Development Methods for Secure Systems

Roger Bishop Jones
ICL Defence Systems
This document consists of the overheads for a presentation to the refinement workshop on March 13 1990.
PRIMARY CURRENT CONCERNS

BROAD SPECTRUM SOFTWARE ENGINEERING

FULL FUNCTIONAL REQUIREMENTS

RIGOROUS PROOF
INTERESTS TO ENCOURAGE

HIGH ASSURANCE APPLICATIONS

CRITICAL REQUIREMENTS ANALYSIS

PROPERTIES AS SPECIFICATIONS

FORMAL PROOFS OF CRITICAL REQUIREMENTS

subproblems

Formalisation of correctness proofs

consistency of specifications

proof theory and tools
MORE EXOTIC ONES

ULTRA HIGH ASSURANCE
(e.g. UKL9 and above)
The OWR

must constrain information flow

from its ‘high’ interface to its ‘low’ interface

to less than 5 bits per second

(statement of requirements

or

policy

or

specification)
The OWR does

constrain information flow...

statement of *conformance*

*correctness* proposition

*correspondence* proposition
FORMALISE

the

CORRECTNESS

PROPOSITION
Such a proposition would traditionally be analysed as having subject-predicate form.

"the OWR"
is the subject,

"constrains the information flow..."
is the predicate

predicates are formally represented:

- in the predicate calculus:
  as predicates!

- in simple type theory:
  as propositional functions

- in set theory:
  as sets
PREDICATES

are

PROPERTIES
In a TYPED, CLASSICAL language
(e.g. Z or HOL)

a property (e.g. secure) must be expressed over some type (say OWR), then:

\[ \text{secure : } \mathbb{P} \text{ OWR} \]

or

\[ \text{secure : } \text{OWR} \rightarrow \text{bool} \]

a system, owr, purporting to have the property, must have the underlying type (OWR)

The specification has a type distinct from that of the entities specified.

\[ \text{secure: } \mathbb{P} \text{ OWR} \]
\[ \text{owr: OWR} \]
The correctness proposition:

\[ \vdash \ ? \ owr \in \text{secure} \]

or

\[ \vdash \ ? \ \text{secure} \ owr \]
(critical requirements)
SPECIFICATIONS

ARE
(should be?)

PROPERTIES
(or sets if you like)
The ICL Defence Systems
HIGH ASSURANCE TEAM

Began as "FORMAL METHODS UNIT"
established in 1985 by Roger Stokes

SLOGAN:

working on real problems
with real tools

CURRENT TEAM:

Dr. Rob Arthan
(tools)
Dr. Kevin Blackburn
(proof technology and secure applications)
Adrian Hammon
(proof HCI and secure applications)
Dr. Barry Homer
(security modelling)
Roger Jones
(leader, foundations)
Geoff Scullard
(hardware verification)
Roger Stokes
(methods)
BACKGROUND
(Apologies for lack of Elegance)

UK/USA evaluation guidelines
intelligibility/fidelity/tractability
expressibility of correctness proposition
provability in HOL
consistency of specification
the problem of instantiation
requirement for ‘conventional’ FTLS
reality/ideals
which reality?
OBJECTIVES

To DEVELOP SECURE SYSTEMS.

To achieve ‘HIGH LEVELS of ASSURANCE’ that SYSTEMS developed ARE SECURE.

‘LEVEL of ASSURANCE’ is DEFINED in CESG evaluation guidelines.

Current focus is on UK LEVEL 6.
STATEMENT of PROBLEM

(a) Formalise Security Requirements

(b) Produce a formal description (specification) of a system meeting the security (and other) requirements.

(c) Produce a (formal machine checked) proof that the system specified meets the requirements stated.

In other words:

(a) Write Security Policy Model (SPM)

(b) Write Formal Top Level Specification (FTLS)
(c) Prove that the system described by the FTLS meets the requirements expressed in the SPM.
CONSTRAINTS on SOLUTION

MODEL must CAPTURE FLOW CONSTRAINTS

(because these are part of the customers requirements)

the TARGET SYSTEM must be DETERMINISTIC

The FTLS must ultimately be in the ‘conventional’ style

(a collection of schemas describing the operations of the system, refinable by ‘standard’ techniques)
THE METHOD PRESCRIBES:

1. Form of Security Policy Model

2. Form of FTLS

3. How to interpret the FTLS in terms of the SPM

4. Form of Correctness Proposition asserting that the system described by the FTLS conforms to the policy prescribed in the SPM.
FORM of SECURITY POLICY MODEL

- Requirements are expressed in the Security Policy Model as a **PROPERTY** over some suitable ‘base’ type (e.g. AUTO).

- The base type is typically a generic schema type, the components of which include:

  (a) A complete BEHAVIOURAL MODEL of target system.
      (e.g., \( tf : \text{IN} \times \text{STATE} \rightarrow \text{STATE} \times \text{OUT} \))

  (b) ‘Abstraction’ maps showing how some of the generic types may be interpreted in terms suitable for the expression of the policy.
      (e.g. \( files : \text{STATE} \rightarrow (\text{NAME} \rightarrow \text{DATA}), \text{class} : \text{STATE} \rightarrow (\text{NAME} \rightarrow \text{CLASS}) \))

  (c) Any OTHER INFORMATION necessary to express the security requirements which will vary from one target system to another.
      (e.g. \( dom : \text{CLASS} \leftrightarrow \text{CLASS} \))

- The properties include information flow con-
straints and other requirements (secure: P AUTO).
FORM of FTLS

(a) A collection of schemas describing operations over some suitable STATE.

(b) The operations must be TOTAL and DETERMINISTIC

(c) They may be loosely specified.

An understanding of the distinction between NON-DETERMINISM and LOOSENESS (sometimes called UNDERSPECIFICATION) and of how to write LOOSE descriptions of DETERMINISTIC operations is essential. See SPIVEY "UNDERSTANDING Z" Section 5.3.
INTERPRETATION by DIRECT CONSTRUCTION

- We need to construct an entity of some instance of ‘Base type’ (AUTO) which corresponds to the system defined in the FTLS.

This involves:

(a) Coercing the behavioural description to a suitable type.
(an instance of $IN \times STATE \rightarrow STATE \times OUT$)

(b) Supplying interpretive maps for particular types used (abstraction maps).
(define files and class)

(c) Supplying auxiliary information (e.g. dominance relation).
(define dom)

(d) Combining these into a schema binding.
(system : AUTO[..] | ...)

INDIRECT INTERPRETATION

Interpretation may be done in stages.

In this case:

An entity of some other type (system:SYSTEM) is constructed from the FTLS in a manner similar to the above.

An interpretation map is defined showing how entities of type SYSTEM may be interpreted as entities of type AUTO.

(interpret: SYSTEM → AUTO[...])

Further intermediate systems may be used if necessary, in which case the interpretation will occur through the composition of more than one interpretation map.
THE CORRESPONDENCE PROPOSITION

The proposition to be proven is the proposition that the entity defined in the FTLS when interpreted as an AUTO is secure.

\[ \text{interpret(system)} \in \text{secure} \]
METHODS of ‘REFINEMENT’

By ADDING REQUIREMENTS

By INSTANTIATION
of security model

By STRUCTURAL DECOMPOSITION
of behavioural model
(maintaining separation of critical properties)

By INTERPRETATION
of target system model
in terms of security model

By ‘conventional’ methods
from FTLS schemas down