Formal Methods in the Robin project: Specification and verification of the Nova microhypervisor

Hendrik Tews
SoS group, Radboud University Nijmegen
http://www.cs.ru.nl/∼tews

Supported by the European Union through PASR grant 104600.

Goals

- verify some properties about a real OS-kernel
- develop formal methods for C/C++
- develop formal methods for low-level programs

History

- 1998 Fiasco case study [1]
- 2002-2006 VFiasco project at Dresden University [2]
- 2006-2008 WP 4 of the Robin project

References

[1] H. Tews. A case study in coalgebraic specification: Memory management in the FIASCO microkernel. Technical Report TPG2/1/2000, SFB 358, April 2000. Available at URL wwwtcs.inf.tu-dresden.de/~tews/vfiasco/.

[2] M. Hohmuth and H. Tews. The VFiasco approach for a verified operating system. In *Proceedings of the 2nd ECOOP Workshop on Programm Languages and Operating Systems*, Glasgow, 2005.

Why micro kernels?

- Size!
  - Minix 3: 4.000 LOC
  - Pistachio: 10.000 LOC
  - Fiasco: 15.000 LOC
  - Linux 2.6: 6.000.000 LOC
- (almost) no device drivers, therefore less dependence on strange hardware behaviour
- very convincing application scenario: micro-kernels or hypervisors can be used to build a trustworthy and secure platform
- personal contact to developers

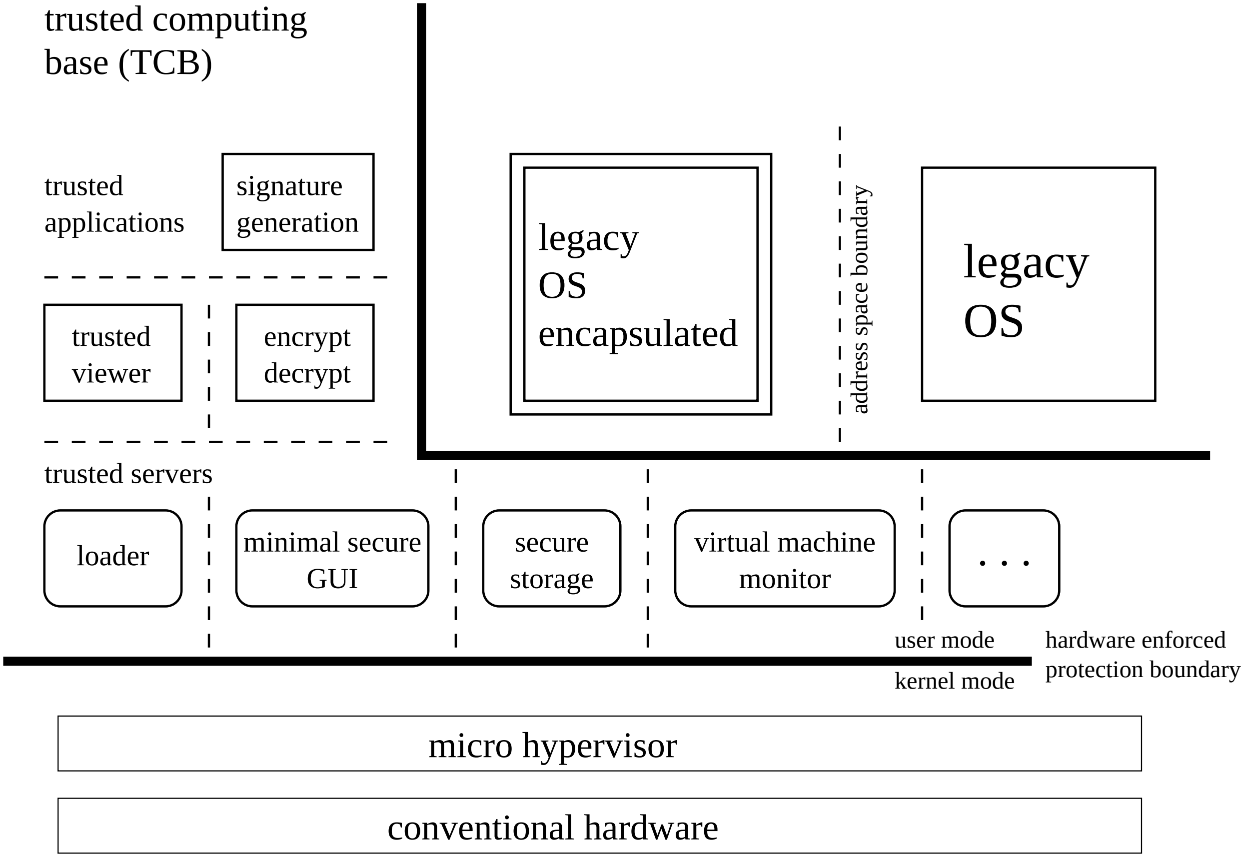
Conflict between Security and Usability

- mobile phone/PDA
  - mobile webbrowser
  - store personal data, used for monetary transactions
- PC at home
  - Internetbanking, private correspondence
  - Internet access console

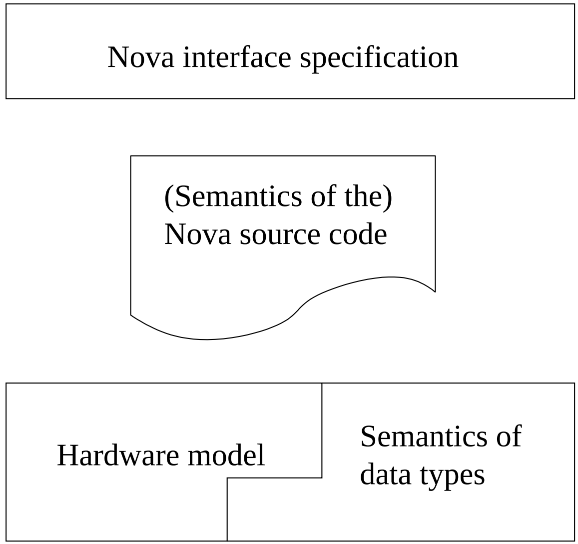
Robin project

- supported in the Preparatory Action in the field of Security Research (PASR)
- Goal: develop a solution to the Security/Usability conflict
- Nizza: a trustworthy, secure, open, and verifiable European platform
- Partners:
  - Dresden University — hypervisor Nova, TCB construction kit
  - Nijmegen University — hypervisor verification/specification
  - Secunet Security Networks AG (Dresden) — use case
  - ST Microelectronics (France) — embedded platform

Nizza Architecture



- use an independent kernel (currently Nova)
- source code verification (of C++)
- develop denotational semantics for a subset of C++
- denotational semantics maps C++ into HOL
- denotational semantics is based on a hardware model and a semantics of C++ data types
- proof properties in the interactive theorem prover PVS



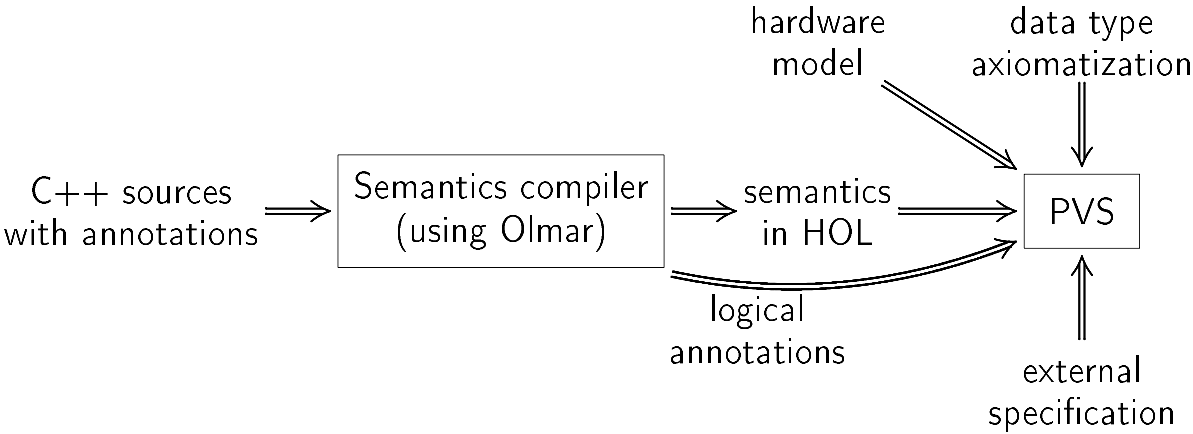
$$\Phi_{data\_types},\quad \Phi_{hardware} \quad \vdash \quad \varphi(\text{hypervisor})$$

- denotational semantics relies on state transformers

$$State \quad \longrightarrow \quad \overset{ok:}{State} \uplus \overset{pagefault:}{State \times Page\_fault\_info} \uplus \overset{hang:}{\mathbf{1}} \uplus \overset{fatal:}{\mathbf{1}} \uplus \cdots$$

- specification for the hypervisor interface developed separately
- base specification in pseudo code (simple set theory with lots of syntactic sugar)

Data flow


hardware model
data type axiomatization
C++ sources with annotations
Semantics compiler (using Olmar)
semantics in HOL
PVS
logical annotations
external specification

- strictly speaking the hardware model is an underspecified hardware specification such that IA32 is a model of it
- physical memory, paging, virtual memory
- TLB
- memory mapped devices
- registers
- provides system state and base operations for source-code semantics:
  - writing in memory
  - reading in memory (which might change the memory state: accessed bits, page faults)
  - special hardware operations: registers (CR3), bits in control registers, ...
- provides a hierarchy of memory interfaces: physical memory, virtual memory, VM with page fault handler
- relies on data type semantics for hardware data types (such as page directory entries)

- highly underspecified specification for each data type
- three levels: uninterpreted data, interpreted data, pod
- consistency proved with PVS theory refinement
- interface

```
size       : nat,
valid?     : [list[Byte], Address -> bool]
uidt       : Uninterpreted_data_type,
to_byte    : [Data, Address -> list[Byte]],
from_byte  : [list[Byte], Address -> lift[Data]]
```

- leaves the object representation of the data completely open
- the object representation might contain type tags (permitted by the C++ standard)
- only functions for conversion to and from the object representation
- converting from the object representation fails for invalid data
- result of interpreting a string as an integer cannot be determined (not even that the conversion does not crash)
- permitted casts must be given as axioms or additional assumptions
- therefore

Normal termination proves dynamic type correctness

Task

- common abstraction of the various memory interfaces for the majority of the code
- deals with virtual memory aliasing
  (two different virtual address regions are mapped to the same physical memory)
- provides shortcut lemmas for well-behaved variable access

Definition, technically

- invariant parameterised with a set of read-only and a set of read-write addresses with additional properties
- a set of system states that is invariant under all memory read and writes within these sets of addresses
- memory accesses within the address sets terminate normally (no page fault occurs infinitely often)
- only expected changes (no virtual memory aliasing)
- in summary

Memory as one would expect

Errors to catch

common programming errors null-pointers, diverging loops

type errors reading data where none or data with a different type has been written (and no permitted cast applies)

virtual memory aliasing shows up as a normal type error in our approach

TLB errors missing INVLPG instructions

allocation errors two different variables overlap

Properties to prove

normal termination no crash

dynamic type correctness implied by normal termination in our approach

only kernel code runs in kernel mode

Unchanged Object Code

- Goal: on every memory write produce a proof obligation:
  the kernel object code is not changed
- work around: prove separately that kernel object code remains unchanged

Connection between the object code and the semantics of the source code

- assume correct compiler(s) currently

Formal Methods in Robin

- source code verification for the Nova hypervisor (separately written)
- denotational semantics for C++ dealing with
  - abrupt termination
  - jumps
  - type casts