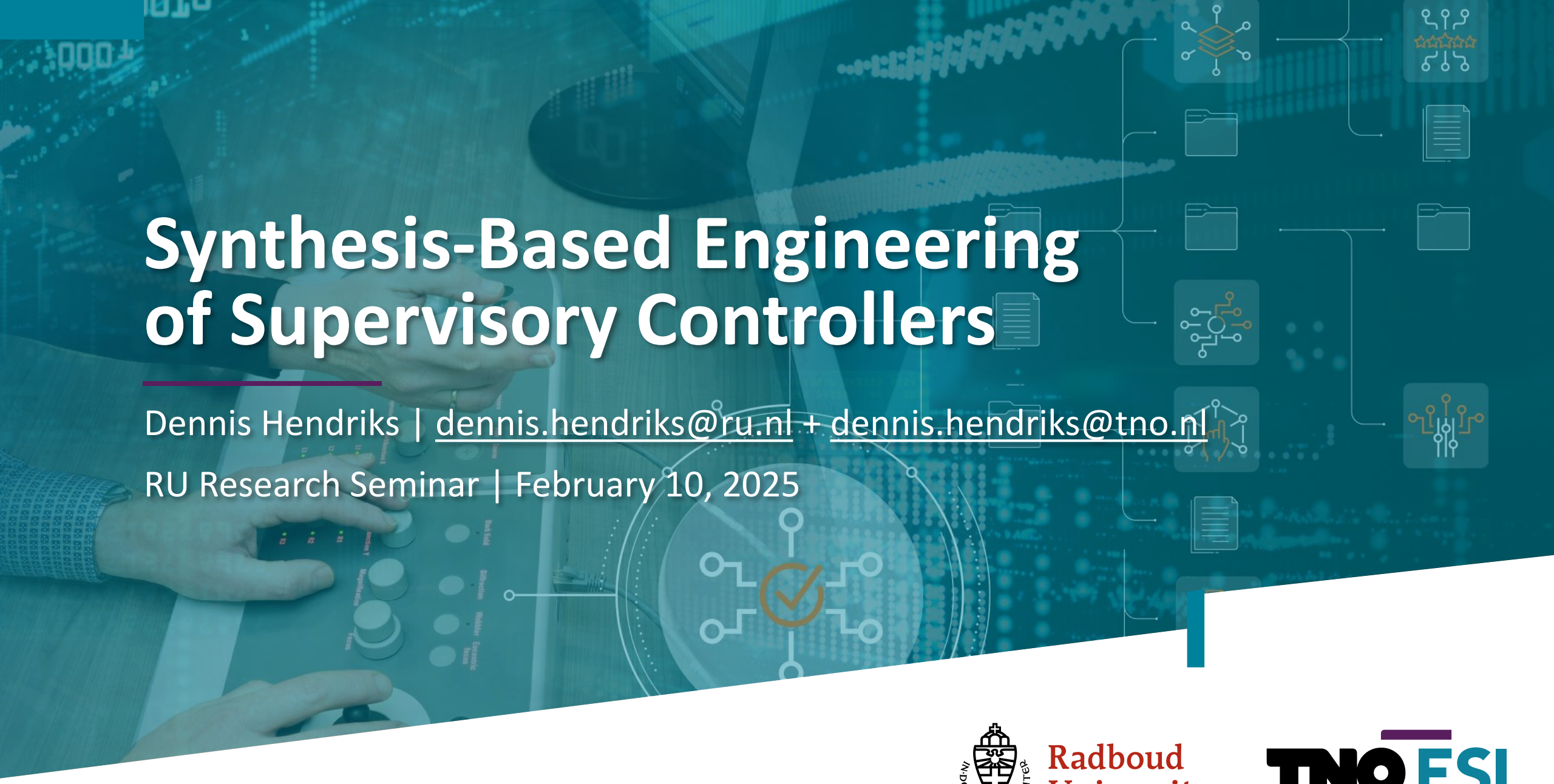


Synthesis-Based Engineering of Supervisory Controllers

Dennis Hendriks | dennis.hendriks@ru.nl + dennis.hendriks@tno.nl

RU Research Seminar | February 10, 2025

Radboud University

TNO ESI
Powered by industry and academia

Public

Dennis Hendriks

- Senior researcher at TNO-ESI | Applied research
- Part-time position at the Radboud University | Academic research and teaching
- Project lead of the Eclipse ESCET open-source project | Tool development and ecosystem orchestration

Public

Outline

1. Synthesis-Based Engineering (SBE)
2. Eclipse ESCET and CIF
3. Where are we with SBE?
4. Ongoing work (if time permits)
5. Roundup

Public

1. Synthesis-Based Engineering (SBE)

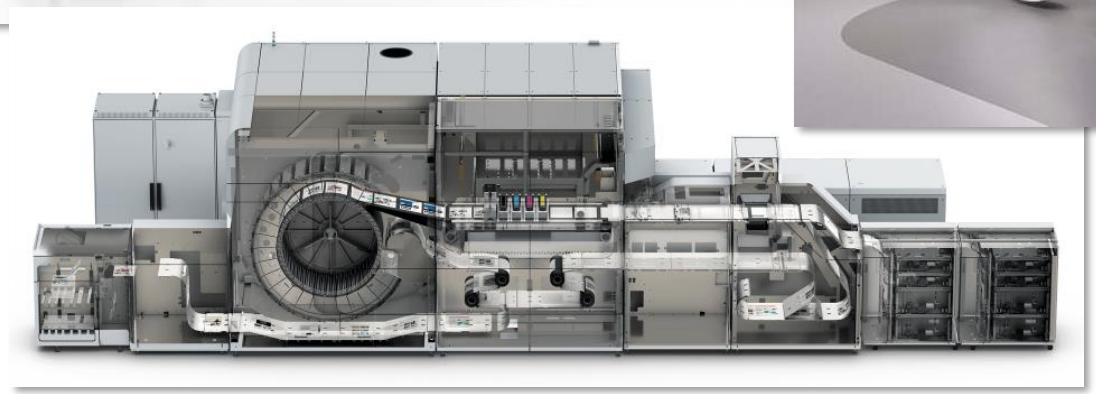
Radboud University

TNO ESI

Powered by industry and academia

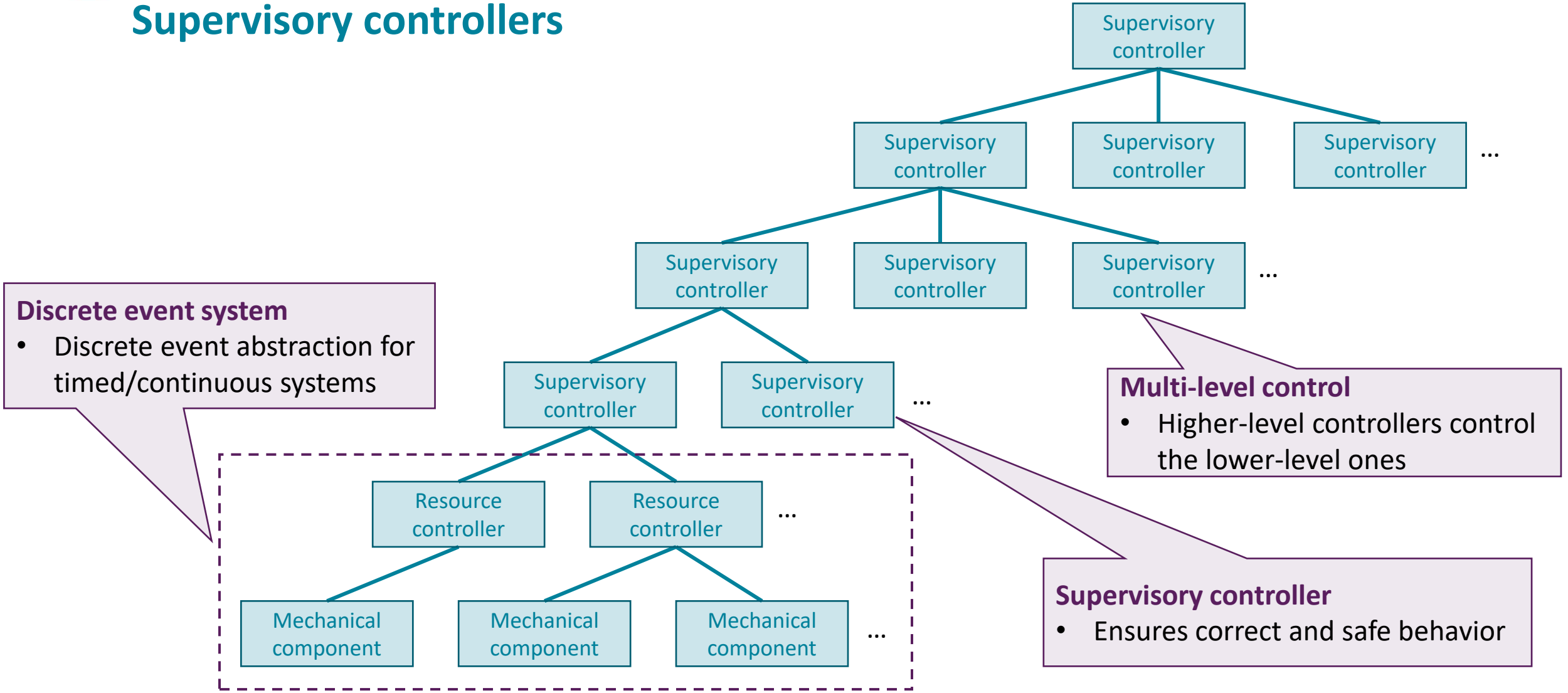
Public

Cyber-physical systems (CPSs)



Public

Supervisory controllers



Public

Engineering supervisory controllers is challenging

• Safety	E.g., prevent collisions
• Performance	E.g., increase parallelism
• Quality	E.g., prevent deadlock
• Complexity	E.g., integrate many components
• Variability	E.g., support many variants
• Exceptional behavior	E.g., handle all situations
• Effort & cost	E.g., work effectively and efficiently
• Lead time	E.g., respond quickly to customer demands
• Shortage of skilled people	E.g., get new people up-and-running quickly
• …	

→ Engineers continuously face difficult trade-offs

Public

Synthesis-Based Engineering (SBE) of supervisory controllers

Engineering approach → / ↓ Development step	Traditional Engineering	Model-Based Engineering	Verification-Based Engineering	Synthesis-Based Engineering
Requirements design	Document-based	Document-based	**Model-based (formal)**	Model-based (formal)
Controller design	Document-based	**Model-based (formal)**	Model-based (formal)	**Computer-aided (formal, synthesized)**
Realization in software (implementation code)	Traditional software engineering (coding)	**Code generation (fault-free code)**	Code generation (fault-free code)	Code generation (fault-free code)
Verification (against requirements)	Testing	Testing + **Model-based testing**	**Formal verification (model checking)**	**Correct-by-construction (guaranteed)**
Validation (of requirements)	Testing	Testing + **Simulation**	Testing + Simulation	Testing + Simulation
		E.g., Ansys SCADE Suite, Enterprise Architect, Cordis SUITE	E.g. ASD/Dezyne, Coco, mCRL2, UPPAAL	E.g. Eclipse ESCET / CIF, Supremica

Legend: Manual work / Focus (Semi-)automatic

(Simplified overview.)

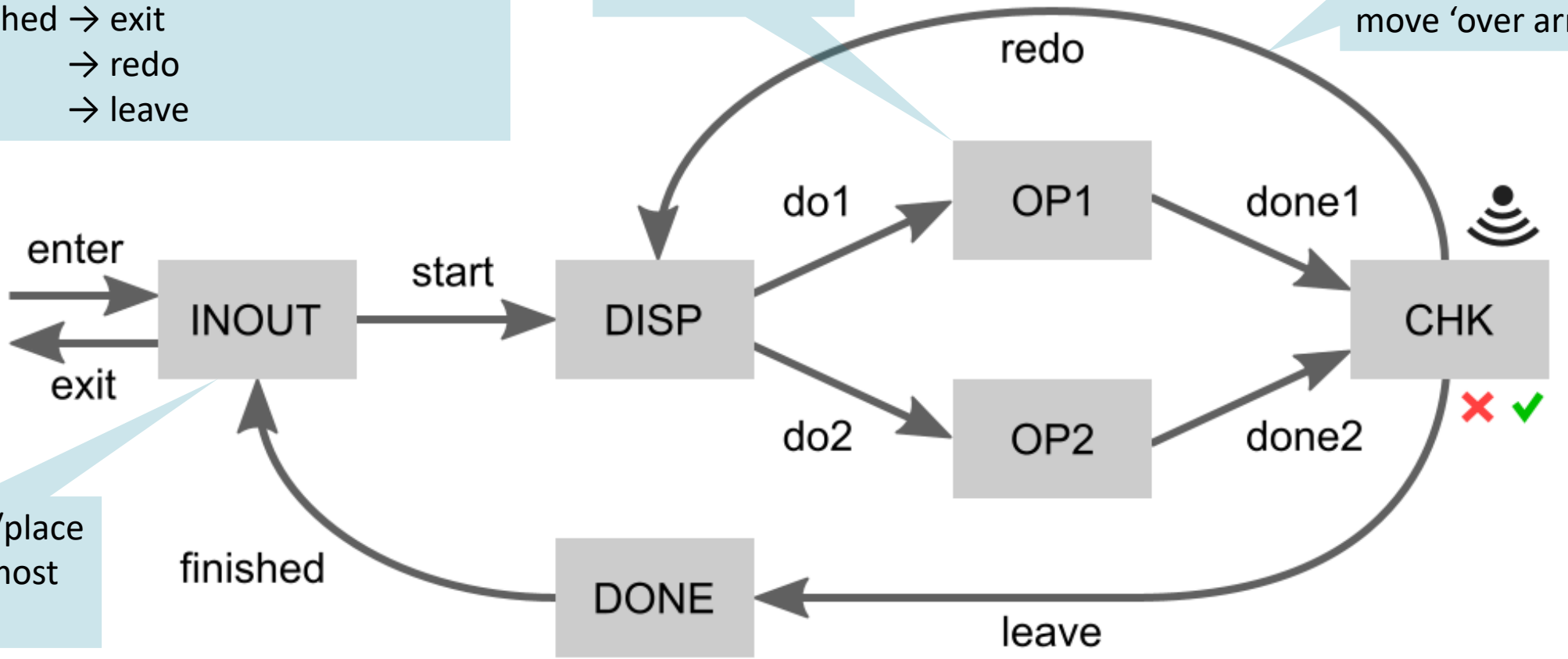
Public

Example: a simple manufacturing system to process products

The system itself already ensures that:
- enter → start
- finished → exit
- ✗ → redo
- ✓ → leave

Both operators are ‘the same’.

Controller decides when products may move ‘over arrows’.



Each station/place can hold at most one product.

Public

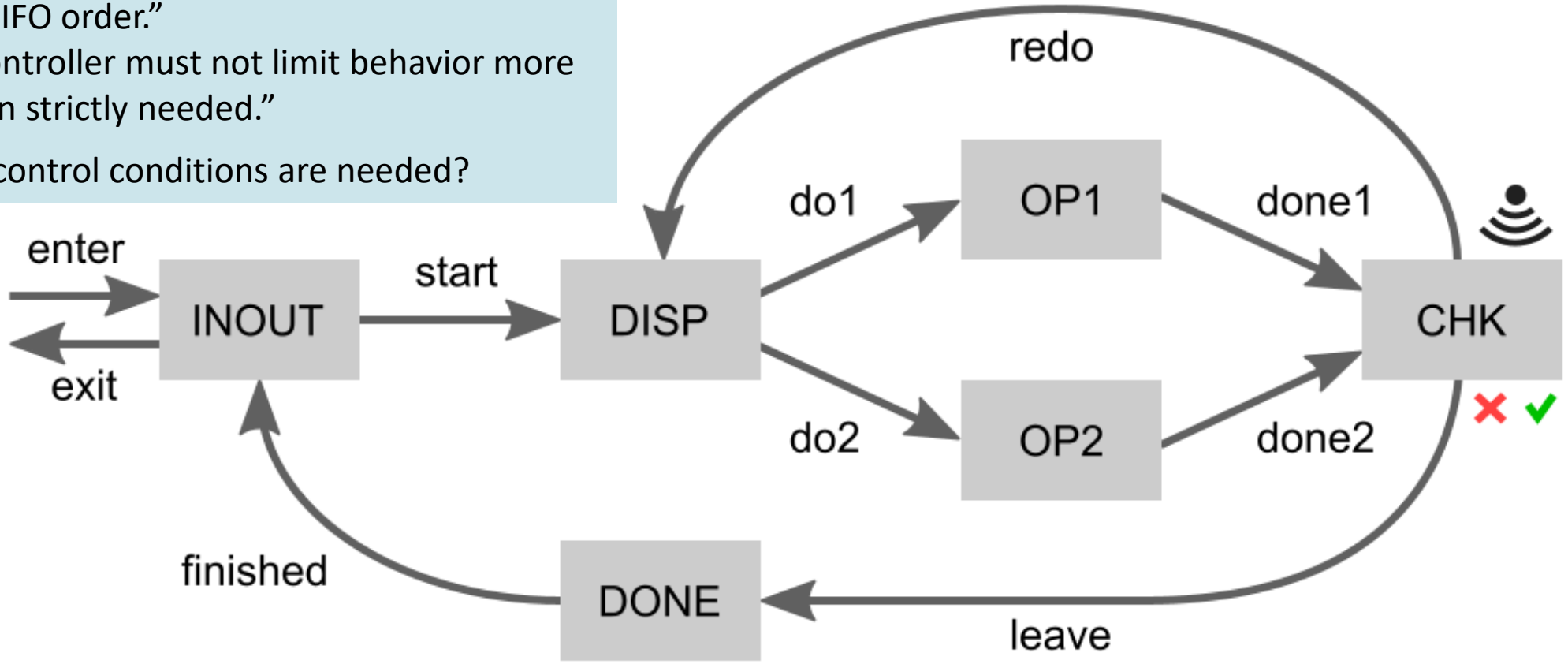
Example: a simple FIFO requirement

- Assign a unique product id to every product that enters.
- Products that exist must have an id that is 1 higher than the last one that exited.

Requirements:

- "Products must *enter* and *exit* the system in FIFO order."
- "Controller must not limit behavior more than strictly needed."

What control conditions are needed?



Public

Example: the control conditions

[Prevent deadlocks]
A product may only *enter* if less than four products are in the system.

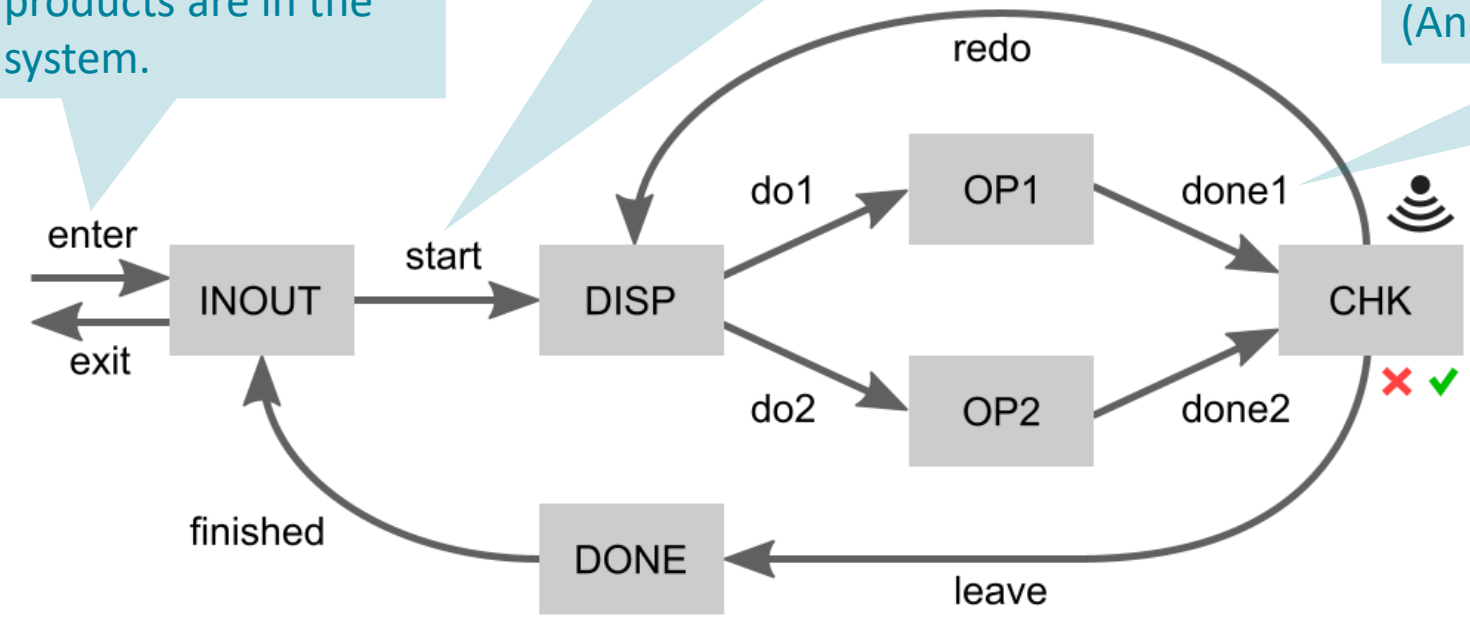
[FIFO: Ensure enough room for a redo]
A product may *start* only if one of two conditions holds:

- DISP + OP1 + OP2 + CHK ≤ 1 products
- DISP + OP1 + OP2 ≤ 1 products, and CHK = 1 product, and CHK result = OK

[FIFO: Only allow earliest product to be checked]
Movement *done1* is only allowed if two conditions both hold:

- Either there is no product at DISP, or it is a later product than at OP1.
- Either there is no product at OP2, or it is a later product than at OP1.

(And similar for *done2.)*

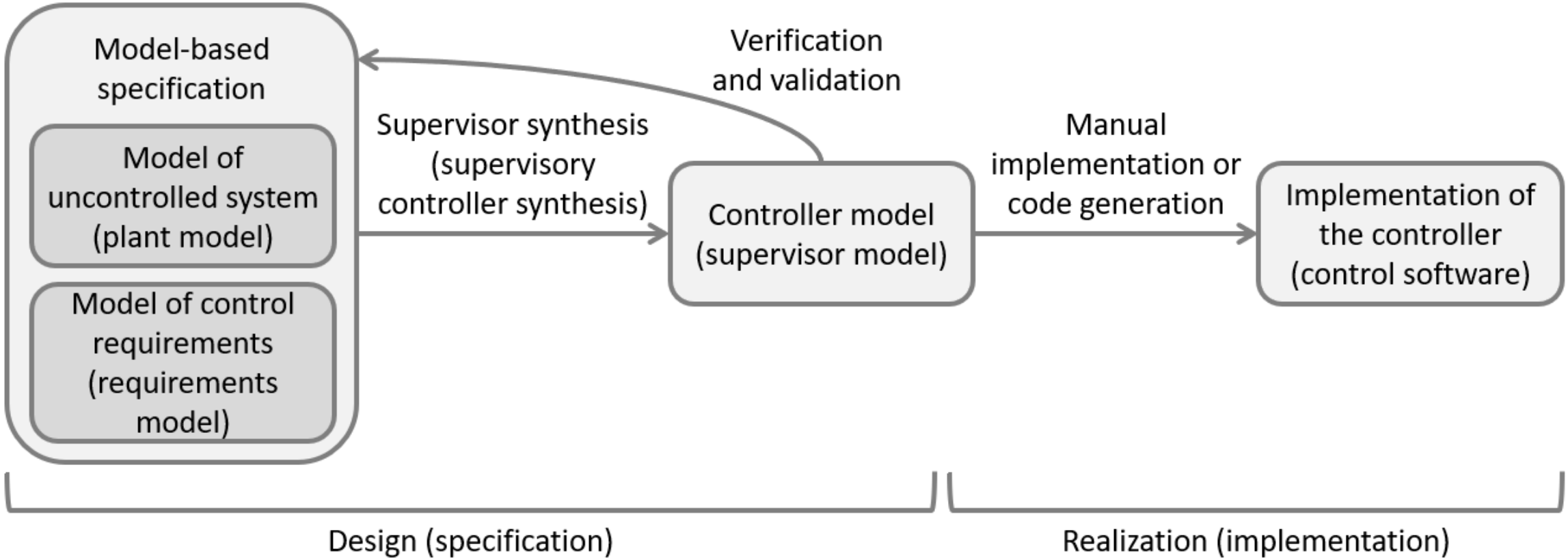


Conclusion

- This is a lot to consider, and quite subtle!
- This is still simple (small system, only two requirements)
- Would you have arrived at this solution?
- How long would it have taken you to?

Public

The Synthesis-Based Engineering process



(Simplified process.)

Public

Supervisor synthesis example: a waterway lock

Fig. 2. Lock system in Maasbracht, the Netherlands.

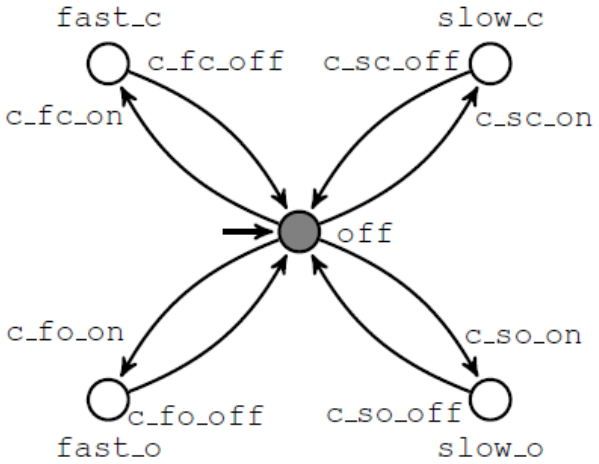


Fig. 12. Model of the gate speed actuator.

4) An incoming traffic light is not allowed to display the green aspect, see Figure 18, if:
   a) an outgoing traffic light displays the green aspect
   b) the gates at the same side are not completely open

```
 out_UN.S.green  ∨ out_UN.A.green     ∨
 out_US.S.green  ∨ out_US.A.green     ∨
¬gate_UN.S.open  ∨ gate_UN.A.closing ∨
¬gate_US.S.open  ∨ gate_US.A.closing
disables {in_UN.c_green, in_US.c_green}
```

Fig. 18. Model of requirement 4: enabling the green aspect for the upstream incoming traffic lights.

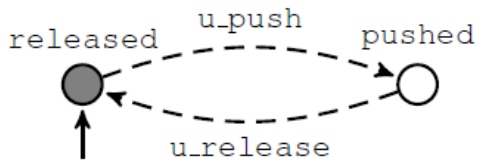
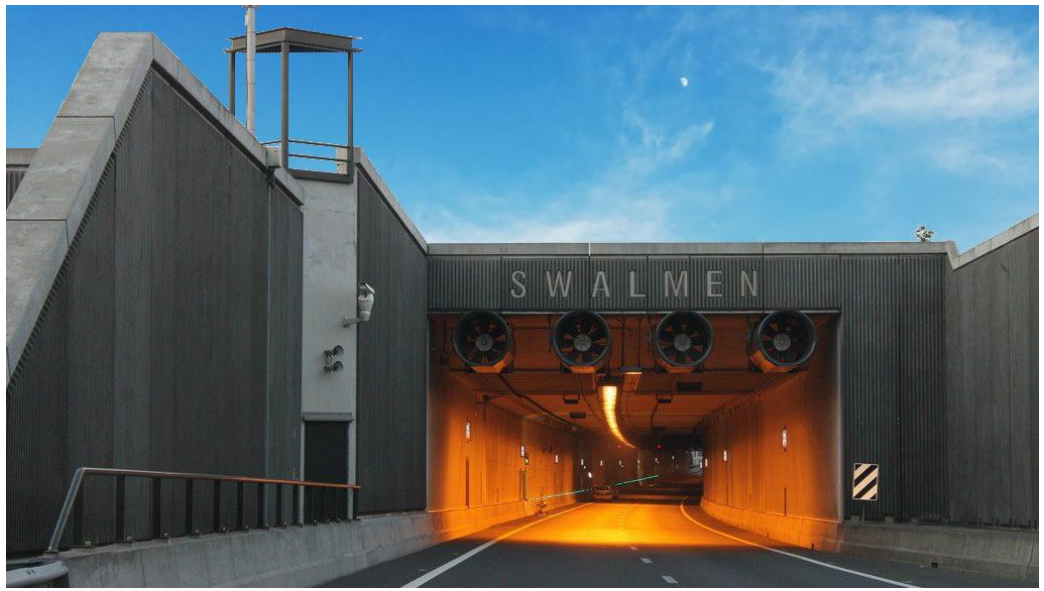


Fig. 14. Model of the GUI button.

- 62 plant components
- 234 requirements
- Synthesis takes a few seconds (simple laptop)
- Source: F.F.H. Reijnen, M.A. Goorden, J.M. van de Mortel-Fronczak and J.E. Rooda, Supervisory Control Synthesis for a Waterway Lock, IEEE Conference on Control Technology and Applications (CCTA), 2017.

Public

Supervisor synthesis example: a tunnel



Swalmen tunnel, A73, near Swalmen, Roermond, Limburg

Textual description:
"The boom barrier may only be lowered when the traffic light is red."

Logical expression:

$$\texttt{BoomBarrier.c\_down} \implies \texttt{TrafficLight.red}$$

CIF code:

```
requirement BoomBarrier.c_down needs TrafficLight.red;
```


Public

Synthesis-Based Engineering: many benefits

SBE combines model-driven engineering with computer-aided design

Raising the abstraction level → **Focus on the essentials ('what rather than how')**

- Using models at proper abstraction levels, bridge the specification/implementation gap to **improve communication**
- **Focus on *what*** the controller should do (the requirements/validation), ***not how*** (controller implementation)

Computer-aided design → **Together tackle the complexity ('complementary strengths')**

- **Design assistance** to automatically detect issues, such as conflicting requirements, livelocks, and more
- **Computer-aided validation/verification** (e.g., simulation) helps get to the requirements right, early on ('shift left')
- **Synthesis** to automatically compute correct-by-construction controllers, for every configuration/scenario

Hyper-automation → **Humans only where needed ('automate all the things')**

- Automation and computer-aided design save time, **shortening the development cycle, reducing efforts/costs**
- **Re-synthesis** upon changes enables incremental engineering, preventing mundane work, **easing evolution**

Model-driven engineering → **Ensure continuity ('single source of truth')**

- **Consolidate modular design/requirements** in intuitive, unambiguous, complete, consistent, up-to-date models

Public

2. Eclipse ESCET and CIF

Radboud University

TNO ESI
Powered by industry and academia

Public

Synthesis-Based Engineering with CIF: the language

CIF is a powerful declarative modeling language for the specification of discrete event, timed, and hybrid systems as a collection of synchronizing automata

Discrete event automata

- Automata
- Synchronizing events
- Point-to-point channels
- Non-determinism
- Monitors

Support for large systems

- Automaton definitions/instantiations
- Groups (+ definitions/instantiations)
- Imports

Extended finite automata

- Data (variables, guards, updates)
- Discrete variables
- Algebraic variables (+ equations)
- Input variables
- Constants
- Rich built-in data-types
- Custom types / type declarations
- Initialization predicates
- Invariants
- Functions (also as data)

Timed automata

- Time variable
- Continuous variables (+ equations)
- Urgency concepts

Other extensions

- Supervisory controller synthesis
- Stochastics
- SVG visualization/interaction
- Print output
- Custom annotations

For more information: https://eclipse.dev/escet/cif/language-tutorial

Public

Synthesis-Based Engineering with CIF: the tools

The CIF tooling supports the entire development process of controllers

Specification
- Textual editor
- Conversion to yEd diagrams

Supervisory controller synthesis
- Data-based synthesis
- Event-based synthesis
- Conversion to Supremica

Simulation, visualization and validation
- Interactive simulation
- Automated simulation
- Powerful visualization/interaction

Verification
- Controller properties checker
- Conversion to mCRL2
- Conversion to UPPAAL

Implementation
- C and Java code generation
- JavaScript/HTML code generation
- PLC and Simulink code generation

Other
- State space generator (explorer)
- CIF to CIF transformations
- Model merger
- Event disabler

For more information: https://eclipse.dev/escet/cif/tools

Public

Eclipse ESCET open-source project

Eclipse ESCET project (pronounced as èsèt)

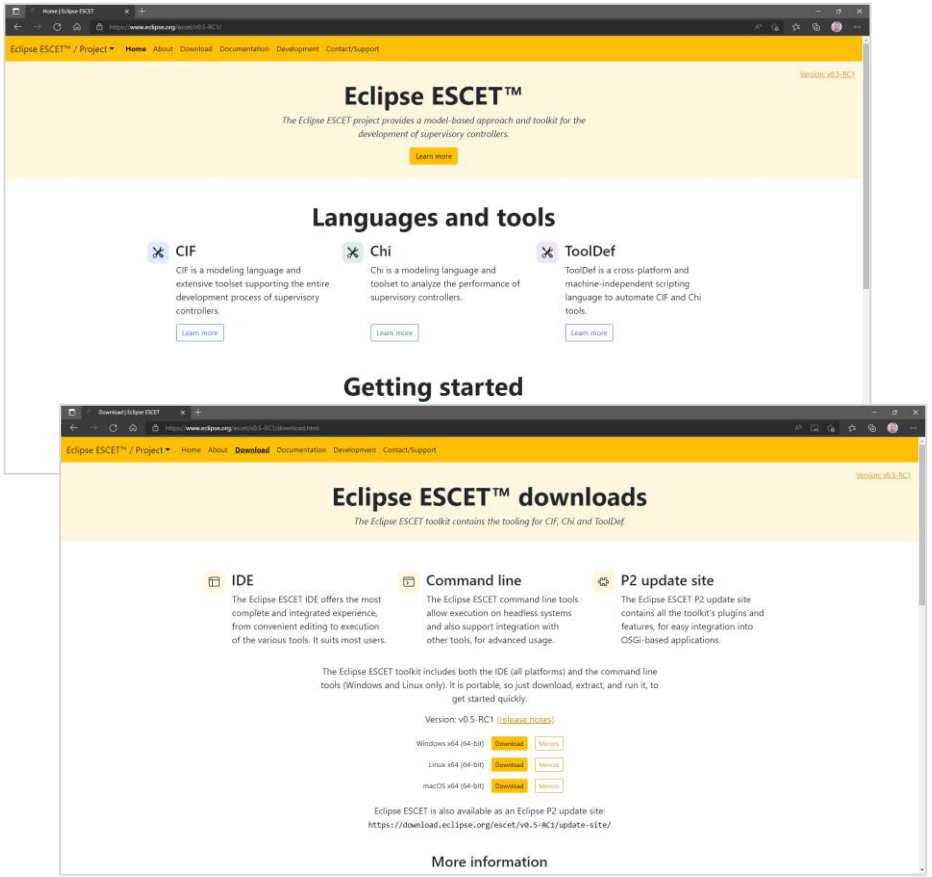
- Eclipse Supervisory Control Engineering Toolkit
- An Eclipse Foundation project since 2020
- Home to CIF, and some other tools (Chi and ToolDef)

Downloads

- Eclipse ESCET IDE and command line tools
- Windows, Linux, macOS
- Portable: extract and use

Active development

- Quarterly releases



For more information: https://eclipse.dev/escet

Public

3. Where are we with SBE?

Radboud University

TNO ESI
Powered by industry and academia

Public

Synthesis-Based Engineering: status of the methodology

A long history

- 37 years of academic and applied SBE research
- Active research at various universities and research institutes world-wide

Maturity of methodology has greatly improved over the past years

- Improved modeling convenience (e.g., with variables/guards/updates, invariants, forms of requirements)
- Improved scalability (e.g., symbolic synthesis, variable ordering, saturation, multi-level synthesis)
- Synthesis of fault-tolerant supervisors (e.g., broken sensors, actuators, wires)
- Correct-by-construction code generation (e.g., controller checks, PLC code semantics)
- Configure-to-order way of working (e.g., generate synthesis/simulation models and code)
- …

Public

Synthesis-Based Engineering: status of the tools

A long history

- 18 years of CIF language/tool development
- Origin at TU/e, now in ESCET open-source project

Maturity of tools

- Thoroughly tested
- Extensive documentation
  - Language tutorial / reference manual
  - SBE documentation
  - Tool documentation
- Examples
- Online self-learning SBE course (https://eclipse.dev/escet/sbe-course/)

Public

Synthesis-Based Engineering: status of application

Real-world application

- Successful case studies in various domains
  - E.g., infrastructure (Rijkswaterstaat)
  - E.g., semiconductor (ASML)
  - E.g., healthcare (Philips)
  - … and many more
- Proven real-world maturity
  - E.g., car drove in real traffic (cruise control) [1]
  - E.g., bridge passed site acceptance test [2]
- Adoption
  - E.g., by RWS for all renovations of water locks (with more to come)

Public

Synthesis-Based Engineering: status of community/ecosystem

Growing community/ecosystem

Community/ecosystem activities

- Academic research
- Applied research
- Joint publications
- Community meetings
- Workshops
- Ecosystem orchestration (roadmap, etc)
- Tool development
- …

Public

4. Ongoing work

Radboud University

TNO ESI
Powered by industry and academia

Public

The Poka Yoke applied research project (2023 – 2026)

Poka Yoke: Japanese term used in Lean manufacturing, meaning ‘mistake-proofing’ or ‘error prevention’

Project goal: investigate to what extent SBE helps to manage the increasing complexity of CPSs

Carrier case: the wafer handler for ASML’s TWINSCAN systems

- Developed by ASML and VDL-ETG together

Current way of working:

- Wafer control is manually specified as UML activities

To ease adoption of SBE:

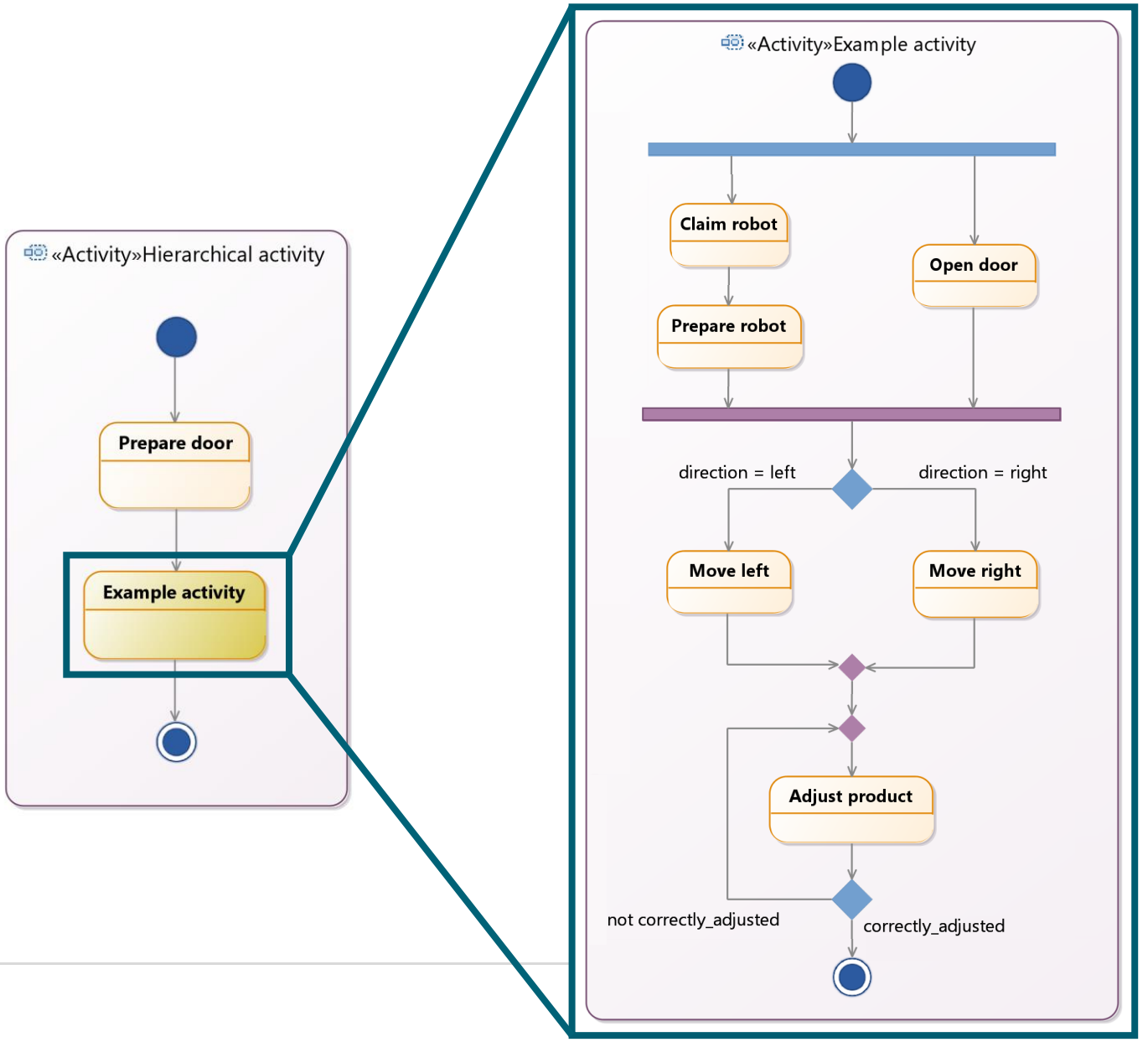
- Stick with familiar modeling concept of activities
- Stick with familiar UML tools

Image source: https://www.lead-engineer.com/

Public

UML activity concepts

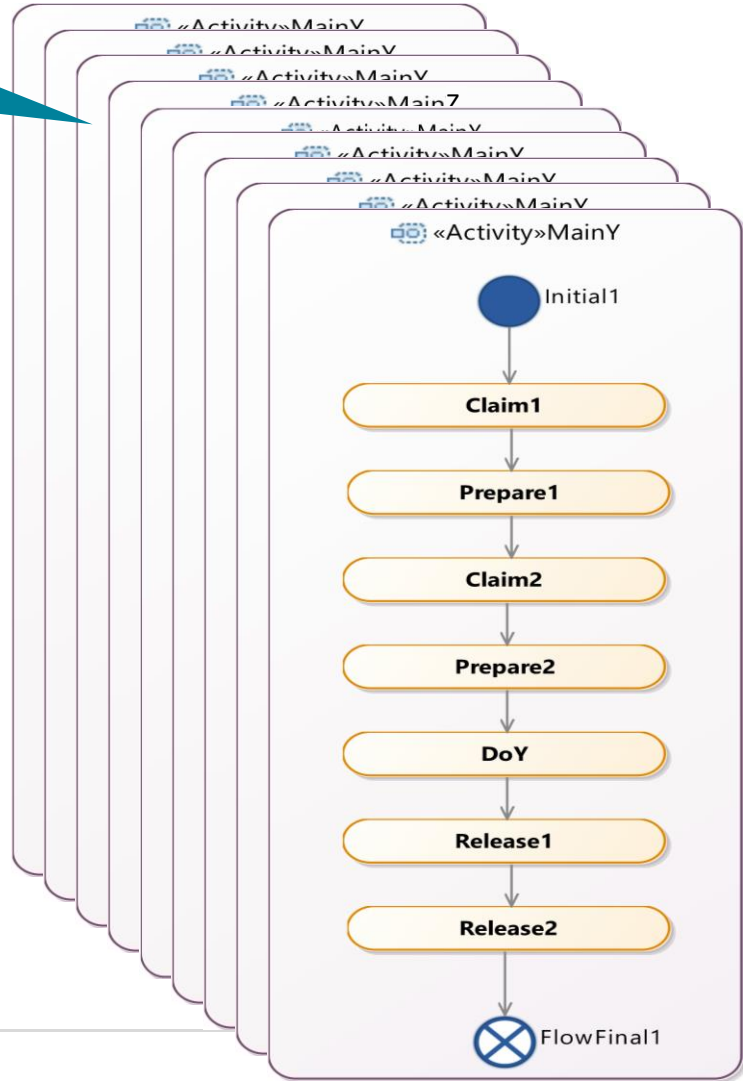
- Action
  - Precondition (required system state)
  - Effect (changes to system state)
- Sequence
- Parallel
- Choice / merge
- Loop
- Initial / final
- Hierarchy



Public

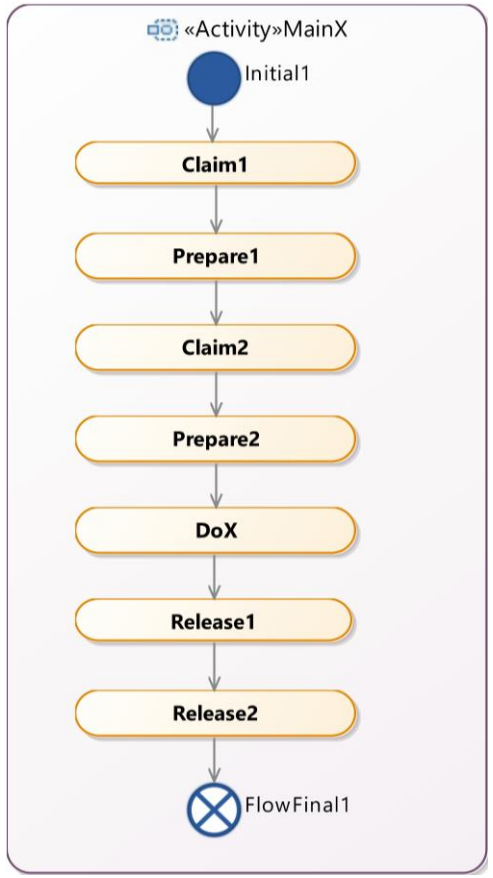
Example: local parallelization → global deadlock risk

Reduce complexity
by divide-and-conquer

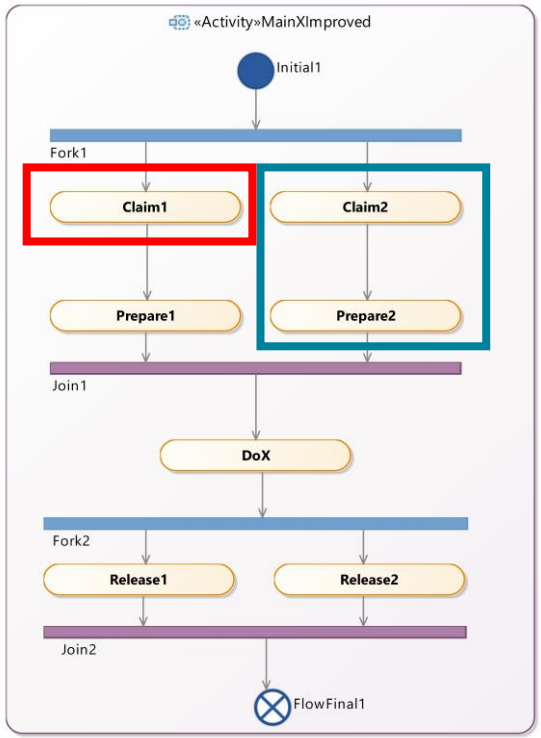


Public

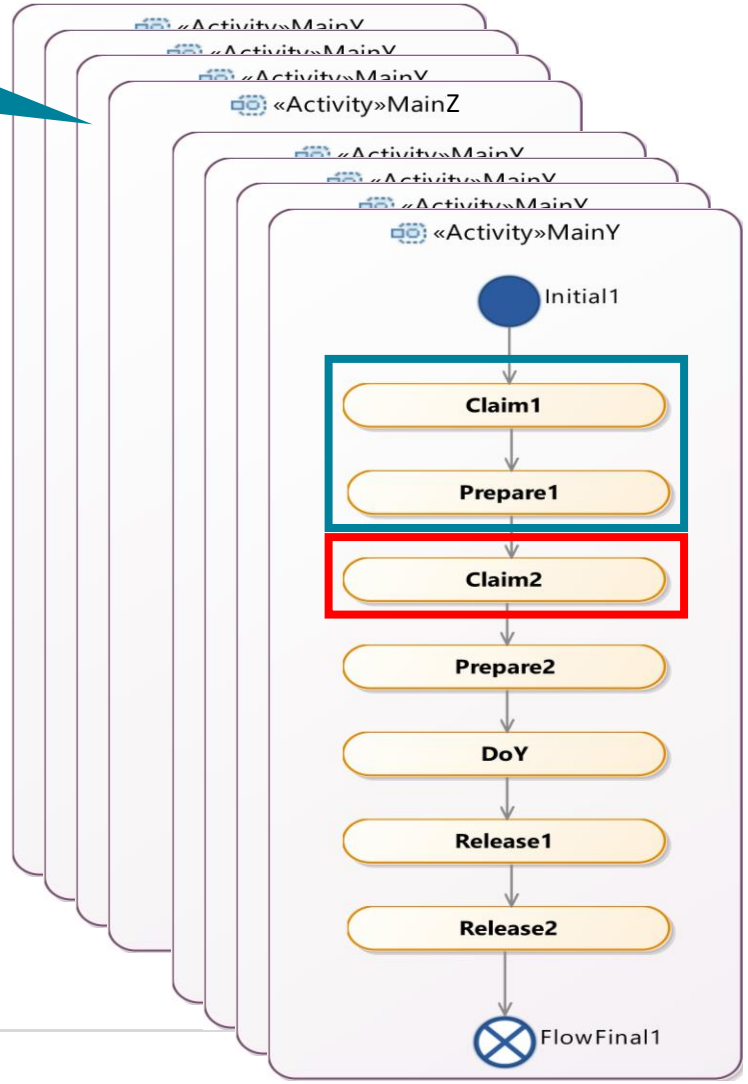
Example: local parallelization → global deadlock risk



Increase performance
by introducing parallelism



Reduce complexity
by divide-and-conquer



Decreased quality
by deadlock

Public

Activity synthesis

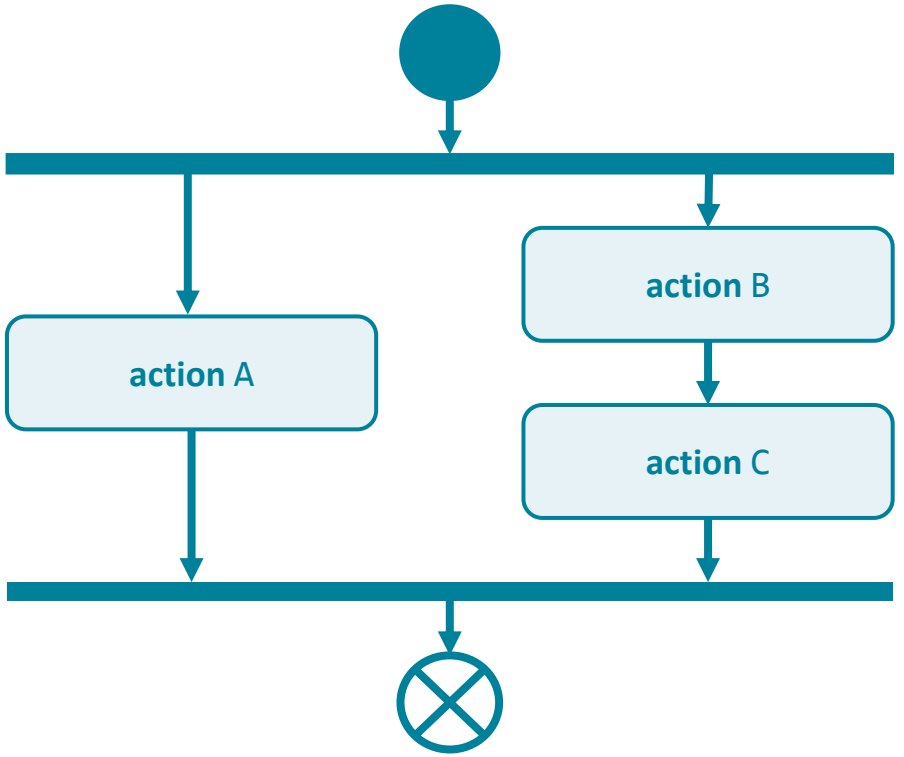
I. **Specify system state**
   - E.g., place holds a wafer or not

II. **Specify available actions**
   - E.g., picking a wafer from a place when it has a wafer, after which it no longer has one

III. **Specify requirements**
   - E.g., robots must not collide

IV. **Specify pre- and postconditions**
   - E.g., unprocessed wafer at start position to processed one at end position

V. **Automatic synthesis**



Public

Activity synthesis embedded in existing UML tooling

<Model> Model
<Enumeration> Position
<Enumeration> UnknownBool
<Class> Spec
{?} <Constraint> safety_requirement_1
{?} <Constraint> safety_requirement_2
{?} <Constraint> safety_requirement_3
{?} <Constraint> safety_requirement_4
{?} <Constraint> ordering_requirement_1
{?} <Constraint> ordering_requirement_2
{?} <Constraint> ordering_requirement_3
{?} <Constraint> ordering_requirement_4
<Property> : Boolean
<Property> : Boolean
<Property> Position
<Property> : Boolean
<Property> : Boolean
<Property> : Boolean
<Property> : Boolean
<Property> : UnknownBool
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Opaque Behavior>
<Activity>
{?} <Constraint> postcondition
{?} <Constraint> precondition

III. Requirements

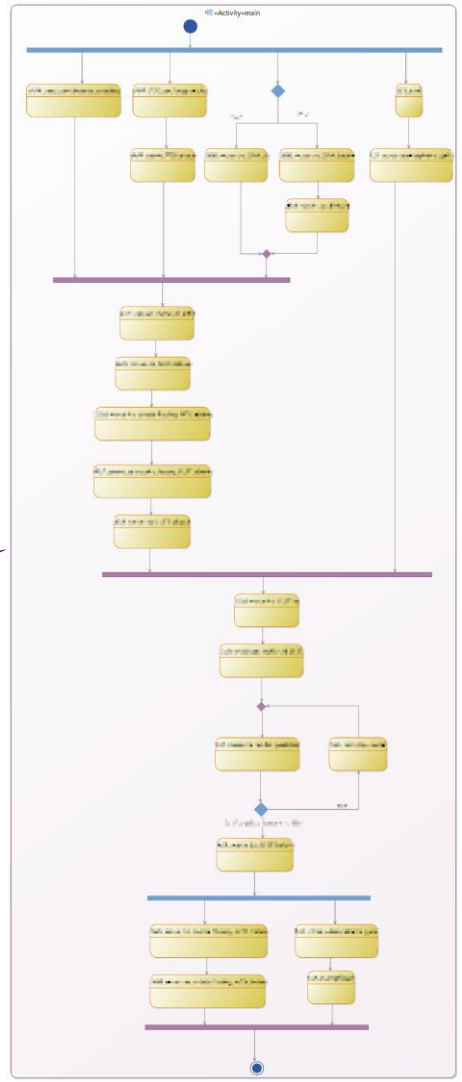
I. System state

II. Available actions

IV. Pre/post conditions

V. Synthesis under the hood, hidden from the user

Activity synthesis

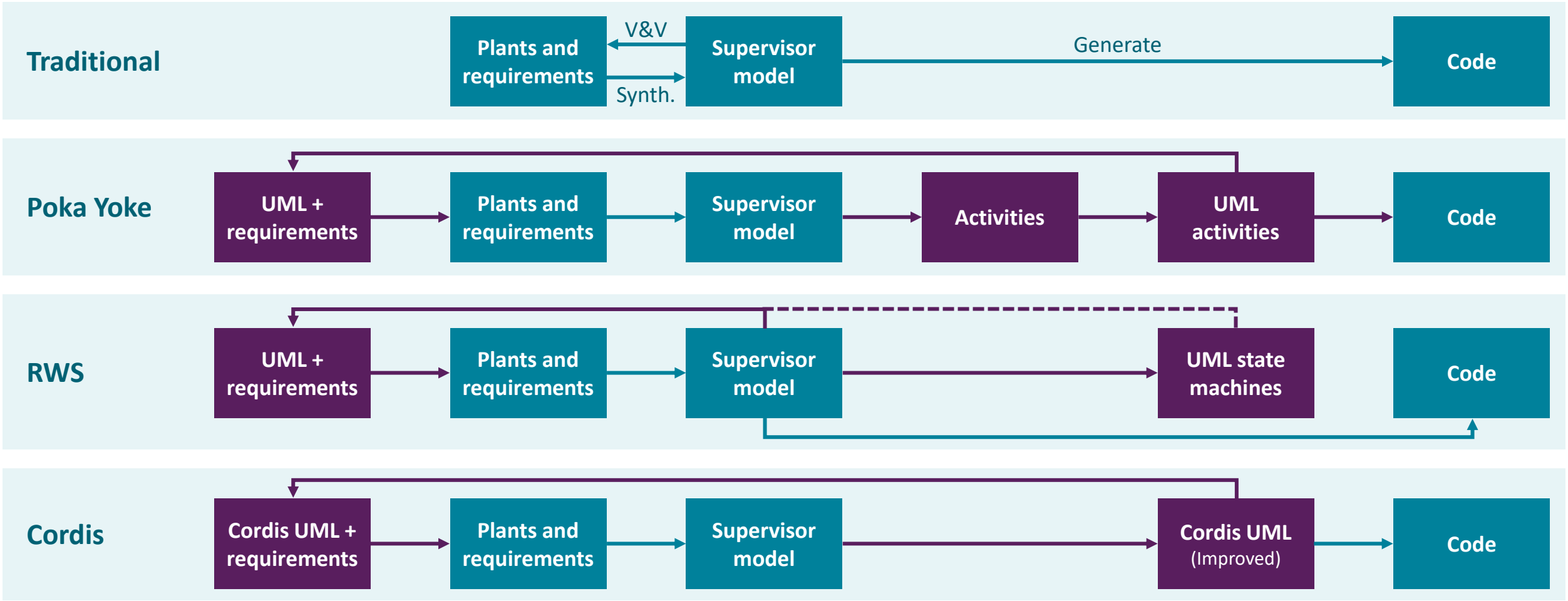


Synthesized UML activity diagram:

- Automatically computed
- Automatically detects rework may be required
- Parallelized as much as possible
- Correct-by-construction with respect to requirements
- Guaranteed to be free of deadlocks
- Result presented in familiar UML notation

Public

Synthesis modeling frontend is adaptable – integration in Way-of-Working



Public

5. Roundup

Radboud University

TNO ESI
Powered by industry and academia

Public

Synthesis-Based Engineering: some more information

- Synthesis-Based Engineering (7 min full video, 1 min teaser video)
  Vimeo, 2024
- Synthesis-based engineering of supervisory controllers
  Mikroniek, 2023
- Supervisor Synthesis: Bridging Theory and Practice
  Computer, 2022
- Supervisory control of discrete-event systems: A brief history
  Annual Reviews in Control, 2018
- Eclipse ESCET™: The Eclipse Supervisory Control Engineering Toolkit
  TACAS, 2023
- Synthesis-based engineering
  Eclipse Foundation, Eclipse ESCET open-source project documentation

Public

Research seminar

You can choose this paper

- Overview and Performance Evaluation of Supervisory Controller Synthesis with Eclipse ESCET v4.0
  CEP, submitted, preprint, 2024

Overview and Performance Evaluation of Supervisory Controller Synthesis with Eclipse ESCET v4.0

Dennis Hendriks[a,b,*,1], Michel Reniers[c], Wan Fokkink[c,d] and Wytse Oortwijn[a,1]

[a]TNO-ESI, High Tech Campus 25, 5656 AE, Eindhoven, The Netherlands
[b]Radboud University, Toernooiveld 212, 6525 EC, Nijmegen, The Netherlands
[c]Eindhoven University of Technology, Groene Loper 5, 5612 AE, Eindhoven, The Netherlands
[d]Vrije Universiteit Amsterdam, De Boelelaan 1111, 1081 HV, Amsterdam, The Netherlands

ARTICLE INFO

Keywords:
Supervisory controller synthesis
Eclipse ESCET
Benchmark models
Performance evaluation

ABSTRACT

Supervisory controllers control cyber-physical systems to ensure their correct and safe operation. Synthesis-based engineering (SBE) is an approach to largely automate their design and implementation. SBE combines model-based engineering with computer-aided design, allowing engineers to focus on 'what' the system should do (the requirements) rather than 'how' it should do it (design and implementation). A main ingredient of SBE is supervisory controller synthesis, which automatically generates correct-by-construction supervisory controllers. In the Eclipse Supervisory Control Engineering Toolkit (ESCET™) open-source project, a community of users, researchers and tool vendors jointly develop a toolkit to support the entire SBE process, particularly through the CIF modeling language and tools. In this paper, we first provide a complete description (up to a certain level of detail) of CIF's symbolic synthesis algorithm, and thereby include aspects that are often omitted in the literature, but are of great practical relevance, such as the prevention of runtime errors, handling different types of requirements, and supporting input variables (to connect to external inputs). This description assists researchers that wish to improve the algorithm. Secondly, we introduce and describe CIF's set of benchmark models, a collection of 23 industrial and academic models of various sizes and complexities, that are freely available, allowing researchers to benchmark synthesis algorithms and their improvements. Thirdly, we describe recent improvements between ESCET versions v0.8 (December 2022) and v4.0 (June 2024) that affect synthesis performance, evaluate them on our set of benchmark models, and show the current practical synthesis performance of CIF. Fourthly, we briefly look at multi-level synthesis, a non-monolithic synthesis approach, evaluate its gains, and show that while it can help to further improve synthesis performance, further performance improvements are still needed to synthesize complex models.

1. Introduction

A supervisory controller, supervisor for short, coordinates the behavior of a cyber-physical system according to discrete-event observations of the system's behavior. Based on such observations, the supervisor decides which events the system can safely perform and which events must be disabled, because they would lead to violations of requirements or to a blocking state. Engineering of supervisors is a challenging task, due to the high complexity of real-life discrete-event systems [3, 20, 21, 56].

Synthesis-based engineering (SBE) is an engineering approach to design and implement supervisors that combines model-based engineering with computer-aided design, to produce correct-by-construction controllers, by automating the engineering process as much as possible. A main ingredient of SBE is supervisory controller synthesis [47], a theory and technique for automatically deriving a model of a supervisor from discrete-event models of the uncontrolled system behavior and the system's requirements, such as functional or safety-related requirements that intend to rule out all undesired behavior. This allows engineers to focus on *what* the system should do (the requirements) rather than *how* the controller should do it (the design and implementation).

The Eclipse Supervisory Control Engineering Toolkit (ESCET™, pronounced *ésèt*) project[2] [19], provides a model-based approach and toolkit for the development of supervisors. It targets the entire model-based engineering process for the development of supervisors, including modeling, synthesis, simulation-based validation and visualization, formal verification, real-time testing and code generation. The ESCET toolkit contains multiple modeling languages and associated tools. In this paper we consider only one of them, CIF[3] [4], which features an automata-based modeling language for convenient specification of large-scale systems, and tools that support the entire SBE process.

The ESCET project, an Eclipse Foundation open-source project since 2020, builds upon decades of research and

*Corresponding author
dennis.hendriks@tno.nl / dennis.hendriks@ru.nl (D. Hendriks); m.a.reniers@tue.nl (M. Reniers); w.j.fokkink@vu.nl (W. Fokkink); wytse.oortwijn@tno.nl (W. Oortwijn)
ORCID(s): 0000-0002-9886-7918 (D. Hendriks); 0000-0002-9283-4074 (M. Reniers); 0000-0001-7443-8978 (W. Fokkink); 0000-0002-5244-2519 (W. Oortwijn)
[1]This research is partly carried out at part of the Poka Yoke program under the responsibility of TNO-ESI in cooperation with ASML and VDL-ETG. These research activities are supported by the Netherlands Organization of Applied Scientific Research TNO, the Netherlands Ministry of Economic Affairs and Climate Policy, and TKI-HTSM.

[2]See https://eclipse.dev/escet. 'Eclipse', 'Eclipse ESCET' and 'ESCET' are trademarks of Eclipse Foundation, Inc.
[3]See https://eclipse.dev/escet/cif.

Public

Internship/graduation assignments

- Both **internal** at RU and **external** projects**,** e.g., TNO, ASML, RWS, Cordis (subject to their availability/interests)
- **Theoretical**, **practical** or both
- Potential to **get your results landed** in the Eclipse ESCET toolkit and be used by industry (depends on project)
- Broad research area. Some examples of current ideas:
  - **Applications/integrations**: synthesis+UML/SysML | synthesis+AI | synthesis+Coco | synthesis+ComMA
  - **Support more models/systems**: synthesis of models with priorities | synthesis of models with tuples/arrays
  - **Synthesis performance**: smart internal data structures | optimize settings (for algorithms/models)
  - **Theory**: define formal semantics and link to concrete syntax
  - ... and more (we can discuss the options)

More information: https://sws.cs.ru.nl/Teaching/SynthesisBasedEngineering

Contact me: dennis.hendriks@ru.nl (for questions, for doing an assignment, …)

Public

Synthesis-Based Engineering (SBE): take-away

SBE combines model-driven engineering with computer-aided design

Raising the abstraction level

→ Focus on the essentials, with clear and explicit requirements (‘*what* rather than *how*’)

Computer-aided design

→ Design assistance: computer-aided validation, verification and correct-by-construction synthesis

Efficient and high-quality engineering:
better quality controllers at reduced effort and cost

Public

Copyright © 2025 TNO-ESI and Radboud University

Some of the work presented in this presentation is carried out at part of the Poka Yoke program under the responsibility of TNO-ESI in cooperation with ASML and VDL-ETG. These research activities are funded by Holland High Tech | TKI HSTM via the PPP Innovation Scheme (PPP-I) for public-private partnerships.

‘Eclipse’, ‘Eclipse ESCET’ and ‘ESCET’ are trademarks of Eclipse Foundation, Inc.

Radboud University

TNO ESI
Powered by industry and academia