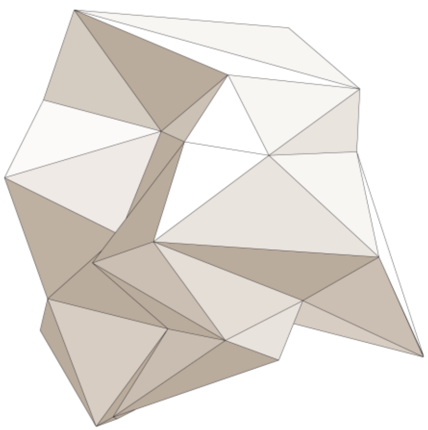
Correctness by Construction for Security

FMATS2

5th February 2013



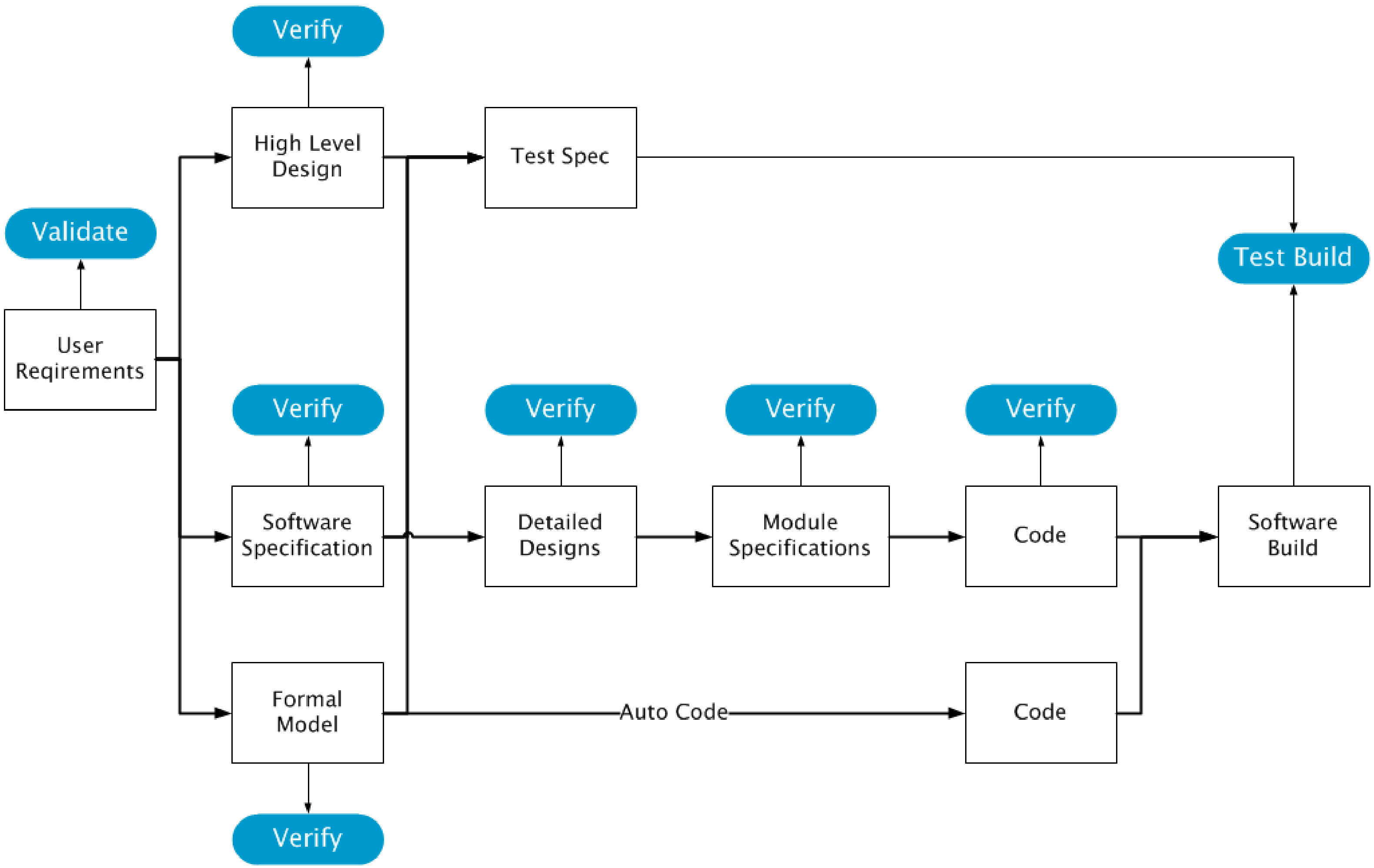
altran

CONTENT

Correctness by Construction

- Avoid introducing defects
- Remove defects as early as possible
  - Unambiguous notations
  - Take small steps
  - Appropriate notations
  - Don’t repeat information
  - Justify decisions
  - Check each stage before progressing
  - Solve difficult problems first

Correctness by Construction



Correctness by Construction

Specification - Z Notation

DoorL...
current...
current...
current...
doorAl...
latchTi...
alarmT...

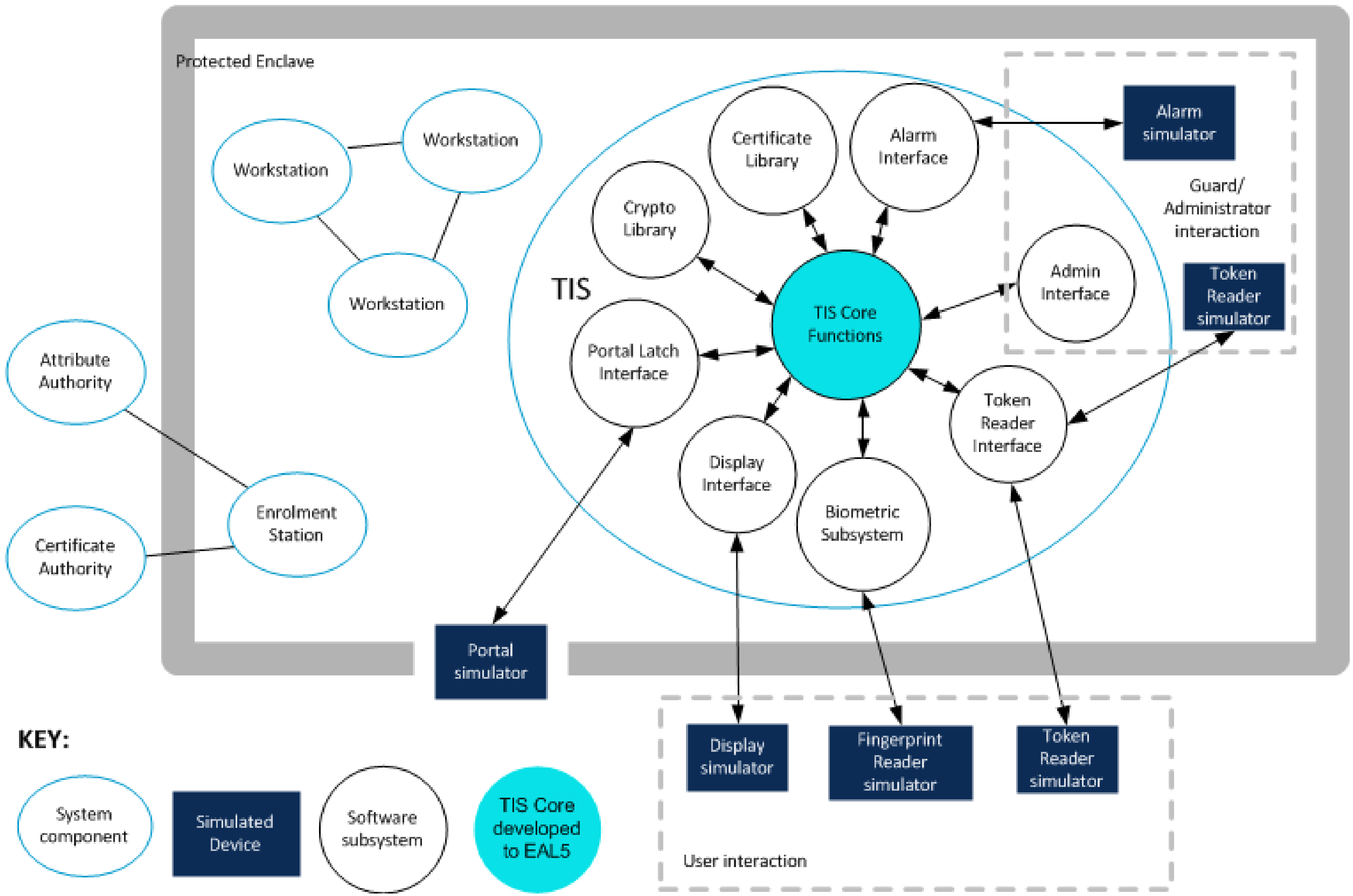
current...
doorAl...
(*c*...

Implementation - SPARK

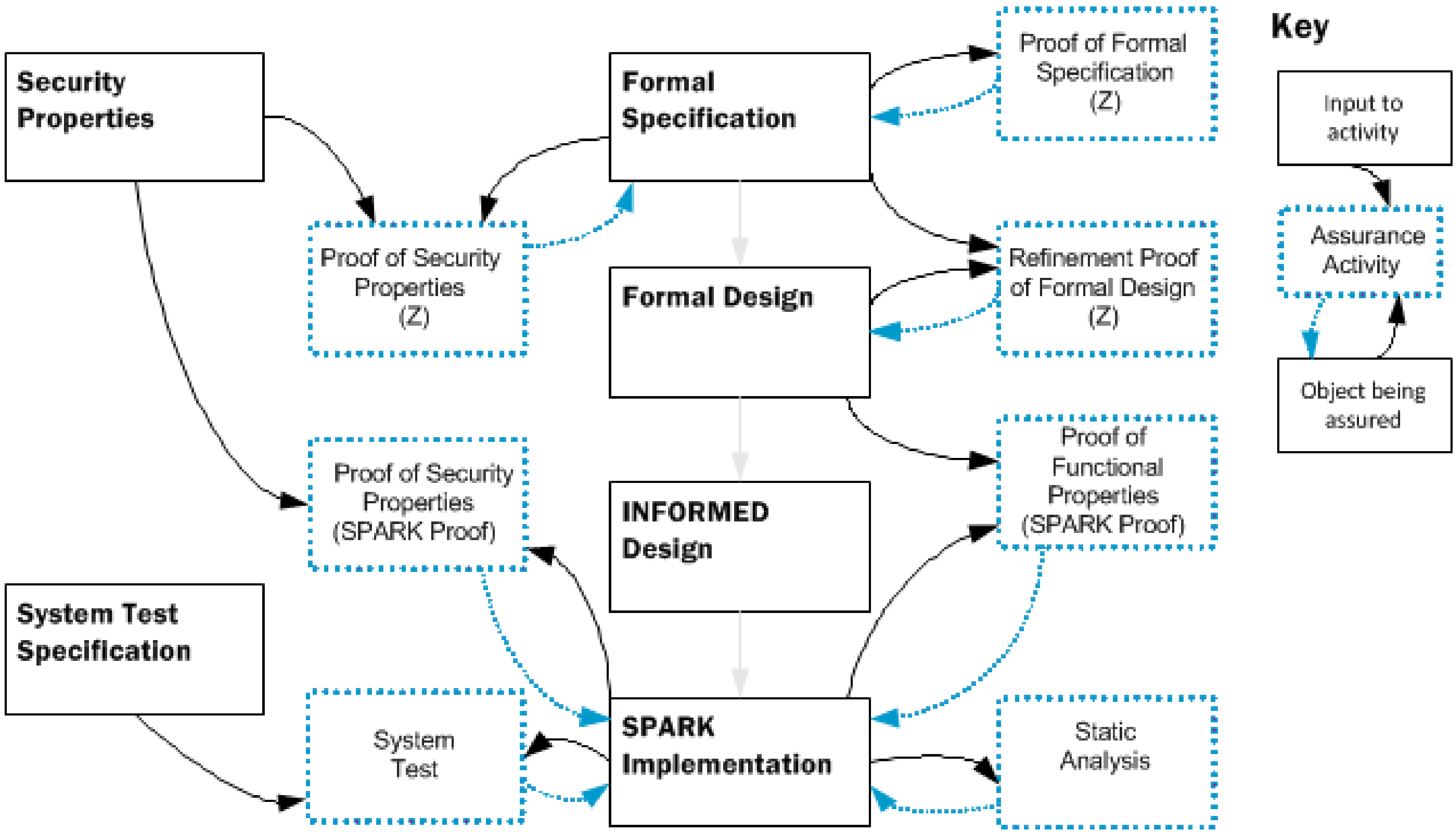
```
procedure UnlockDoor;
--# global in      Clock.CurrentTime;
--#        in      Clock.Now;
--#        in      ConfigData.State;
--#        in out  State;
--#        in out  Latch.State;
--#        ...
--# derives State,
--#         Latch.State from *,
--#                          Clock.CurrentTime,
--#                          Latch.State,
--#                          ConfigData.State &
--#         ...
--# post
--#       ( Latch.IsLocked(Latch.State) <->
--#         Clock.GreaterThanOrEqual
--#           (Clock.TheCurrentTime(Clock.CurrentTime),
--#            Latch.prf_LatchTimeout(Latch.State)) );
```


altran

Tokeneer System



Assurance Process



Tokeneer Assurance in SPARK

- Tokeneer had a number of security properties all of which were functional in nature.
  - Eg. An alarm will be raised whenever the door/latch is insecure.
- Application architecture was a simple cyclic scheduler, regularly polling for inputs, processing and generating outputs.
- Security properties were formulated as post conditions on the procedure implementing a single interaction through the scheduling loop
- Use of the SPARK language automatically eliminated a number of potential language insecurities.
- Performing proof of absence of run-time errors provided an efficient way of ensuring the program would not raise exceptions.

Tokeneer Experiment Results

- Lines of code : 9939
- Total effort (days) : 260
- Productivity (lines of code / day) : 38
- Process assessment : EAL5 +

- Defects found to date : 5

Industrial Challenges to using Formal Methods

- Challenges to adopting formal methods can be divided into those introduced by the **Notation** and those introduced by the **Tools**.

- Scalability
  - Will the notation and tools cope with a large system?
- Familiarity of Notation
  - Can we hide the formalism from users?
- Expressiveness
  - How easy is it to say what we want?

- Speed
  - How long will it take to get results?
- Support
  - Where do we get help when tools don't work as expected?
- Ease of Interpretation
  - How easy is it to understand the output?

Security and SPARK

- To show a system is secure we need to demonstrate that it satisfies a number of security properties.
  - These can be functional where they capture things the system must or must not do to be secure
    - E.g. Door only unlocked when valid token presented.
  - Or information driven often requiring non-interference of data from different security contexts
    - E.g. Unclassified context must not include classified information
- SPARK Static Analysis can help with both classes of problem
  - Post conditions capturing functional properties are added to SPARK specifications and proven.
  - Information flow analysis can be used to demonstrate non-interference between different classes of data.

Security and SPARK Information Flow Analysis

- Information flow analysis can be difficult to use effectively
- Derives contracts can be difficult to maintain
  - Without abstraction they expose detail of the information flow through the whole program
  - A small change at the bottom of a calling hierarchy can ripple up through the system.
- Derives contracts for complex structures can obscure the true information flow
  - Either the use of data abstraction or structures such as arrays can result in “phantom” dependencies being identified.
- Derives contracts do not take into account declassification of data

Security and SPARK Information Flow Analysis

- Use of complex structures can obscure the true information flow.

```
type KS_T is array ( KS_Range ) of Key_T;
Key_Store : KS_T;

procedure Load_Key (Index : in KS_Range; Key : in Key_T);
--# global in out Key_Store;
--# derives Key_Store from *, Index, Key;

procedure Get_Key (Index : in KS_Range; Key : out Key_T);
--# global out Key_Store;
--# derives Key from *, Index, Key_Store;

procedure Manage_Keys (Key1 : in Key_T; Key2 : out Key_T)
--# global in out Key_Store;
--# derives Key2, Key_Store from Key1, Key_Store;
is
begin
   Load_Key (1, Key1); Get_Key (2, Key2);
end Manage_Keys;
```

Security and SPARK Information Flow Analysis

- Derives contracts do not take into account declassification of data

```
procedure Produce_Output
--# global in Secret_Data;
--#        out Unclassified_Output;
--# derives Unclassified_Output from Secret_Data;
is
begin
   Unclassified_Output := Declassify (Secret_Data);
end Produce_Output;
```

- The function Declassify converts secret data to unclassified data.
- This is not apparent from information flow.

A new SPARK Language

- The new generation of SPARK and the toolset based on Ada2012 provides an opportunity for change.
- Language and Tool development is a collaborative project involving Altran, AdaCore and KSU.

A new SPARK Language – new features

- The language will support several profiles aimed at different communities including a security profile.
- Proof contracts will be expressed as Ada aspects that can be interpreted by the compiler as well as the SPARK toolset.
  - Ada pre- and post- condition aspects can be checked by the compiler at execution time.
  - Improves confidence in the specifications given to non-SPARK fragments of code by testing against the specification contract used by SPARK.
  - Better support for mixed language programming.

```
procedure Swap_Array_Elements(A: in out Array_Type)
with Global => (Input => (X, Y)),
     Pre => X /= Y and X in Index and Y in Index,
     Post => A = A'Old'Update(X => A'Old(Y),
                              Y => A'Old(X));
```

A new SPARK Language – opportunity for change

- Should it be possible to perform information flow at the level of array elements?

- Should the tools provide an option to allow information flow contracts to be reverse engineered from the code?

- Would the addition of ghost variables to the proof contexts aid reasoning about system properties?

- Could data be tagged with a security classification and the tools police data non-interference?

What are your thoughts?

Altran UK Limited
22 St Lawrence Street
SouthGate
BATH BA1 1AN

Tel: +44 1225 466991
Fax: +44 1225 469006
Website: www.altran.com

Email: janet.barnes@altran.com

INNOVATION MAKERS