Synthesis-Based Engineering of Supervisors for System Product Lines

Michel Reniers

joint work with Sander Thuijsman

Eindhoven University of Technology, April 10, 2024

Who am I?

- Master and PhD from TU/e in Computer Science
- Associate professor in Control Systems Technology group of Mechanical Engineering
- Associate editor of Automatica, Discrete Event Dynamic Systems and Open Journal of Control Systems
- Research topics:
    - Model-based Engineering of Supervisory Control
    - Discrete-event systems
    - Supervisory control theory and synthesis
    - Simulation-based performance analysis

Motivation

- until 2050 in NL 37 locks reach EoL and additionally 15 have too low capacity
- investment 2-4 billion €
- in the past, each lock separate process also for (supervisory) control part

Overview

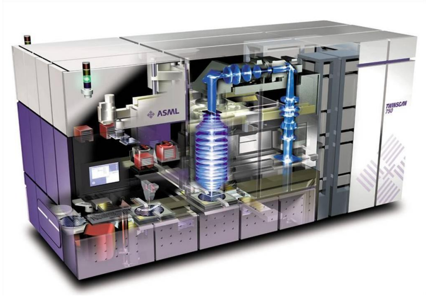
Supervisory control

Synthesis-based engineering

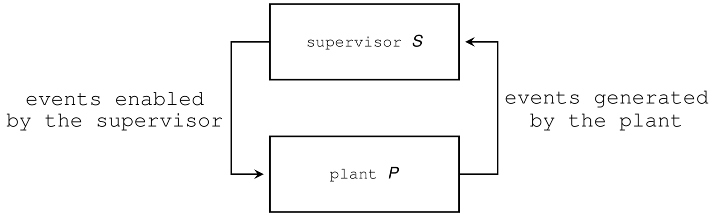
Supervisory controller synthesis

Supervisory control synthesis for product lines

What is supervisory control and where do you encounter it?



Supervisory control loop


supervisor S
plant P
events enabled by the supervisor
events generated by the plant

- Plant automaton describes the physically possible behavior of the system to be controlled
- Requirements describe allowed states or event sequences
- Supervisor should prevent these by exercising control (i.e., disabling events)
- Interaction between supervisor and plant is by synchronizing shared events

Supervisory controller synthesis

Based on model of plant and requirements, supervisory control synthesis provides a supervisory controller that is

1. *Safe*: the requirements are always adhered to
2. *Nonblocking*: a marked state can always be reached
3. *Controllable*: the supervisor does not disallow uncontrollable events to occur
4. *Maximally permissive*: no behavior is disabled that does not strictly need to be disallowed to satisfy the aforementioned properties

[1] P. J. Ramadge, W. M. Wonham, Supervisory control of a class of discrete event processes, SIAM Journal on Control and Optimization 25 (1) (1987)

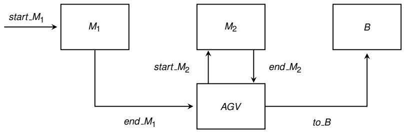
[2] L. Ouedraogo et al, Nonblocking and safe control of discrete-event systems modeled as extended finite automata, IEEE TASE 8 (3) (2011)

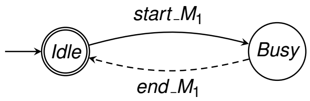
Basic supervisory control problem

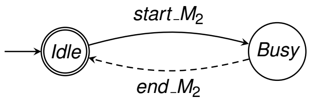
$$\boxed{\text{For plant } P\text{, find a maximally permissive proper supervisor } S}$$

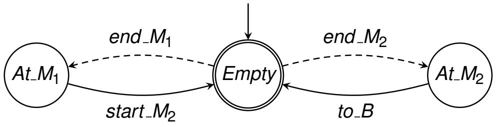
- Supervisory control synthesis is a method / algorithm that delivers a specific maximally permissive proper supervisor for a given plant, if it exists
- It uses the uncontrolled system as a starting point

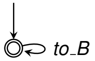
Synthesis by example: Workcell with an AGV



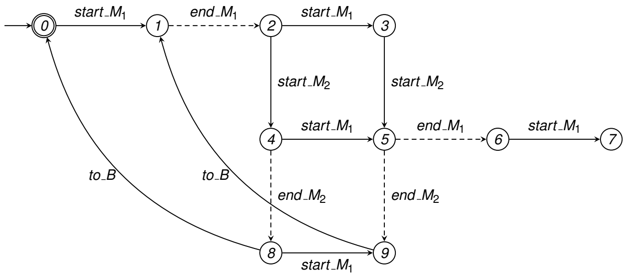






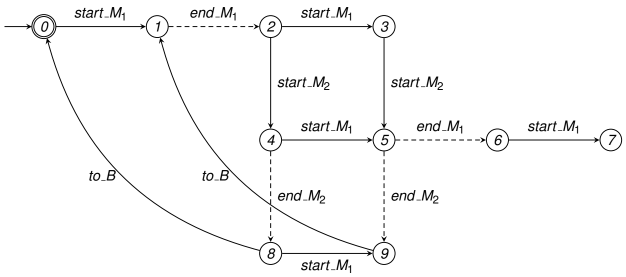


Workcell with AGV: uncontrolled system

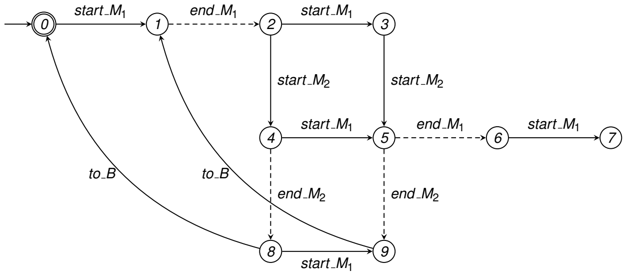


	0	1	2	3	4	5	6	7	8	9
M_1	**Idle**	**Busy**	**Idle**	**Busy**	**Idle**	**Busy**	**Idle**	**Busy**	**Idle**	**Busy**
M_2	**Idle**	**Idle**	**Idle**	**Idle**	**Busy**	**Busy**	**Busy**	**Busy**	**Idle**	**Idle**
AGV	**Empty**	**Empty**	**At_M$_1$**	**At_M$_1$**	**Empty**	**Empty**	**At_M$_1$**	**At_M$_1$**	**At_M$_2$**	**At_M$_2$**

Is the uncontrolled system nonblocking?

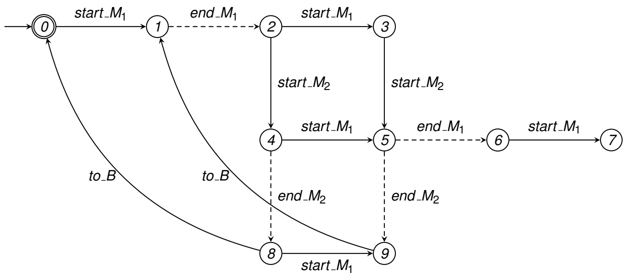


Is the uncontrolled system nonblocking?



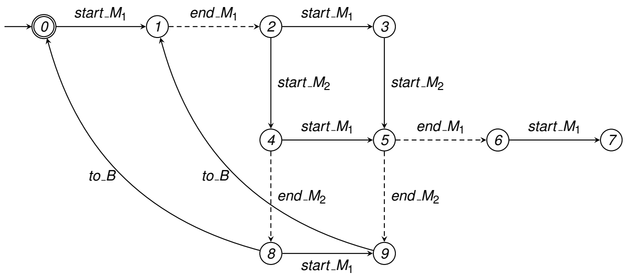
- Location 7 is blocking!

Is the uncontrolled system nonblocking?



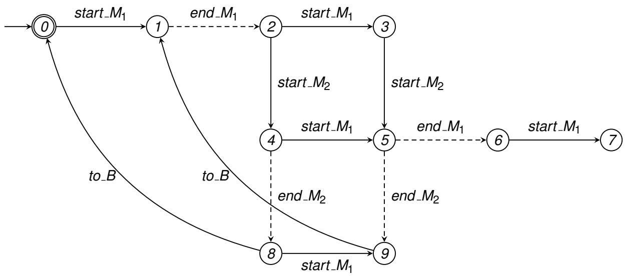
- Location 7 is blocking!
- Location 6 is blocking!

Is the uncontrolled system nonblocking?



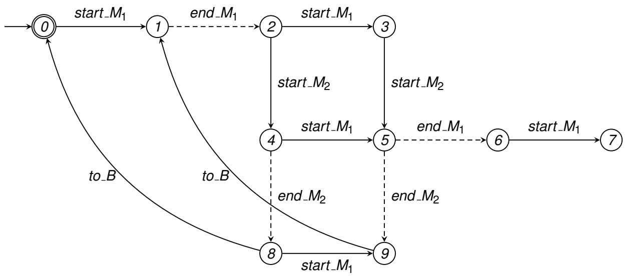
- Location 7 is blocking!
- Location 6 is blocking!
- Disabling event *start_M1* from location 6 solves blocking of location 7

Is the uncontrolled system nonblocking?



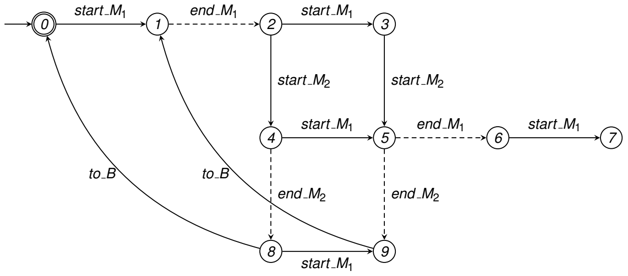
- Location 7 is blocking!
- Location 6 is blocking!
- Disabling event *start_M1* from location 6 solves blocking of location 7
- Event *end_M1* is uncontrollable and may not be prevented

Is the uncontrolled system nonblocking?



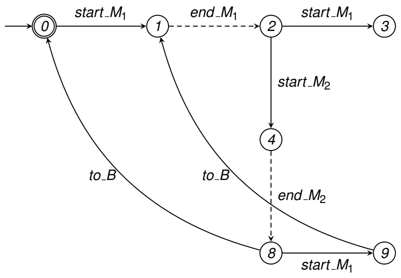
- Location 7 is blocking!
- Location 6 is blocking!
- Disabling event *start_M1* from location 6 solves blocking of location 7
- Event *end_M1* is uncontrollable and may not be prevented
- Location 5 is a bad state and should not be entered!

Is the uncontrolled system nonblocking?

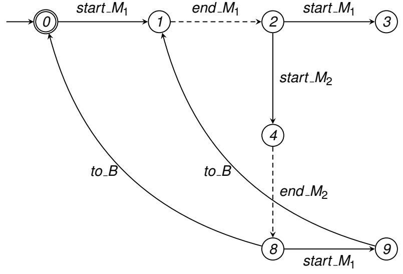


- Location 7 is blocking!
- Location 6 is blocking!
- Disabling event *start_M1* from location 6 solves blocking of location 7
- Event *end_M1* is uncontrollable and may not be prevented
- Location 5 is a bad state and should not be entered!
- Prevent incoming events *start_M1* and *start_M2*

Intermediate result

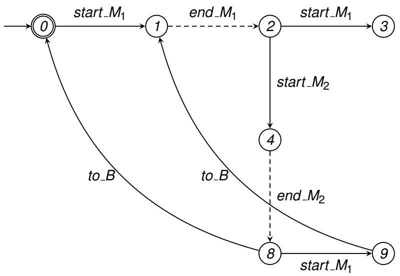


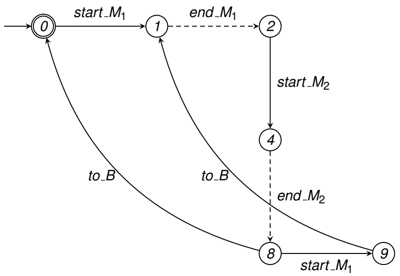
Intermediate result



Repeat the previous reasoning!

Intermediate result





Repeat the previous reasoning!

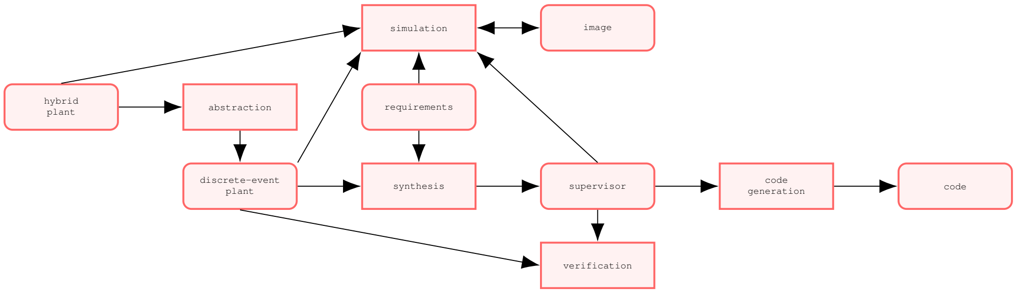
Iterative synthesis procedure

Starting from the uncontrolled system as a candidate supervisor

Repeat the following steps:

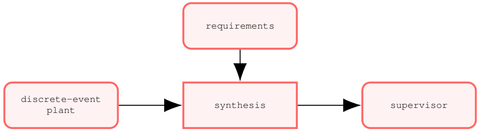
1. Compute set of *blocking* states. If no blocking states are found, synthesis has finished, otherwise continue
2. Compute set of *bad* states, which are the blocking states and all states from which a bad state can be reached by a sequence of uncontrollable events
3. Remove all controllable events that target a bad state
4. Remove unreachable locations and edges between those

SBE workflow for supervisory controllers



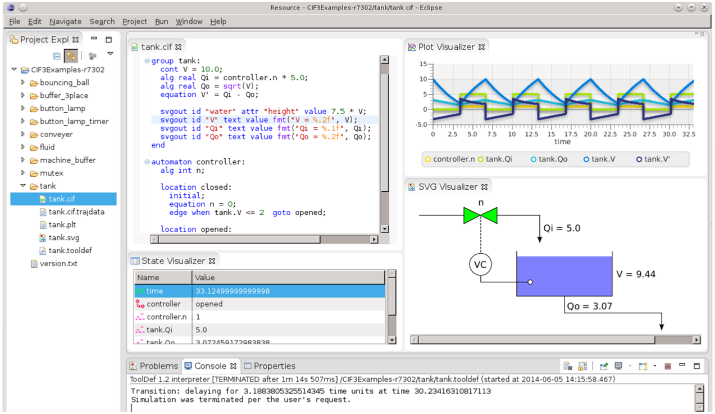
[3] D. A. van Beek et al, CIF 3: Model-based engineering of supervisory controllers, TACAS, 2014

SBE workflow for supervisory controllers



[3] D. A. van Beek et al, CIF 3: Model-based engineering of supervisory controllers, TACAS, 2014

Tool support: CIF as part of ESCET



```
group tank:
  cont V = 10.0;
  alg real Qi = controller.n * 5.0;
  alg real Qo = sqrt(V);
  equation V' = Qi - Qo;

  svgout id "water" attr "height" value 7.5 * V;
  svgout id "V" text value fmt("V = %.2f", V);
  svgout id "Qi" text value fmt("Qi = %.1f", Qi);
  svgout id "Qo" text value fmt("Qo = %.2f", Qo);
end

automaton controller:
  alg int n;

  location closed:
    initial;
    equation n = 0;
    edge when tank.V <= 2  goto opened;

  location opened:
```

Name	Value
time	33.12499999999998
controller	opened
controller.n	1
tank.Qi	5.0
tank.Qo	3.072459172983838

```
ToolDef 1.2 interpreter [TERMINATED after 1m 14s 507ms] /CIF3Examples-r7302/tank/tank.tooldef (started at 2014-06-05 14:15:58.467)
Transition: delaying for 3.1883805325514345 time units at time 30.23416310817113
Simulation was terminated per the user's request.
```

† ESCET toolset and documentation is open source and freely available at `eclipse.org/escet`
[4] W. J. Fokkink et al, Eclipse ESCET™: The Eclipse Supervisory Control Engineering Toolkit, TACAS, 2023

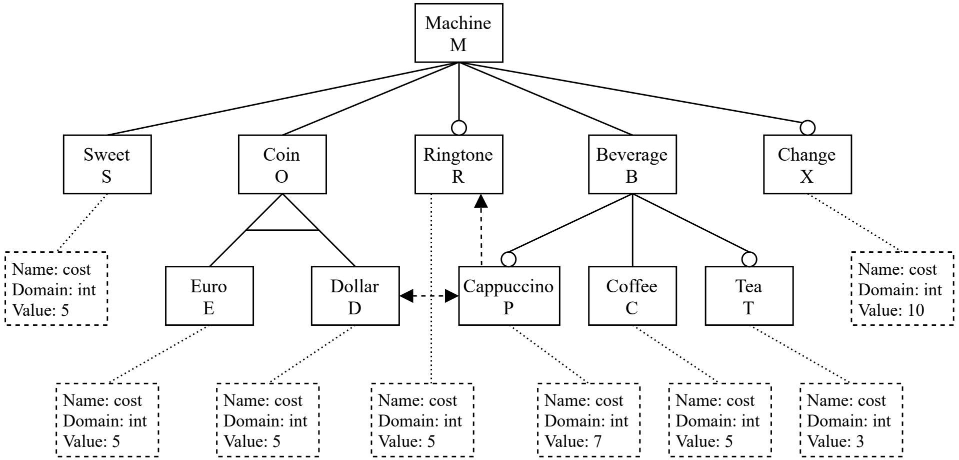
Supervisory Control Synthesis for Product Lines

This approach consists out of the following steps:

1. *Representing the feature model in extended finite automata*
2. *Capturing dynamic configuration of features in the models*
3. *Modeling uncontrolled system behavior taking the configuration into account*
   - component-wise specification of system behavior
   - component behavior (events) linked to the presence of features
4. *Modeling requirements depending on presence of features*
   - specification of behavioral requirements
   - specification of requirements for transitional phase during reconfiguration
5. *Applying supervisory controller synthesis*

[5] M. ter Beek, M. Reniers, E. de Vink, Supervisory controller synthesis for product lines using CIF 3, ISoLA 2016
[6] M. Reniers and S. Thuijsman, Supervisory control for dynamic feature configuration in product lines, FDL, 2020
[7] S. Thuijsman, M. Reniers, Supervisory control for dynamic feature configuration in product lines, ACM TECS (2023)

Static feature models in CIF



Machine M
Sweet S
Coin O
Ringtone R
Beverage B
Change X
Euro E
Dollar D
Cappuccino P
Coffee C
Tea T
Name: cost Domain: int Value: 5
Name: cost Domain: int Value: 10
Name: cost Domain: int Value: 5
Name: cost Domain: int Value: 5
Name: cost Domain: int Value: 5
Name: cost Domain: int Value: 7
Name: cost Domain: int Value: 5
Name: cost Domain: int Value: 3

[8] M. ter Beek and E. de Vink, Using mCRL2 for the analysis of software product lines. FormaliSE 2014

Features and Feature constraints

```
plant def FEATURE():
  disc bool present in any;
  location: initial ; marked;
end

plant def FEATURE_ATTRIBUTED(alg int x):
  disc bool present in any;
  alg int cost = if present : x else 0 end;
  location: initial ; marked;
end

F1: FEATURE();
F2: FEATURE_ATTRIBUTED(7);
...


alg bool r1 = F0.present <=> true; //root
alg bool r2 = F1.present <=> F2.present; //mandatory
alg bool r3 = F2.present => F1.present; //optional
alg bool r4 = (F1.present <=> (not(F2.present) and F.present)) and (F2.present <=> (not(F1.present) and F.
      present)); //alternative
alg bool r5 = F.present <=> (F1.present or F2.present); //or
alg bool r6 = F1.present => F2.present; //requires
alg bool r7 = not (F1.present and F2.present); //excludes
```

Valid configurations

```
alg bool sys_valid = r1 and r2 and r3 and ...;

alg int cost_sum = F2_ATTRIBUTED.cost+F3_ATTRIBUTED.cost+F5_ATTRIBUTED.cost;
alg bool cost_valid = cost_sum <= 30;

plant automaton Validity:
  location: initial sys_valid and cost_valid; marked;
end
```

Features with parameters:

```
enum colordomain =  red, yellow, blue, NA;

plant def BallFeature(alg colordomain clr):
  disc bool present in any;
  alg colordomain color = if present : clr else NA end;
  location: initial; marked;
end

RedBall: BallFeature(red);
YellowBall: BallFeature(yellow);
```

Static feature model for coffee machine in CIF

```
plant def FEATURE():                     plant def FEATURE_ATTRIBUTED(alg int x):
  disc bool present in any;                disc bool present in any;
  location: initial ; marked;              alg int cost = if present : x else 0 end;
end                                        location: initial ; marked;
                                         end

FM, FO, FB : FEATURE();
FS, FR, FE, FD, FC : FEATURE_ATTRIBUTED(5);
FX : FEATURE_ATTRIBUTED(10);
FP : FEATURE_ATTRIBUTED(7);
FT : FEATURE_ATTRIBUTED(3);

alg bool r1 = FM.present <=> true;                  alg bool r8 = FP.present => FB.present;
alg bool r2 = FM.present <=> FS.present;            alg bool r9 = FB.present <=> FC.present;
alg bool r3 = FM.present <=> FO.present;            alg bool r10 = FT.present => FB.present;
alg bool r4 = FR.present => FM.present;             alg bool r11 = FP.present => FR.present;
alg bool r5 = FM.present <=> FB.present;            alg bool r12 = not(FD.present and FP.present);
alg bool r6 = FX.present => FM.present;
alg bool r7 = (FE.present <=> (not(FD.present) and FO.present))
                and (FD.present <=> (not(FE.present) and FO.present));

alg bool sys_valid = r1 and r2 and r3 and r4 and r5 and r6 and r7 and r8 and r9 and r10 and r11 and r12;

alg int cost_sum = FS.cost+FR.cost+FX.cost+FE.cost+FD.cost+FP.cost+FC.cost+FT.cost;
alg bool cost_valid = cost_sum <= 30;

plant automaton Validity:
  location: initial sys_valid and cost_valid; marked;
end
```

Dynamic configuration - Single feature reconfiguration

```
plant def FEATURE():
  uncontrollable come, go;
  disc bool present in any;
  location: initial; marked;
    edge come when not present do present := true;
    edge go when present do present := false;
end
```

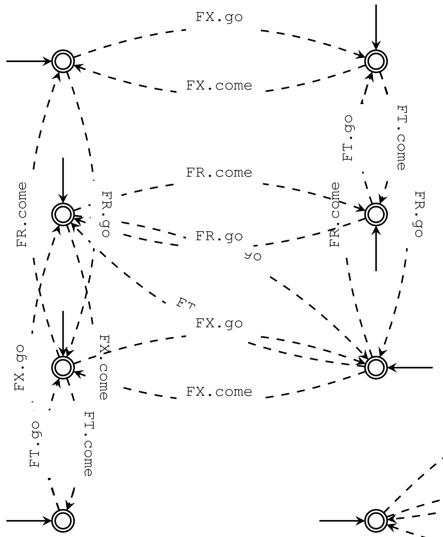
- `come` and `go` event for each feature (that can be removed or added)
- `come` and `go` events can be controllable or uncontrollable ⇒ system dependent (any mix is possible)
- if plant `Validity` is included only single feature reconfigurations from valid to valid configurations are possible
- if plant `Validity` is absent all (valid and invalid) configurations are included
- restricting reconfiguration to valid configurations is discussed later

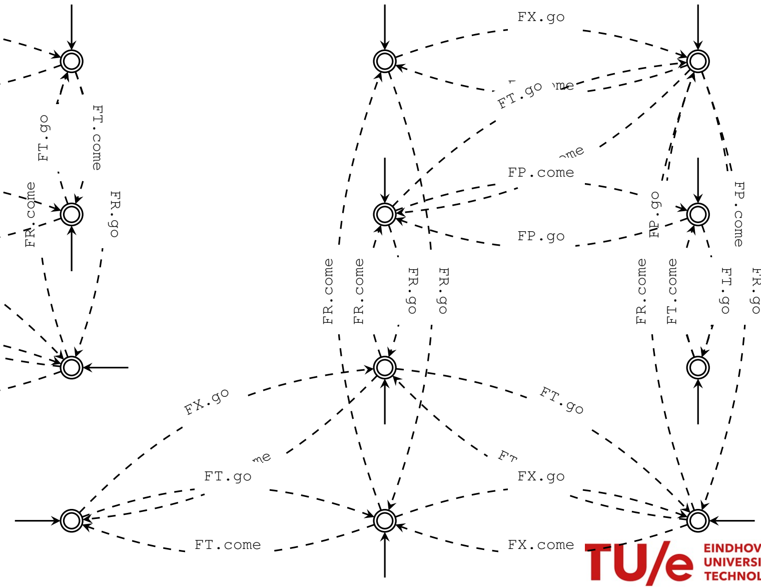
Dynamic configuration - Multi feature reconfiguration

- simultaneously remove a feature and add another feature without violating the feature constraint that exactly one of these is always present
- a global event (`swap`) can be introduced on which the automata (`F1` and `F2`) synchronize and their presence variables are updated

```
uncontrollable swap;

plant F1:                                                 plant F2:
  disc bool present in any;                                 disc bool present in any;
  location: initial ; marked;                               location: initial ; marked;
   edge swap when present do present:=not(present);          edge swap when present do present:=not present;
end                                                       end

alg bool r1 = (F1.present and not(F2.present)) or (F2.present and not(F1.present));
alg bool sys_valid = r1;

plant automaton Validity:
  location: initial sys_valid; marked;
end
```

Reconfiguration with strict feature configuration - Coffee machine





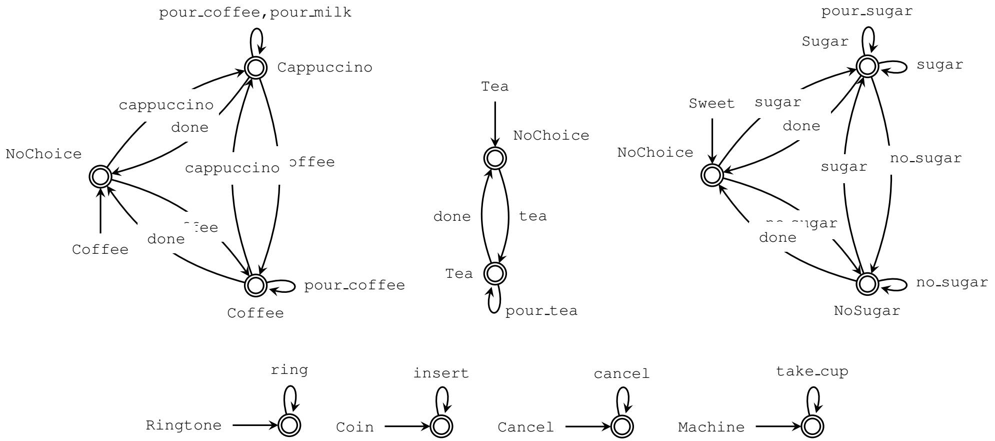
Constraints during reconfiguration

- One may allow invalid configurations to be reached
- for the coffee machine there may be a constraint when the change feature is present, the coin feature must always be present
- resulting state space consists of 1,364 states, among which 16 initial states. There are 13,440 come and go transitions
- Given the possibilities offered by CIF and the modularly defined feature constraints, it is possible to make more complex constraints

```
plant invariant FX.present => FO.present;
```

Modeling of uncontrolled behavior - Coffee system

The component-wise behavioral specification is taken from [5]



Linking events to features

- events cannot occur if they are ‘connected’ to features that are not present
- in coffee machine example, for each component there is a one-to-one correspondence with the features
- modeler should indicate for each event which features need to be present
- the availability of an event may also be dependent on the value of some attribute

```
plant automaton event_feature_conditions:
  location: initial; marked;
    edge e when F1.present and F2.A = x;
end
```

Connection between events and features for the coffee machine

```
plant automaton event_feature_link:
  location: initial; marked;
    edge Coffee.cappuccino when FC.present;
    edge Coffee.coffee when FC.present;
    edge Coffee.done when FC.present;
    edge Coffee.pour_coffee when FC.present;
    edge Coffee.pour_milk when FC.present;

    edge Tea.done when FT.present;
    edge Tea.pour_tea when FT.present;
    edge Tea.tea when FT.present;

    edge Sweet.done when FS.present;
    edge Sweet.no_sugar when FS.present;
    edge Sweet.pour_sugar when FS.present;
    edge Sweet.sugar when FS.present;

    edge Ringtone.ring when FR.present;

    edge Coin.insert when FO.present;

    edge Cancel.cancel when FX.present;

    edge Machine.take_cup when FM.present;
end
```

Specification of behavioral requirements - coffee machine

1. The coffee and tea component can not both be ready to pour:

```
requirement not(Coffee.Coffee and Tea.Tea);
```

2. Coffee can only be selected when no choice has been made yet:

```
requirement Coffee.coffee needs Coffee.NoChoice and Tea.NoChoice;
```

3. When the ringtone feature is present, it may only ring (once) after the coffee or tea component is finished:

```
requirement automaton RingAfterBeverageCompletion:
  location NotCompleted:
    initial; marked;
    edge Coffee.done when FR.present goto Completed;
    edge Tea.done when FR.present goto Completed;
    edge Coffee.done, Tea.done when not FR.present;
  location Completed:
    edge Ringtone.ring goto NotCompleted;
end
```

Requirements during configuration - coffee machine

1. It may be unsafe to cancel the order in case both the euro and dollar feature are present, as it is unclear from which feature the coin should be returned:

```
requirement Cancel.cancel needs not(FE.present and FD.present);
```

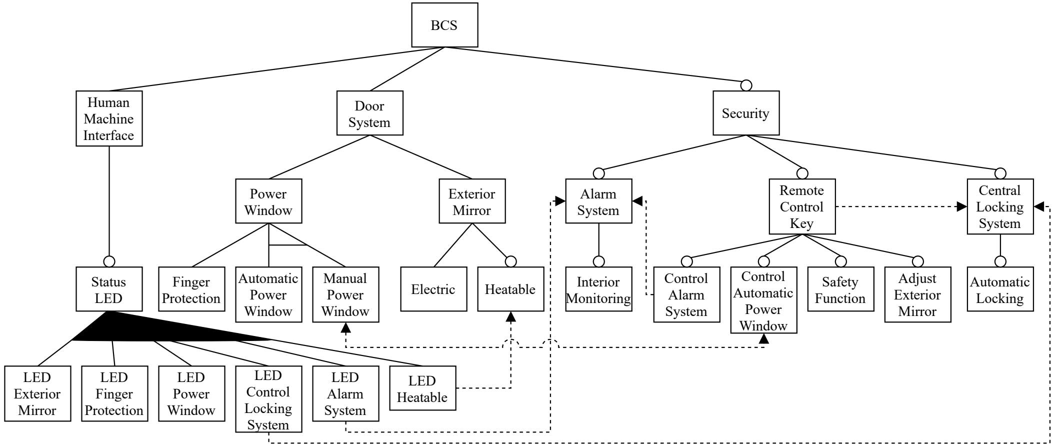
2. In case the sweet feature is ready to pour sugar, it should not be removed:

```
requirement Sweet.Sugar => FS.present;
```

Coffee machine, single feature reconfiguration, valid configurations

```
supervisor automaton sup:
  alphabet ...;
  location:
    initial;
    marked;
    edge Cancel.cancel when ...;
    edge Coffee.cappuccino when CoinPresence.CoinPresent and not Coffee.Coffee and (Tea.NoChoice and
      TakeCupWhenCoffeeOrTeaDone.NotPoured);
    edge Coffee.coffee when CoinPresence.CoinPresent and not Coffee.Cappuccino and (Tea.NoChoice and
      TakeCupWhenCoffeeOrTeaDone.NotPoured);
    edge Coffee.done when not Coffee.Cappuccino and CoffeePoured.Poured or Coffee.Cappuccino and (
      CoffeePoured.Poured and MilkPoured.Poured);
    edge Coffee.pour_coffee when CoffeePoured.NotPoured;
    edge Coffee.pour_milk when MilkPoured.NotPoured;
    edge Coin.insert when true;
    edge Machine.take_cup when true;
    edge Ringtone.ring when true;
    edge Sweet.done when true;
    edge Sweet.no_sugar when CoinPresence.CoinPresent and not Sweet.Sugar and (PourSugarTwice.Idle and
      TakeCupWhenSugarDone.NotPoured) or (CoinPresence.CoinPresent and (not Sweet.Sugar and PourSugarTwice.
      SugarNeeded) or CoinPresence.CoinPresent and (Sweet.Sugar and PourSugarTwice.count = 2));
    edge Sweet.pour_sugar when true;
    edge Sweet.sugar when CoinPresence.CoinPresent and TakeCupWhenSugarDone.NotPoured;
    edge Tea.done when true;
    edge Tea.pour_tea when TeaPoured.NotPoured;
    edge Tea.tea when CoinPresence.CoinPresent and (Coffee.NoChoice and TakeCupWhenCoffeeOrTeaDone.
      NotPoured);
end
```

Case study: Body Comfort System



[9] S. Lity, R. Lachmann, M. Lochau, and I. Schaefer, Delta-oriented Software Product Line Test Models - The Body Comfort System Case Study. Technical Report. TU Braunschweig, 2013

Numbers ...

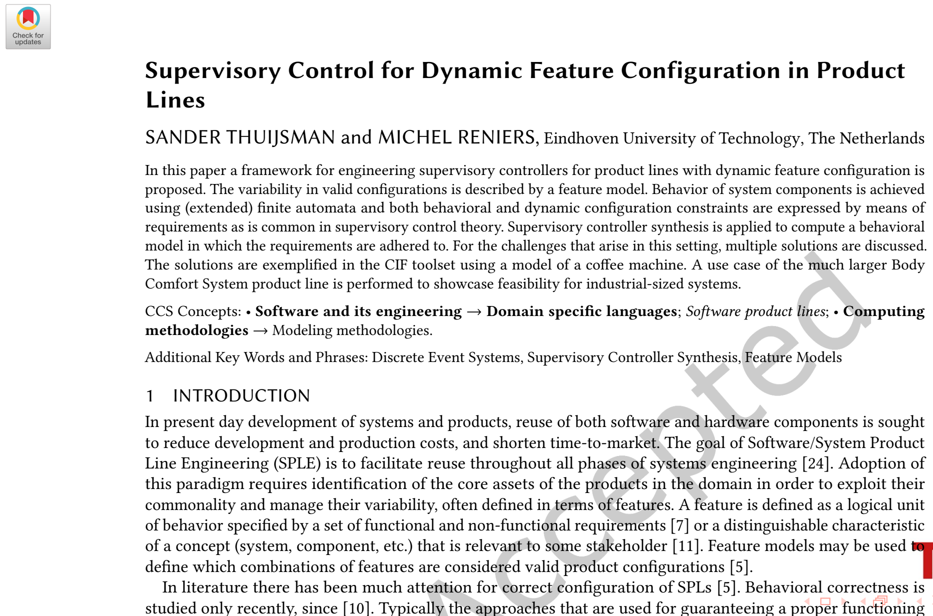
- 134,217,728 configurations from which 11,616 are valid
- 31 plant automata representing the behavior of the components, 27 feature automata, 18 plant automata that link the component events to the presence of the features, and 55 requirements
- supervisor is obtained in roughly 0.3 seconds and requires no more than 0.5 GB of memory

State space	**Number of states**
Worst-case	$7.7 \cdot 10^{20}$
Uncontrolled static	$3.2 \cdot 10^{14}$
Uncontrolled dynamic	$6.2 \cdot 10^{20}$
Controlled static	$7.6 \cdot 10^{13}$
Controlled dynamic	$1.1 \cdot 10^{20}$

Concluding Remarks

- Synthesis-based engineering of supervisory controllers for product lines described by a feature model
- Allowing dynamic reconfiguration
- Several solutions for control of/during reconfiguration
- CIF language shown to be adequate
- Presented small case study, feasibility shown for (much) larger Body Comfort System

... ACM Transactions on Embedded Computing Systems (2023)



Supervisory Control for Dynamic Feature Configuration in Product Lines

SANDER THUIJSMAN and MICHEL RENIERS, Eindhoven University of Technology, The Netherlands

In this paper a framework for engineering supervisory controllers for product lines with dynamic feature configuration is proposed. The variability in valid configurations is described by a feature model. Behavior of system components is achieved using (extended) finite automata and both behavioral and dynamic configuration constraints are expressed by means of requirements as is common in supervisory control theory. Supervisory controller synthesis is applied to compute a behavioral model in which the requirements are adhered to. For the challenges that arise in this setting, multiple solutions are discussed. The solutions are exemplified in the CIF toolset using a model of a coffee machine. A use case of the much larger Body Comfort System product line is performed to showcase feasibility for industrial-sized systems.

CCS Concepts: • **Software and its engineering** → **Domain specific languages**; *Software product lines*; • **Computing methodologies** → Modeling methodologies.

Additional Key Words and Phrases: Discrete Event Systems, Supervisory Controller Synthesis, Feature Models

1 INTRODUCTION

In present day development of systems and products, reuse of both software and hardware components is sought to reduce development and production costs, and shorten time-to-market. The goal of Software/System Product Line Engineering (SPLE) is to facilitate reuse throughout all phases of systems engineering [24]. Adoption of this paradigm requires identification of the core assets of the products in the domain in order to exploit their commonality and manage their variability, often defined in terms of features. A feature is defined as a logical unit of behavior specified by a set of functional and non-functional requirements [7] or a distinguishable characteristic of a concept (system, component, etc.) that is relevant to some stakeholder [11]. Feature models may be used to define which combinations of features are considered valid product configurations [5].

In literature there has been much attention for correct configuration of SPLs [5]. Behavioral correctness is studied only recently, since [10]. Typically the approaches that are used for guaranteeing a proper functioning

Synthesis-Based Engineering of Supervisors for System Product Lines

Michel Reniers

joint work with Sander Thuijsman

Eindhoven University of Technology, April 10, 2024