Mobility and Security Group

David Aspinall, Robert Atkey, Brian Campbell, Stephen Gilmore, Patrick Maier, Kenneth MacKenzie,
Alberto Momigliano, Randy Pollack, Don Sannella, Jaroslav Ševčík, Ian Stark
Laboratory for Foundations of Computer Science, School of Informatics, University of Edinburgh
http://www.lfcs.inf.ed.ac.uk/research/mobility+security/

Mobile Code Security

The Mobility and Security Group is engaged in building secure foundations for the next generation of mobile applications, using proof-carrying code to give mathematical guarantees of program safety.

The internet is now a major channel for software distribution: interactive web pages, automatic software updates; even complete applications and operating systems. All this is mobile code: fantastically convenient, but such a dynamic environment hugely magnifies the challenge of ensuring that software runs safely, securely and reliably.

Mobile Devices

Modern mobile phones come equipped with Java Virtual Machines for running downloaded games and other applications.

Small mobile devices have limited resources, such as processor speed and memory space, as well as expensive resources such as network usage.

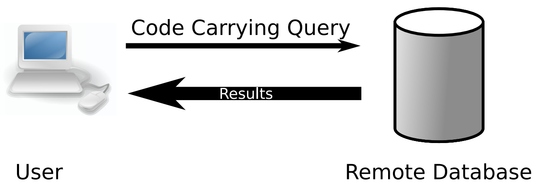
Knowing beforehand that a downloaded game will have enough resources to run and that it will not cost the user unexpected amounts of money is extremely useful.

Databases and the Grid

Grid computing aims to commoditise super-computing-level processing power and large shared databases.

The Grid is built on code being executed remotely by untrusting hosts. Of particular importance is safe use of computing resources such as CPU time and memory space. A rogue piece of code must not be allowed to monopolise a shared resource.

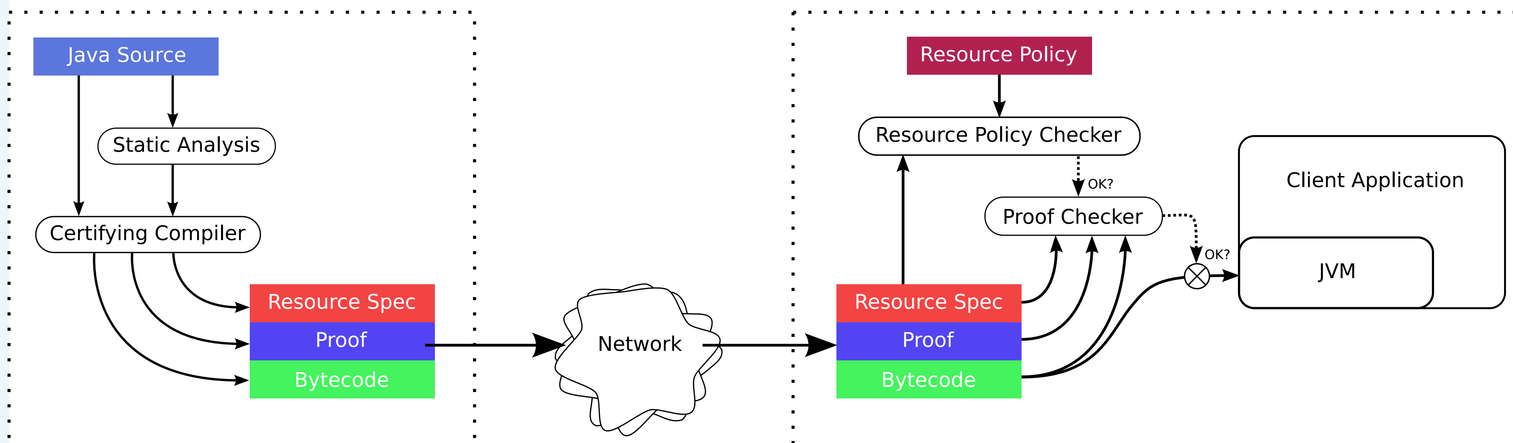
We are focusing on mobile code security for databases. We intend that users submit queries along with custom analysis code that it is more efficient to run near the data. Such code is run directly inside the server and so database providers must ensure that a single user's query does not dominate the server.



Trust, but Verify: Proof-Carrying Code

The current trust standard for mobile code is a cryptographic certificate, a "signature" that identifies the code provider. This works well but has limitations: it requires a supporting infrastructure to distribute cryptographic keys; and although it tells us the origin of some mobile code, it can say nothing about the code itself.

We aim to add an entirely new level of code assurance by using Proof-Carrying Code (Necula POPL'97): programs carry with them a mathematical proof of their safety. These are inherently tamper-proof and unforgeable — they do not refer to an external authority but describe the code itself.



Code Producer

The code producer is responsible for packaging their code with its resource specification and a proof that the code satisfies the specification. We generate these from resource aware type systems on high-level code.

Resource Aware Type Systems

We have developed Camelot, a functional language for resource constrained programming. The type system ensures that heap space is used correctly. Joint work with LMU Munich enables the inference of resource bounds for some programs. The basic idea is shown to the right. We are also working on stack space inference.

More recently, we have been working in Java and also on non-memory resources such as the amount of text messages sent on mobile phones. Our mechanism for this is the use of dynamic `ResourceManager` objects. We are developing a type system for checking these statically.

```
Cons(-,-)@- : 'a * 'a list * <> -> 'a list

let rev l acc = match l with
  | Nil -> acc
  | Cons(h,t)@d -> rev t (Cons (h,acc)@d)
```

The list constructor `Cons` takes a head and tail, plus a free memory cell: a "diamond". Pattern matching frees a diamond, so `rev` operates in constant space.

ResourceManager API (simplified)

```
 - enable (MultiSet r)
 - use (MultiSet r)
```

A client application must call `enable` to allocate resources before the system code calls `use`. The type system ensures that calls to `use` never fail.

Code Consumer

The code consumer receives the packaged bytecode from the code producer. It checks that the resource specification matches its resource policy and that the purported proof of the resource specification with respect to the code is correct. The code is only executed if these checks succeed.

Resource Policies

Resource Policies specify the resources the consumer will allow for mobile code. We have developed a method to check resource policies against specifications provided by the producer that are parametric with respect to platform-dependent system calls.

Bytecode Logics

To state the proofs that are checked on the client side, we are developing a Bytecode Logic for Java bytecode in conjunction with our partners in the Mobius project.

Previously, we developed a logic for a language called Grail -- a functional representation of Java bytecode. In order to state and prove resource properties we introduced the generic notion of Resource Algebra which can be instantiated with resources such as heap space or instruction counts.

Formalisation

To ensure soundness we have formalised the Grail logic in the interactive theorem prover Isabelle. We are formalising the new bytecode logic in Coq, and we are using it to develop a formal executable specification of the Java Virtual Machine.

The work presented here is funded by the European Community as part of the Mobius project (FP6-015905) and by a Engineering and Physical Sciences Research Council grant ReQueST (Resource Quantification for e-Science Technologies) (EP/C537068).

Ian Stark is funded by an EPSRC Advanced Fellowship grant "Mathematical Models for Concurrent and Mobile Computation" (GR/R76950/1).