Formal Stuff can be Surprisingly Useful

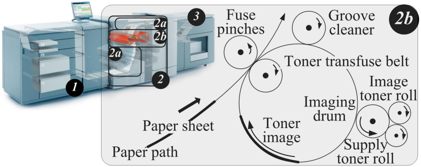
Wan Fokkink

Joint work with Martijn Goorden, Ferdie Reijnen, Lars Moormann, Jeroen Verbakel, Bert van Beek, Asia van de Mortel-Fronczak, Michel Reniers, Koos Rooda

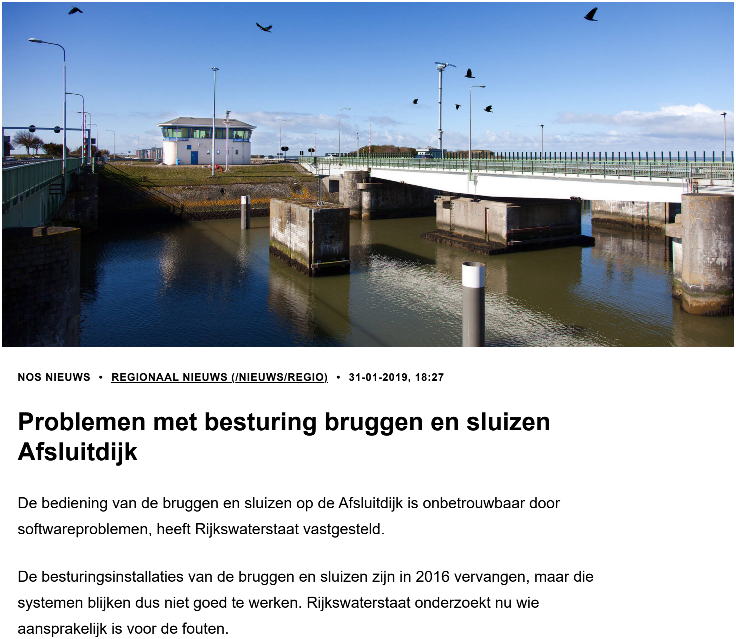
PART I

Supervisor synthesis

Controller software



Software problems at Rijkswaterstaat



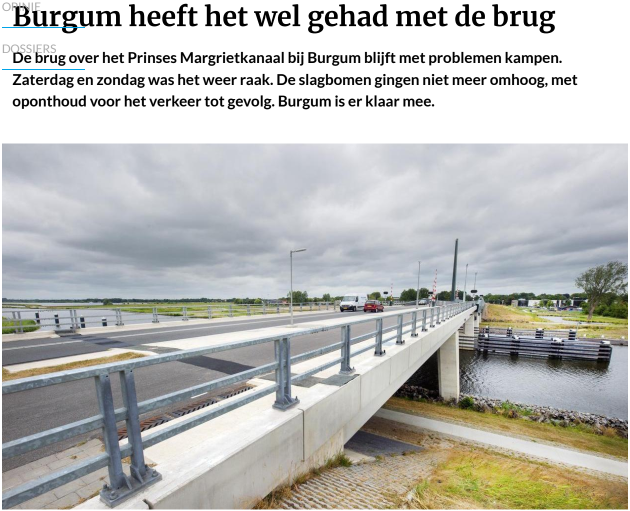
NOS NIEUWS • REGIONAAL NIEUWS (/NIEUWS/REGIO) • 31-01-2019, 18:27

Problemen met besturing bruggen en sluizen Afsluitdijk

De bediening van de bruggen en sluizen op de Afsluitdijk is onbetrouwbaar door softwareproblemen, heeft Rijkswaterstaat vastgesteld.

De besturingsinstallaties van de bruggen en sluizen zijn in 2016 vervangen, maar die systemen blijken dus niet goed te werken. Rijkswaterstaat onderzoekt nu wie aansprakelijk is voor de fouten.

Software problems at Rijkswaterstaat



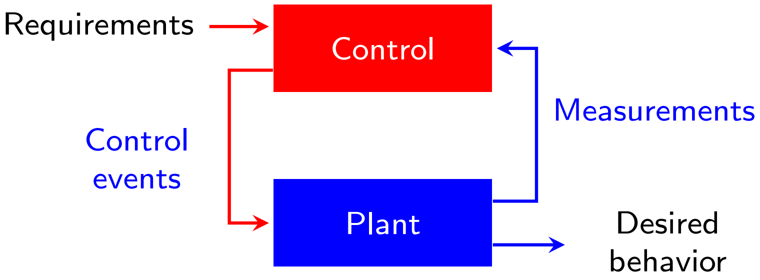
Burgum heeft het wel gehad met de brug

De brug over het Prinses Margrietkanaal bij Burgum blijft met problemen kampen. Zaterdag en zondag was het weer raak. De slagbomen gingen niet meer omhoog, met oponthoud voor het verkeer tot gevolg. Burgum is er klaar mee.

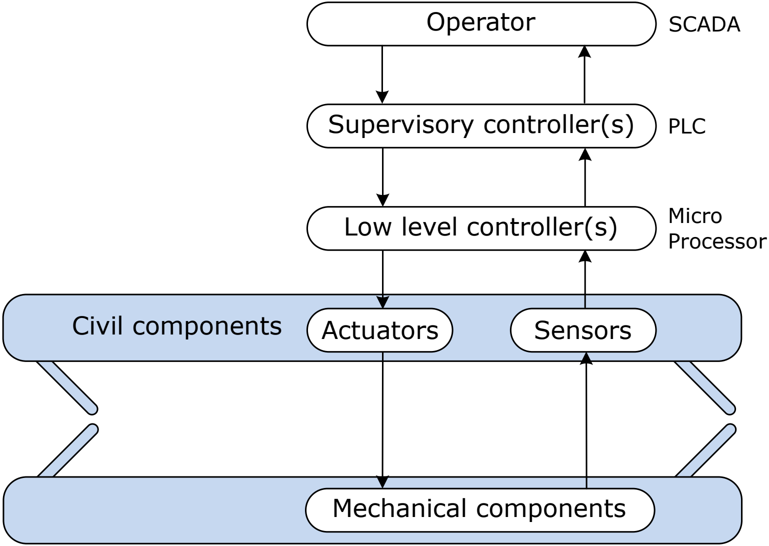
Resident: *"Wy hawwe der gjin ferstân fan. It sil wol technyk wêze."*

Rijkswaterstaat: *"It appears to be a software problem."*

Plant, requirements, and control



System view


Operator
SCADA
Supervisory controller(s)
PLC
Low level controller(s)
Micro
Processor
Civil components
Actuators
Sensors
Mechanical components

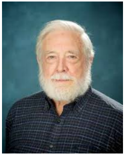
Supervisor synthesis

A supervisor is *synthesized automatically* out of finite automata (with variables) that specify

- the parallel composition of the *unrestricted behavior* of the system components, and
- functional *safety requirements* on the system behavior.

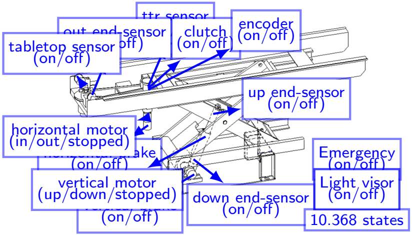
The restricted behavior is guaranteed to meet all requirements.

Peter J. Ramadge, W. Murray Wonham
Supervisory control of a class of discrete event processes
SIAM Journal on Control and Optimization, 1987



Example: MRI scanner

Example: MRI scanner



ttr sensor

out end-sensor (on/off)

tabletop sensor (on/off)

clutch (on/off)

encoder (on/off)

up end-sensor (on/off)

horizontal motor (in/out/stopped)

horizontal brake (on/off)

vertical motor (up/down/stopped)

vertical brake (on/off)

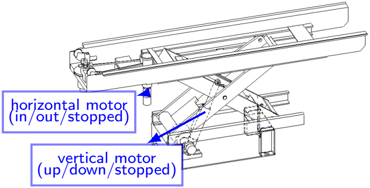
down end-sensor (on/off)

Emergency (on/off)

Light visor (on/off)

10.368 states

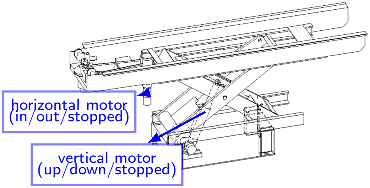
Example: MRI scanner



horizontal motor
(in/out/stopped)

vertical motor
(up/down/stopped)

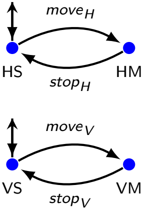
Example: MRI scanner



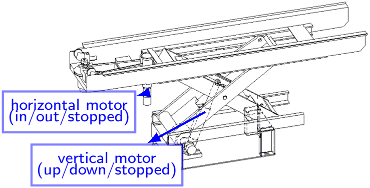
horizontal motor (in/out/stopped)

vertical motor (up/down/stopped)

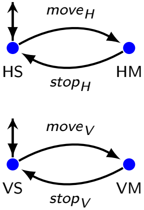
Plant model



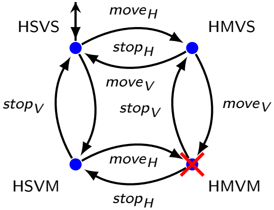
Example: MRI scanner



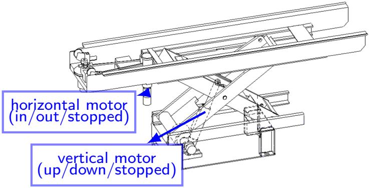
Plant model



Uncontrolled behavior



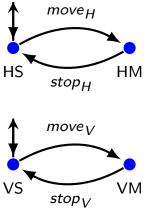
Example: MRI scanner



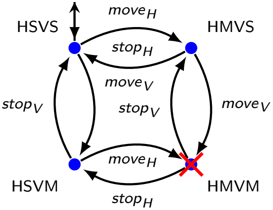
horizontal motor
(in/out/stopped)

vertical motor
(up/down/stopped)

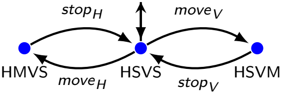
Plant model



Uncontrolled behavior



Supervised behavior



Plant, requirements, and control

A plant consists of a number of distributed components.

E.g., a bridge consists of a road deck, barriers, traffic lights, . . .

Requirements rule out undesired behavior.

E.g., the bridge should only be open if its barriers are closed.

A controller blocks (controllable) events that would lead to a violation of requirements.

Uncontrollable events

Some events are uncontrollable:

- An **input** event that cannot be blocked.
- An **urgent** event that should never be delayed.

A supervisor can only block controllable (typically **output**) events.

Marked locations

Marked locations in an automaton represent a complete work cycle.

They are used to express *liveness properties*:

> It must always be possible to reach a marked state, where all plant automata are in a marked location.

For example, for a bridge: the deck is closed, the barriers are open, and the lights are green.

Automaton

An automaton consists of:

- a set of *locations*
- an *initial* location
- *marked* locations
- a set of *events*, which are *controllable* or *uncontrollable*
- *transitions* between locations, carrying an event

Synchronous product of automata

The synchronous product $A \parallel B$ of automata A and B:

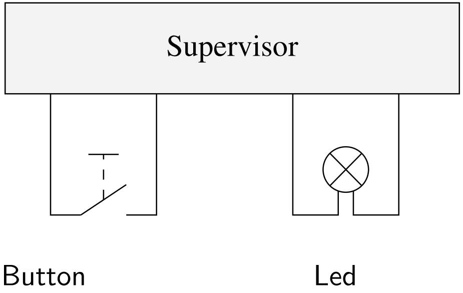
- if p and q are *locations* of A and B, then (p, q) is a *location* of $A \parallel B$
- if p and q are *initial*, then (p, q) is *initial*
- if p and q are *marked*, then (p, q) is *marked*
- if $p \xrightarrow{a} r$ and $q \xrightarrow{a} s$, then $(p, q) \xrightarrow{a} (r, s)$

Supervisor

A supervisor disables events of the plant, to meet 4 conditions:

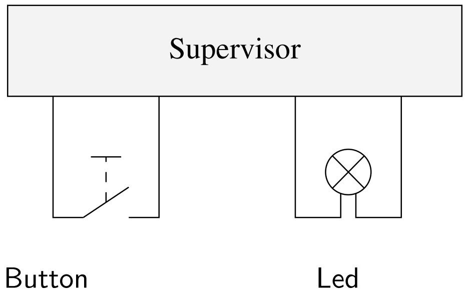
- Compliant: All requirements are satisfied.
- Controllable: Only controllable events are disabled.
- Non-blocking: From any reachable system state a *marked state* can be reached, with all plant automata in a marked location.
- Minimally restrictive: As few controllable events as possible are disabled.

Supervised plant



Pushing the button alternates between switching the light on and off.

Supervised plant



Pushing the button alternates between switching the light on and off.

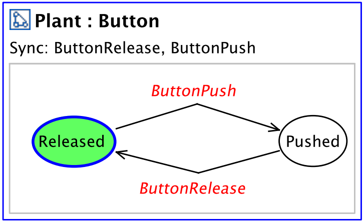
Initial locations are green.

Marked locations have a blue circle.

Controllable events are blue, uncontrollable events red.

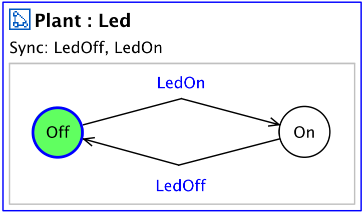
Plant automata

The plant is modeled by (the synchronous product of) automata.



Plant : Button

Sync: ButtonRelease, ButtonPush

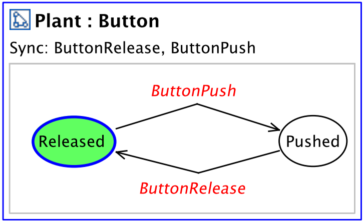


Plant : Led

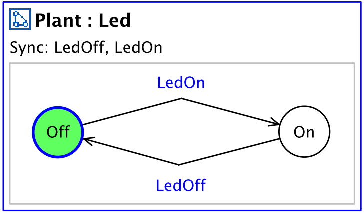
Sync: LedOff, LedOn

Plant automata

The plant is modeled by (the synchronous product of) automata.


Plant : Button

Sync: ButtonRelease, ButtonPush


Plant : Led

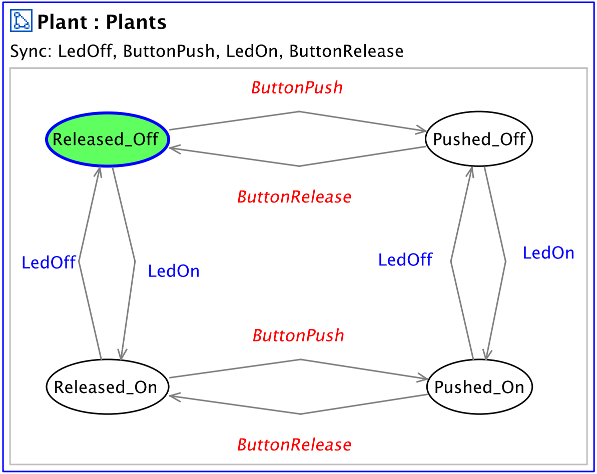
Sync: LedOff, LedOn

In the first automaton, `LedOn` and `LedOff` are absent from “Sync”.

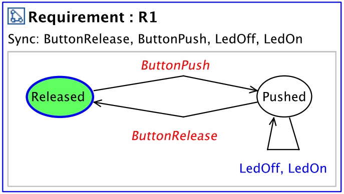
All locations in **Button** have implicit self-loops with these events.

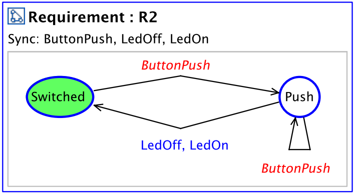
Likewise for `ButtonPush` and `ButtonRelease` in **Led**.

Synchronous product of plant automata



Requirement automata

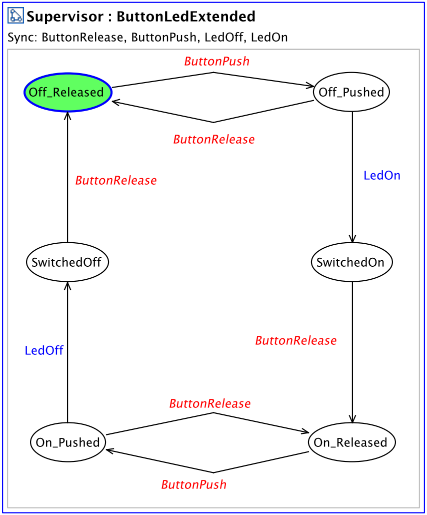




Requirement **R1** ensures that the light can only change when the button is down.

Requirement **R2** ensures that at most one light change can take place between button pushes.

Supervisor synthesis



Correct supervisor

Requirements **R1** and **R2** are enforced.

The supervisor is *compliant*, *controllable*, *non-blocking*, and *minimally restrictive*.

Correct supervisor

Requirements **R1** and **R2** are enforced.

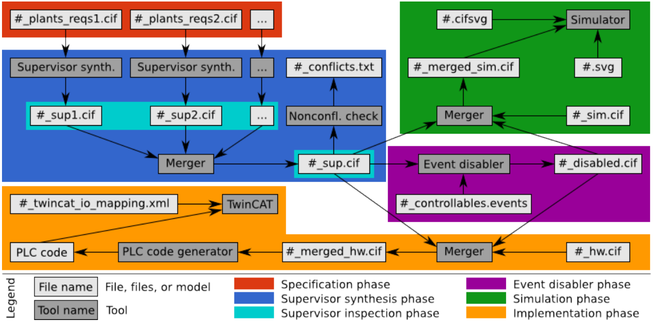
The supervisor is *compliant*, *controllable*, *non-blocking*, and *minimally restrictive*.

Here the supervisor happened to be non-blocking straight away.

Moreover, no uncontrollable event led to a removed state.

Generally, states with an uncontrollable event to a removed state need to be trimmed, leading to a cascade of removed states.

ESCET toolset

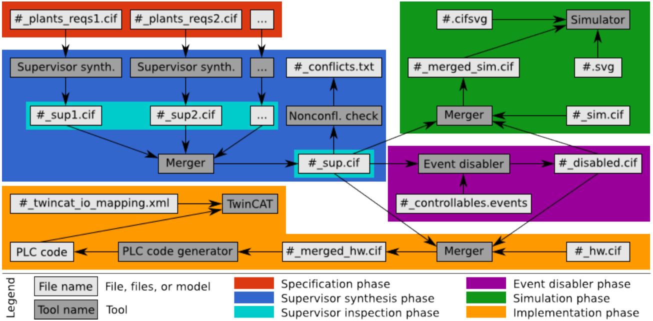


Wan Fokkink, Dennis Hendriks, Martijn Goorden, et al.

Eclipse ESCET^{TM}: The Eclipse Supervisory Control Engineering Toolkit

Proc. TACAS 2023

ESCET toolset



Wan Fokkink, Dennis Hendriks, Martijn Goorden, et al.

Eclipse ESCET^{TM}: The Eclipse Supervisory Control Engineering Toolkit

Proc. TACAS 2023

The toolset exploits optimizations from model checking, such as the BDD datastructure.

PhD students

Lennart Swartjes, *Model-Based Design of Baggage Handling Systems*, 2018

Martijn Goorden, *Supervisory Control System for Large-Scale Infrastructural Systems*, 2019

- multi-level supervisory control
- dependency structure matrices
- control problem dependency graph
- Princess Marijke waterway lock

PhD students

Lennart Swartjes, *Model-Based Design of Baggage Handling Systems*, 2018

Martijn Goorden, *Supervisory Control System for Large-Scale Infrastructural Systems*, 2019

- multi-level supervisory control
- dependency structure matrices
- control problem dependency graph
- Princess Marijke waterway lock

Ferdie Reijnen, *Putting Supervisor Synthesis to Work: Controller Software Generation for Infrastructural Systems*, 2020

- fault-tolerant supervisory control
- generation of a PLC controller
- safety PLCs
- revolving bridge Oisterwijksebaan

PhD students

Lars Moormann, 2022

- Heinenoord tunnel
- Koning Willem Alexander tunnel
- exploit symmetry of components
- digital twins

PhD students

Lars Moormann, 2022

- Heinenoord tunnel
- Koning Willem Alexander tunnel
- exploit symmetry of components
- digital twins

Jeroen Verbakel, 2024

- road information systems
- connecting supervisor synthesis and linear integer programming
- formal verification of communication protocols

PhD students

Lars Moormann, 2022

- Heinenoord tunnel
- Koning Willem Alexander tunnel
- exploit symmetry of components
- digital twins

Jeroen Verbakel, 2024

- road information systems
- connecting supervisor synthesis and linear integer programming
- formal verification of communication protocols

Marzhan Baubekova, Marijn Minkenberg: storm surge barriers

PART II

Multi-level supervisory control

Research challenge: To tame the size of the model

Models tend to get very large (say 10^{100} states), because

- real-life systems are complicated, with many components

  (e.g., multiple lock chambers, lock-bridge combinations, tunnels)

- fault-tolerant control is needed (e.g., Algera bridge)

- safety PLCs are used (e.g., Oisterwijksebaan bridge)

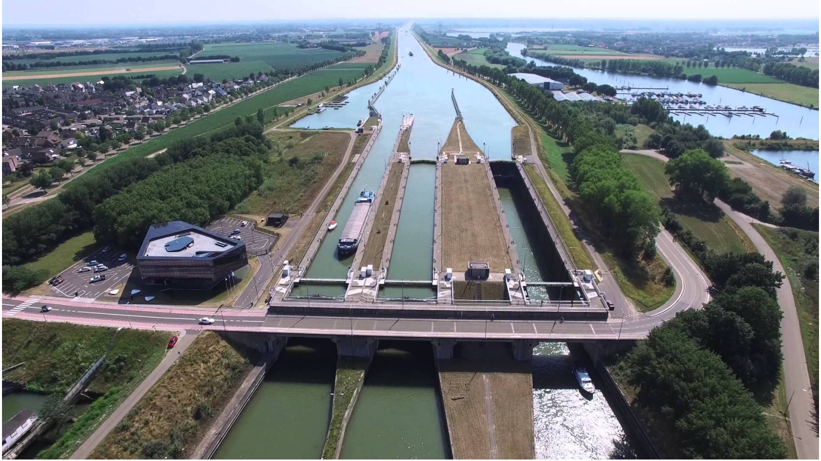
We encounter computational limits with supervisor synthesis, notably out-of-memory issues.

Waterway locks in the Netherlands

There are 16 sea locks and numerous fresh water locks in the Netherlands.

- *Schutsluis* : allows ships to overcome a height difference.
- *Keersluis* : regulates the water level.
- *Uitwateringssluis* : lets excess water flow away during low tide.
- *Spuisluis* : avoids silting up of a river.
- *Inundatiesluis* : can flood an area for military protection.

Waterway lock at Maasbracht



Simplified lock system

The following components at both sides of the lock, are modeled in 10 plant automata:

- entering light
- leaving light
- door actuator

  door sensor
- sewer actuator

  sewer sensor
- equal water sensor

The safety requirements are modeled in 26 requirement automata.

Matrix of plant and requirement automata

Identify the relationships between the plant automata and the requirement automata.

A requirement is related to a plant if it refers to an event in the plant automaton.

PR	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1						1	1	1	1		1															
2						1				1	1	1	1													
3		1	1		1		1	1	1	1		1	1				1	1								
4	1			1										1	1											
5			1																							
6																			1	1	1	1		1		
7																			1				1	1	1	1
8				1	1										1	1		1		1	1	1	1		1	1
9	1	1												1			1									
10																1										

Design Structure Matrix (DSM) of plant automata

Two plant automata are related if there exists a requirement mentioning them both.

P	1	2	3	4	5	6	7	8	9	10
1	5	2	3							
2	2	5	3							
3	3	3	11		1			2	2	
4				4				2	2	
5			1		1					
6						5	2	3		
7						2	5	3		
8			2	2		3	3	11		1
9			2	2					4	
10								1		1

Clustered DSM

Cluster closely related plant automata together.

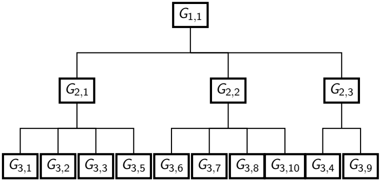
P_C	1	2	3	5	6	7	8	10	4	9
1	5	2	3							
2	2	5	3							
3	3	3	11	1			2			2
5			1	1						
6					5	2	3			
7					2	5	3			
8			2		3	3	11	1	2	
10							1	1		
4							2		4	2
9			2						2	4

Multi-level discrete event system

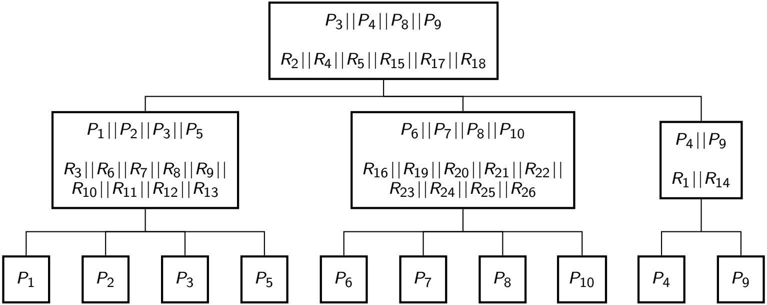
The clustered DSM is transformed into a tree structure.

Each requirement R is assigned to the lowest node that includes all plants related to R.

P	1	2	3	5	6	7	8	10	4	9
1	5	2	3							
2	2	5	3							
3	3	3	11	1			2			2
5			1	1						
6					5	2	3			
7					2	5	3			
8			2		3	3	11	1	2	
10							1	1		
4							2		4	2
9			2						2	4



Multi-level discrete event system in more detail



$P_3||P_4||P_8||P_9$
$R_2||R_4||R_5||R_{15}||R_{17}||R_{18}$
$P_1||P_2||P_3||P_5$
$R_3||R_6||R_7||R_8||R_9||$
$R_{10}||R_{11}||R_{12}||R_{13}$
$P_6||P_7||P_8||P_{10}$
$R_{16}||R_{19}||R_{20}||R_{21}||R_{22}||$
$R_{23}||R_{24}||R_{25}||R_{26}$
$P_4||P_9$
$R_1||R_{14}$
P_1
P_2
P_3
P_5
P_6
P_7
P_8
P_{10}
P_4
P_9

Instead of 1 monolithic supervisor, we get 4 multi-level supervisors.

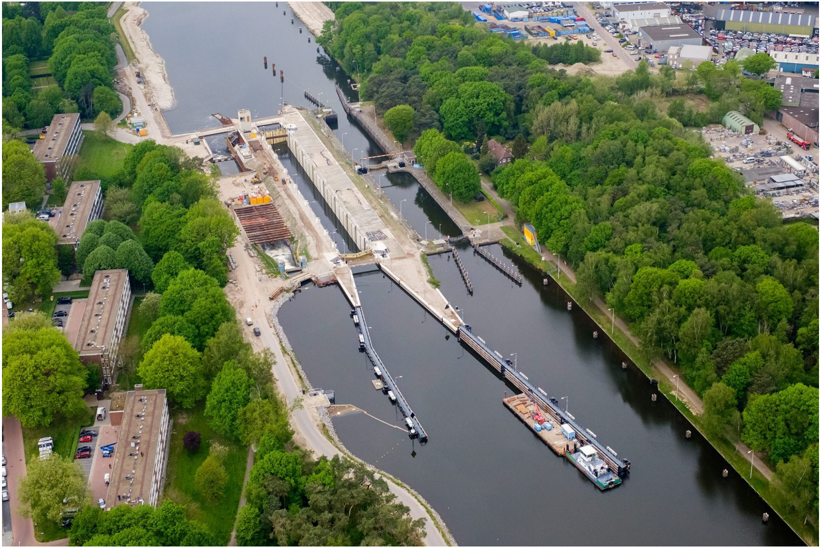
Bus structure

A bus structure, connecting many components, tends to be placed at the top in the multi-level approach.

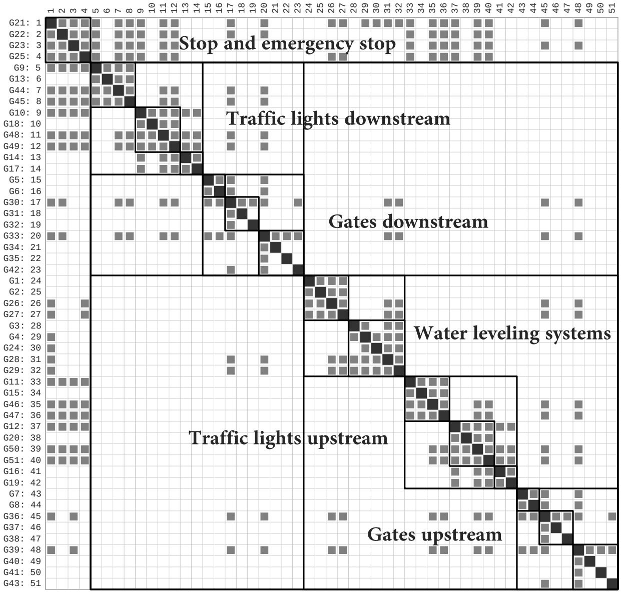
This results in a large supervisor associated to the top node.

Therefore the bus and non-bus components are kept separate in two trees, which are joined at an empty root node.

Waterway lock at Tilburg



DSM of the waterway lock at Tilburg



Numerical results for the lock at Tilburg

Model	$\lvert P\rvert$	$\lvert R\rvert$	monolithic	no bus	with bus	ns
lock at Tilburg	51	198	$6.0 \cdot 10^{24}$	$3.1 \cdot 10^{9}$	$7.7 \cdot 10^{6}$	34

$\lvert P\rvert$ = number of plants $\lvert R\rvert$ = number of requirements ns = number of supervisors

Splitting requirements

Both in the DSM approach and the check for non-blockingness, a requirement establishes relations between plant automata.

The fewer relations, the better these methods work.

It is advisable to split a requirement into smaller requirements.

This gives rise to fewer relations between plant models.

Requirement for opening a gate in the lock at Tilburg

It is *unsafe* to open a gate when

1. the water-leveling system at the other side is not closed, or
2. the gate at the other side is not closed, or
3. there is no equal water over the gate, or
4. the incoming traffic light at that lock head is not showing a red or red-red aspect, or
5. the outgoing traffic light at that lock head is not showing a red aspect.

Split the requirement into 17 parts

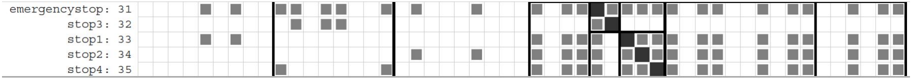
This requirement can be split into 17 requirements:

- The condition can be split into its 5 disjuncts.
- At places it refers to multiple system states.

  E.g. “not closed” or “not a red aspect”.
- At places it refers to multiple system components.

  E.g. it addresses multiple lights.

A closer look at the DSM of the lock at Tilburg



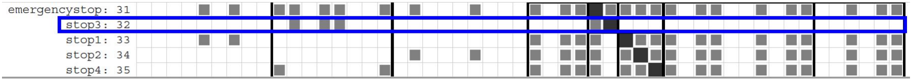
emergencystop: 31					■		■			■	■		■	■			■		■				■				■		■	■	■	■	■	■	■	■		■	■			■		■	■			■		■	■
stop3: 32											■		■	■																	■	■																			
stop1: 33					■		■																				■		■	■	■		■	■	■	■		■	■			■		■	■			■		■	■
stop2: 34																			■				■				■		■	■	■		■	■	■	■		■	■			■		■	■			■		■	■
stop4: 35										■							■										■		■	■	■		■	■	■	■		■	■			■		■	■			■		■	■

Consider only the stops and emergency stop of the original DSM.

The goal of each stop in the system is similar.

But stop3 had significantly fewer dependencies than the other stops.

A closer look at the DSM of the lock at Tilburg



emergencystop: 31
stop3: 32
stop1: 33
stop2: 34
stop4: 35

Consider only the stops and emergency stop of the original DSM.

The goal of each stop in the system is similar.

But stop3 had significantly fewer dependencies than the other stops.

A typo was identified: At several places, stop4 should be stop3.

Three modeling guidelines based on DSMs

1. *Similar components should have similar relationships in a DSM.*

This led to the detection of flaws in plant and requirement models.

2. *A DSM should not contain any empty row or column.*

This led to the detection of wrongly formulated requirements.

3. *A DSM should not have independent clusters.*

This led to the detection of missing requirements.

Conclusions

Supervisory control synthesis generates a controller from automata.

State space explosion is combatted with several methods:

- multi-level synthesis based on DSMs
- special treatment of bus components
- splitting requirements

This is employed at Rijkswaterstaat for automatic generation of controller software at large-scale infrastructural projects.

Fokkink, Goorden, van de Mortel, Reijnen, Rooda
Supervisor synthesis: Bridging theory and practise
Computer, 2022

Oisterwijksebaan bridge

A small demo of the revolving Oisterwijksebaan bridge in operation:

https://www.youtube.com/watch?v=sdVGl51SVBI

Reijnen, Leliveld, van de Mortel, van Dinther, Rooda, Fokkink
Synthesized fault-tolerant supervisory controllers, with an application to a rotating bridge
Computers in Industry, 2021