Formal Methods in Practice: the Missing Link. A Perspective from the Security Area

D. Bolignano, D. Le Métayer,
C. Loiseaux

Trusted Logic, France
www.trusted-logic.fr

The bright side: potential benefits of formal methods

- Development methodology
  - Modelisation
  - Early bug detection
- Software validation
  - Consistency checking
  - Verification of properties
  - Testing
- Communication
  - Clarity (no ambiguity)
  - Completeness
  - Document generation

The dark side:
the obstacle race of
the determined formalist

8 What help can I expect for

8 building on existing documentation?
8 understanding what to specify and how?
8 specifying and proving in the large?
8 dealing with systems rather than individual components?
8 using appropriate tools to process the spec?
8 explaining and justifying specs and proofs?
8 integrating the formalisation into a traditional development environment?
8 assessing the benefits of the formalisation?

Evaluation Assurance Levels

- Each level provides an internally consistent general purpose assurance package
- Assurance components are levelled on the basis of the degree of formality and degree of detail

EAL1: functionally tested

EAL2: structurally tested

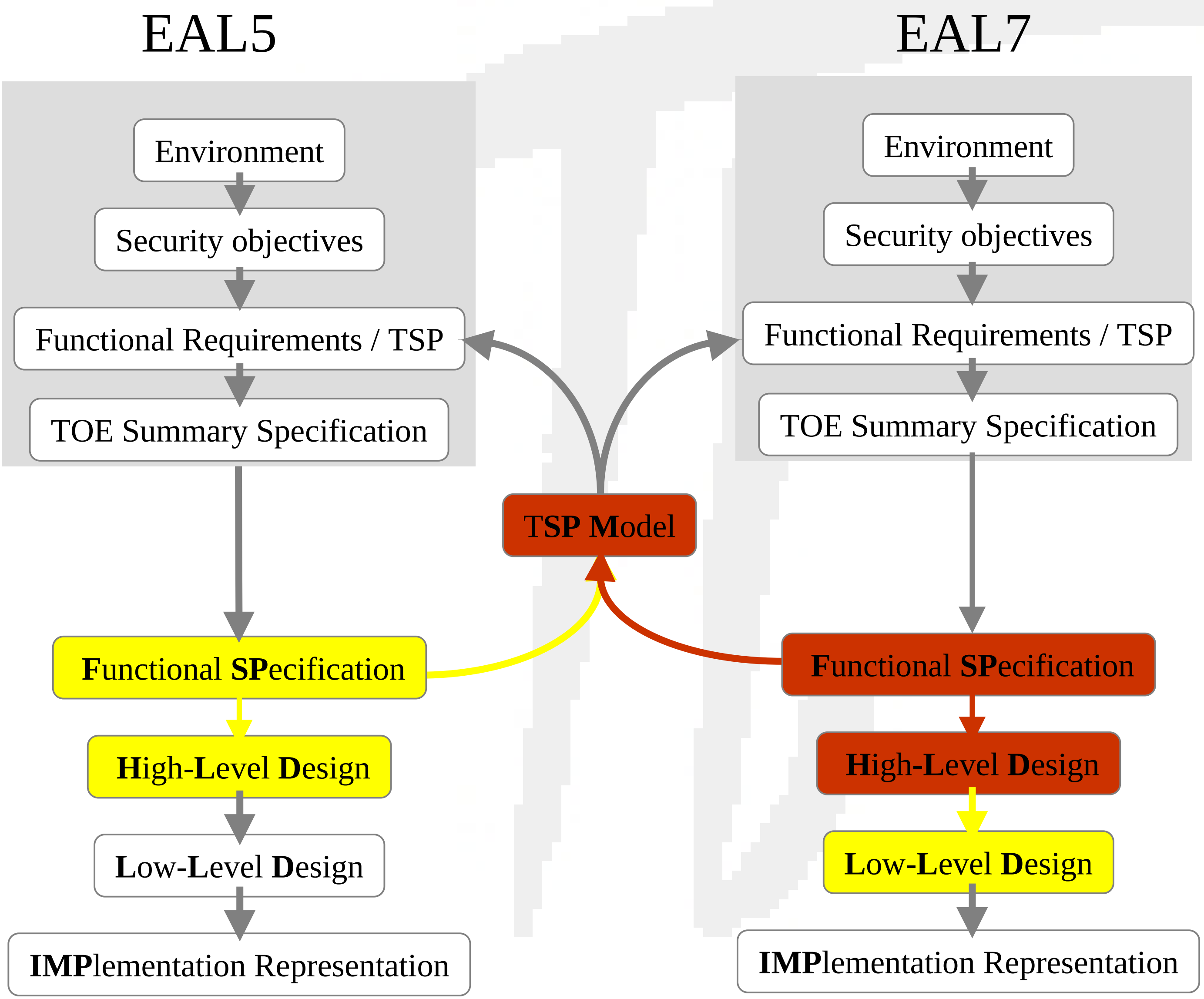
EAL3: methodically tested and checked

EAL4: methodically designed, tested and reviewed

EAL5: semiformally designed and tested

EAL6: semiformally verified design and tested

EAL7: formally verified design and tested


EAL5
EAL7
Environment
Security objectives
Functional Requirements / TSP
TOE Summary Specification
TSP Model
Functional SPecification
High-Level Design
Low-Level Design
IMPlementation Representation
Environment
Security objectives
Functional Requirements / TSP
TOE Summary Specification
Functional SPecification
High-Level Design
Low-Level Design
IMPlementation Representation

Building on existing documentation

8 **Two observations concerning the CC:**

- Higher level certifications don't start from scratch: informal and/or semi-formal documents are available
- Difficult to build on existing certification documents to prepare certifications at higher levels
- Even higher level certifications do not require only formal documents

8 **Formal methods never live in a vacuum:**

- Requirements are informal
- The ultimate goal is to convince (clients, certification bodies, etc.)

8 **Formal, semi-formal and informal methods: integration rather than opposition**

- The key issue is rigour, not formality

Establishing links between informal, semi-formal and formal methods

- **Current work in the context of UML**
  - Semantics of UML diagrams
  - Enhancing consistency checkings
  - Integration of formal and informal notations (UML-Z)
- **The TL-FIT environment**
  - Smooth integration of informal, semi-formal and formal documents through an appropriate modelisation discipline
  - Automatic consistency checkings (internal, refinement, traceability)
  - Automatic generation of documents for CC certifications
  - Capitalisation of efforts

Trusted Logic

What to specify and how

8 **Different options for modelisation**
- Focus on security relevant issues
- Choose an appropriate level of abstraction
- Provide a rationale for these choices

8 **Different specification languages**
- General purpose languages lead to decidability problems and require heavy user assistance
- Domain specific specification languages for automatic verification (ex: cryptographic protocol verification)?

8 **Different interpretations of the CC**
- The added value of formal methods is manifest only if they tackle the most complex issues

Example: the Java Card VM (I)

8 **Excerpts from Sun documentation:**

- “There is no way to forge pointers to enable malicious programs to snoop around inside memory.”
- “The Java Card firewall provides protection against the most frequently anticipated security concern: developer mistakes and design oversights that might allow sensitive data to be leaked to another applet”.
- “The firewall also provides protection against incorrect code. If incorrect code is loaded onto a card, the firewall still protects objects from being accessed by this code”.

Example: the Java Card VM (II)

8 **Modelisation and CC:**

- What are the key security properties (Req)?
- How should they be designed to map the CC components (SPM, FSP, HLD, etc.)?
- Should they be expressed as access control policies, information flow policies, combinations of those, etc.?
- What are the key components (type checker, linker, firewall, interpreter, etc.)?
- How do these components cooperate to achieve the key security properties?
- Is it possible to prove that “incorrect” applets are really harmless for other applets on the card?

Trusted Logic

Example: the Java Card VM (III)

8 **One possible option:**

- Security policy model (SPM):
  - Collection of dedicated abstract machines with the associated invariant properties
  - Consistency checking of the abstract machines
  - Global properties based on the collection of invariants
- Functional specification (FSP):
  - External behaviour of the key components
- Correspondence between SPM and FSP:
  - Link between the SPM and the key components
- High level design (HLD):
  - Collection of high level descriptions of the key components
- Correspondence between FSP and HLD:
  - Correctness of the design of the key components

Further research topics on modelisation

- **How to reason about:**
  - Denial of service
  - Authentication
  - Key management
  - Risk analysis (attack trees?)
  - Native methods

  etc.
- **Domain specific languages for specifying security policies (subjects, objects, roles, life cycles, etc.)?**

Specifying and proving in the large

- **Lack of a methodology to design and manage large proofs.**
  - Large proofs take a long time to design and to debug
  - Little use for communication
  - Difficult to maintain and reuse
- **Mechanised formal methods are pieces of software themselves**
  - Inspiration should be taken from the software engineering area: proof architecture, proof debugging, proof reusability, object-oriented proof techniques, etc.?
  - Evolution should be considered from the start: varying assumptions

Dealing with systems rather than components

- Current trend: open systems
- Specific needs for certifying systems rather than complete products:
  - Independent validation of the architecture and the individual components
  - Conditions on the components to ensure that they can be integrated within the architecture
  - Different security requirements on components
  - Composition of security polocies
- Compositionality issues in formal methods:
  - Software architectures
  - Compositional verification techniques

Using appropriate tools (I)

- **Variety of tools:**
  - Proof assistants
  - Model checkers
  - Static analysers
  - Type checkers
  - Constraint solvers
- **No panacea:**
  - Variety in expressive power
  - Variety in the degree of mechanisation
- **Need for cooperation between tools:**
  - Foundation level
  - Strategy level
  - Interface level

Using appropriate tools (II)

8 **Domain specific rather than general purpose verification environments?**

- Suitable expressive power
- High level of mechanisation
- Capture the expertise of a given domain

8 **Illustration: verification of cryptographic protocols**

- Step 1: modelisation of the protocol (with explicit assumptions on the environment) in a domain specific language
- Step 2: abstraction of the initial system (based on trust assumptions on the actors of the protocol)
- Step3: model checking of the abstract system

Key issues:

- Integrated formal model
- Varying assumptions

Explaining and justifying specs and proofs

- The ultimate goal is to convince (users, partners, certification bodies, etc.)
- Variety of languages and verification tools: is it possible to convey the essence of a spec or a proof to a non-expert?
- Is it really necessary to get into the intricacies of a specific tool to grasp the essence of a proof?
- Is the project of a “neutral” proof style just an impossible dream?

Integration of formal methods in a traditional development environment

- For a long time formal methods have been designed with the (wrong) assumption that they should supersede traditional methods
- "Seven myths of formal methods" [Hall, IEEE Software 1990]:
  - "Formal methods can guarantee perfect software and eliminate the need for testing"
  - "Formal methods are all about proving programs correct",
  - ...
- Formal methods should be designed to improve traditional methods rather than to replace them
  - Evolutionary process
  - Build on existing expertise
  - Improve the cost/benefit ratio of formal methods

Illustration: test suite generation

- **TL-CAT: automatic test suite generation**
  - Relieves the tester from the burden of choosing specific test data
  - Builds not only test cases but also test suites
  - Link with APDU-Script to automatically send the scripts to the card or to a card simulator
- **Two parameters: specification and test strategy**
  - Simple specification language well suited to card applications
  - Strategies expressed in terms of data domain coverage and control path coverage
- **Benefits of using TL-CAT**
  - Reduce testing costs
  - Improve the quality of the process (explicit test strategies)
  - Help in the production of the CC test assurance component

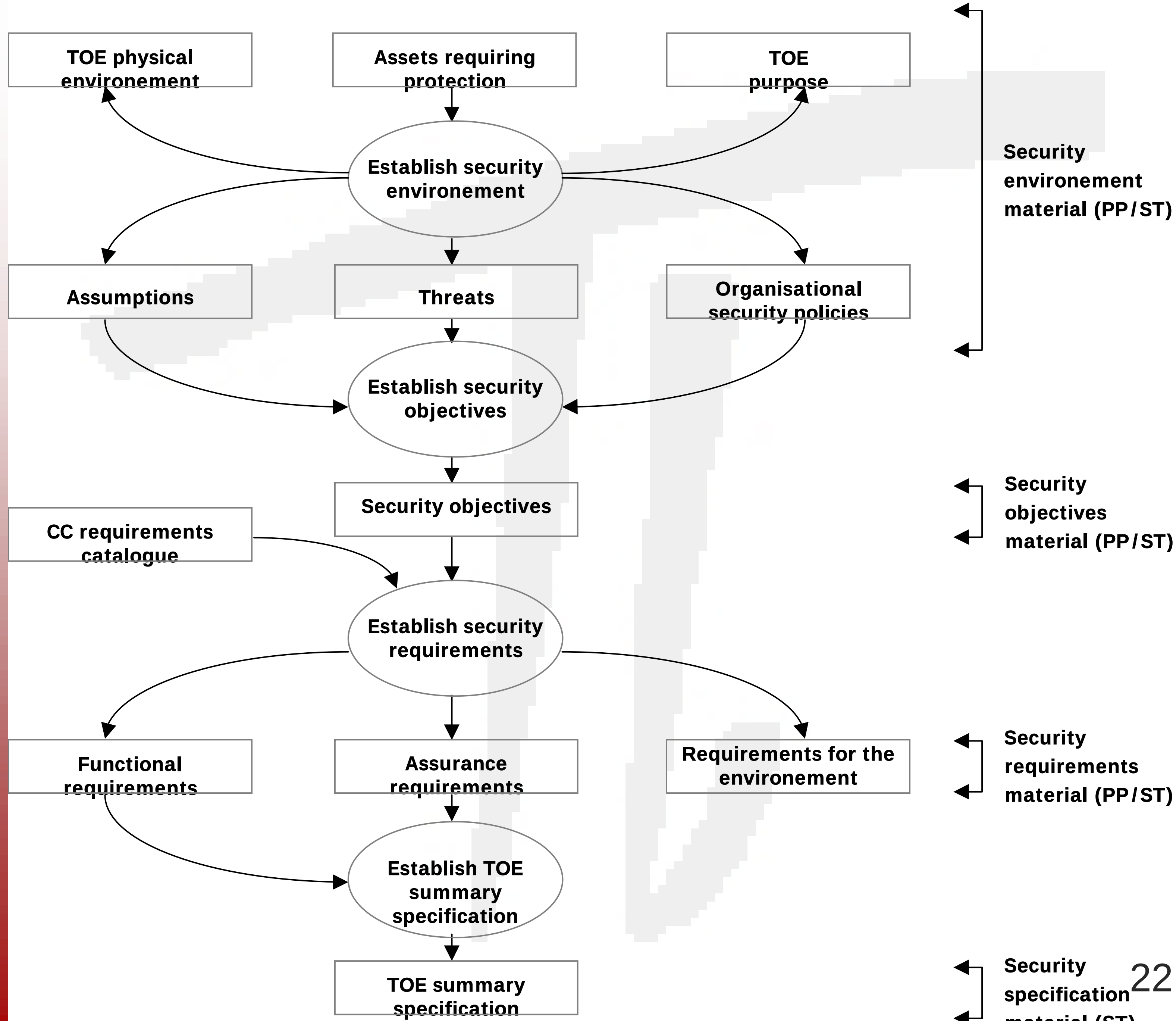
Assess the benefits of the formalisation

- Formalisation is always partial: need to justify choices and assumptions about the parts which are not formalised (eg. SPM in the CC).
- The CC also require justifications for the methods and tools that have been used.
- The ultimate goal is to increase the level of confidence: is it possible to quantify this improvement?
- Can inspiration be taken from the semi-formal development methods? A modest approach to formal methods: check consistency constraints between a variety of representations (or views) produced by different actors.
- “Social processes and proofs of theorems and programs” [R. De Millo, CACM 1979] .

Conclusion

- **The real challenge today is to improve the formalisation infrastructure rather than to design more powerful proof techniques**
  - Integration with semi-formal methods
  - Methodology for modelisation in the context of the CC
  - Methodology for proving in the large
  - Better communication with non-experts
  - Better integration between tools and guidelines to use them in an appropriate way
  - Domain specific environments
- **These problems are not only engineering issues, they are true research challenges**

Trusted Logic



Formal Methods in Common Criteria

- EAL7 — Formally verified design
- EAL6 — Semi-formal verified design
- EAL5 — Semi-formal designed
- EAL4 — Methodically designed
- •
- •
- •
- EAL1 — Functionally tested

Assurance Class	Assurance Family	Evaluation assurance level		
		EAL5	EAL6	EAL7
Configuration management				
Delivery and operation				
Developpement	ADV_FSP	SF	SF	SF
	ADV_HLD	SF	SF	F
	ADV_IMP	I	I	I
	ADV_INT	I	I	I
	ADV_LLD	I	SF	SF
	ADV_RCR	SF	SF	F
	ADV_SPM	F	F	F
Guidance document				
Life cycle support				
Tests				
Vulnerability assesment				