MBSE ⊃ MDA®

Model-based Software Engineering, Formal Methods, and Mechatronic Systems

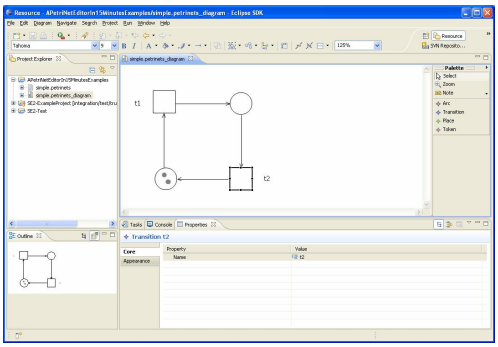
"PB speak" for embedded systems"

Ekkart Kindler
Technical University of Denmark
Informatics and Mathematical Modelling

Competence Center in Model-based Software Engineering

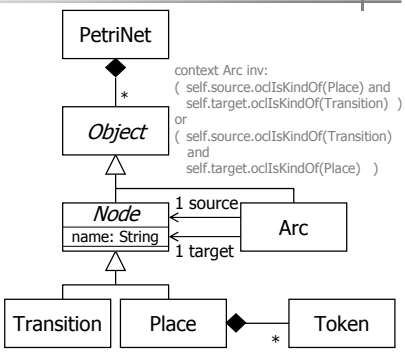
Table of Contents

A Petri Net Editor



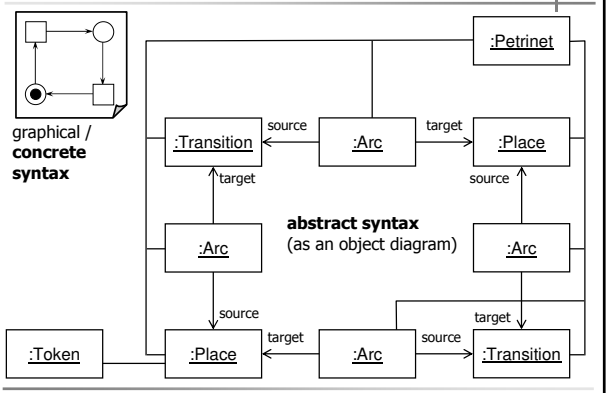
Models and Meta Models

Petri net model

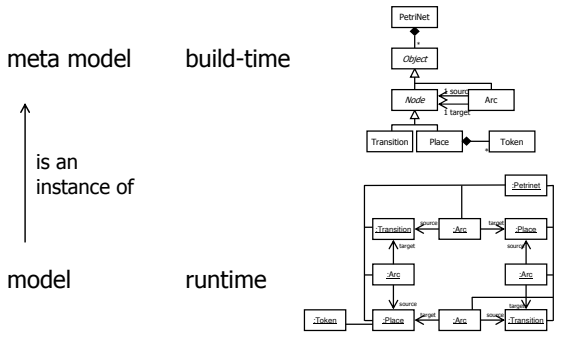


Meta model for Petri nets

Syntax (abstract and concrete)



Overview

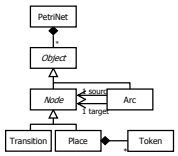


Benefits of Modelling

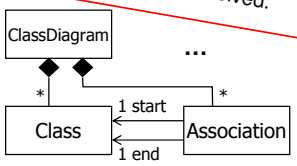
- Better understanding
- Mapping of instances to XML syntax (XMI)
- Automatic code generation
  - API for creating, deleting and modifying model
  - Methods for loading and saving models (in XMI)
  - Standard mechanisms for keeping track of changes (observers)

Class Diagrams are Models too

Disclaimer: The real model of UML is much more evolved.



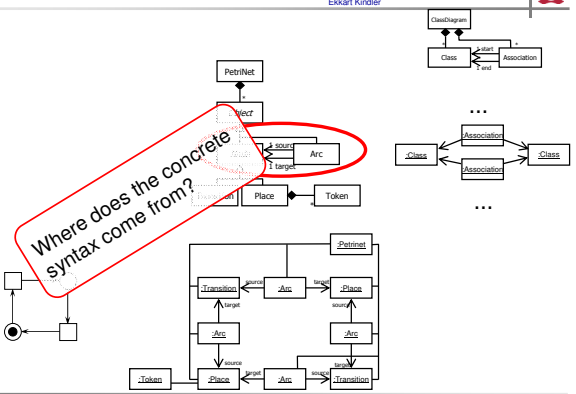
UML model



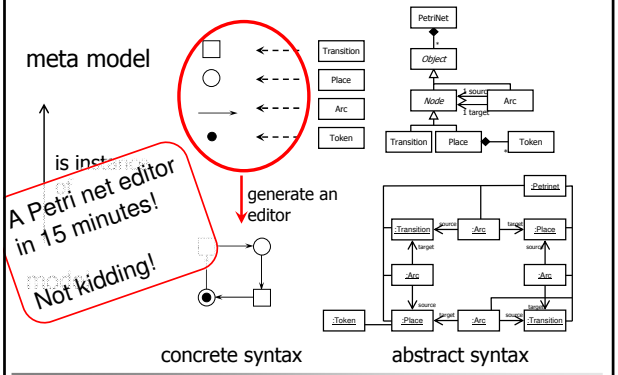
Meta model for UML (class diagrams)

Now, the term "meta" model makes sense!

Different Meta-levels: MOF

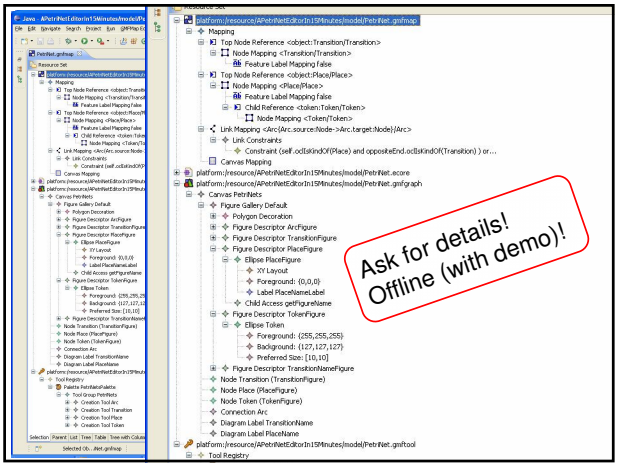


EMF/GMF Technology



concrete syntax

abstract syntax



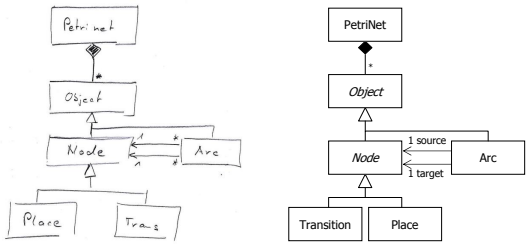
Benefits of Modelling (cntd.)

- Better Understanding
- Mapping of instances to XMI)
- A Generation
  - for creating, deleting and modifying model
  - Methods for loading and saving models (in XMI)
  - Standard mechanisms for keeping track of changes (observers)
  - Editors and GUIs

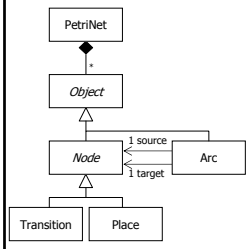
NB: All this is "standard functionality"?

No programming at all!

From Sketches to Models



From Models to Code

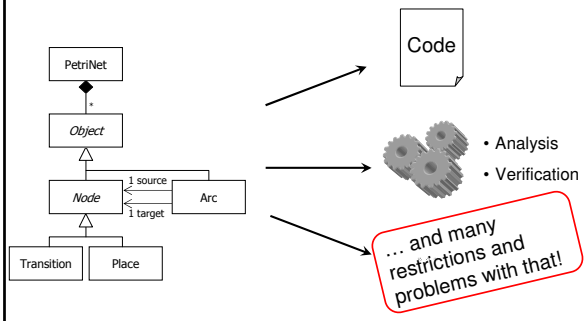



```
...
public class ArcImpl extends EObjectImpl implements Arc {

protected Node source;

protected Node target;

...

protected ArcImpl() {
  super();
}

...

public Node getSource() {
  if (source != null && source.eIsProxy()) {
    InternalEObject oldSource = (InternalEObject) source;
    source = (Node) eResolveProxy(oldSource);
    if (source != oldSource) {
      if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, ... ));
    }
  }
  return source;
}
...
```


Table of Contents

Forget about programming!

From Models to …

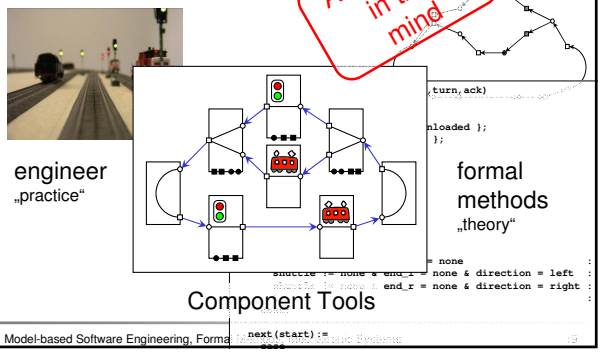


Challenges

- Modelling non-standard functionality (modelling behaviour)
- Adequate modelling notations
  - Coarse grain
  - Fine grain
- Clear interfaces and integration of behaviour models
- Change mentality
  → Model interpretation vs. code generation

Table of Contents

Bridging the Gap



Formal Methods

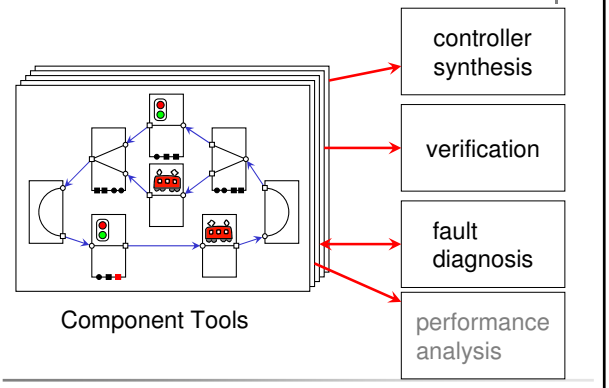
- Formal Methods provide many techniques for designing, analyzing, validating and verifying systems.
- Formal Methods complement each other
- For getting the full support of Formal Methods, we need to make use of
  - different models in different formalisms
  - different techniques and tools
  - models on different levels of detail

Formal Methods

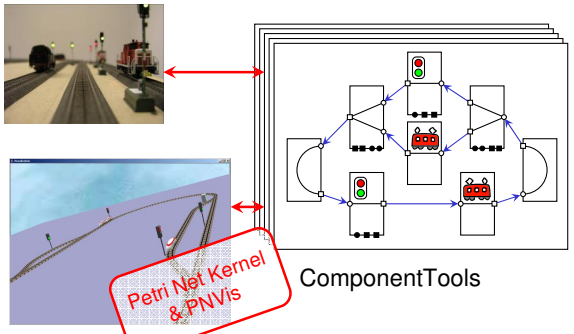
➔ need to be combined with each other

➔ need a unified front-end to the engineer

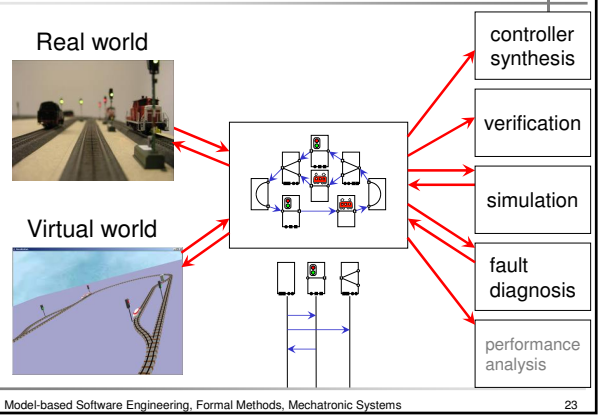
Tasks



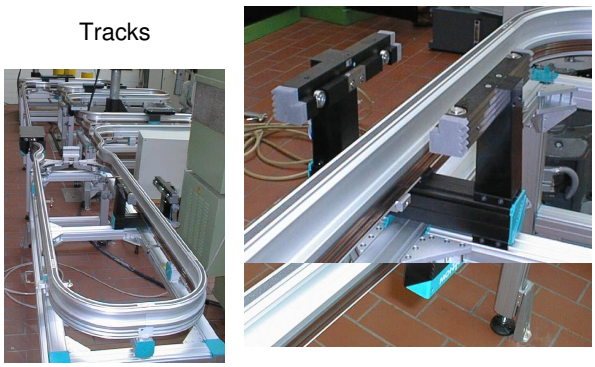
Real and Virtual World



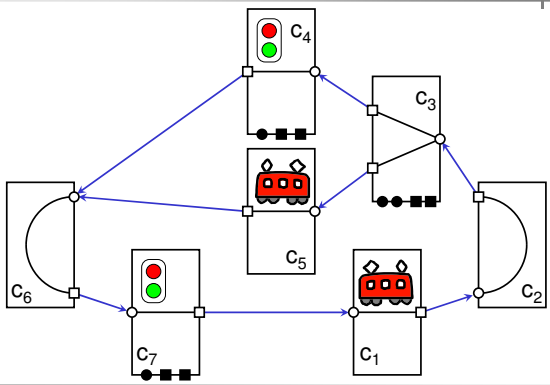
Integration



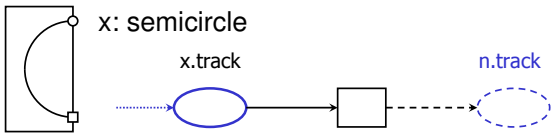
Example: FMS MonTrac



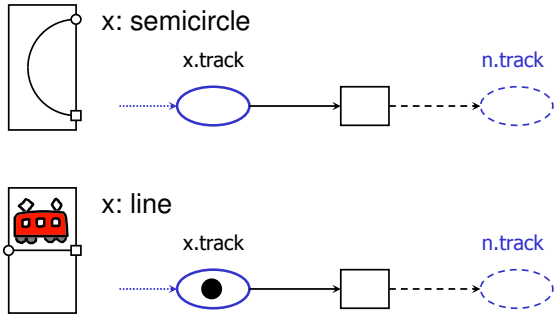
Circuit diagram



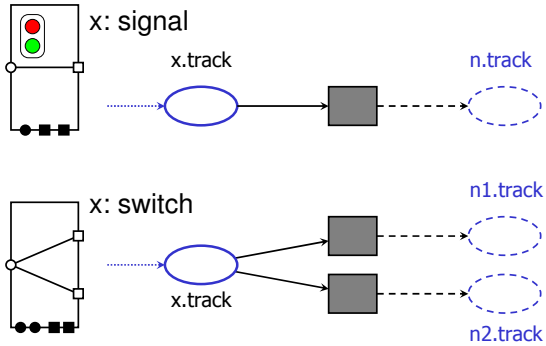
Models for components



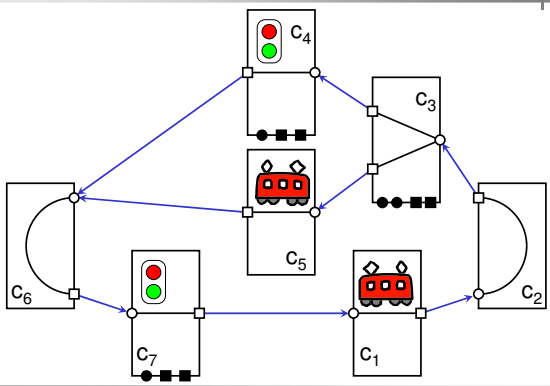
Models for components



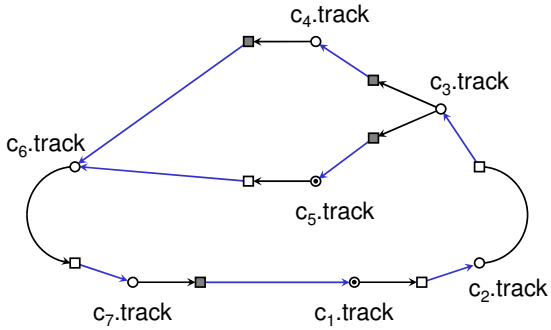
Models for components



Circuit diagram



Overall model

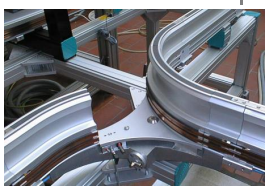


Purpose of these models

- Analysis of controllability
- Verification
  - for a concrete circuit
    → model checking
  - for a class of circuits
    → automated theorem proving
- Controller synthesis
  (on an abstract level)

Problems

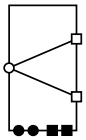
- Shuttle must not cross a switch while the switch is turning.



- Two shuttles must not travel on a semicircle at the same time.

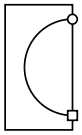
- …

Constraints



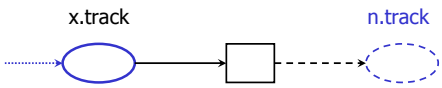
x: switch

AG (x.track ⇒ ¬ (x.turnL ∨ x.turnR))



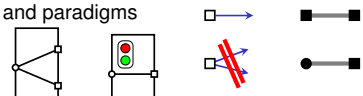
x: semicircle

AG | x.track | ≤ 1



Component Tools

- Component library
  - Port types
  - Connection types and paradigms
  - Components
  - Views

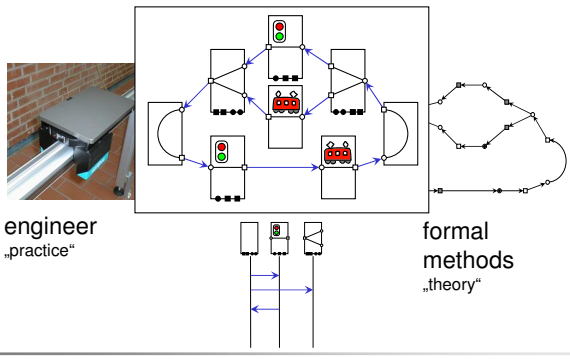


  - Model types
  - Models for components (for each type)
  - Constraints

- Transformations (in either directions)
- Formal Methods
- Visualisation

Table of Contents

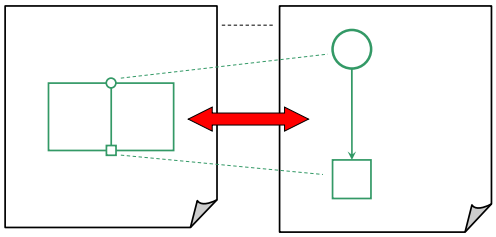
Transformations



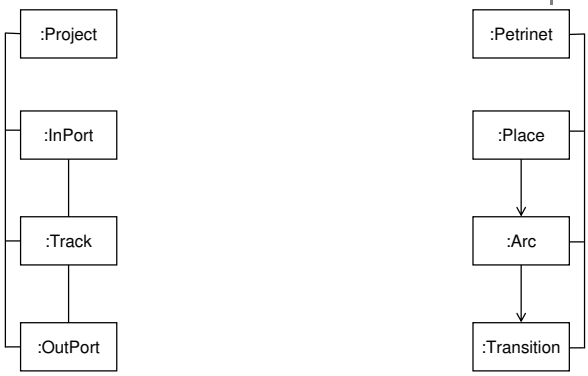
engineer
„practice“

formal methods
„theory“

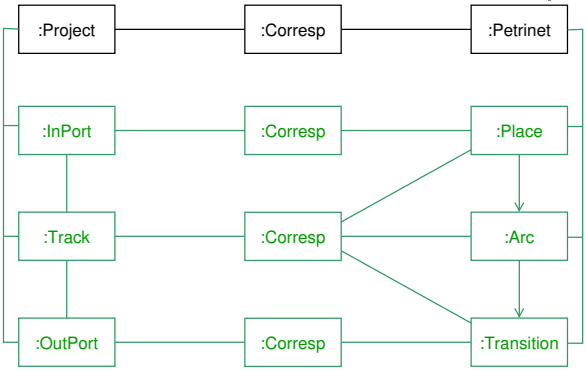
Transformation Rule 1



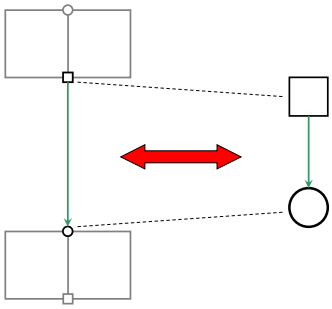
Transformation Rule 1



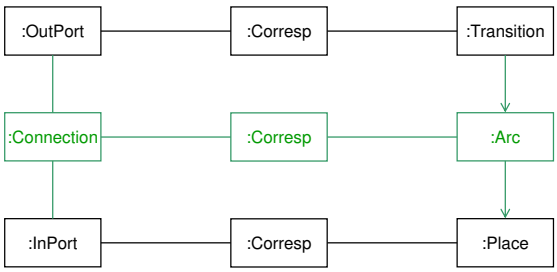
Triple Graph Grammar (TGG)



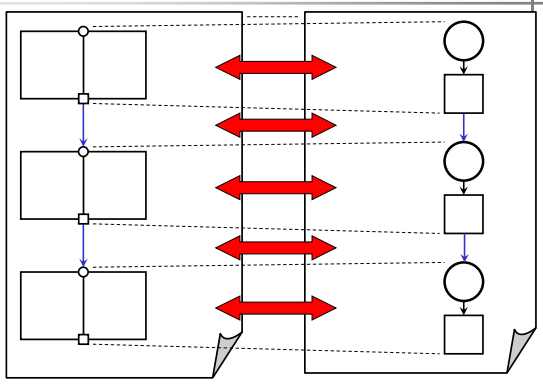
Transformation Rule 2



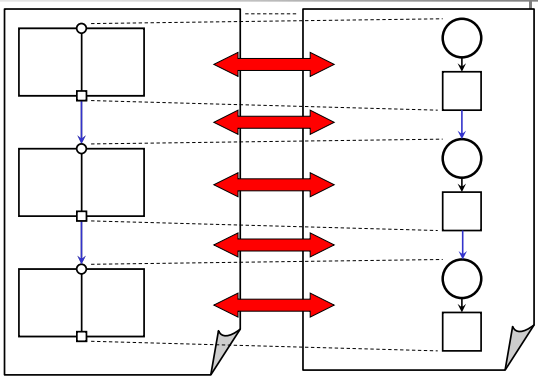
Triple Graph Grammar



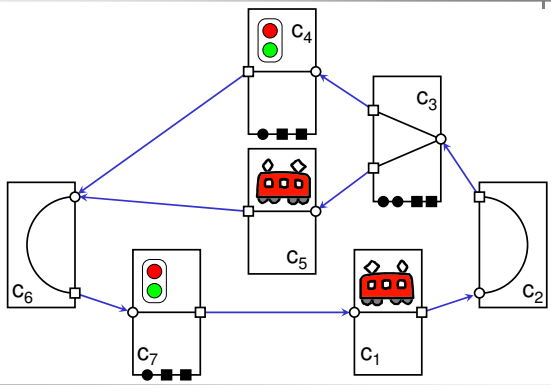
Semantics of TGG-Rules



Transformation

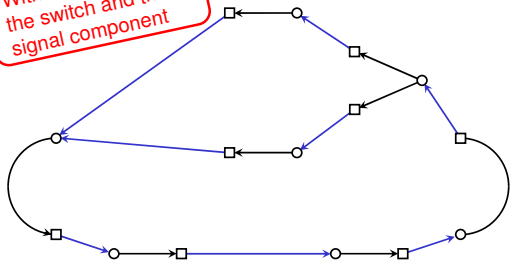


Source (project)

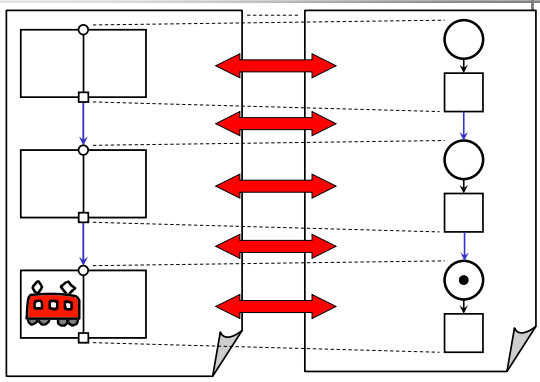


Target (Petri Net)

With similar rules for the switch and the signal component



Incremental application



TGG-Transformations

- Declarative definition
- TGG-rules close to “intuition”
- Transformation is executed by a TGG-rule interpreter (both ways)

→Simple and flexible definition of transformations (“easy” to change)

→Transformation in both directions (backwards translation of results)

Conclusions (Vision)

- Models become more and more important in SE!
- Behaviour modelling is still a problem!
- Reasons
  - No good understanding for different “kinds” and levels of behaviour
  - No clear interfaces and mechanisms for integration
  - To focussed on the final “code”

Conclusions (Vision)

Technical University of Denmark
Informatics and Mathematical Modelling
Ekkart Kindler

Requirements for modelling notations (2):