Universität Bremen

Verified

The Future of Modelling Languages in Industry

Why Practitioners do not Use Your Favourite Process Algebra

Jan Peleska – University of Bremen and Verified Systems International – peleska@uni-bremen.de

αFM 2021

2021-11-23

HiDyVe

Acknowledgements

This work was partially supported in part by the German Ministry of Economics, Project "HiDyVe – Highly Dynamic Virtual and Hybrid Validation and Verification" under grant agreement 20X1908E.

While I am solely responsible for the content of this presentation, I am grateful to several experts in the field who have significantly influenced my view on Formal Methods in general and modelling formalisms in particular.

I sincerely apologise to all of you that – despite the brilliant input from your side – I still believe that we are not "already there", as far as modelling formalisms are concerned

Jean-Raymond Abrial, Dines Bjorner, Ana Cavalcanti, Hubert Garavel, Mario Gleirscher, Anne E. Haxthausen, C. A. R. Hoare, Wen-ling Huang, Alexander Knapp, Hans Langmaack, Peter Gorm Larsen, Mohammad Reza Mousavi, Amir Pnueli, Jaco van de Pol, Markus Roggenbach, Bill Roscoe, Uwe Schulze, Jim Woodcock

Objectives

Objectives

in accordance with the Manifesto for Applicable Formal Methods . . .

. . . this talk focusses on one of the most serious show stoppers related to application for formal methods in practice

The lack of widely accepted modelling formalisms

I do not think that there should be one universal formalism, but their number should be as “small” as that of currently accepted programming languages (Java, C/C++, C#, Python, …), so that each formalism would have its well-defined application domain and a reasonably large group of “followers”

My main claim is that

Now is a good time to invest you energy into a new modelling formalism

Overview

Overview

Three main parts in this talk

Part I. What's wrong with current modelling formalisms ?

I try to explain why both the FM communities and industry have failed to come up with universally accepted modelling formalisms

Part II. Ingredients of a successful modelling formalism

We need more than just a clever language design …

Part III. How to approach this

I advertise something that I did not invent myself : SysML V2, and explain how it could become a real success

Part I.
What’s wrong with current modelling formalisms?

What’s Wrong With Current Modelling Formalisms?

Why should you trust my assessment ?

- I started to apply formal modelling languages (Z, CSP) in 1984, since the complexity of certain software projects at Philips suggested that textual informal specifications would lead to unmanageable software
- I applied Structured Analysis, Z, CSP, UML, SysML, MATLAB/Simulink/Stateflow, SCADE, Astree (abstract interpretation) in real-world projects for Philips, Siemens, Airbus, Daimler, Hella and others
- I worked with VDM, CCS, CirCus, RSL, B, Isabelle for research purposes
- Our company Verified Systems International founded in 1998 is specialised on V&V of safety-critical systems, and we offer application of formal methods as a service for
  - model-based testing
  - bug finding
  - protocol verification
- I have programmed core components of Verified Systems’ flagship tool RT-Tester and of the open source MBT library libfsmtest

What's Wrong With Current Modelling Formalisms?

Common problems of FM-based and industrial formalisms

- Current **modelling tools cannot compete with programming IDE's** (Xcode, IntelliJ IDEA, Eclipse), because programming IDEs
  - provide on-the-fly syntax checking while you program
  - offer very effective lookup functions for existing types and operations
  - have powerful static analysers uncovering potential runtime errors
  - allow for immediate test and execution without leaving the IDE
- With today's tool support
  - **programming is fun, while modelling is painful**

What my Xcode static analyser for C/C++ can do for me …

Similar or even more impressive capabilities are available for Java with IDEs IntelliJ IDEA or Eclipse

```
std::unique_ptr<Fsm> RandomCompletelySpecifiedFsm::createFsm() {
    random_device rd;
    default_random_engine gen(rd());
    uniform_int_distribution<size_t> distributeOutputs(0, numberOfOutputs - 1);
    uniform_int_distribution<size_t> distributeInputs(0, numberOfInputs - 1);
    uniform_int_distribution<size_t> distributeStates(0, numberOfStates - 1);
    uniform_int_distribution<size_t> distribution(0, 1);

    States states;
    map<Index, bool> reachable;

    states.reserve(numberOfStates);
    for (Index i = 0; i < numberOfStates; ++i) {
        states.emplace_back(to_string(i), i);
        reachable.emplace(i, false);
    }

    // initial state is always reachable
    reachable[0] = true;

    // We create transitions by starting from reachable nodes
    // and trying to reach at least one not reachable node from there.
    deque<Index> queue;
    queue.push_back(0);

    while ( not queue.empty() ) {

        Index source = queue.front();
        queue.pop_front();

        // Select a random state
        Index current = distributeStates(gen);
        // Store index of first selected state
        Index start = current;
        // Prepare the target state for the transition
        Index target = 0;

        // try to find a non-reachable state, otherwise use the starting state
        do {
            if ( !reachable[current] ) {
                // The picked state is not reachable, so use it as target
                target = current;
                reachable[target] = true;
                queue.push_back(target);
                break;
            } else {
                // Try the next state
                current = (current + 1) % (numberOfStates);
            }
```

2 2. Loop body executed 0 times

2 3. Assuming the condition is true

5. Assuming the condition is false

6. Division by zero

What’s Wrong With Current Modelling Formalisms?

Common problems of FM-based and industrial formalisms

- For current modelling languages, **action / expression languages are too weak**, when compared with Java and C++ (think of pipes, lambda expressions, operator overloading, address arithmetic…)
  - Why should you learn a restrictive action language when you are already a Java/C++ expert?
  - Nobody will accept, for example, the *Action Language for Foundational UML (Alf)* – see http://www.omg.org/spec/ALF/1.1
  - Why do modelling formalisms not come with an (optional) memory model. just like standardised programming languages?
    - As one consequence, models are rarely used for developing operating system code or driver code

What's Wrong With Current Modelling Formalisms?

The problem with UML-style syntax

- The focus on graphical notation as favoured by UML tools (though a textual abstract syntax exists) has turned out to reduce productivity for several reasons
  - Instead of expressing the desired system structure and behaviour in the most effective way, system designers are forced to deal with drawing problems
  - In particular, re-factoring textual code is much simpler than re-factoring graphical diagrams
  - Graphical representations are inadequate for
    - Very large numbers of interacting components
    - Dynamic component creation/deletion
    - Mobile processes with changing communication links

What’s Wrong With Current Modelling Formalisms?

Common problems of FM-based and industrial formalisms

- **Modelling without additional tool support is not worth the effort!**
- Most important: efficient **code generator** for simulation and on-target execution

  “If you can't execute it, it's not worth anything.”
  [A system designer from Astrium – now Airbus Defence and Space]

- Further important tool components
  - On-the-fly syntax checks while modelling
  - Static analyser
  - Verification support: testing – simulation – model checking – theorem proving
  - Requirements tracing

What's Wrong With Current Modelling Formalisms?

FM-based formalisms

- Formal modelling languages have **insufficient support for object orientation** – because
  - static process networks semantics is well understood, whereas
  - dynamic object creation, destruction, inheritance, and polymorphism are extremely hard to capture in a formal semantics
- **But** formal treatment of object orientation cannot be avoided, since
  - **increasingly complex systems** require OO modelling to cope with size and complexity
  - **collaborative autonomous systems and mobile processes** require changing configurations and dynamic re-allocation of their communication channels
  - **skilled programmers are used to apply OO concepts** in C++, C#, Java, Python – why should they refrain from using them in modelling?

What's Wrong With Current Modelling Formalisms?

FM-based modelling formalisms – the problem with mathematical syntax

Mathematical syntax (as used in Z, VDM, B, CirCus,…) is too far away from programming syntax

$$\textit{BirthdayBook}$$

$$\textit{known} : \mathbb{P}\,\textit{NAME}$$
$$\textit{birthday} : \textit{NAME} \pfun \textit{DATE}$$

$$\textit{known} = \mathrm{dom}\,\textit{birthday}$$

Let's compare Z and Java

```
public class BirthdayBook {
    public Set<BBName> known = new HashSet<BBName>();
    public Map<BBName, BBDate> birthday = new HashMap<BBName, BBDate>();
    void assertInvariant() {
        if ( ! known.equals(birthday.keySet()) )
            throw new RuntimeException("BirthdayBook invariant violated");
}}
```

What's Wrong With Current Modelling Formalisms?

FM-based modelling formalisms – the problem with mathematical syntax

Mathematical syntax (as used in Z, VDM, B, CirCus,…) is too far away from programming syntax

BirthdayBook

$known : \mathbb{P}\ NAME$

$birthday : NAME \pfun DATE$

$known = \mathrm{dom}\ birthday$

But the
expressive power of modern
programming languages is just as good – or
even better than that of formal
modelling languages

```
public class BirthdayBook {
    public Set<BBName> known = new HashSet<BBName>();
    public Map<BBName, BBDate> birthday = new HashMap<BBName, BBDate>();
    void assertInvariant() {
        if ( ! known.equals(birthday.keySet()) )
            throw new RuntimeException("BirthdayBook invariant violated");
}}
```

What's Wrong With Current Modelling Formalisms?

FM-based modelling formalisms – the problem with mathematical syntax

It seems that the focus on mathematical notation instead of programming style notation resulted in FM communities and programming communities drifting away from each other

```
void addBirthday(BBName newName, BBDate date) {
    if ( ! known.contains(newName) ) {
        known.add(newName);
        birthday.put(newName, date);
        assertInvariant();}}
```

AddBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$

What’s Wrong With Current Modelling Formalisms?

FM-based modelling formalisms – the problem with mathematical syntax

It seems that the focus on mathematical notation instead of programming style notation resulted in FM communities and programming communities drifting away from each other

```
void addBirthday(BBName newName, BBDate date) {
   if ( ! known.contains(newName) ) {
      known.add(newName);
      birthday.put(newName, date);
      assertInvariant();}}
```

AddBirthday

$\Delta \mathit{BirthdayBook}$

$\mathit{name?} : \mathit{NAME}$

$\mathit{date?} : \mathit{DATE}$

$\mathit{name?} \notin \mathit{known}$

$\mathit{birthday}' = \mathit{birthday} \cup \{\mathit{name?} \mapsto \mathit{date?}\}$

… and this is already executable …

What’s Wrong With Current Modelling Formalisms?

FM-based modelling formalisms – the problem with mathematical syntax

It seems that the focus on mathematical notation instead of programming style notation resulted in FM communities and programming communities drifting away from each other

```
void addBirthday(BBName newName, BBDate date) {
   if ( ! known.contains(newName) ) {
      known.add(newName);
      birthday.put(newName, date);
      assertInvariant();}}
```

AddBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$

… whereas that is not!

What’s Wrong With Current Modelling Formalisms?

FM-based formalisms

- Some modelling paradigms considered to be essential by the FM community are not ensured / do not even occur in the wilderness of real-world system development
  - Synchronous unbuffered communication
  - Full compositionality
- This complicates code generation and endangers the trustworthiness of verification results

What’s Wrong With Current Modelling Formalisms?

FM-based formalisms – verification

- **Model checking often fails** (even bounded MC, k-induction, …), because
  - the model is too complex
  - OO-aspects or certain syntactic features are not supported by the model checker
- As a consequence, no verification result at all is obtained
- In contrast to this, **coverage-guided fuzz testing** just requires a code generator, and you can even test for temporal properties specified in LTL by creating observer code added to the model code
  - Observer fails if formula is fulfilled / violated by the model code
  - Fuzz tester (e.g. Libfuzzer in LLVM/clang) records data leading to this failure

Part II.
Ingredients of a successful modelling formalism

Four Success Factors

Non of these may be disregarded

Since the 1970s, we have seen modelling formalisms come and fail or at least turn out be not quite as successful as we hoped them to be.

We have understood the reasons for failure – now is the time to exploit this experience and create a formalism that can compete in its popularity with C++, C#, Java, Python

Part III.
An Approach to Create and Market a Novel Modelling Formalism

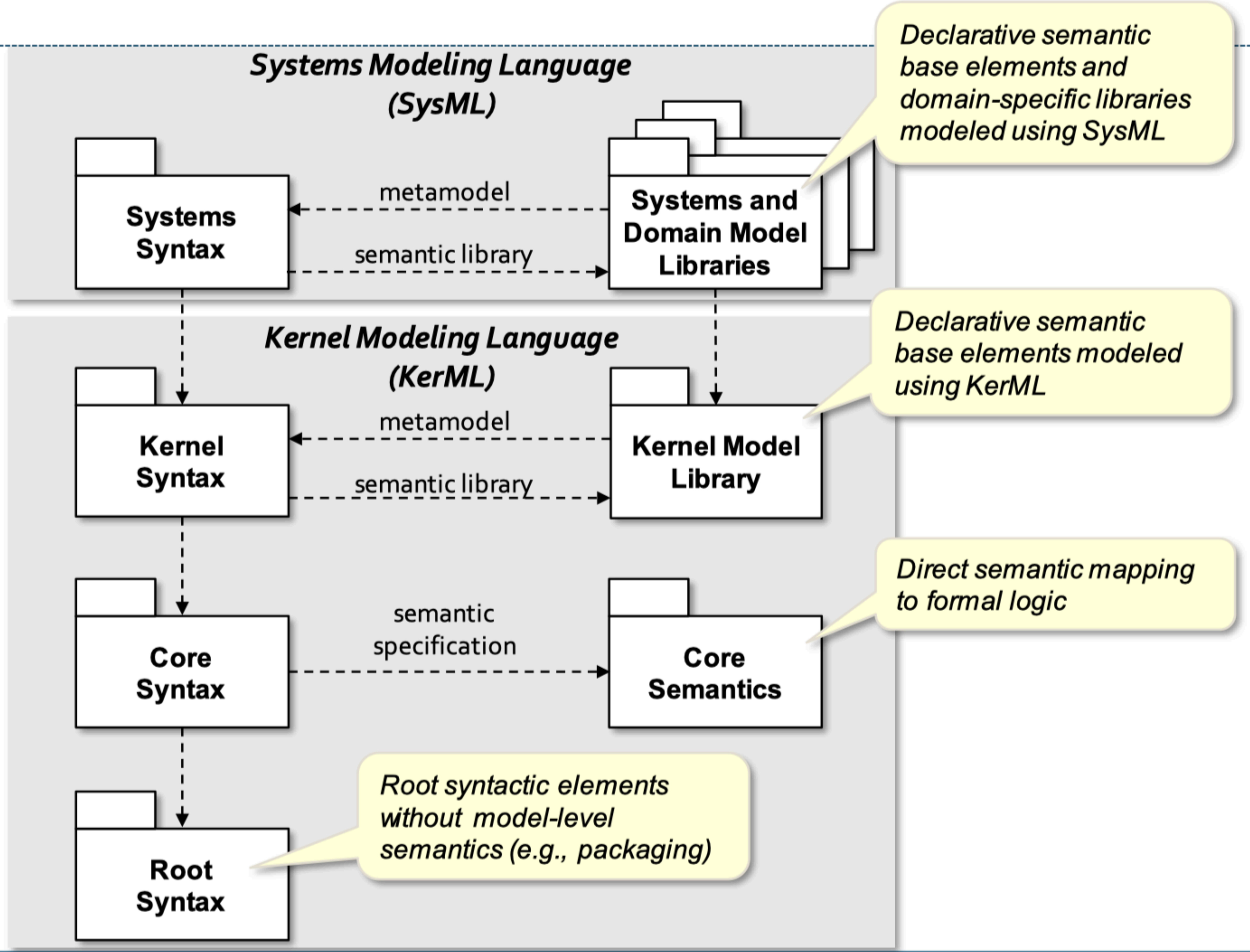
Four Success Factors

Language Design

- My suggestion: **support the novel SysML V2 development**, because SysML V2
  - is a wide spectrum language with a well-defined, extensible structure
  - can be alternatively used as textual or graphical language
  - has all desirable features for large industrial developments – just as UML, but significantly easier to understand and improved
    - Package and library structuring mechanism, parameterised, formalised requirements specifications, requirements tracing support, behavioural, structural, non-functional specification elements, full object orientation support, . . .
  - has a simplified semantics which is independent on the UML meta model

SysML v2 Language Architecture

SST



Requirement Definitions (1)

SST

A *requirement definition* is a special kind of constraint definition.

A textual statement of the requirement can be given as a documentation comment in the requirement definition body.

```
requirement def MassLimitationRequirement {
   doc /* The actual mass shall be less than or equal
    * to the required mass. */

   attribute massActual : MassValue;
   attribute massReqd : MassValue;

   require constraint { massActual <= massReqd }
}
```

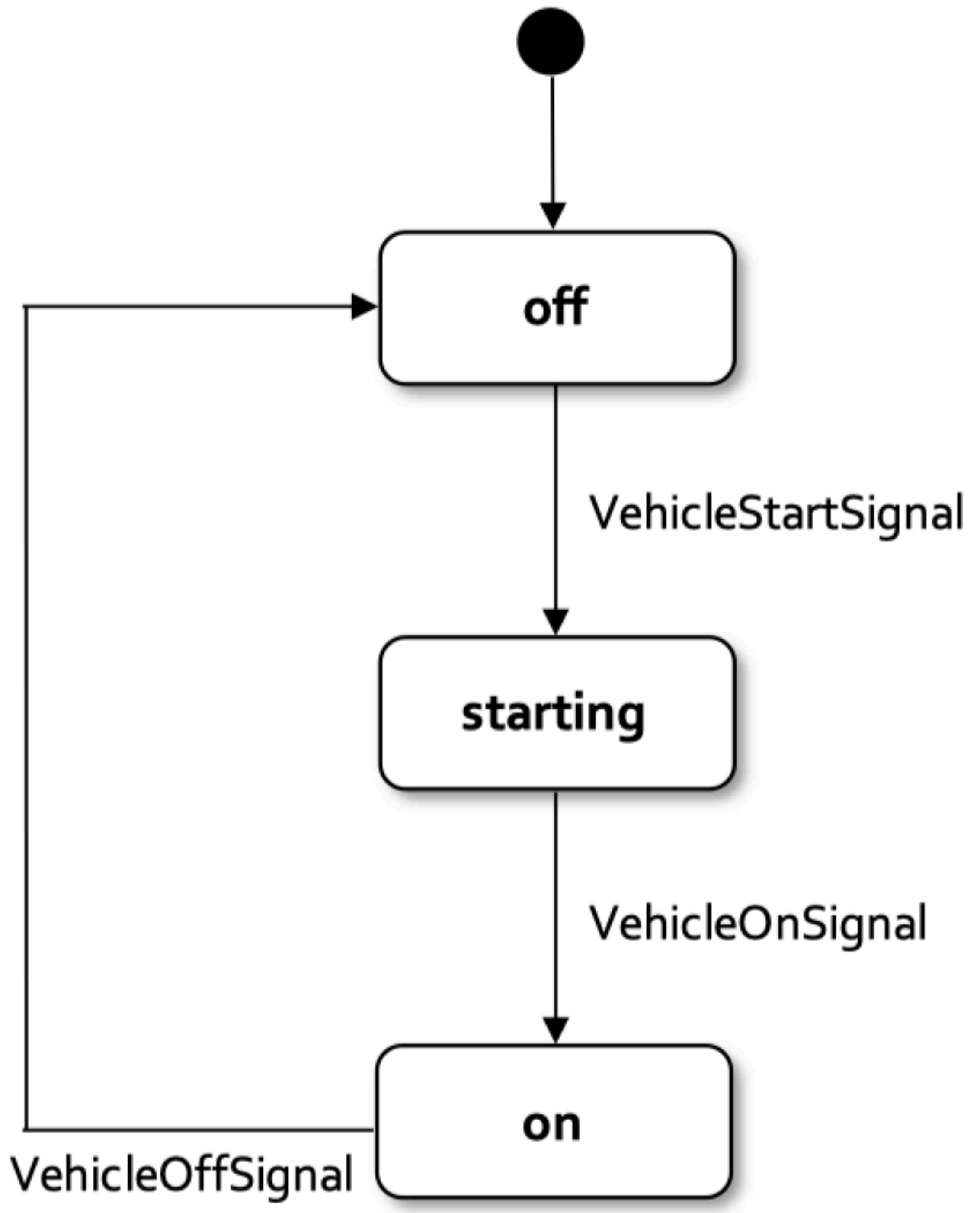
Like a constraint definition, a requirement definition can be parameterized using features.

The requirement can be formalized by giving one or more component *required constraints*.

State Definitions (1)

SST

A *state definition* is like a state machine in UML and SysML v1. It defines a behavioral state that can be exhibited by a system.




```
state def VehicleStates {
   entry; then off;

   state off;

   transition off_to_starting
      first off
      accept VehicleStartSignal
      then starting;

   state starting;

   transition starting_to_on
      first starting
      accept VehicleOnSignal
      then on;

   state on;

   transition on_to_off
      first on
      accept VehicleOffSignal
      then off;
}
```

This indicates the the initial state after entry is "off".

A state definition can specify a set of discrete nested *states*.

States are connected by *transitions* that fire on acceptance of item transfers (like accept actions).

Four Success Factors

Missing features in SysML V2

- **A powerful action language**
  - Currently, the Action Language for Foundation UML (Alf) has been selected to specify operation bodies, constraints, guard conditions, actions etc.
  - We expect that Alf will not be accepted by software developers who are experts for C++ or Java
  - Could we use C++ as action language for SysML V2 ?
- **Operational behavioural semantics suitable for OO designs**
  - Apply slicing techniques to develop “partial understanding” of complex models
- **Incremental modelling** with refinement relations similar to **ioco**
  - This allows to compare incomplete reference models with incomplete refinements

Four Success Factors

Missing features in SysML V2

- **Added communication concepts** for signals – synchronous CSP-style communication ?
- A **runtime environment** (similar to JVM) where all behavioural language features are efficiently implemented ?
- **Code generators** for runtime environment that enforce compositionality by adding protection for critical sections ?
- Proof support
- Model checking Support, MBT support

Four Success Factors

Business model for industry

- **Open access language standard** – e.g. distributed by OMG
- **Open source, free tool core** comprising
  - Basic modelling
  - Static semantics checker
  - Model simulation – code generator for standard platform (Linux, Windows, MacOS)
  - Model verification support
  - Model-based testing – MiL, SiL

Four Success Factors

Business model for industry

- Tool core maintained by community (similar to Linux or Eclipse Foundation)
- Commercial distribution of maintained baselines (similar to Linux)
- Tool vendors can create added value components, e.g.
  - Optimised code generators for specific runtime environments or embedded target platforms
  - HiL testers

Four Success Factors

Business model for academia

- Finding solutions for the missing features listed above would lead to significant academic success!
- More difficult: create an academic environment where you can be successful even when supporting inventions of others

Four Success Factors

Supporting community

- Joint venture of industry and academia – large support community created by joint dissemination strategy
  - SysML V2 has core team of 160 individuals from academia, industry, government organisations
- Avoidance of factions in academia
  - Could the FM community agree that this is the best chance to gain the impact that has been missing so far ?
- Research topics would focus on added value, improved V&V support etc., but **not** on alternative language designs

Conclusion

Conclusion
Summary

- Several important deficiencies of existing modelling formalisms have been identified and explained
- The need for a new modelling formalism has been motivated
- A possible path to success has been sketched

Conclusion
Ongoing work

- Currently, the approach described in Part III is investigated by
  - Anne E. Haxthausen, Alexander Knapp, Mohammad Reza Mousavi, Jan Peleska, Markus Roggenbach, Uwe Schulze, Jörg Brauer
  - Would you like to join us?

THANK YOU VERY MUCH FOR YOUR ATTENTION!