SafetyNet: Designing an Object-Oriented Language for Network Programming

Alan Jeffrey

CTI, DePaul University, Chicago

In collaboration with:
Ian Wakeman, Sussex
Tim Bateman, Sussex
Rory Graves, Sussex
Sam Type, Sussex
David Gawlinski, DePaul
Ed Thoele, DePaul

Partially funded by Hewlett-Packard and British Telecom
Overview

Semantics vs Networks.
Active networks.
The SafetyNet project.
Type safety and subject reduction.
SafetyNet models.
Further work.
Life as a semantics gun-for-hire.
Semantics vs Networks

Concerns of semantics:

- Mathematical models of programming languages
- Proofs proofs proofs...
- Category theory, type theory, formal logic, topology...

Concerns of network engineers:

- Reliability
- Scalability
- Integrity
- Mechanisms for rolling out new services...

What do these have to do with each other?
The SafetyNet project is about one area of intersection: active networks.
Active Networks

In a traditional network topology, ‘smart’ hosts are linked by ‘dumb’ routers:

In reality, a router is often pretty smart. An AN allows users to make use of the computing power in the routers:
Example Active Networks

Implement multicast on top of broadcast:

Active multicast networks can take advantage of processing power at the relays.
For example, an active relay can perform application-specific compression for low-bandwidth links.
Example Active Networks

‘On the fly’ firewalls:

![Diagram of a network showing active nodes]

Active nodes
- \( T \) Trusted node
- \( F \) Firewall
- \( U \) Untrusted router

Makes firewall deployment lightweight.
Example Active Networks

Intelligent agents.

Scalable VR.

Application-specific protocols.

Application-specific proxies.

...
Problems with ANs

Switch managers are right to be worried about ANs. To be usable, ANs must not:

Add a notable performance overhead.
Compromise network integrity.

For example, if arbitrary C code were executed:

Security violations from dereferencing arbitrary memory.
Denial of Service attacks from flooding the network with packets or processes.
Why not Java?

Java solves some problems, e.g. banning pointer arithmetic, but at a run-time cost.

For example, each array assignment requires a bounds test and a run-time type test.

Can we achieve security and safety without requiring costly run-time checks?
SafetyNet

The goals of the SafetyNet project are to apply semantic techniques to networking problems.
To develop safety and security policies for ANs.
To design a typed language for programming ANs.
To show that any type-safe program will satisfy the safety and security policies.
To provide prototype implementations and simulation tools.
Type safety and subject reduction

There are two components to a proof that type-safe programs satisfy safety and security policies:

- **Type safety**: if a program is well-typed, then it satisfies the policies now.
- **Subject reduction**: if a program is well-typed, then it will be well-typed in all possible futures.

Together, these properties ensure that:

- If a program is well-typed then it will satisfy the policies in all possible futures.

Hooray!
Example: subject reduction

We have types:

\[ \vdash 1 : int \]
\[ \vdash 2 : int \]
\[ x : int; y : int; \vdash x + y : int \]

and a possible execution:

\[ (1 + 1) \mapsto 2 \]

so this system satisfies subject reduction.

Whoop de doo.
Example: subject reduction failing

In Java:

```java
void put (Object[] a, Object x) {
    a[0] = x;
}
Hashtable[] a = new Hashtable[1];
String x = "Fred";
put (a, x);
```

this type-checks, but evaluates to:

```java
Hashtable[] a = new Hashtable[1];
String x = "Fred";
a[0] = x;
```

which does not type-check!

So Java does not satisfy subject reduction (it uses run-time typing to get round this).
Object-oriented model

Class-based polymorphic OO language.
Looks ‘enough like Java’ to fool programmers.
Translates down to a core language with a formal model.
Network model

A network is a graph of locations.
At each location there is a heap of objects and a pool of executing threads.
Threads can die, can spawn new threads, either locally or remotely.
Remote thread spawning is best-effort with no acknowledgment (can be implemented with TCP/UDP).
RPC is derived, not primitive!
Threads cannot discover the network topology or roll their own routing.
Formal model

The language specification consists of:

- A grammar for the user syntax
- A grammar for a core language
- A translation from the user syntax to the core language
- Formally defined type rules for the core language.
- Formally defined execution rules for the core language.

For ‘interesting’ subsets of the language we have subject reduction results.
Example formal model: trust

For example, we have a subject reduction result for a subset of the language with:

• Networks of nodes, some nodes are ‘bad guys’ trying to subvert the type system.
• Good nodes are well-typed.
• Bad nodes can do whatever they like.
• A trust relation between nodes, where good nodes only trust other good nodes.
• When code migrates from a trusted node, it is run without type-checking.
• When code migrates from an untrusted node, it is type-checked.

This is formalized as a small language, and we prove subject reduction, thus showing that good nodes stay well-typed.
Related work

Language design work is based on:

Milner, Parrow and Walker’s pi-calculus.
Boudol’s blue-calculus.
Fournet, Gonthier, Lévy, Maranget and Rémy’s join calculus.
Moggi’s monadic meta-language.
Girard / Reynolds’ 2nd order lambda-calculus.
Hennessy and Riely’s / Sewell’s distributed pi-calculus.
Cardelli and Gordon’s ambients.
Abadi and Cardelli’s object calculus
Gordon’s concurrent object calculus
Odersky, Wadler et al’s Generic Java

Closest AN work is Gunter et al’s PLAN.
Future work

There is still a lot of work to be done:

‘Scale up’ to a real programming language. **Done**

Develop and formalize ‘real world’ safety and security policies. **Done some.**

Add linear types (for time-to-live counts or charging mechanisms). **Still to do**

Include security and digital signatures. **Still to do**

Include a ‘web of trust’ model based on Hennessy and Riely. **Done some**

Try out on real examples! **Done some.**
Life as a semantics gun-for-hire

Interdisciplinary work is hard!
Both sides need to learn a new language and a new way of thinking.
Both sides need to respect each other.
Communication is very important.
‘Network researchers are from Saturn, semanticists are from Neptune’. 
What is semantics good for?

Semanticists can:

• do proofs
• think rigorously
• come up with counterexamples
Interdisciplinary work

Pros:

- Learn a new problem domain
- Make new friends and influence people
- Fun.

Cons:

- Takes a lot of work
- Initial requirements capture generates zero papers
- Difficult to get funded
Conclusions

SafetyNet is an ongoing project.

We have a prototype compiler, and a draft language spec (which disagree with each other).

Semantics can be useful.

Interdisciplinary work is fun.

http://klee.cs.depaul.edu/an/