Formal Development of a Social Network Application
Progress Report

Sorren Harvey, Nestor Catano, Camilo Rueda, Timothy Wahls

1Universidad Javeriana-Cali 2Madeira University 3Dickinson College

SecSy 2010
∀pe. (pe ∈ dom(friendship)) ⇒
∀bf. (bf ∈ best_friends[{pe}]) ⇒
∀sf. (sf ∈ social_friends[{pe}]) ⇒
∀rc. (rc ∈ rawcontent) ⇒
∀ac. (ac ∈ OPS) ⇒
((pe = owner(rc) ∧ pe ↦→ rc ∈ content
 ∧ rc ↦→ ac ↦→ pe ∈ act ∧ bf ↦→ rc ∈ content
 ∧ sf ↦→ rc ∈ content ∧ rc ↦→ ac ↦→ sf ∈ act)
 ⇒
 rc ↦→ ac ∈ act⁻¹[[{bf}]])))))

∧

∀pe. (pe ∈ dom(friendship)) ⇒
(wallaccess[{pe}] ∩ social_friends[{pe}] ≠ ∅ ⇒
 best_friends[{pe}] ⊆ wallaccess[{pe}]))
Motivation: SN proliferation

Social Networks

- Have become extremely popular
  - E.g. Facebook, MySpace, LinkedIn, Hi5, Twitter, Sapo
- Each supporting millions of active users
- Used to
  - Publish media content
  - Share personal info
  - Make business contacts
Motivation: SN privacy?

Social-networks and Media in general have replaced personal communication as communication force, but...

Information in social-networks is security and privacy-sensitive
Privacy: SN user behavior

  - Analyzed the behaviour of 4,000 CMU students on a social-network catered to colleagues
  - Evaluated information students disclose and study how they use social-network site privacy settings

A minimal percentage of users change the highly permeable privacy preferences
SN Privacy

Existing Social Networks

- *Do not enforce privacy of media content*
- *They have conflicting goals*
  - E.g. Expanding the network vs. exposing users content
Developing a SN with parental control facilities, that is Secure, extensible and privacy-enforcing in such a way that those properties can be formally proved.
Outline
Outline

1. Motivation

2. Model architecture
   - Social Network Core
   - The B to JML translator

3. Extending with Plug-ins

4. Verifying Plug-ins in Yices

5. Conclusions
Architecture

- Specification of SN Core in B
- Code generator
- SN Core in Java
- B to JML Translator
- JML specification
- Plug-in Java code
- Plug-in verifier
- Proof obligations in Yices (certificate)
- YICES
Social network Core

- Write **privacy** and **security** social network policies as an initial predicate calculus-based **abstract specification**
- **Refine** the initial abstract specification and obtain a social network **core application** that adheres to stipulated policies
- **Privacy** is modeled as **access permissions** on content
The core in B

Model architecture

Social Network Core

The core in B

Abstract Model

Principal Content, Page Field

Mandatory contents

Friendship Relations

Permissions according to Friendship

User Wall

Suggest, Fiend Friends

Extend Functionality

Add Plug-in
Abstraction: state

Some defined types, i.e. sets: PERSON, RAWCONTENT, OPS
And some basic variables:

\[ \text{person} \subseteq \text{PERSON} \]
\[ \text{rawcontent} \subseteq \text{RAWCONTENT} \]
\[ \text{content} \in \text{person} \leftrightarrow \text{rawcontent} \]
\[ \text{visible} \subseteq \text{content} \]

\[ \text{act} \in (\text{rawcontent} \times \text{OPS}) \leftrightarrow \text{person} \]
Abstraction: operations

- Create content, make it visible, edit, comment, hide,...
- Transmit some content $rc$ from the owner $ow$ to some person $pe$

```
transmit_rc(rc, ow, pe) =
    pre rc ∈ rawcontent ∧ pe ∈ person ∧
        ow = owner(rc) ∧ ow ≠ pe ∧
        (pe, , rc) ∉ content ∧ pe ∉ act[{(rc, , view)}]
    then
        content := content ∪ {(pe, , rc)} ||
        act := act ∪ {((rc, , view), , pe)}
    end
```
Abstraction: invariants

**The owner of some data has all permissions for it:**

\[ \forall rc \in \text{rawcontent} \Rightarrow \forall op \in OPS \Rightarrow ((rc, op), \text{owner}(rc)) \in \text{act} \]

**If you can see some content, then you have “view” permission:**

\[ \forall (rc, pe). (rc \in \text{rawcontent} \land pe \in \text{person}) \Rightarrow ((pe, rc) \in \text{visible}) \Rightarrow ((rc, \text{view}, pe) \in \text{act}) \]
Modeling friendship relations

Content is partitioned by three relations

\[ \text{friendship} \subseteq \text{friend} \leftrightarrow \text{friend} \]
\[ \text{best\_friends} \subseteq \text{friendship} \]
\[ \text{social\_friends} \subseteq \text{friendship} \]
\[ \text{acquaintances} \subseteq \text{friendship} \]
\[ \text{best\_friends} \cap \text{social\_friends} = \varnothing \]

etc.

Your best friends are not your social friends
Enforcing privacy

$p_{e}$: the owner of some content $r_{c}$,

$b_{f}$: a best friend of $p_{e}$,

$s_{f}$: a social friend of $p_{e}$,

$a_{c}$: a permission $s_{f}$ has over $r_{c}$

\[
((r_{c}, a_{c}), s_{f}) \in act \Rightarrow ((r_{c}, a_{c}), b_{f}) \in act
\]

*Your best friends have always at least the permissions you have given to your social friends*
The whole B specification

- 6 machines
- About 450 Proof Obligations (all discharged)
- Some interactive proofs were rather elaborate
Drawback

How to express the property:

\[
\text{if you have now permission op for data d then}
\]
\[
(1) \text{the owner gave } d \text{ to you with the op permission previously}
\]
\[
(2) \text{the owner did not take the op permission away later}
\]

Solution (may be):
implment the core in the ProB model-checker
express the property in ProB’s temporal logic
the notation translated into may permit the use of tools that do not support the notation translated from.

the notation translated from may not directly support implementation in the desired programming language.

the user may simply be more proficient and comfortable with the notation translated into.

One would like the same specification languages for Plug-in and Core
B to JML translator: JML

**JML is a way to annotate java programs with specifications**

- Specifications are of the form:
  - precondition: \texttt{requires} P
  - poscondition: \texttt{ensures} Q

- there are also \texttt{class invariants} (with so-called \texttt{ghost} variables)

**JML tools verify an annotated java program**
## B to JML

<table>
<thead>
<tr>
<th>B Model</th>
<th>JML Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine</td>
<td>Class</td>
</tr>
<tr>
<td>Machine Refinement</td>
<td>Sub-class</td>
</tr>
<tr>
<td>PRE</td>
<td>requires</td>
</tr>
<tr>
<td>Operation Body</td>
<td>ensures + \old invariant</td>
</tr>
<tr>
<td>INVARIANT</td>
<td>JML Method specification</td>
</tr>
<tr>
<td>Operation ANY x</td>
<td>\forall x</td>
</tr>
</tbody>
</table>
B to JML rules

\[ jml \ (PRE \ P \ THEN \ S \ END) = \]

public normal\_behavior; requires \( jml(P) \):
assignable \( mod(S) \); ensures \( jml(S) \);
also public exceptional\_behavior; requires \( !jml(P) \);
assignable \( \text{\textbackslash nothing} \);
signals \( \text{(java.lang.Exception e)} \);

What \( S \), above, could be:

\[ jml(\ x := E \ || \ y := F ) = \]
\[ x == \ \text{old}(jml(E)) \ \&\& \ y == \ \text{old}(jml(F)) \]
B to JML rules (2)

\[
jml \ (\text{ANY } x \ \text{WHERE } P \ \text{THEN } S \ \text{END}) = \\
\exists \ type\_of(x) \ x; \ \\ol (jml(P)) \&\& jml(S)
\]

Example:

\[
jml \ (\text{ANY } x \ \text{WHERE } x \in PERSON \land x \notin person \ \text{THEN } person := person \cup \{p\} \ \text{END}) = \\
\exists PERSON \ x; \ !\ol (person.\text{has}(x)) \ \&\& person == \ \ol (person.\text{union}(\text{singleton}(x)))
\]
Specifications in JML: translating a B operation

```java
public abstract JMLEqualsSequence< Object >
    transmit_rc(Integer rc, Integer ow, Integer pe);
/*@ requires RAWCONTENT.has(rc) && rawcontent.has(rc) &&
PERSON.has(pe) && person.has(pe)
&& content.has(pe, rc) && !visible.has(pe, rc)
&& act.has(new JMLEqualsEqualsPair< Integer, OPS >
    (rc, OPS.view), pe);
assignable visible;
ensures visible.equals(
    old(visible.union(JMLEqualsToEqualsRelation.singleton(pe, rc)))))
&& \result.int.size() == 0;
```
Outline

1. Motivation

2. Model architecture
   - Social Network Core
   - The B to JML translator

3. Extending with Plug-ins

4. Verifying Plug-ins in Yices

5. Conclusions
Extending the Core

- Plug-ins implementing functionalities
- Social Network Plug-in Validator
  - Proof Carrying Code (PCC), Necula, G.-C.
  - Plug-in consists of *java* code implementing the functionality + proof of adherence to the B model of social-networks
PCC architecture
Extension adherence to policies

- **Non-bypassable**: the security functions cannot be circumvented
- **Tamper-proof**: subversive code cannot alter the function of the security functions by exhausting resources or overrunning buffers.
VC generator

- As VC we use Dijkstra’s *weakest precondition* calculus

  \[ WP(S,Q) : \text{weakest condition that must hold before } S \]
  \[ \text{to ensure that } Q \text{ holds after executing } S \]

- We devise rules to give WP semantics to a *subset* of *java*

- and translate WP rules to Yices formulae

  **Given a plug-in** \( S \):
  \[
  \begin{align*}
  (1) & \text{ compute } P = WP(S, \text{true}) \\
  (2) & \text{ use Yices to verify } P
  \end{align*}
  \]
Plugin verification: architecture

- Code producer
- Source program (Java)
- Safety policy (B -> JML)
- Verification conditions (WP -> Yices)
- VCGen (WP)
- Prover Yices
- Code consumer
### WP Java Semantics

<table>
<thead>
<tr>
<th>WP(S,⟨Q, R, L⟩)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
</tr>
<tr>
<td>Q</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>L</td>
</tr>
</tbody>
</table>

- Relation $R$ associates a WP with each caught Java exception
- $L$ keeps track of assignable variables
Extending with Plug-ins

WP java semantics: the annotation of a method

/*@ public normal_behavior;
  @requires P1;
  @assignable L1;
  @ensures Q;
  @also
  @public exceptional_behavior;
  @requires P2;
  @assignable L2;
  @signals R;
  @/

public T m(x) throws E { block }
WP java semantics: some rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WP(t, ⟨Q, R, L⟩) = Q[result\t]</td>
<td>terms</td>
</tr>
<tr>
<td>WP(x = e, ⟨Q, R, L⟩) = WP(e, ⟨Q[x\result], R, L⟩)</td>
<td>assignment</td>
</tr>
<tr>
<td>WP(S₁; S₂, ⟨Q, R, L⟩) = WP(S₁, WP(S₂, ⟨Q, R, L⟩), R, L)</td>
<td>sequence</td>
</tr>
<tr>
<td>WP(throw e, ⟨Q, R, L⟩) = WP(e, ⟨R(e.type), R, L⟩)</td>
<td>raise exc</td>
</tr>
<tr>
<td>WP(try {S₁} catch e {S₂}, ⟨Q, R, L⟩) =</td>
<td></td>
</tr>
<tr>
<td>WP(S₁, ⟨Q, R.add(e, WP(S₂, ⟨Q, R, L⟩), L)⟩)</td>
<td></td>
</tr>
<tr>
<td>add (exception, predicate) pair</td>
<td></td>
</tr>
</tbody>
</table>
WP java semantics: method call rule

\[ WP(m(y), \langle Q, R, L \rangle) = \]

\[ m.P_1 \land m.L_1 \subseteq L \land \forall l \in L_1 \quad \text{method precondition must be true} \]
\[ \lor \]
\[ m.P_2 \land m.L_2 \subseteq L \land \forall l \in L_2 \quad \text{method postcondition must imply Q} \]
\[ m.q[x\backslash y] \Rightarrow Q \]
\[ m.R(m.E)[x\backslash y] \Rightarrow Q \]
VC generator: plug-in example

/*@ PRECONDITION true; @ POSTCONDITION Q; @*/

public void install_plugin(SocialNetwork sn) {
    integer ow1 = sn.create_content(); // I1
    integer rc1 = sn.create_rc(); // I2
    sn.upload_rc(ow1,rc1) // I3
    integer pe1 = sn.create_content(); // I4
    sn.transmit_rc(rc1,ow1,pe1); } // I5

VC = (true ⇒ WP(I1; I2; I3; I4, Q))
≡
WP(I1, WP(I2, WP(I3, WP(I4, WP(I5, Q)))))
Outline

1. Motivation
2. Model architecture
   - Social Network Core
   - The B to JML translator
3. Extending with Plug-ins
4. Verifying Plug-ins in Yices
5. Conclusions
Yices SAT solver
Sets (and relations) are represented as bit-vectors.

B operations on sets are translated to bit-vector ops.

**Challenge:** do the above efficiently (i.e. avoid recursion!)

<table>
<thead>
<tr>
<th>$r_1 \subseteq r_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(define \ jmlrel-is-subset::(\rightarrow \ jmlrel \ jmlrel \ bool) \ (\lambda (r1::jmlrel \ r2::jmlrel) \ (= \ r1 (bv-and \ r1 \ r2)))$ )</td>
</tr>
</tbody>
</table>

**How to implement** $S \triangleleft r$ ??
Verifying Plug-ins in Yices

VC generator in Yices: example

\[ WP(I_1, WP(I_2, WP(I_3, WP(I_4, WP(I_5, Q)))))) = \]
\[
(\text{define } Q5 \]
\[
(\text{let } \) (r5::bool (translation-pre (transmit_rc sn rc1 ow1 pe1)))
\[
(t5::bool (translation-post (transmit_rc sn rc1 ow1 pe1)))) )
\[
(\text{and } r5 (\Rightarrow t5 Q ) \))
\]

WP(I_1, WP(I_2, WP(I_3, WP(I_4, Q5))))
VC proof in Yices

(assert
  (and
    (= person4 person-prestate5)
    (= rawcontent4 rawcontent-prestate5)
    (= content4 content-prestate5)
    ...
    Q5 Q4 Q3 Q2 Q1
  ))
(check )
Outline

1. Motivation
2. Model architecture
   - Social Network Core
   - The B to JML translator
3. Extending with Plug-ins
4. Verifying Plug-ins in Yices
5. Conclusions
Help! we need somebody,
not just anybody

We need students!!!