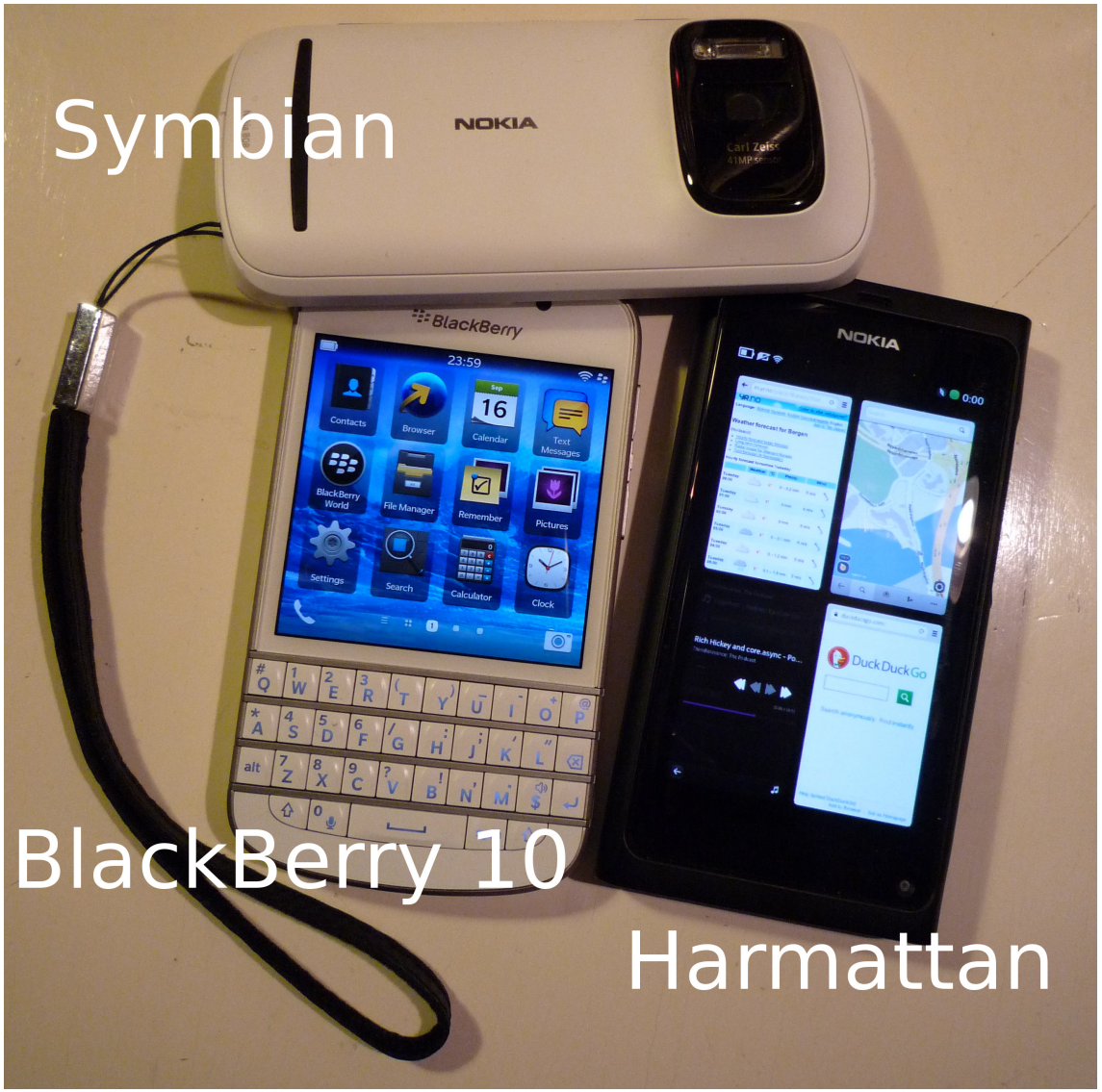
- too small a set $\rightsquigarrow$ runtime errors
- too large a set $\rightsquigarrow$ worried users
- optimal set $\rightsquigarrow$ maintenance hassle

Permissions

Permission	Identifier
BlackBerry Messenger	bbm_connect
Calendar	access_pimdomain_calendars
Camera	use_camera
Contacts	access_pimdomain_contacts
Device Identifying Information	read_device_identifying_information
Email and PIN Messages	access_pimdomain_messages
GPS Location	read_geolocation
Internet	access_internet
Location	access_location_services
Microphone	record_audio
Notebooks	access_pimdomain_notebooks
Post Notifications	post_notification
Push	_sys_use_consumer_push
Run When Backgrounded	run_when_backgrounded
Shared Files	access_shared
Text Messages	access_sms_mms

Select All Deselect All

hassle compounds in a cross-platform setting



- permission requirements vary between platform releases
  - often inadequately documented
- an app may come in multiple variants
  - sometimes because of permission restrictions
    - can differ per distribution channel

permission analysis tools availability

Android	Stowaway, Permission Check Tool (both 3rd party)
bada	API and Privilege Checker
BB10	none
Harmattan	`aegis-manifest` (automatically generates a declaration)
Symbian	Capability Scanner
Tizen	API and Privilege Checker
WP7	Store Test Kit (managed code only in WP7 apps)
WP8	none

vendor-supported permission inference

- infer required permissions from a program's platform API use
  - examine either binaries or source code
  - current tools for scanning *native* programs rely on heuristics
    - dynamic loading and invocation (when allowed) make accurate analysis difficult/impossible

baseline requirements for a cross-platform permission management solution

- the same solution must work for *all* platforms
- must make it *cheaper* to deal with app variants
  - since there can easily be many in a cross-platform setting
- we want to request an *optimal* permission set for each variant
- we do not want an app to *crash* due to runtime permission errors

many differences, many things beyond our control

- How do we get permissions?
  - just ask, system policy grants according to certificate or download source, user grants on install or at launch or per operation, manufacturer grants, ...
- Are there permissions we cannot ask for?
  - Does platform release affect what we can ask for?
- Is our running app guaranteed to have requested permissions?
- Can granted permissions be queried at runtime?
- Can we specify install time hardware requirements?
- Can we do install time adaptation (e.g., which binary)?
- Is app submission process arduous?
- In app store, can we specify which devices are supported?
- Will a build only be deployable to specific devices (IMEI codes)?
- etc.

how we might manage permissions

on a uniform, platform-independent basis with tools

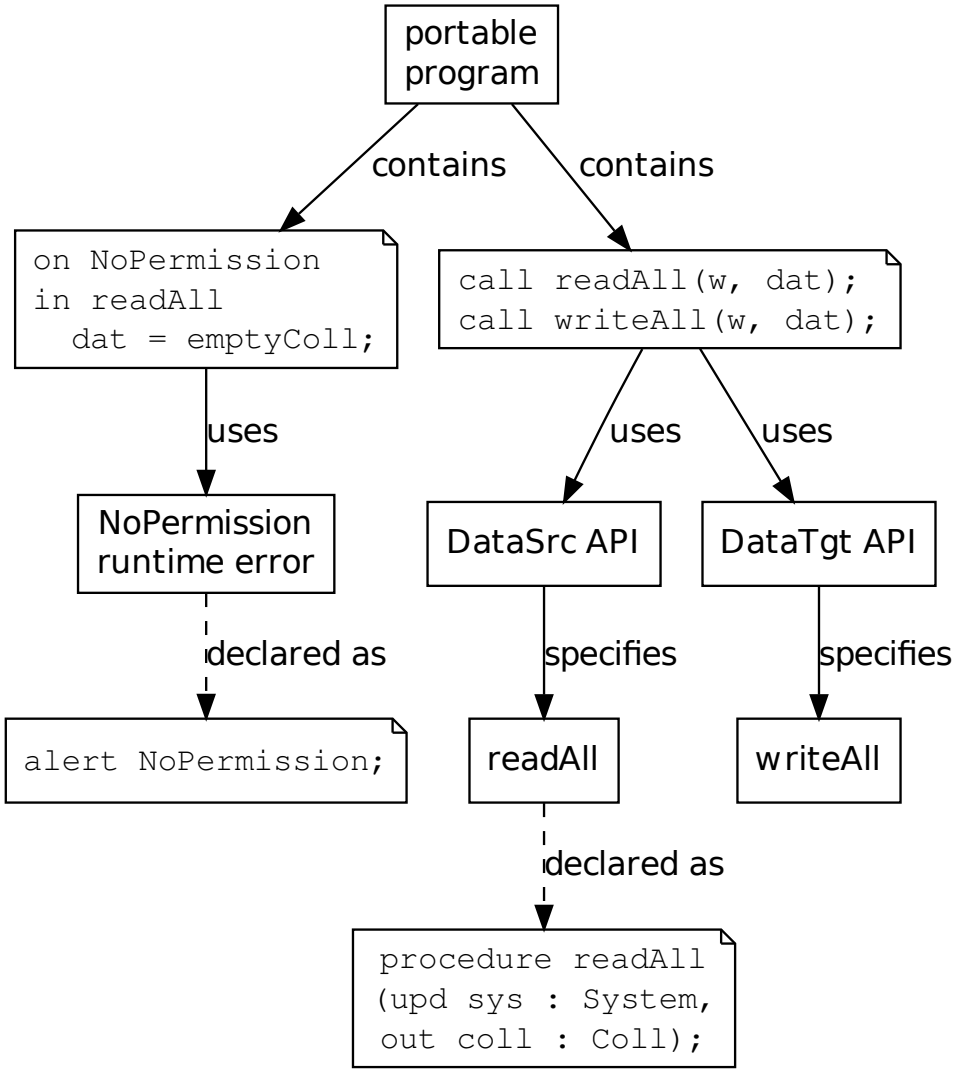
- app variant specific permission manifests
  - automate manifest file generation
- ungrantable permissions
  - automatically leave them out
- ungranted permissions
  - support portable implementation of error handling code
- *hardware requirements*
  - treat uniformly to permissions
    - both are *access capabilities*

how we might manage permissions

on an ad hoc, platform-specific basis by developers

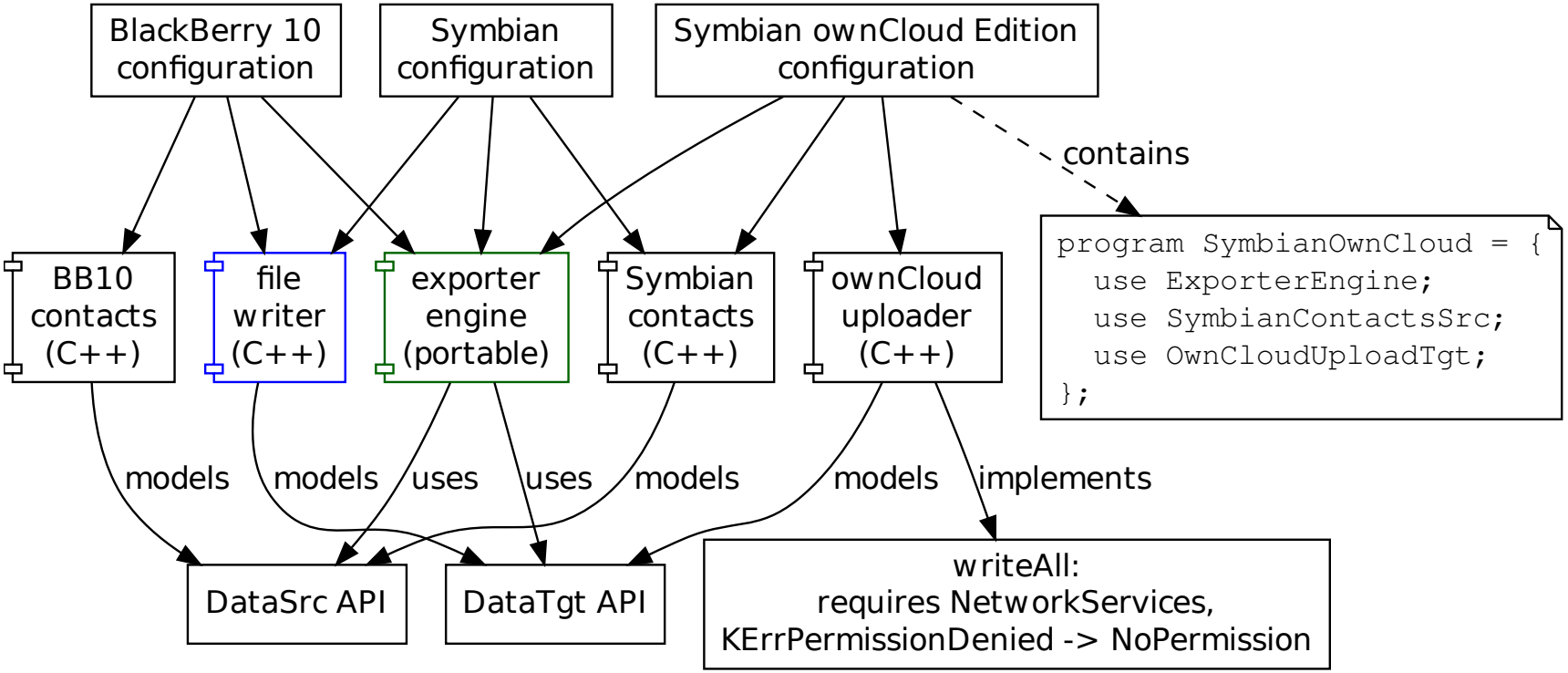
- compose software configurations that make sense
  - leave out functionality that is never accessible on a platform
    - perhaps find workarounds
- dynamic (or install time) adaptation to specific device models and user preferences (e.g., denied permissions)
  - particularly desirable if app submission process is arduous
  - useful if granted permissions can be queried at runtime
    - e.g., consider UI adaptation

portable permission-aware programming



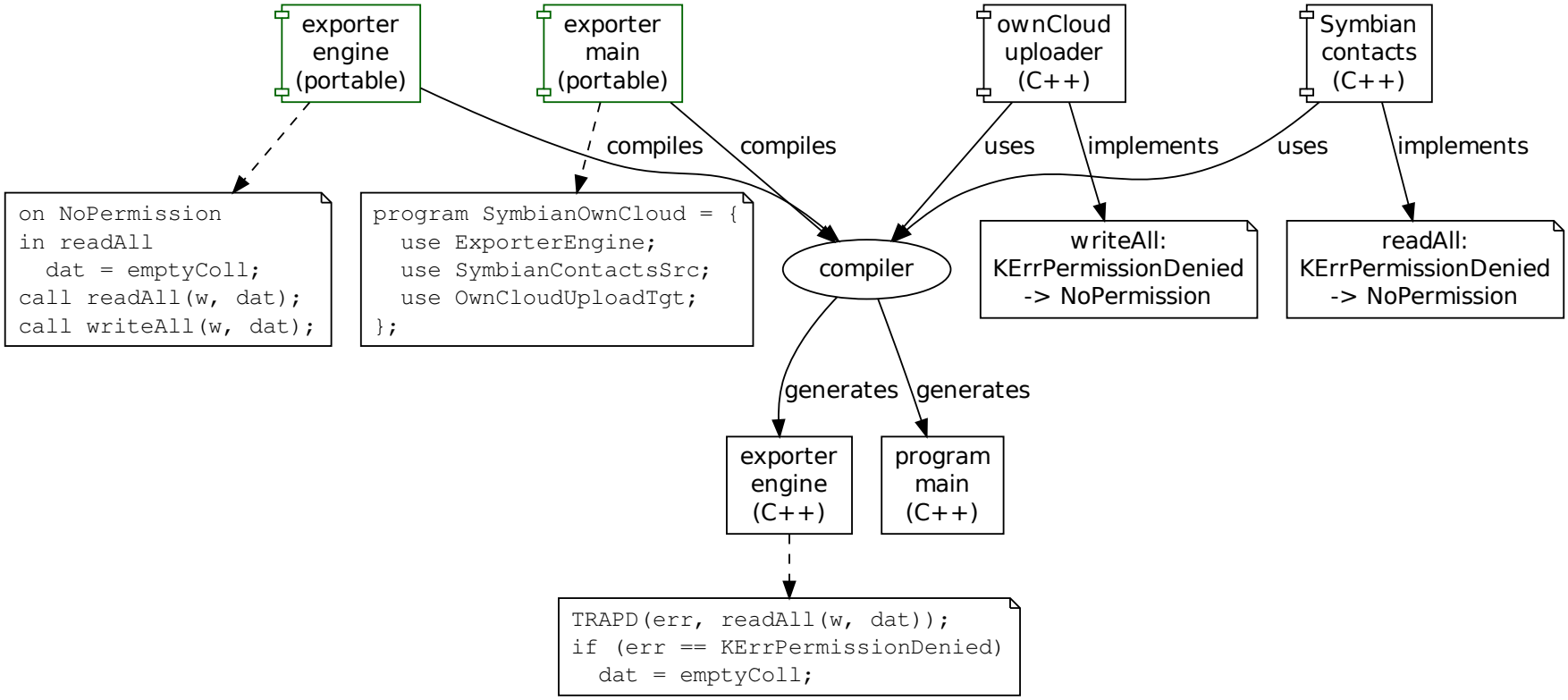
1. interfaces for abstracting over platform-specific implementations of components
2. language abstraction for portable runtime permission error handling

permission-aware product line



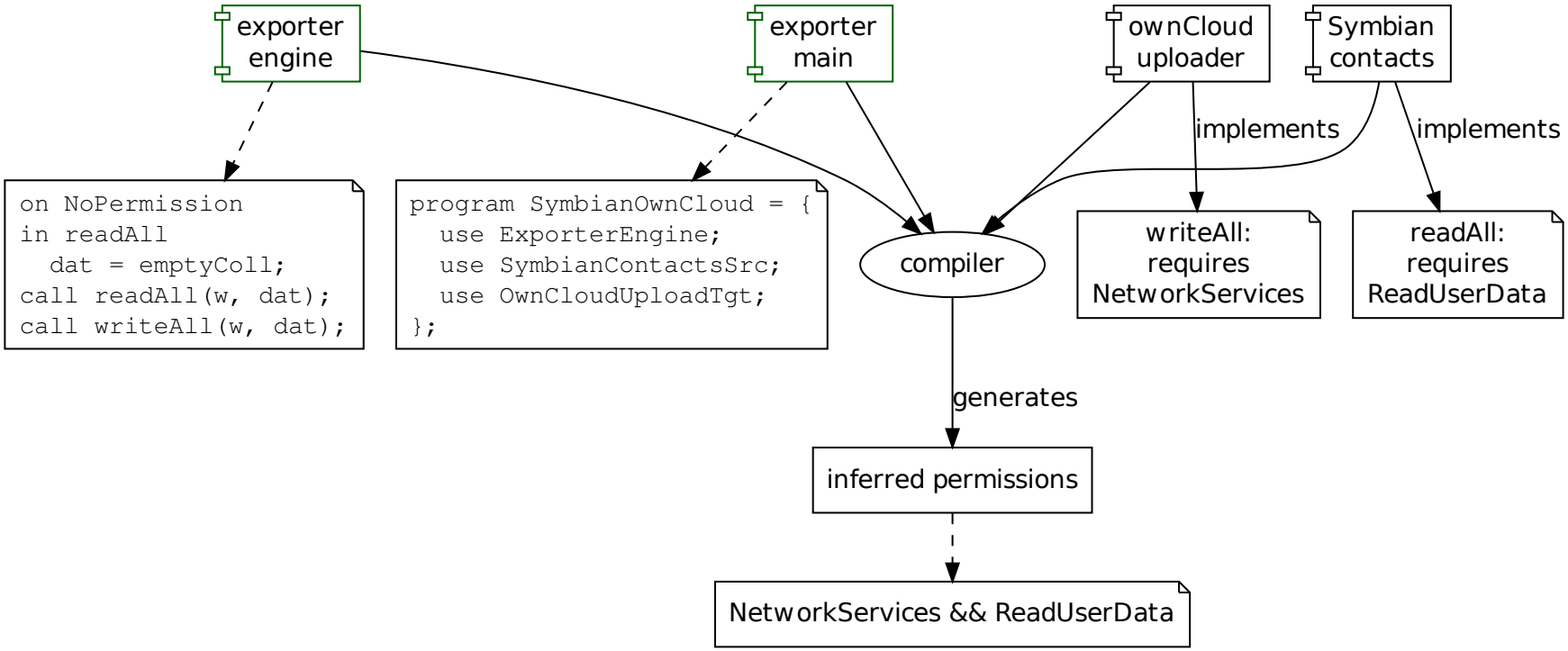
1. multiple implementations of components, reusable in different compositions

permission-aware compilation



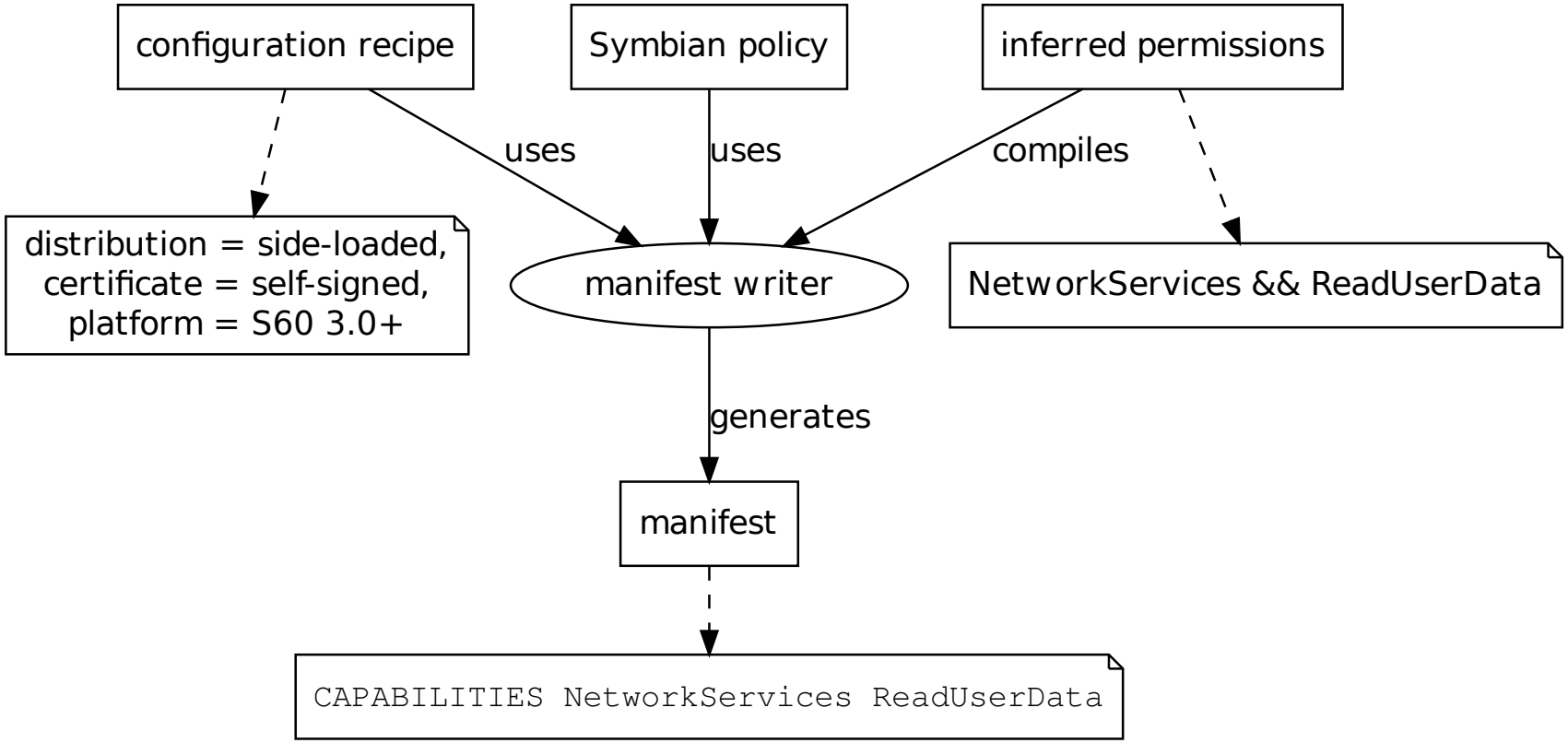
1. abstract-to-concrete permission error handling translation

permission inference



1. program analysis for determining reachable invocations of operations, and associated permissions

permission resolution



1. automated decision making on permission requests
   - configuration recipes may require additional information

vendor-specific permission inference

- infer required permissions from a program's platform API use

cross-platform permission inference

- infer required permissions from a program's platform-agnostic API use
  - implementations encapsulate platform API use
- and: declare permissions for each implementation of said APIs
- and: program against said APIs in a language you can analyze to determine API use

Can reuse the same API:

- for multiple platforms (if can implement)
- in multiple apps (if suitably general)

favorable language characteristics

interface-based abstraction

- to support organizing cross-platform codebases

static analysis friendliness

- to allow for accurate inference

adopting the approach

- adopt a favorable language, preferably
  - coding conventions or explicit information about programmer assumptions may help otherwise

in-source permission annotations

- as an extra-language feature (probably within comments)
- using any language-provided annotation support
- by extending the language

our proof of concept: based on Magnolia

- general-purpose research programming language Magnolia
  - `http://magnolia-lang.org/` 
- its implementation provides the required language infrastructure
- permission management is just one application for Magnolia
  - perhaps: address error handling in general (not just permission errors)
    - separate idea of *partiality* from concrete details of error reporting—Bagge: Separating exceptional concerns (2012)
    - abstract over different mechanisms—Hasu: Concrete error handling mechanisms should be configurable (2012)

Magnolia's interface-based abstraction

- a Magnolia interface is declared as a `concept`
  - each `concept` may have multiple `implementation`s
  - one `implementation` may satisfy multiple `concept`s

Magnolia's static analysis friendliness

- Magnolia avoids "dynamism"
  - no pointers, carefully controlled aliasing
  - no runtime passing of code (e.g., no higher-order functions)
  - abstract data types, not objects
    - concrete type and operations known at compile time
  - makes up for restrictions with extensive support for static "wiring" of components
- Magnolia promotes use of semantically rich concepts
  - a `concept` may specify (some) semantics as `axiom`s
  - an operation may specify use limitations as `guard`s

what & how to declare (static requirements)

- platform-specific required permission information (per operation, per `implementation`)
  - as a predicate expression—commonly need `&&`, sometimes `||`
  - for the Magnolia compiler to statically infer permission requirements for a `program`
  - e.g.,
    ```
    alert RequiresPermission unless pre SNS_SERVICE()
    ```

what & how to declare (dynamic behavior)

- platform-agnostic, abstract permission error names
  - to allow for error-handling in portable code
  - e.g., `alert NoPermissionSocial <: NoPermissionCloud;`
- mappings between platform-specific, concrete errors and error names (per operation, per `implementation`)
  - for the Magnolia compiler to implement the mapping
  - e.g., `alert NoPermissionSocial if post value == E_PRIVILEGE_DENIED`

domain engineering an exporter: data extraction and outputting

```
concept DataSrc = {
  use World;
  use DataCollection;

  procedure readAll(upd sys : System, out coll : Coll);
};

concept DataTgt = {
  use World;
  use DataCollection;

  procedure writeAll(upd sys : System, obs coll : Coll);
};
```

runtime permission errors

```
implementation Permissions = {
  alert NoPermission;
};
```

platform-specific permissions

```
implementation HarmattanPermissions = {
  use Permissions;
  predicate TrackerReadAccess() = Permission; // Harmattan
  predicate TrackerWriteAccess() = Permission; // Harmattan
  predicate GrpMetadataUsers() = Permission; // Harmattan
  // ...
};

implementation SymbianPermissions = {
  use Permissions;
  predicate ReadUserData() = Permission; // Symbian
  // ...
};
```

Pardon the verbose syntax!

Symbian-native contacts reader implementation

```
implementation SymbianNativeContactsSrc =
  external C++ datasrc.SymbianContacts {
    require type System;
    require type Coll;
    require SymbianPermissions;
    procedure readAll(upd sys : System, out coll : Coll)
      alert RequiresPermission unless pre ReadUserData()
      alert NoPermission if leaving KErrPermissionDenied
      /* more alerts ... */;
  };

satisfaction SymbianNativeContactsIsDataSrc = {
  use DataCollection; use World; use SymbianPermissions;
} with SymbianNativeContactsSrc
  models DataSrc;
```

same for Harmattan

```
implementation HarmattanQtContactsSrc =
  external C++ datasrc.HarmattanContacts {
    require type System;
    require type Coll;
    require HarmattanPermissions;
    procedure readAll(upd sys : System, out coll : Coll)
      alert RequiresPermission unless pre
        TrackerReadAccess() && TrackerWriteAccess() &&
        GrpMetadataUsers()
      alert NoPermission unless pre haveQtContactsPerms()
      /* more alerts ... */;
  };

satisfaction HarmattanQtContactsIsDataSrc = {
  use DataCollection; use World; use HarmattanPermissions;
} with HarmattanQtContactsSrc
  models DataSrc;
```

portable code, against platform-agnostic interfaces

```
implementation DefaultEngine = {
  require DataSrc;
  require DataTgt;

  procedure exportData() {
    var sys : System = initialState();
    var dat : Coll;
    on NoPermission in readAll
      dat = emptyColl();
    call readAll(sys, dat);
    call writeAll(sys, dat);
  }
};
```

one program configuration

```
program SymbianContactsSaver = {
  use DefaultEngine;
  use DefaultWorld;
  use DefaultDataCollection;
  use SymbianNativeContactsSrc;
  use CxxFileOut;
};
```

permission inference

- Magnolia compiler assembles a `program`—only relevant `implementations` are included from codebase
- permission inference accounts for all operations that appear in the `program`

build integration

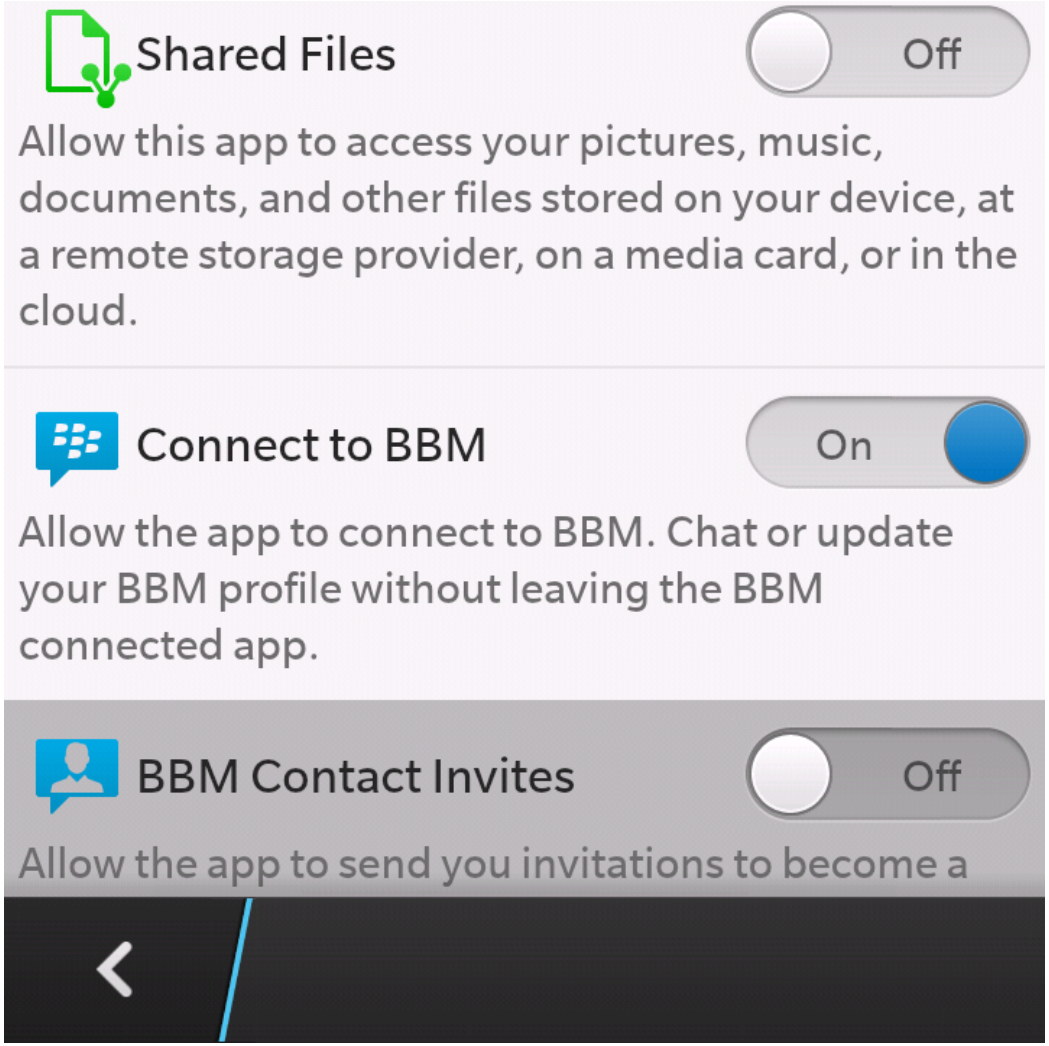
- compiler outputs a permission *expression*
- build tool writes a *set* of permissions into a manifest file
  - in a format expected by vendor toolchain

examples

- $(\texttt{ACCESS\_COARSE\_LOCATION} \lor \texttt{ACCESS\_FINE\_LOCATION}) \land \texttt{BLUETOOTH} \xrightarrow{decide} \{\texttt{ACCESS\_FINE\_LOCATION}, \texttt{BLUETOOTH}\}$, as need not ask for both coarse and fine location on Android
- $\texttt{AllFiles} \land \texttt{ReadUserData} \xrightarrow{decide} \{\texttt{ReadUserData}\}$, as getting `AllFiles` on Symbian requires manufacturer approval

gain: permission management solution

- tools support for avoiding runtime errors due to permission underdeclaration
  - assuming correct and complete annotations, and grantable & granted permissions (toggleable in BB10 and iOS)
- language support for handling runtime permission errors portably



cost: annotation effort

- may be able to amortize annotation cost over many projects and configurations
  - unlike when manually declared in a per-project-configuration manifest file
- a way to store and perhaps share domain knowledge
  - "I know this implementation of this API requires these permissions"

conclusion

- permissions are a concern to smartphone app devs
- we proposed a solution for permission management
  - requires no pre-existing permission tooling
  - can be applied to cross-platform codebases
  - no separately declaring permissions for each program
- we have tried out the solution
  - by integrating permission support into Magnolia

Anyxporter—permission management test app

`https://github.com/bldl/anyxporter`

- idea: different data sources, different permissions—can create variants
- cross-platform codebase, organized as concepts
- currently:
  - contact data export
  - for Harmattan and Symbian
  - one "Magnoliafied" build configuration, with permission inference

Anyxporter—contact data export

```
<?xml version="1.0" encoding="UTF-8"?>
<Contacts> ...
  <Contact> ...
    <ContactDetail>
      <DefinitionName>DisplayLabel</DefinitionName>
      <Label>Tero Hasu</Label>
    </ContactDetail>
    <ContactDetail>
      <DefinitionName>EmailAddress</DefinitionName>
      <EmailAddress>tero.hasu@ii.uib.no</EmailAddress>
    </ContactDetail>
    <ContactDetail>
      <DefinitionName>Guid</DefinitionName>
      <Guid>000000003e7be123-00e18ae873575ee5-41</Guid>
    </ContactDetail> ...
  </Contact> ...
</Contacts>
```