Safety and Security Case Study Experiences with Event-B and Rodin

Rodin Workshop, 8 June 2021

Jonathan Hammond

Capgemini engineering

Agenda

01

Introduction

- Capgemini engineering
- Network Rail
- HICLASS
- Event-B tooling

02

Safety Case Study: ERTMS & RCA

- Case study description
- Model overview
- Results

03

Security Case Study: Tokeneer

- Case study description
- Model overview
- Results

04

Summary of Language & Tooling Challenges

- Language
- Scalability
- Test generation
- Instantiation

Capgemini engineering

Capgemini and Altran join forces in **Engineering and R&D** to create the **global digital transformation leader** for industrial and tech companies

52,000+
people

30+
countries

Network Rail Target 190plus

Network Rail owns, operates and develops Britain's railway infrastructure

Target 190plus is an R&D programme on signalling sustainability, aiming to reduce whole life costs & enable ERTMS (ETCS) long-term deployment

1 of the 29 Target 190plus projects (DMAP) is assessing possible formal methods use

DMAP: Data & Mathematical Assurance Process

DMAP aims to develop a process and method to:

- Allow some mathematical assurance of future signalling systems
- Reduce risks of system failures (& thus also cost / time issues) by earlier fault identification
- Achieve more automation of assurance activities

Capgemini is supporting Network Rail on DMAP by:

- assessing candidate formal methods
- proposing a process for DMAP
- establishing how to integrate DMAP into wider engineering & procurement context

Event-B was shortlisted as a candidate method for future Network Rail use

Capgemini did an ERTMS-related case study using Event-B

HICLASS

HICLASS (High-Integrity, Complex, Large, Software and Electronic Systems) is a 4-year industry-led research project funded by InnovateUK to "enable UK industry to build and support the most complex, connected, cyber-secure avionic systems in the world"

It follows on from the SECT-AIR project, which looked at reducing barriers for using formal specification (see ABZ'18 paper: ABZ Languages and Tools in Industrial-Scale Application)

Within SECT-AIR in 2017, Capgemini did an Event-B evaluation with mixed results

Further tools development (notably CamilleX) was underway but not yet available

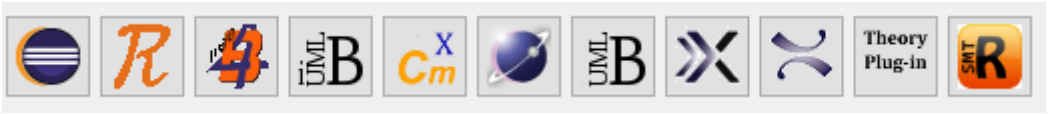
In HICLASS in 2021 we've revisited the SECT-AIR Event-B evaluation, using the latest tools

Key driver for Capgemini is to get more value from formal specification effort

- do current tools give more validation (e.g. animation) & verification (e.g. test generation) support?

Acknowledgement: This work was supported by the HICLASS project, funded by the Aerospace Technology Institute and Innovate UK, as project number 113213.

Event-B Tooling



For both case studies we primarily used Rodin with the following additional plug-ins:

- Atelier-B and SMT solvers for additional proof capability
- UML-B support for class diagrams and state machines
- CamilleX primarily for model structuring using machine inclusion
- ProB for animation support and some model checking
- Scenario Checker for saving and replay of scenarios
- Event-B Theory Plug-in to extend the notation (for sequences & optionality)

For the 2nd case study we also used the standalone ProB 2 UI

- for animation, model checking and test generation

Safety Case Study Description: ERTMS & RCA

ERTMS is well-known European standard for interoperable advanced railway signalling

EULYNX is standardising signalling interfaces

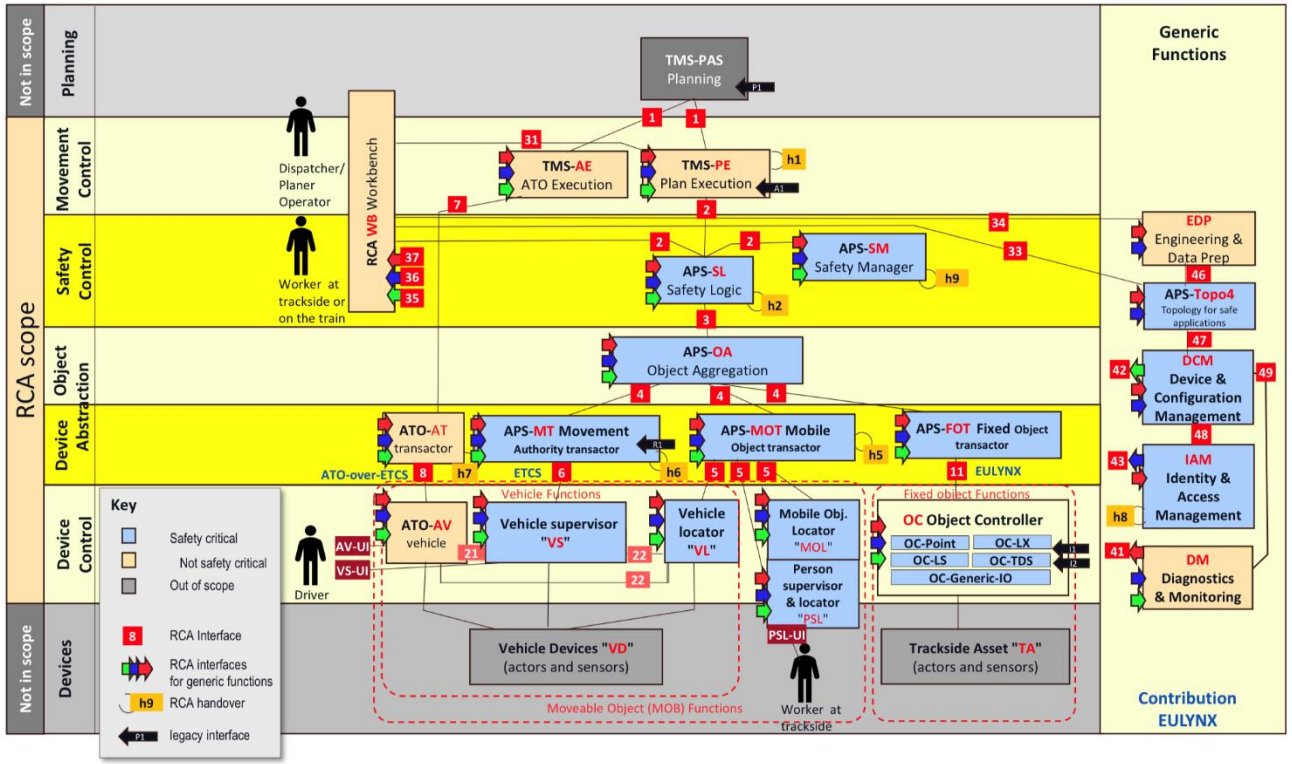
Reference CCS Architecture (RCA) is a new architecture based on ERTMS & EULYNX for modular standard products

Our DMAP case study requirements combine

- Hybrid ERTMS case study (from ABZ 2018)
- ASTRail ERTMS+ATO comms case study
- Using RCA components as the subsystems
- Generic track layout with points & crossings

Aim is **not** to do a complete formal model, but to cover enough to enable Event-B evaluation against Network Rail's criteria

- Need a system of systems to assess criteria



RCA overview – showing main components

Model Overview: Structure

CamilleX machine inclusion splits model into more manageable pieces and permits reuse

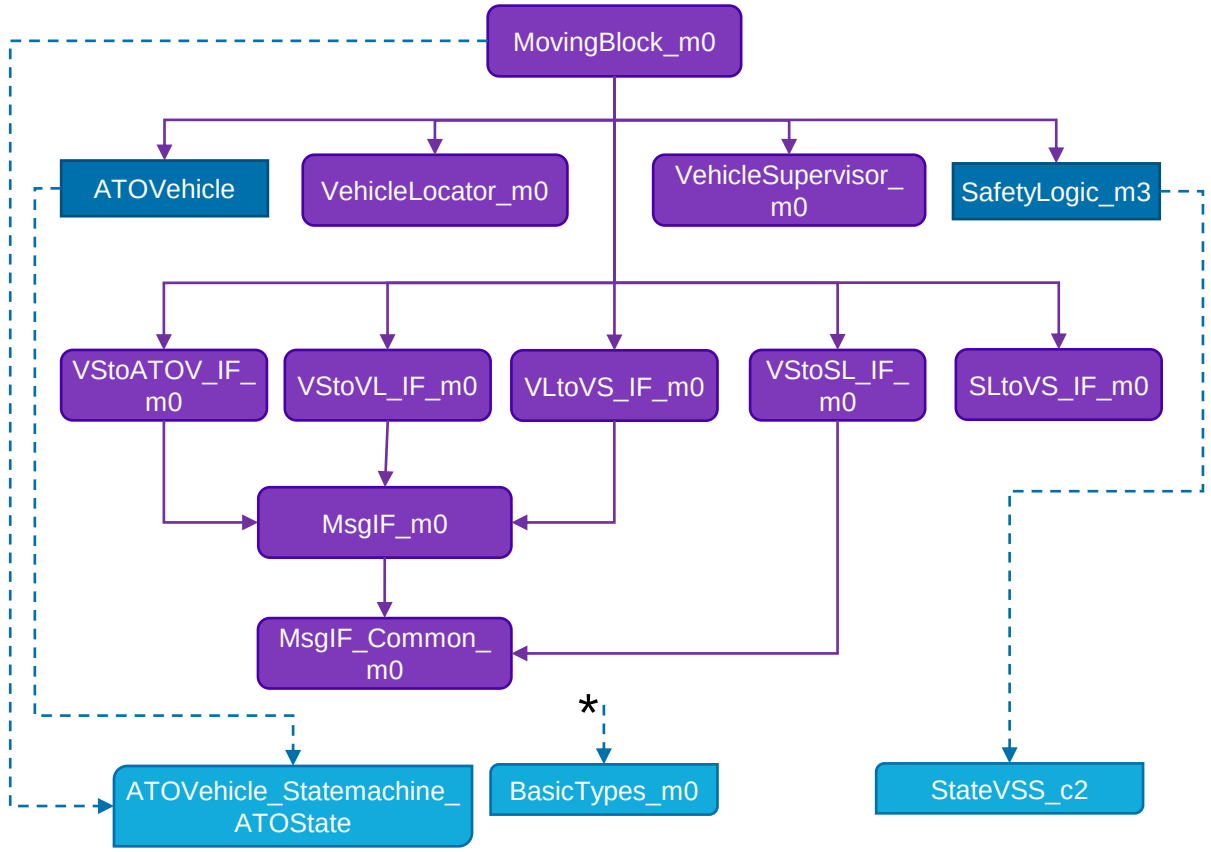
System of systems level machine, which combines all lower level machines

1 machine per subsystem from the RCA

1 machine per interface between RCA subsystems

Common interface definition for 1:1 interfaces (1 sender & 1 recipient)

Common interface definition for n:1 interfaces (n senders & 1 recipient)



Key:

CamilleX machine

Event-B machine

Context

Note: not all contexts are shown

Machine includes another machine

Machine sees context

Model Overview: Structure – CamilleX Machine Inclusion

Machine inclusion is simple to specify (we use prefixes to avoid any name clashes)

```
machine MovingBlock_m0

sees
     BasicTypes_m0
     StateVSS_c2
     ATOVehicle_Statemachine_ATOState

/* include machines for all subsystems */
includes VehicleSupervisor_m0 as vs
includes VehicleLocator_m0 as vl
includes SafetyLogic_m3 as sl
includes ATOVehicle as av
```

```
/* include machines for each interface
   between subsystems */
includes VStoSL_IF_m0 as vsToSL
includes SLtoVS_IF_m0 as slToVS
includes VStoVL_IF_m0 as vsToVL
includes VLtoVS_IF_m0 as vlToVS
includes VStoATOV_IF_m0 as vsToATOV
```


Model Overview: Example Safety Property

Some safety properties may be captured as invariants, such as:

```
inv21: ∀ t1, t2 · t1 ≠ t2 ∧ {t1, t2} ⊆ dom (trainMA)
                    ⇒ pathLoc [{trainMA (t1)}] ∩ pathLoc [{trainMA (t2)}] = ∅
```

The safety property breaks down as follows:

`∀ t1, t2 · t1 ≠ t2`	for all items `t1` & `t2` that are different
`  ∧ {t1, t2} ⊆ dom (trainMA)`	and both `t1` & `t2` are trains with current Movement Authorities
`⇒`	it is true that
`  pathLoc [{trainMA (t1)}]`	the set of locations on `t1`'s Movement Authority (MA)
`   ∩`	that are common with
`  pathLoc [{trainMA (t2)}]`	the set of locations on `t2`'s MA
`  = ∅`	is empty (i.e. there is no overlap between the MAs)

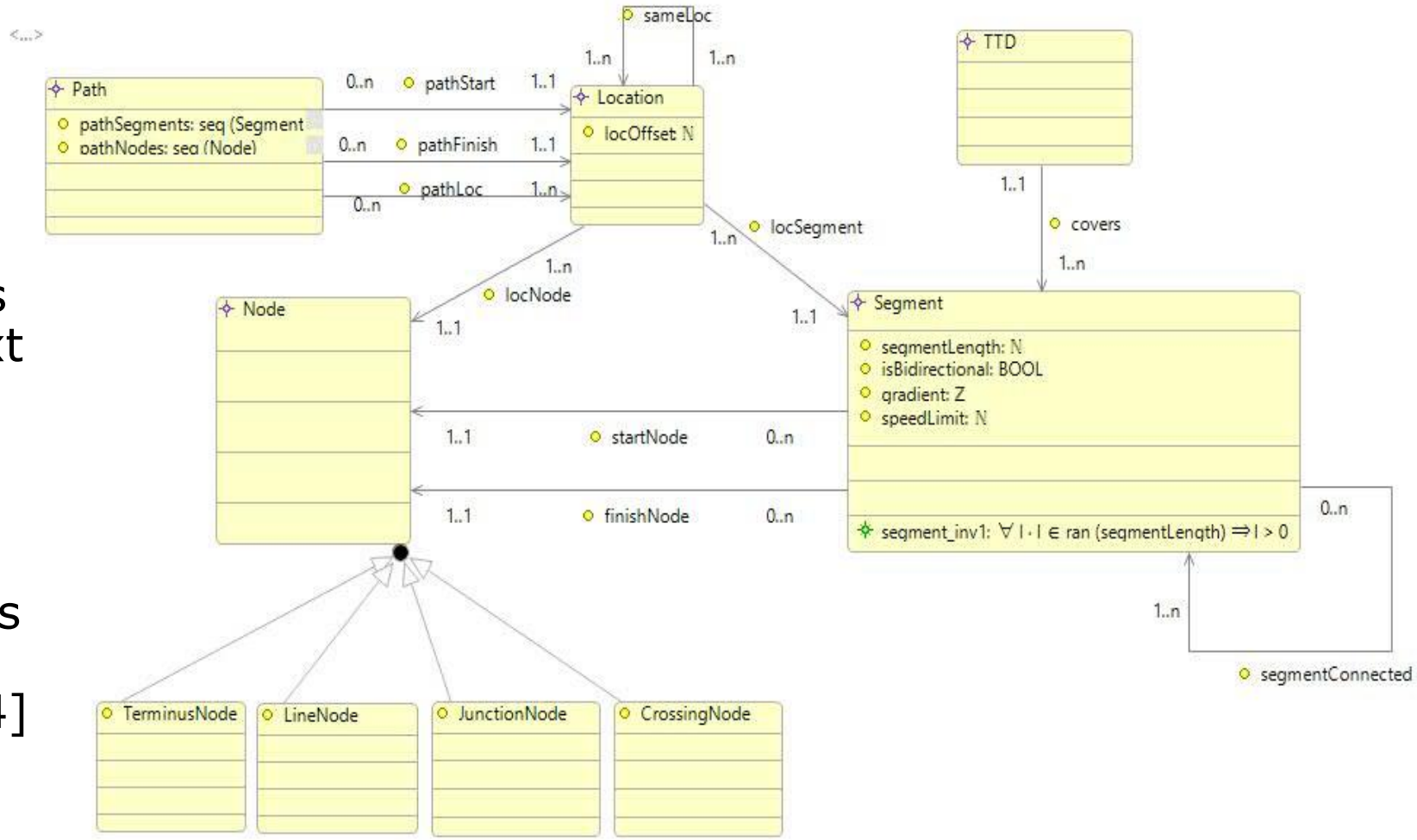
Model Overview: Track Layout UML-B Class Diagram

Application is key rail concept: configuring generic systems / products for a particular layout

Model includes a generic track layout: segment & node graph models track topology, gradients etc defined in an Event-B context

Graph can be instantiated for a specific rail scheme / project

[Aside: segment & node model is reused from Z spec we did for British Rail in 1994! See FME'94]

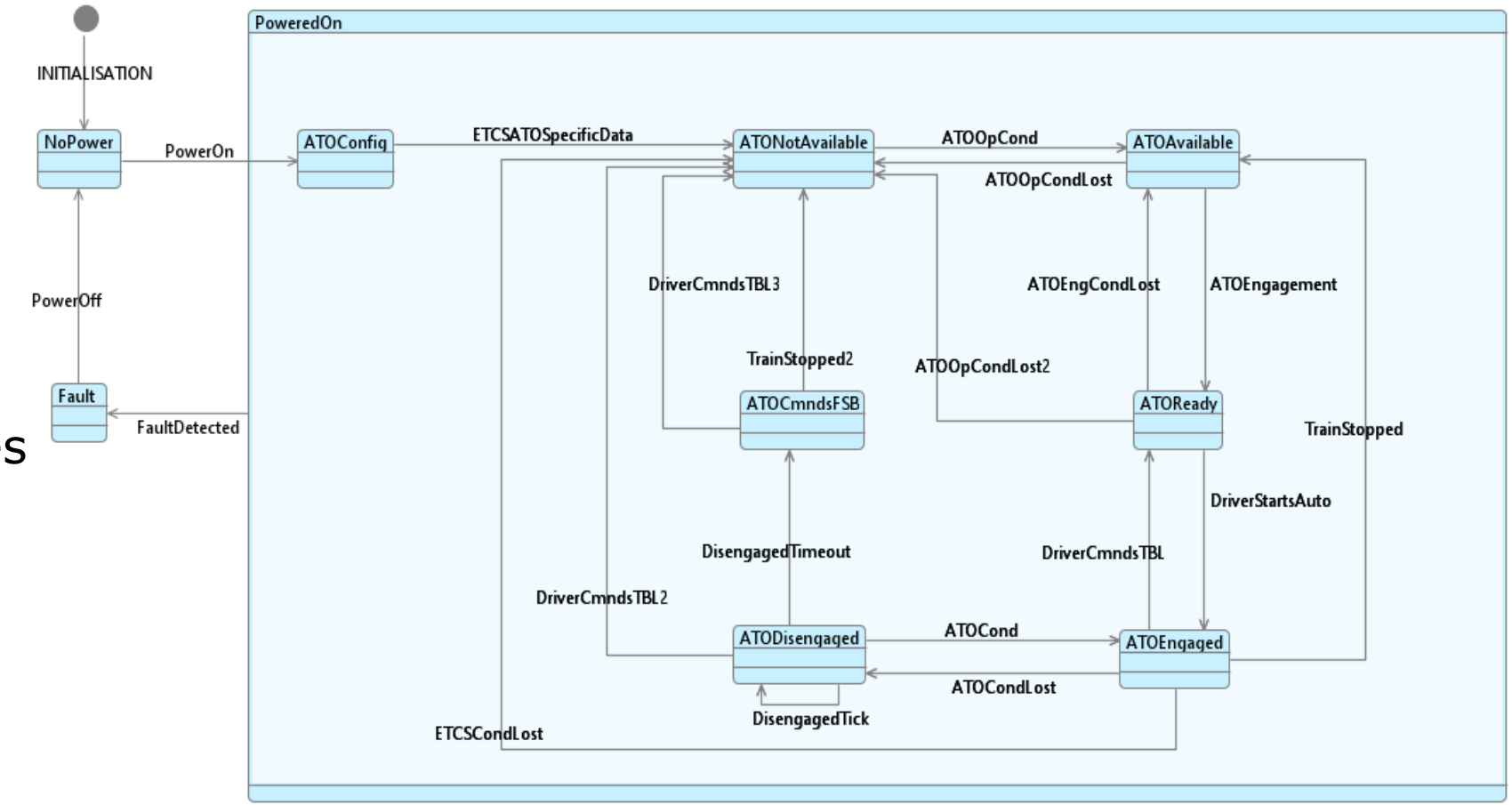


Model Overview: ATO UML-B State Machine

Automatic Train Operation (ATO) is autonomous driving

State machine covers: when ATO is available, driver engagement / deselection & various error cases

States are enumeration type values in Event-B model and machine transitions are names of Event-B events



Results: Aspects that worked well

Machine inclusion is vital to be able to structure a large model into digestible pieces

- Also enables subsequent independent refinement of subsystems (out of scope for case study)

Class diagrams are very useful for presenting data models that involve entities / objects

Similarly, state machines – where applicable – really aid understanding of event sequences

Autogenerated Event-B integrates well with user Event-B, e.g. invariants between classes

Theory plug-in enabled definition of key notation not in Event-B (e.g. generic sequences)

Proof tools generally worked well (as expected is labour & specialist skill intensive)

Animation is great for both validation (with SMEs) & verification (by specifiers) but …

Results: Challenges – proof & model checking tension

To fully specify behaviour to enable proof may require properties that cannot be evaluated

- Important as evaluation is a pre-requisite for all model checking analysis, including animation
- Property may be as simple as function totality (where it's impractical to list all function values)

Cannot evaluate the trackside train detection (TTD) topological connectedness property:

```
covers_contiguous:
∀ t, s1, s2 · t ∈ TTD ∧ s1 ≠ s2 ∧ {s1, s2} ⊆ covers [{t}]
  ⇒ (∃ ss · ss ∈ seq (Segment) ∧ ss ≠ ∅ ∧ s1 = ss (1) ∧ s2 = ss (card (ss))
      ∧ ss ∈ ℕ ⇸ covers [{t}]
      ∧ (∀ i · {i, i+1} ⊆ dom (ss) ⇒ ss(i) ↦ ss(i+1) ∈ segmentConnected))
// All the segments that a TTD covers are connected
```

We specify contexts/machines without such properties & include in proof-only versions

- illustration of how this works, in terms of model structure, is given with the security case study

Results: Challenges – valid model instantiations

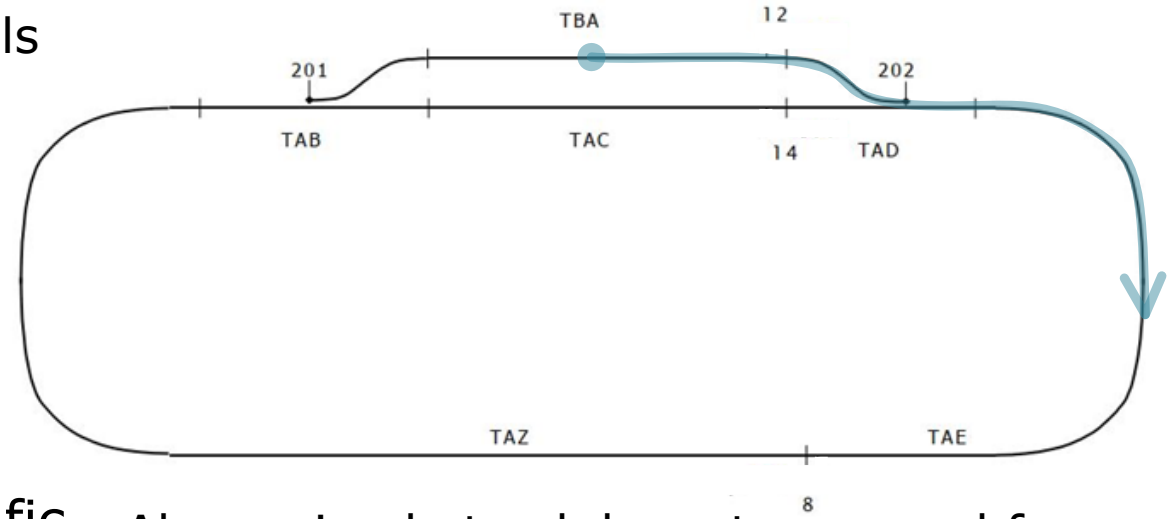
Defining an instantiation, e.g. for animation, can be difficult as it may be hard to:

- predict how much instantiation is needed for the tools to be able to evaluate
- understand from an error which parts of the model need to be further defined/constrained
- define instantiated values that satisfy all the model constraints/properties
- ensure the instantiation is rich enough to cover scenarios/cases needing analysis

Last 2 points are particularly domain/model specific

The track layout was intricate to get correct; many constants have complex relationships

- domain specific tooling would help, e.g. autogenerate instantiation from graphical track layout



Above simple track layout was used for a model instantiation, took several attempts to get all the detail correct

Results: Challenges – structuring complex events

Machine inclusion enables specifications of separate pieces to be combined into a whole

We think a natural specification structure is to have a machine per subsystem

- common for whole system level (atomic) events to involve changes in multiple subsystems
- we'd like to specify each subsystem change separately & combine to define whole system effect

However, Event-B limitations greatly constrain how different events can be combined:

- event synchronisation in machine inclusion is effectively parallel composition of all actions
- limited data flow between events; can use guards to equate parameters from different events

So cannot combine events where the action of one depends on the action of another

This reduces atomicity/increases fragmentation resulting in:

- an increased number of events
- more difficulty understanding how events fit together; what permitted sequencing of events is
- some desired invariants not holding, as they're only preserved once event sequence completes

Animation helps, but we'd much prefer to avoid the fragmentation

Security Case Study: Tokeneer

Security Case Study Description: Tokeneer

Demonstrator project for the US NSA; showed practical to achieve security std with formal methods

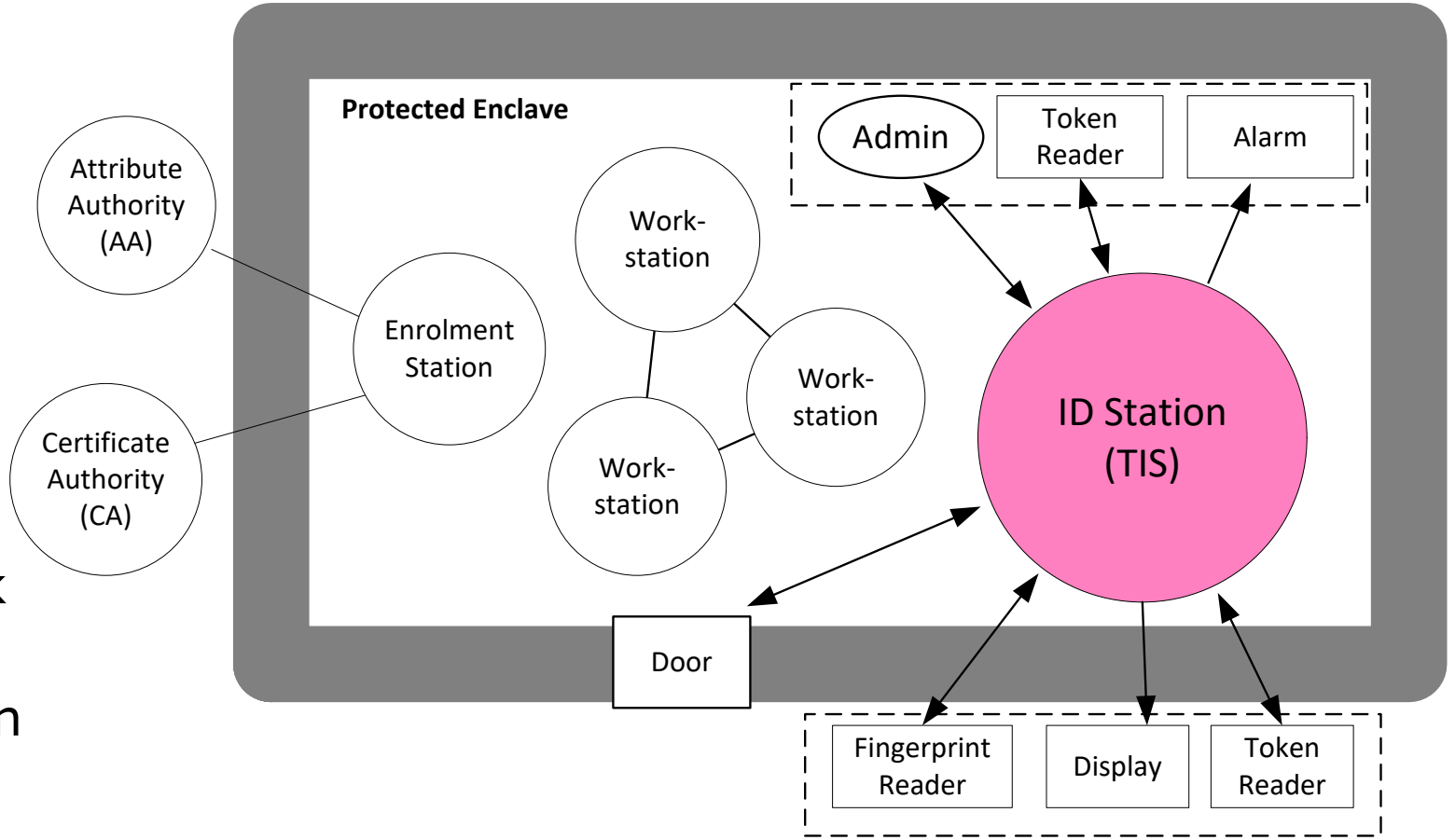
NSA open sourced the project in 2008; available to all researchers

SECT-AIR did (partial) Tokeneer Event-B model in 2017

HICLASS updated & expanded model 3.5 years later in 2021

Tokeneer has workstation network in a protected enclave

Access granted via biometric token & fingerprint reader

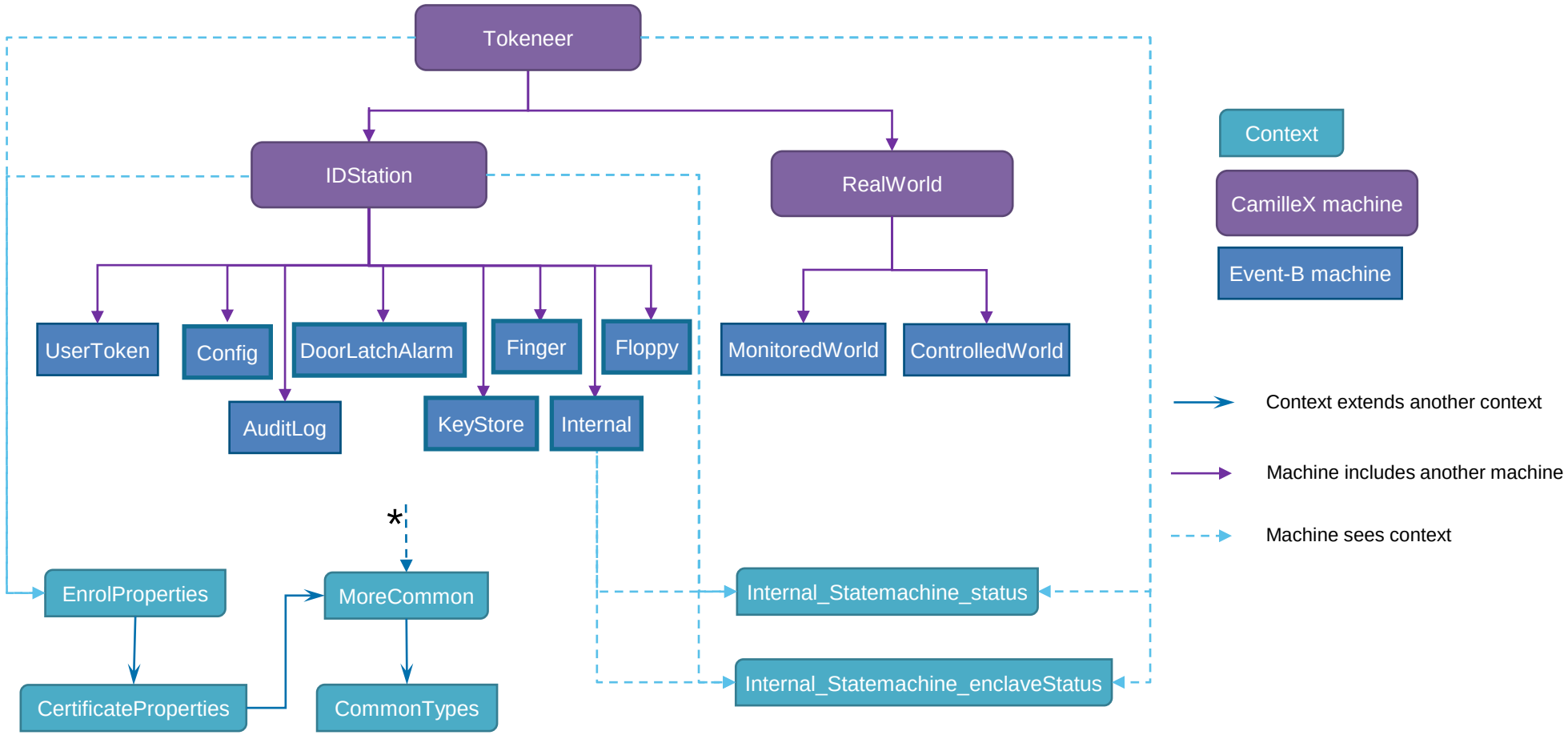


Model Overview: Structure

IDStation is the system to specify

- 12 subsystems
- 8 modelled for case study

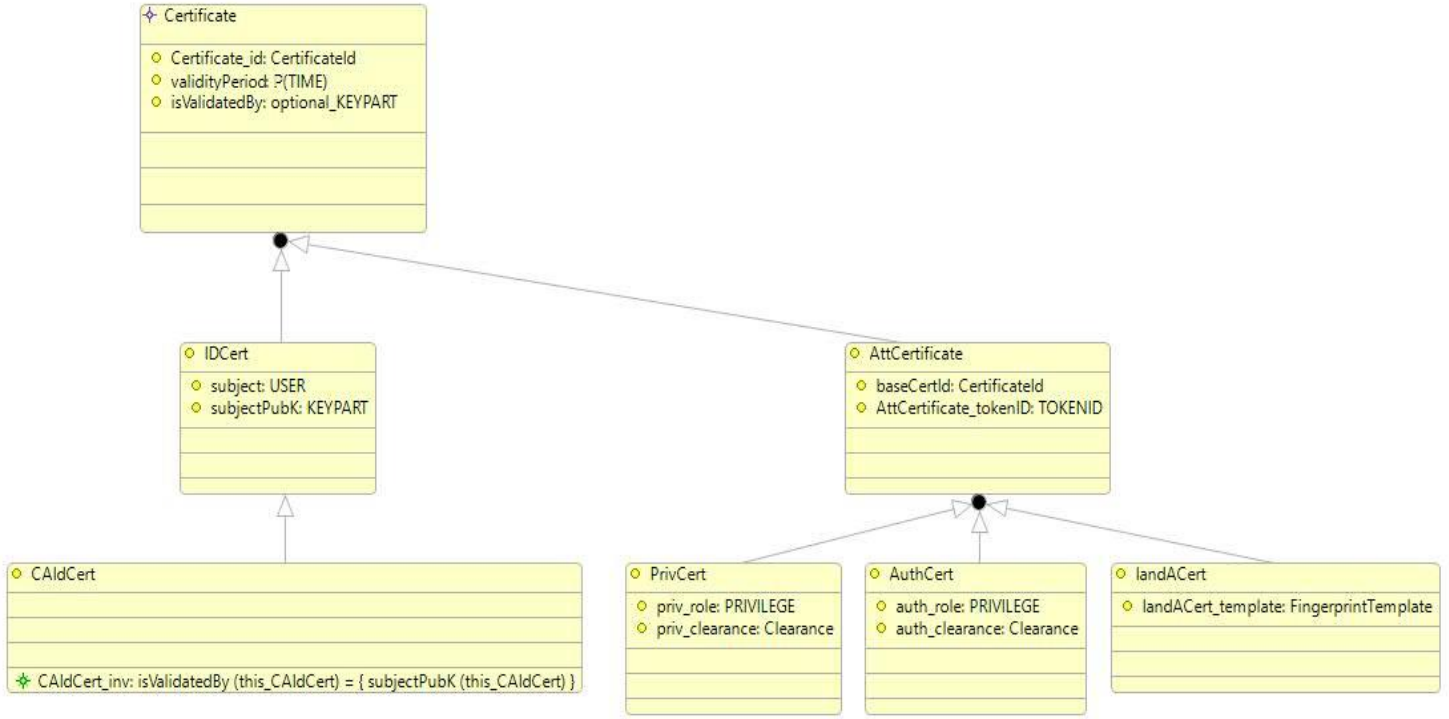
RealWorld models key environment inputs (sensors) & outputs (door latch and alarm)



Model Overview: Certificate Types UML-B Class Diagram

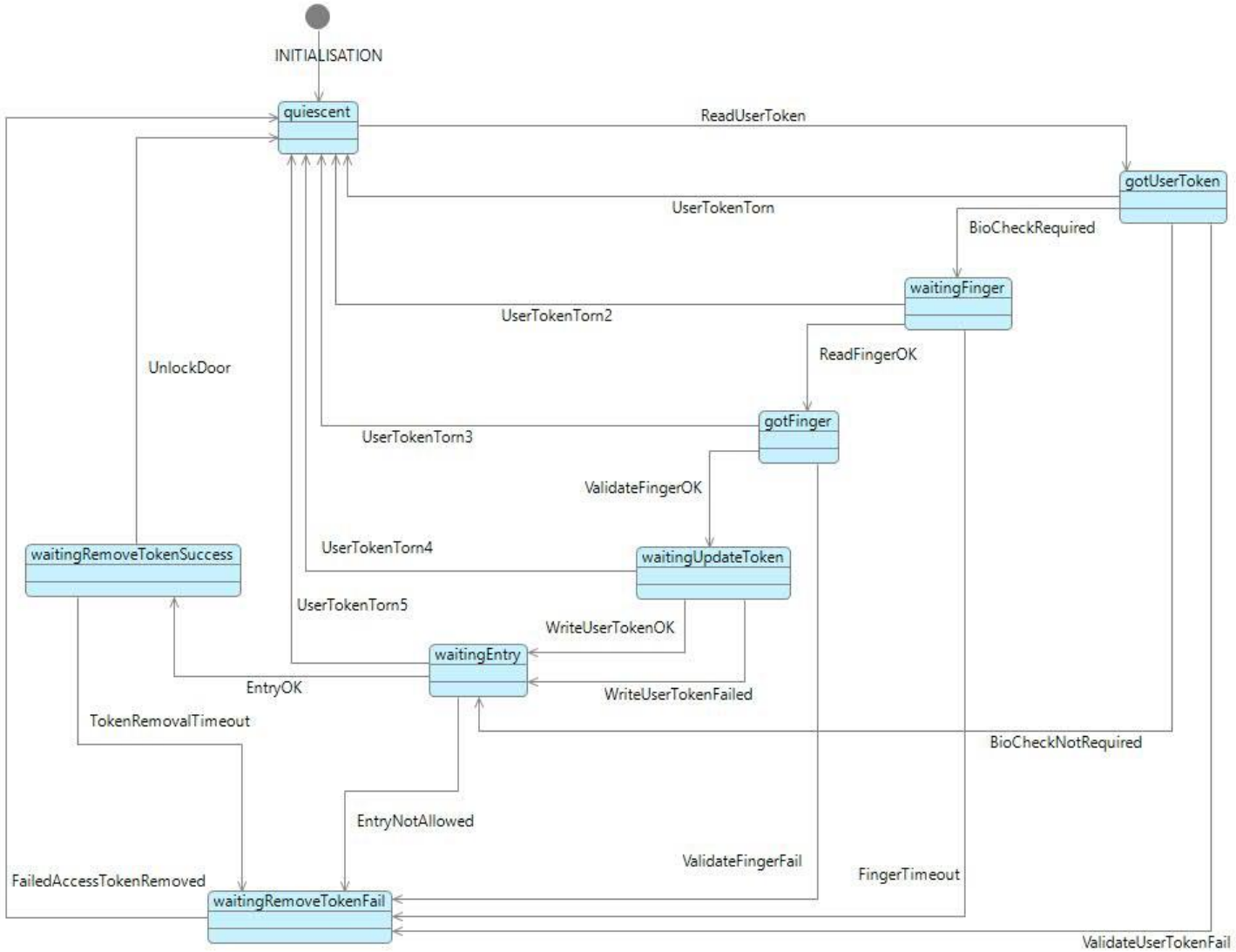
Tokeneer has quite a rich certificate model

Certificates identify users & define what users are authorised to do



Model Overview: User Entry status UML-B State Machine

Key state machine showing user entry process



Results: Aspects that worked well

Great deal of consistency in our experiences with both case studies:

All the positive points from the ERTMS case study apply equally to Tokeneer

2021 Tokeneer model is notably more successful than 2017 version, main reasons are:

- machine inclusion giving needed structure
- UML modelling also adding structure & making specification more accessible
- completing an animation of the model (2017 work struggled to find a useable instantiation)
- plug-in compatibility issues impacted work in 2017, much less of an issue in 2021
- more substantial proof work completed (due to needing less time working other issues)

Also true that the challenges affecting the ERTMS case study apply to Tokeneer as well

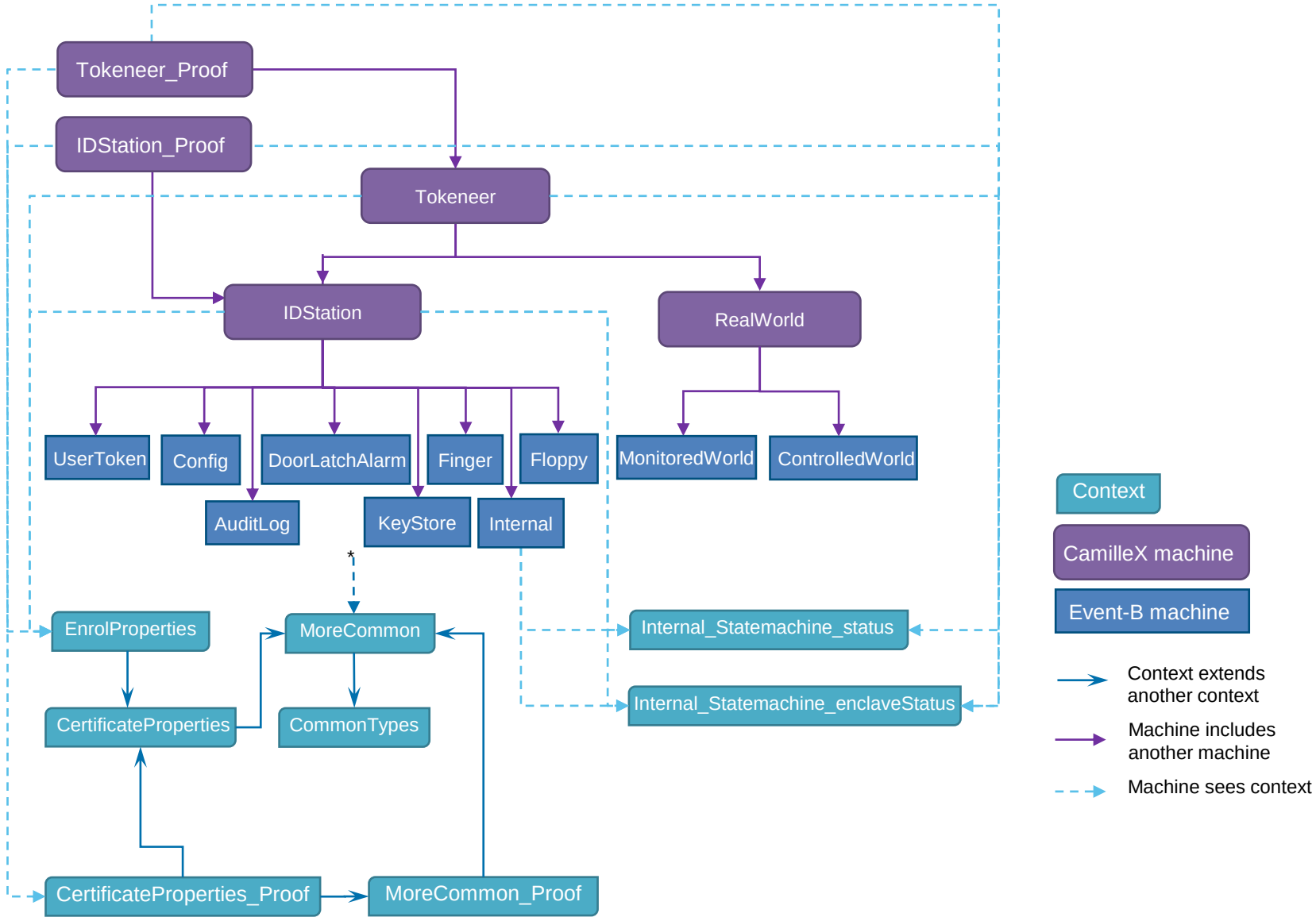
Results: Challenges – proof & model checking tension

Like ERTMS, Tokeneer has context elements too complex for model checking

- Time model within fn parameter

Figure shows extra contexts & machines needed to fully specify properties for proof

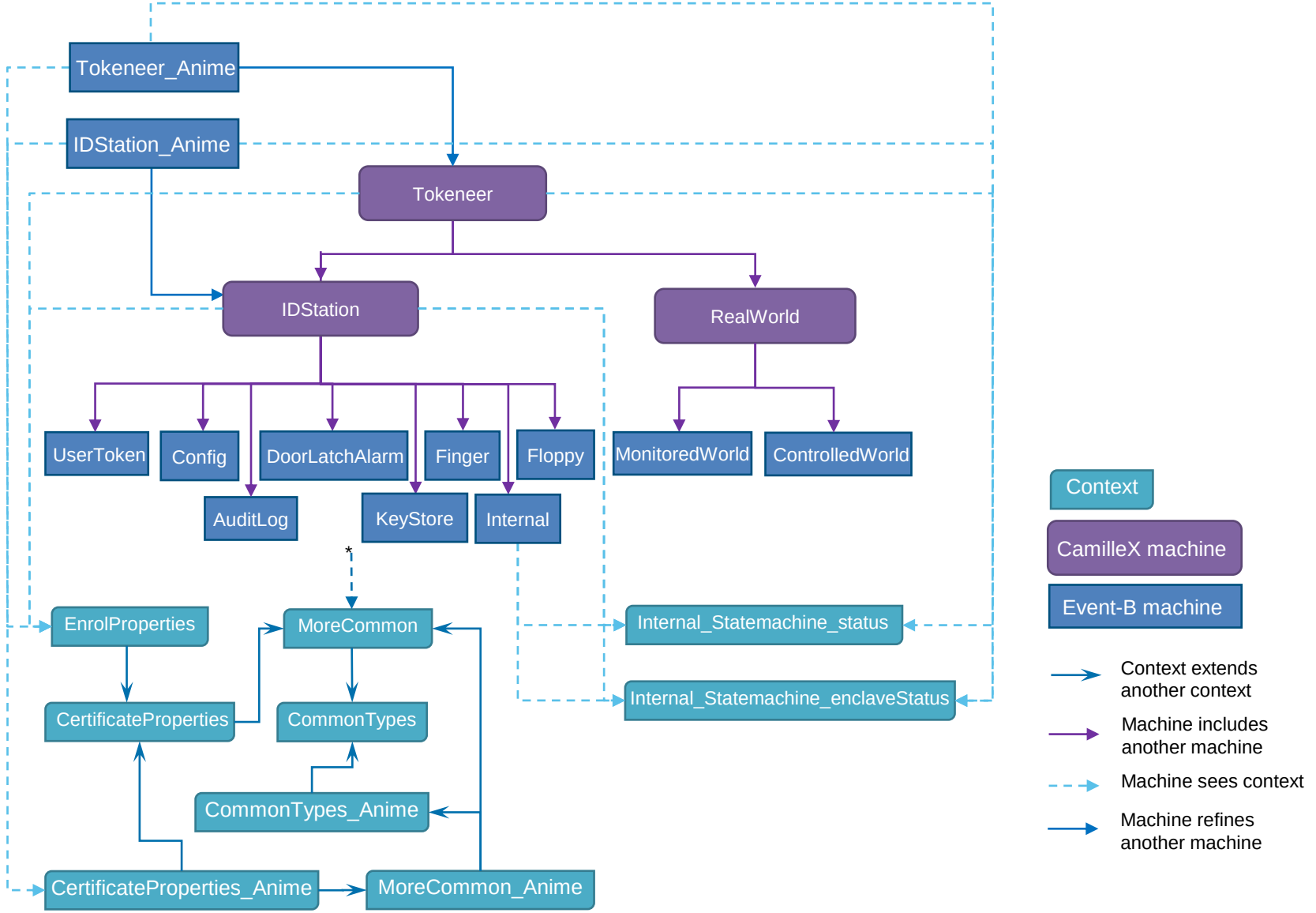
- names end '_Proof'
- proof contexts include only the missing properties
- proof machines see the proof contexts to generate models with all required constraints
- proof machines have to list events too (but using CamilleX 'synchronises' avoids need to duplicate event contents)



Results: Challenges – valid model instantiations

Figure shows the extra contexts & machines needed to define a model instantiation

- names end '_Anime' as main purpose is to support animation
- extra contexts simply extend those to be instantiated
- have anime versions of both Tokeneer & IDStation to allow animation at different levels
- the instantiated machines refine the original ones, to preserve behaviour, whilst also seeing the anime contexts
- so no duplication between original and anime models



Results: Challenges – valid model instantiations

Like ERTMS case study, defining a valid instantiation for Tokeneer proved tricky

- ProB plug-in error reporting could be unhelpful, but notably better (more specific) with ProB 2 UI

For Tokeneer it's relationships between certificate types and tokens that's complex

- as with track layout for ERTMS, domain/model specific tooling would address this

Is the fact that both case studies had this issue bad luck or indicative of general issue?!

- certainly true that both case studies have complex configuration data

Results: Challenges – class identity vs attribute values

One limitation with the Event-B model iUML-B generates for classes is a lack of identification between a class instance and the instance's attribute values

There are at least a couple of different ways of doing this:

- Value semantics: two class instances are identical if their attributes have the same values, so the identity of an instance is equivalent to the combination of all the values of the attributes of that instance
- Reference semantics: each instance has an identity and different instances, with different identity values, can still have all the same attribute values

Capturing this semantic equivalence was needed in Tokeneer to ensure all the desired properties are provable

- UML-B's flexibility allowed us to add the missing properties manually, would be nice to automate
- we use value semantics – to match original Tokeneer Z spec – although where needed Z spec includes explicit identity attributes within classes (modelled in Z as schema types)

Results: Challenges – property verification model checking

ProB supports LTL and CTL (but was not yet in ProB 2 UI), useful for liveness properties

- we think CTL likely to be more useful for the types of specs we write

For the user entry state machine want to show states are reachable

- as no CTL in ProB 2 we wrote LTL: `G ({in_status /= gotUserToken})` expecting a counterexample
- initially model checking gave no results
- with a restructuring of one event some properties (including above one) were correctly analysed
- later instantiation change (slightly larger set) reverted model checking to being unsuccessful
- counterexample for above property needs event sequence of length 6 (including initialisation)

Clearly model checking properties is fragile to how spec written (combinatorial explosion?)

State machine was fully animated with same instantiation tried for property verification

Results: Challenges – test generation

Had more success with test case generation
- on right is 7 step event sequence that covers several user entry state machine transitions
- first 5 steps form a counterexample to LTL property on previous slide
- again used same common instantiation

Not clear why LTL failed but test succeeded?

Test cases requiring longer event sequences were not possible to generate
- guess due to exponential search space growth

Test case auto-generation is highly desirable
- takes a lot of manual effort

```
<test_case id="1" targets="[ReadUserToken]">
  <global>
    <step id="1" name="INITIALISATION">
    <value name="fi_currentFinger">fptry1</value>
    <value name="SCREENTEXT_value"><set><pair>…
  </step>
    <step id="2" name="PollFloppy">
    <value name="fl_floppy">floppy2</value>
  </step>
    <step id="3" name="ReadEnrolmentFloppy" />
    <step id="4" name="ValidateEnrolmentDataOK">
    <value name="ks_enrolId">enrolId2</value>
  </step>
    <step id="5" name="PollUserToken">
    <value name="ut_userToken">tokEnt4</value>
  </step>
    <step id="6" name="ReadUserToken" />
    <step id="7" name="BioCheckRequired" />
  </global>
</test_case>
```


Results: Challenges – detailed notational issues

Some notational features of Event-B require more work than we'd like, for example:

No specific enumerated type syntax

- yes can model using partition, but can be bulky & requires more work by both specifier & reader

Sequences and optionality

- concept of ordering (sequences) is used frequently, as is optionality
- Theory plug-in allowed us to address this (with potential risk of tool/plug-in compatibility issues)

Derived state – i.e. state whose value is determined by invariant

- Event-B's use of actions for dynamic behaviour means any change must be explicitly stated
- however, derived state can be very useful to simplify understanding and avoid the need to repeatedly state the same derivation
- e.g. Tokeneer Z spec invariants derive state variables *currentLatch* and *doorAlarm*, current latch and alarm status are referred to in various places
- in Event-B we can state the invariants, but have to write explicit actions too for any updates (at least proof support means that if we get any updates wrong then there'll be a proof failure)

Summary of Language & Tooling Challenges

Challenges: Language

We like black-box specifications where all the system behaviour resulting from a single environmental stimulus can be specified as an atomic event / operation / set of actions

Our impression is a lack of structure for large / complex events forces event fragmentation

- so whole system behaviour for single stimulus gets split into multiple events (unless specified in a monolithic single machine)

Also some more detailed notational issues

- such as derived state

Challenges: Scalability

As we discuss in our ABZ 2018 paper, a major issue for formal specification of large/complex systems is scalability

- Very large, successful formal specs are possible (e.g. iFACTS Z spec, see ABZ'18), but only tools we've found workable at that scale are document generation & type-checking

Both our ERTMS and Tokeneer case studies were successful, but we are unsure if similar results are achievable with full specifications (especially for ERTMS)

Machine inclusion expands to 1 flattened Event-B machine; possible tool resource issues:

- Any limits on number of variables, invariants or events in 1 machine?
- Checking instantiations prior to animation
- Evaluating event guards to determine event availability in a given state
- Number and/or size of instantiations needed to cover all scenarios needing analysis
- Event sequence combinatorial explosion for model checking

Refinement meant to manage scalability but key properties may rely on less abstract detail

- especially in a system of systems context

Challenges: Test generation

Combinatorial explosion of possible event sequences inevitably limits sequence lengths that are practical to analyse / model check

To break this limitation, why not allow user to specify point to start searching from?

- user could specify initial event sequence (& parameter values)
- tool would then search from state resulting from applying user input

Such an approach could give full test coverage for the Tokeneer user entry state machine

- tool can already cover first few states
- feeding that back as initial sequence(s) to tool would enable coverage of next few states etc

Challenges: Instantiation

Instantiation clearly vital to unlock potential of Event-B

- as pre-requisite for animation, property verification, test generation etc

We have limited experience generating instantiations for animation, model checking etc

- much to learn about how much instantiation to define & impact of spec styles on ease of doing it
- can others' experience be captured as guidance?

Was time consuming and error prone to do

- model specific tooling would help, such as converting track layouts to model sets
- not sure whether there's potential for more generic tooling / framework?

Think tooling could assist with managing/generating multiple context/machine versions

- especially with proof versions too (if certain properties/definitions too difficult for evaluation)

Q&A

About Capgemini

Capgemini is a global leader in partnering with companies to transform and manage their business by harnessing the power of technology. The Group is guided everyday by its purpose of unleashing human energy through technology for an inclusive and sustainable future. It is a responsible and diverse organization of 270,000 team members in nearly 50 countries. With its strong 50 year heritage and deep industry expertise, Capgemini is trusted by its clients to address the entire breadth of their business needs, from strategy and design to operations, fuelled by the fast evolving and innovative world of cloud, data, AI, connectivity, software, digital engineering and platforms. The Group reported in 2020 global revenues of €16 billion.

Get the Future You Want | www.capgemini.com

Jonathan Hammond

High Integrity Software Expertise Centre
Capgemini UK
22 St Lawrence Street
Bath
BA1 1AN

jonathan.hammond@capgemini.com