Using model checking techniques to analyze interface moding and timing problems in interactive systems

Michael Harrison, Karsten Loer & Jose Campos

The Interdisciplinary Research Collaboration in Dependability (DIRC)

York, Newcastle and University of Minho at Braga

iri

York HCI Group

- Alistair Edwards, Andrew Monk, Peter Wright
- Dependability Interdisciplinary Research Collaboration
  - Newcastle, City, Edinburgh, Lancaster, York
  - Six year project since 2000
- ADVISES
  - EU Human error in interactive systems
  - Also Glasgow, Liege, Paderborn, Pisa, Risoe,Toulouse
- Dependable home
  - Funding: Joseph Rowntree Trust, focus assisting elderly
- Focus:
  - Mathematically based models, structured methods, dependability arguments in a interdisciplinary context
- Now move to establish Informatics Research Institute, Newcastle: continuing dependability research

iri

DIRC

Overview

- Two examples:
  - Mode problem on flight-deck
  - Mobile device in the context of process control, using information relevant to spatial context to interpret user action
- The actions that the system might perform may depend on previous operator actions or context
- Require ways to check the design of such devices in order to understand these contexts better and the effect that they have
- Talk discusses the role that model checking can play and different modelling notations

Altitude bust problem (Palmer, Degani, Rushby)

- MCP influences aircraft ascent/descent depending on operating pitch mode
- VERT_SPD (vertical speed pitch mode): instructs the aircraft to maintain the climb rate indicated in the MCP (the airspeed will be adjusted automatically)
- IAS (indicated airspeed pitch mode): instructs the aircraft to maintain the airspeed indicated in the MCP (the climb rate will be adjusted automatically)
- ALT_HLD (altitude hold pitch mode): instructs the aircraft to maintain current altitude
- ALT_CAP (altitude capture mode): internal mode used by the aircraft to perform a smooth transition from VERT_SPD or IAS to ALT_HLD (see ALT below)
- A capture switch (ALT) when armed causes the aircraft to stop climbing when the altitude indicated in the MCP is reached

MCP (includes, attributes, actions)

interactor MCP
includes
aircraft **via** plane
dial(ClimbRate) **via** crDial
dial(Velocity) **via** asDial
dial(Altitude) **via** ALTDial
attributes
vis pitchMode: PitchModes
vis ALT: Boolean
actions
vis enterVS, enterIAS, enterAH, toggleALT, enterAC

MCP (action effects and permissions)

axioms

```
  # Action effects
   (0) [] plane.altitude = 0
   (1) [crDial.set(t)] pitchMode'=VERT_SPD ^ ALT'=ALT
   (2) [asDial.set(t)] pitchMode'=IAS ^ ALT'=ALT
   (3) [ALTDial.set(t)] pitchMode'=pitchMode ^ ALT'
   (4) [enterVS] pitchMode'=VERT_SPD ^ ALT'=ALT
   (5) [enterIAS] pitchMode'=IAS ^ ALT'=ALT
   (6) [enterAH] pitchMode'=ALT_HLD ^ ALT'=ALT
   (7) [toggleALT] pitchMode'=pitchMode ^ ALT' neq ALT
   (8) [enterAC] pitchMode'=ALT_CAP ^ ~ALT'
  # Permissions
   (9) per(enterAC) ->
              (ALT ^ (|ALTDial.needle - plane.altitude| <= 2))
```

MCP (obligations and invariants)

\# Obligations

(10) (ALT ^(|ALTDial.needle - plane.altitude| <= 2) -> obl(enterAC)

(11) (pitchMode=ALT_CAP ^ (plane.altitude=ALTDial.needle) -> obl(enterAH)

\# Invariants

(12) pitchMode=VERT_SPD -> plane.climbRate=crDial.needle

(13) pitchMode=IAS -> plane.airSpeed=asDial.needle

(14) pitchMode=ALT_HLD -> plane.climbRate=0

(15) (pitchMode=ALT_CAP ^ plane.altitude<ALTDial.needle) -> plane.climbRate=1

(16) (pitchMode=ALT_CAP ^ plane.altitude>ALTDial.needle) -> plane.climbRate=-1

Modelling the environment

```
interactor aircraft
    attributes
      altitude: Altitude
      airSpeed: Velocity
      climbRate: ClimbRate
    actions
      fly
    axioms
      [fly] (altitude' >=altitude - 1 ^ altitude' <=altitude + 1) ^
          (altitude' <altitude -> climbRate' <0) ^
          (altitude'=altitude -> climbRate'=0) ->
          (altitude' >altitude -> climbRate' >0)
```

Pilot expectation about how the system operates

- "Whenever the pilot sets the automation to climb up to a given altitude, the aircraft will climb until such altitude is acquired and then maintain it."
- Are there situations when this does not occur?
- Are there features of the design which might conspire against this happening?

- Rather than focus on the user's expectation or performance set constraints on the behaviours that are possible in order to explore whether there are possible areas in which the user might have problems

Checking constraints

- If the altitude capture (ALT) is armed the aircraft will stop climbing at the desired altitude (selected in ALTDial). This can be expressed as the CTL formula:
  - AG((plane.altitude<ALTDial.needle^ALT)->
    AF(pitchMode=ALT_HLD^
    plane.altitude=ALTDial.needle))
- A trace generates a situation in which the pilot continuously changes the climb rate when altitude armed
- An additional condition excludes the possibility of descending
  - AG((plane.altitude<ALTDial.needle^ALT)->
    AF((pitchMode=ALT_HLD^
    plane.altitude=ALTDial.needle)
    v (plane.climbRate=-1))

An interesting trace

- Checking leads to trace indicative that changing the pitch mode to VERT_SPD (for instance by setting the corresponding dial) when in ALT_CAP terminates the request to stop climbing at the target altitude
- When the pitch mode changes to ALT_CAP, altitude capture is switched off (see Axiom 8) even though the aircraft is still climbing.
- Subsequent pilot action causing change to pitch mode means aircraft climbs past the target altitude
- The counterexample prompts the designer to consider whether there is enough information provided by the MCP so that the pilot may be kept in the loop.
- No assumed model of pilot interacting with device, however trace highlights situation that may be of human factors concern

Property checking

- Exhaustive behavioural usability analysis of interactive systems
  - Moding, visibility, recoverability, consistency, predictability ...
  - Analysis typically performed by usability experts

- For “traditional” dependability there is often formal analysis of
  - system-theoretic properties: e.g. stability/continuity, robustness
  - dynamic temporal properties: safety, liveness, timing
- … and the analysis is performed by formal methods experts

- Several issues are *related*, e.g. recoverability and robustness

Formalising Usability Requirements

"Users often choose system functions by mistake and will need a clearly marked 'emergency exit' to leave the unwanted state […]. Support undo and redo." (Nielsen and Mack 94)

↓

In all possible execution paths it is possible to reach a previously visited *configuration* **after an <u>unwanted</u>** *user_input* **occurred.**

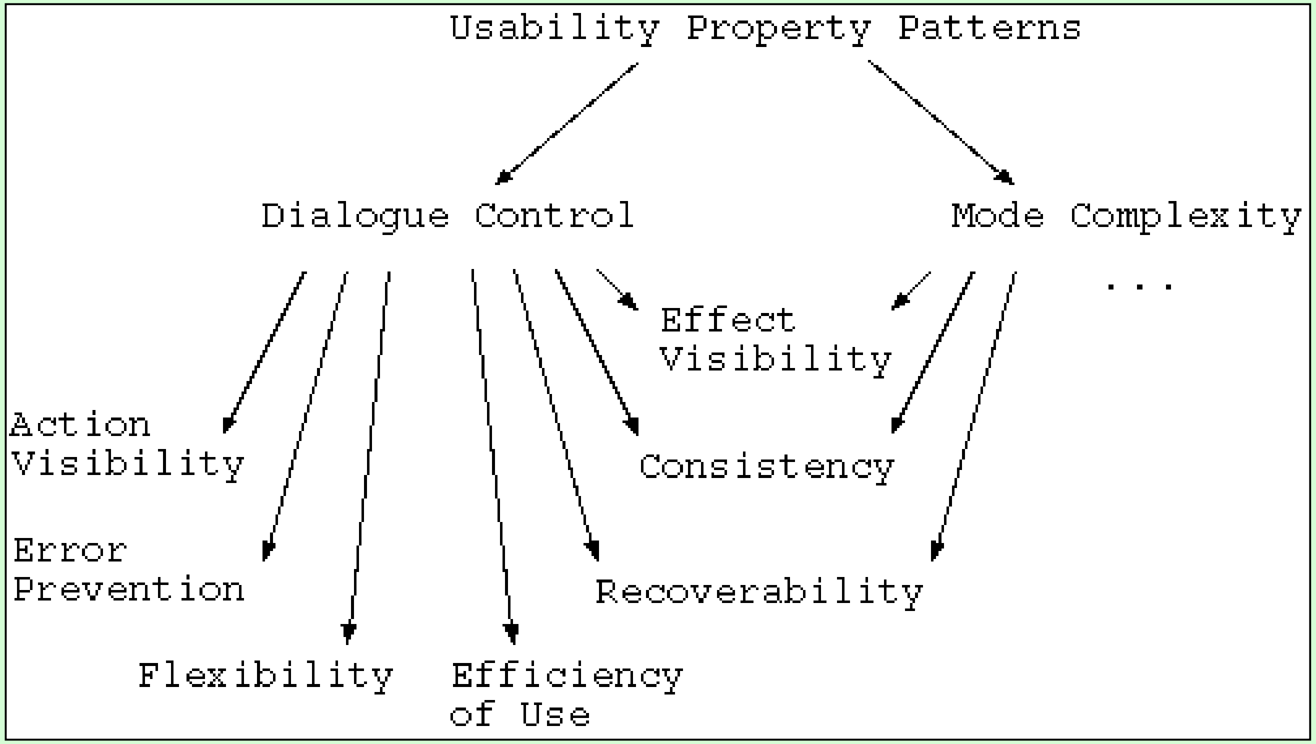
↓

recovery template:

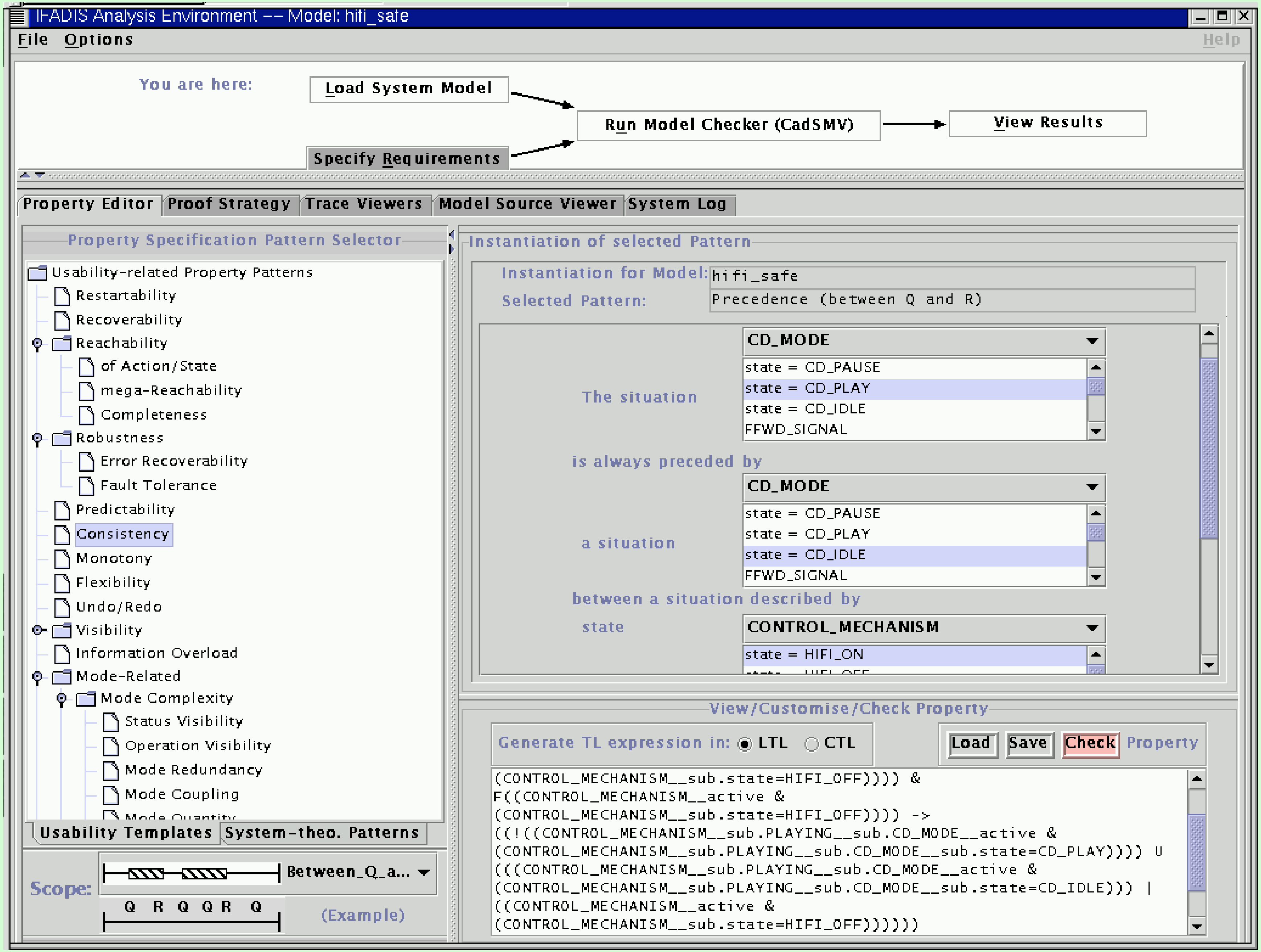
AG(<*configuration*> **&** <user_*input*> **)**
-> AX EF(<*configuration*> **)**

Templates for Temporal Logic properties

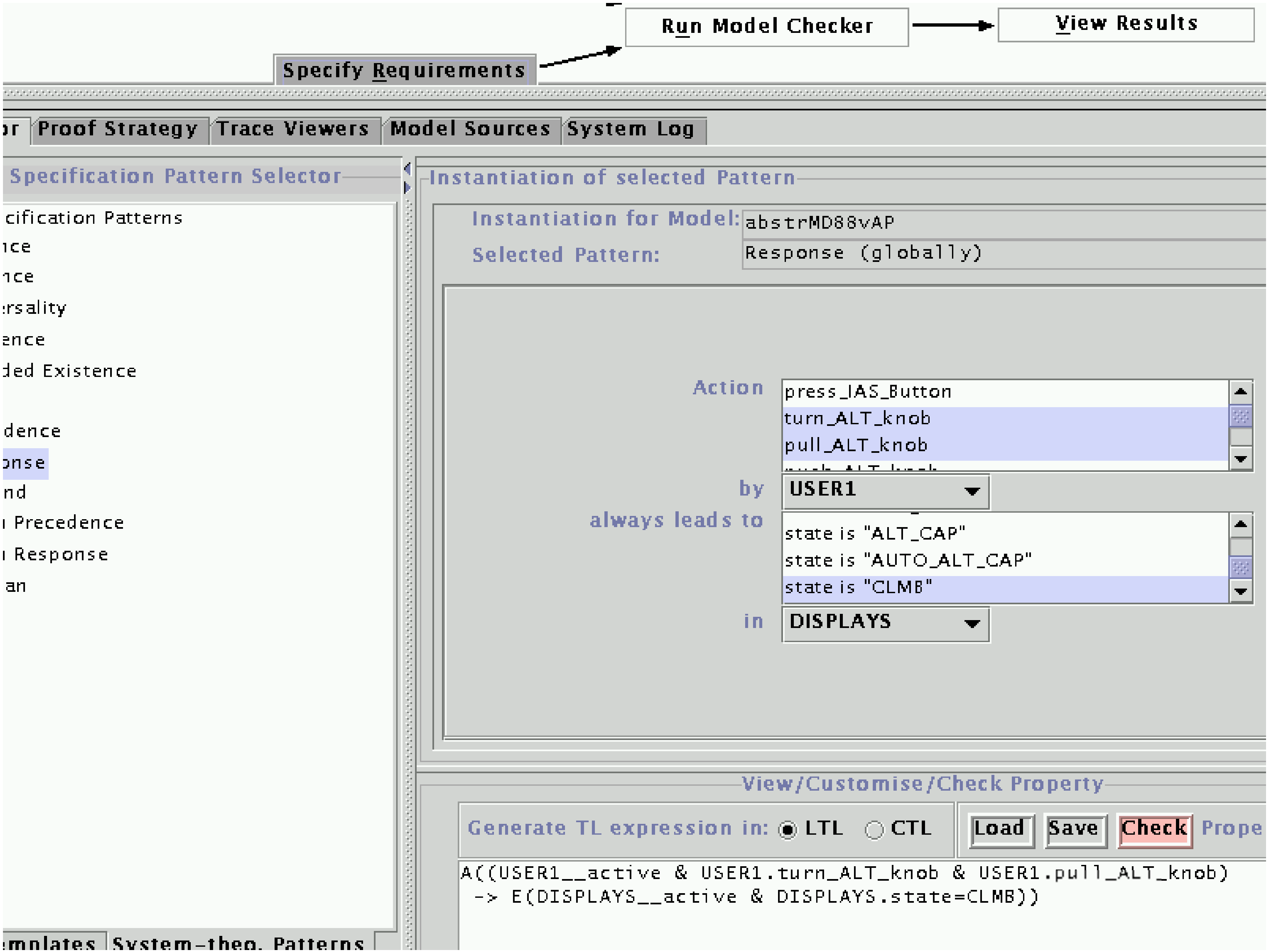
- Dwyer's templates can also be addressed from a usability point of view:



- Based on such templates a CTL property editor can be developed



IFADIS Analysis Environment -- Model: hifi_safe
File
Options
Help
You are here:
Load System Model
Run Model Checker (CadSMV)
View Results
Specify Requirements
Property Editor
Proof Strategy
Trace Viewers
Model Source Viewer
System Log
Property Specification Pattern Selector
Usability-related Property Patterns
Restartability
Recoverability
Reachability
of Action/State
mega-Reachability
Completeness
Robustness
Error Recoverability
Fault Tolerance
Predictability
Consistency
Monotony
Flexibility
Undo/Redo
Visibility
Information Overload
Mode-Related
Mode Complexity
Status Visibility
Operation Visibility
Mode Redundancy
Mode Coupling
Usability Templates
System-theo. Patterns
Scope:
Between_Q_a...
Q R Q Q R Q
(Example)
Instantiation of selected Pattern
Instantiation for Model: hifi_safe
Selected Pattern: Precedence (between Q and R)
The situation
CD_MODE
state = CD_PAUSE
state = CD_PLAY
state = CD_IDLE
FFWD_SIGNAL
is always preceded by
a situation
CD_MODE
state = CD_PAUSE
state = CD_PLAY
state = CD_IDLE
FFWD_SIGNAL
between a situation described by
state
CONTROL_MECHANISM
state = HIFI_ON
View/Customise/Check Property
Generate TL expression in: LTL CTL
Load
Save
Check
Property
(CONTROL_MECHANISM__sub.state=HIFI_OFF)))) &
F((CONTROL_MECHANISM__active &
(CONTROL_MECHANISM__sub.state=HIFI_OFF))) ->
((!((CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__active &
(CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__sub.state=CD_PLAY))) U
(((CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__active &
(CONTROL_MECHANISM__sub.PLAYING__sub.CD_MODE__sub.state=CD_IDLE)) |
((CONTROL_MECHANISM__active &
(CONTROL_MECHANISM__sub.state=HIFI_OFF)))))



Run Model Checker
View Results
Specify Requirements
Proof Strategy
Trace Viewers
Model Sources
System Log
Specification Pattern Selector
Instantiation of selected Pattern
Instantiation for Model:
abstrMD88vAP
Selected Pattern:
Response (globally)
Action
press_IAS_Button
turn_ALT_knob
pull_ALT_knob
by
USER1
always leads to
state is "ALT_CAP"
state is "AUTO_ALT_CAP"
state is "CLMB"
in
DISPLAYS
View/Customise/Check Property
Generate TL expression in:
LTL
CTL
Load
Save
Check
A((USER1__active & USER1.turn_ALT_knob & USER1.pull_ALT_knob)
-> E(DISPLAYS__active & DISPLAYS.state=CLMB))
System-theo. Patterns


Run Model Checker
View Results
Specify Requirements
r
Proof Strategy
Trace Viewers
Model Sources
System Log
Strategy for complex Proofs
Prove properties under certain assumptions.
roperty
reachCLMBviaALTknob
under assumptions
neverReset
onlyOneInputAt_a_time
onlyTwoInputsAtOnce
notCrash
neverALTbelowZero
alwaysBothPanelsAccessible
Strategy
Save Strategy
Run Model Checker
ary of this strategy:
eset: assert
A(!CTRL_MECH.reset);
LTbelowZero: assert
A(!ALT<0);
LMBviaALTknob: assert
A((User1__active & USER1.turn_ALT_knob & USER1.pull_ALT_knob)
-> E(DISPLAYS__active & DISPLAYS.state=CLMB));
neverReset, neverATLbelowZero prove reachCLMBviaALTknob;

```
/* Query */

AG(playing_state=CD_IDLE)&
AF(~PLAY_SIGNAL) -> (~EF
(playing_state=CD_PLAY))

/* state 1 */
CTRL_MECH.state         = OFF,
CTRL_MECH.playing_state = INACTIVE,
CTRL_MECH.CD_MODE       = 0,
USER.pressONOFF_Button  = 1,
CTRL_ELEM.ONOFF_SIGNAL  = 1,
USER.pressPAUSE_Button  = 0,
CTRL_ELEM.PAUSE_SIGNAL  = 0,
USER.pressPLAT_Button   = 0,
CTRL_ELEM.PLAY_SIGNAL   = 0,
DISPLAYS.AUDIO_state    = QUIET,
[...]

/* state 4 */
CTRL_MECH.state         = ON,
CTRL_MECH.playing_state = CD_IDLE,
CTRL_MECH.CD_MODE       = 1,
USER.pressPAUSE_Button  = 1,
CTRL_ELEM.PAUSE_SIGNAL  = 1,
USER.pressPLAT_Button   = 0,
CTRL_ELEM.PLAY_SIGNAL   = 0,
DISPLAYS.AUDIO_state    = QUIET,
[...]
/* state 5 */
CTRL_MECH.state         = OFF,
CTRL_MECH.playing_state = INACTIVE,
USER.pressPAUSE_Button  = 1,
CTRL_ELEM.PAUSE_SIGNAL  = 1,
USER.pressPLAT_Button   = 0,
CTRL_ELEM.PLAY_SIGNAL   = 0,
DISPLAYS.AUDIO_state    = QUIET,
[...]
/* state 6 */
CTRL_MECH.state         = OFF,
CTRL_MECH.playing_state = INACTIVE,
USER.pressPAUSE_Button  = 1,
CTRL_ELEM.PAUSE_SIGNAL  = 0,
USER.pressPLAT_Button   = 0,
CTRL_ELEM.PLAY_SIGNAL   = 0,
DISPLAYS.AUDIO_state    = MUSIC,
[...]
```

- Counter example
  - Can traces point to interaction problems?
  - Traces contain information about:
    - all system states that are relevant
    - users involved
    - environmental factors
- traces can be quite long and hard to read

Trace comparison

Trace Visualisation Tool - M. Kermelis v1.1

File Help

traffic lights

Options Help

specification AG (SWITCH_ON -> AF TRAFFIC_LIGHTS__sub.ON__sub.state = RED) is false

	State 1	State 2	State 3	State 4	State 5	State 6
TRAFFIC_LIGHTS__active	1	1	1	1	1	1
state	TRAFFIC_L...	TRAFFIC_L...	TRAFFIC_L...	TRAFFIC_L...	TRAFFIC_L...	TRAFFIC_L...
SWITCH_ON	0	1	0	0	1	0
SWITCH_OFF	0	0	0	1	0	0
enAMBER	0	0	0	1	0	0
enGREEN	0	0	1	0	0	1
enRED	0	0	0	0	0	0
UPWARDS	0	0	1	1	1	1
TRAFFIC_LIGHTS__sUPWARDS	0	0	1	0	0	1
ON__sUPWARDS	0	0	0	0	0	0
ON__rUPWARDS	0	0	0	0	0	0
D_out__TRAFFIC_LIGHTS	1	0	0	0	0	0
TRAFFIC_LIGHTS__sub.ON__active	0	0	1	1	0	1
TRAFFIC_LIGHTS__sub.state	OFF	OFF	ON	ON	OFF	ON
TRAFFIC_LIGHTS__sub.D_out__ON	1	0	1	0	0	1
TRAFFIC_LIGHTS__sub.ON__sub.state	RED	GREEN	GREEN	AMBER	AMBER	GREEN
default	1	0	0	0	0	0

traffic lights 2

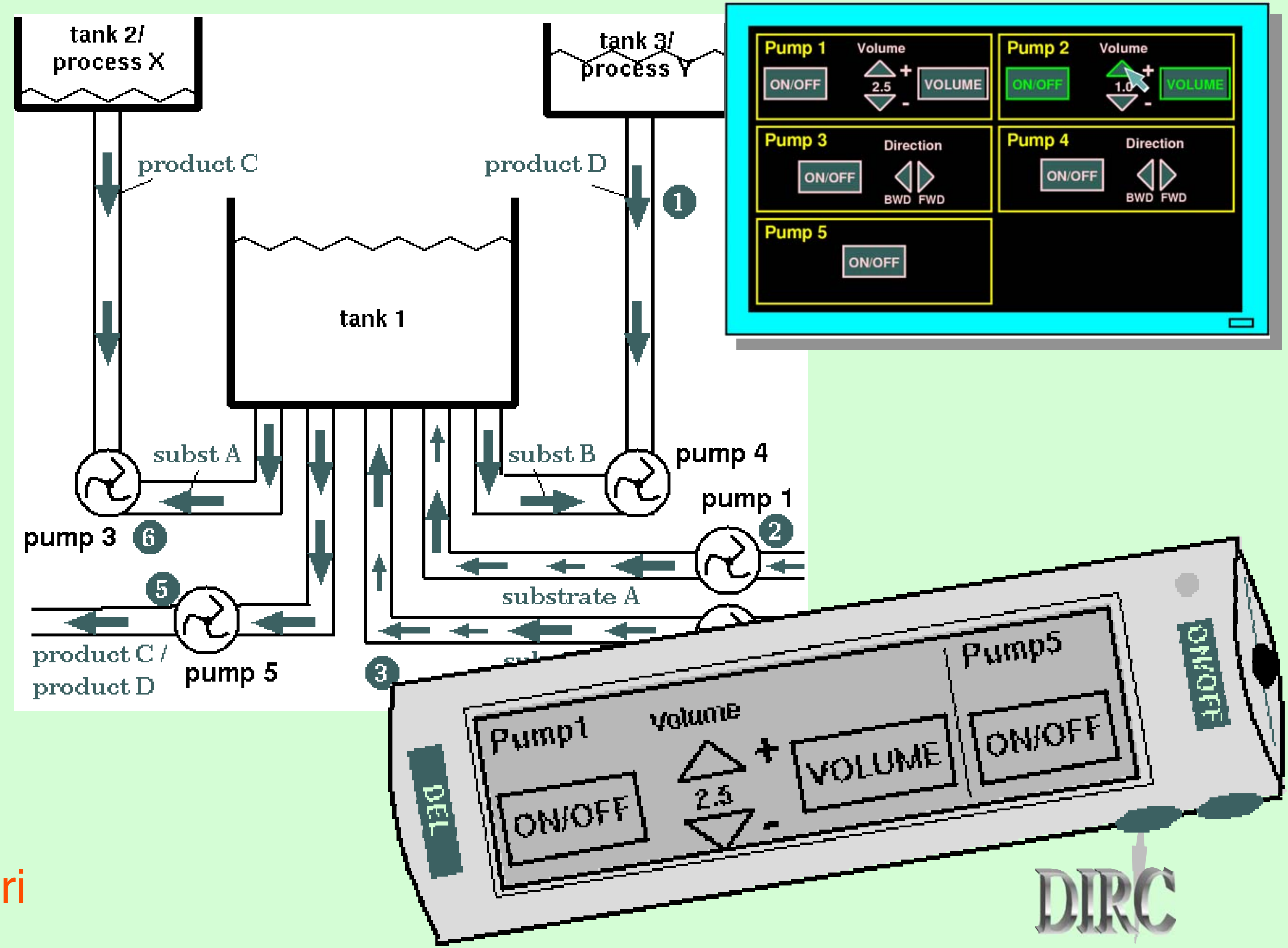
Options Help

specification !EF (TRAFFIC_

TRAFFIC_LIGHT				
state				
SWITCH_				
SWITCH_				
enAMBE				
enGREEN	0	0	1	0
enRED	0	0	0	0
UPWARDS	0	0	1	1
TRAFFIC_LIGHTS__sUPWARDS	0	0	1	0
ON__sUPWARDS	0	0	0	0
ON__rUPWARDS	0	0	0	0
D_out__TRAFFIC_LIGHTS	1	0	0	0
TRAFFIC_LIGHTS__sub.ON__active	0	0	1	1
TRAFFIC_LIGHTS__sub.state	OFF	OFF	ON	ON
TRAFFIC_LIGHTS__sub.D_out__ON	1	0	1	0
TRAFFIC_LIGHTS__sub.ON__sub.state	RED	RED	GREEN	AMBER
default	1	0	0	0

Welcome to TVT...by M. Kermelis

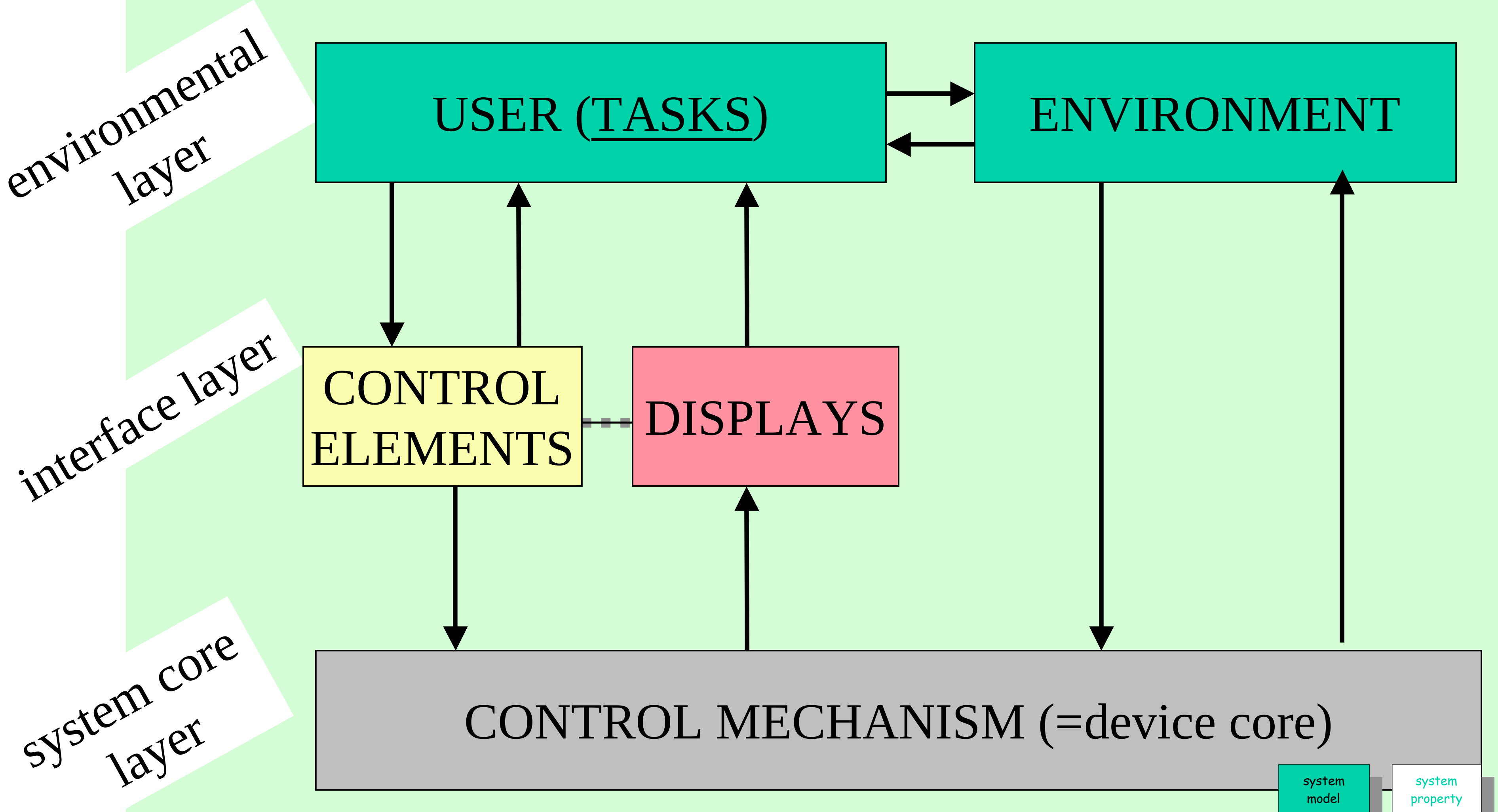
Sample domain: A processing plant



Modelling a mobile device

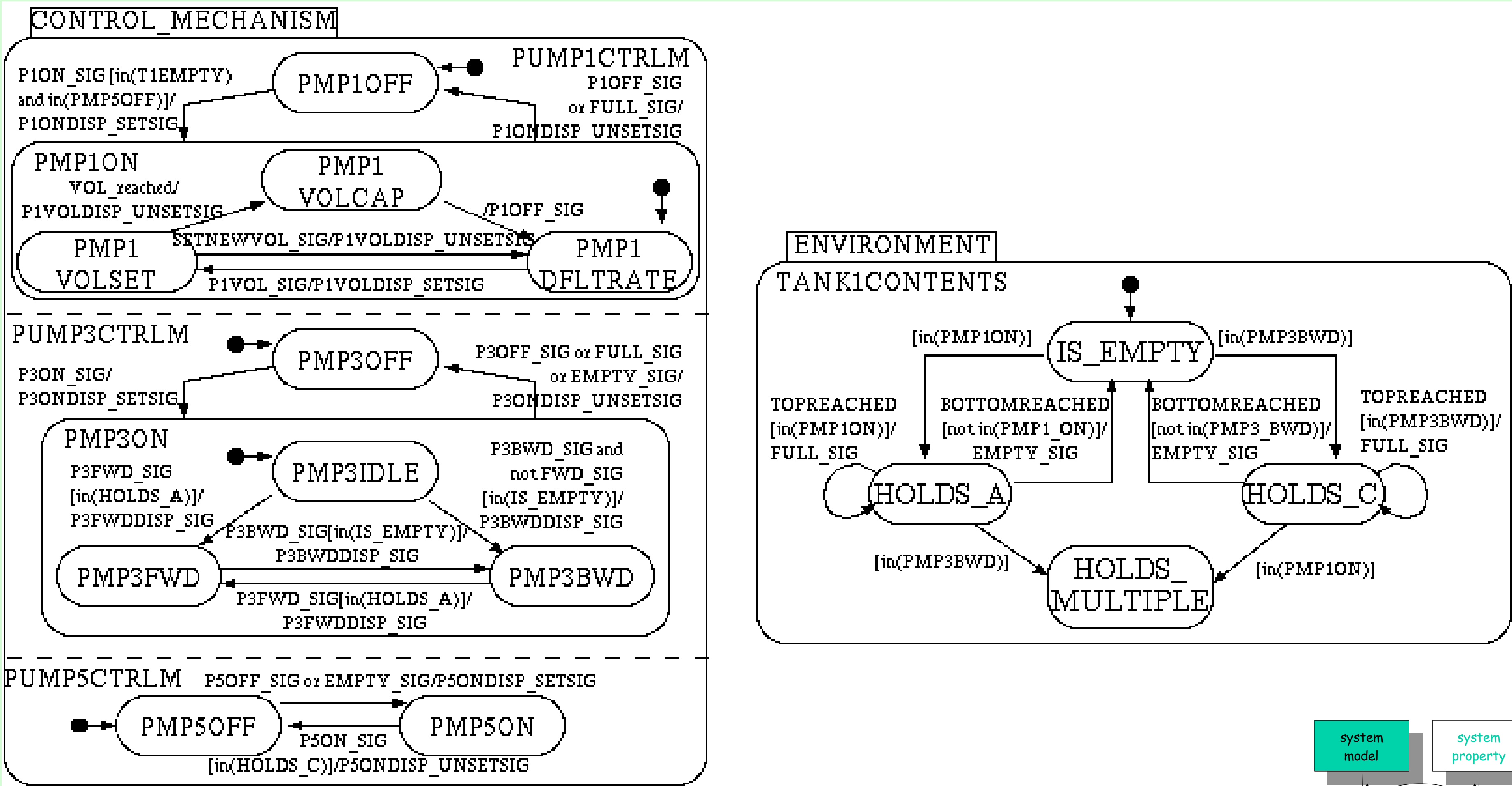
- Ubiquitous control of a sewage plant
- Control device implements a “bucket” metaphor
  - Buckets filled with status information relating to pumps, valves and displays passed as the operator does rounds in the plant
  - Monitor role and control role, buttons also collected into buckets – currently limited to two controls at a time
- Need to model the context in order to understand how the device relates to the context

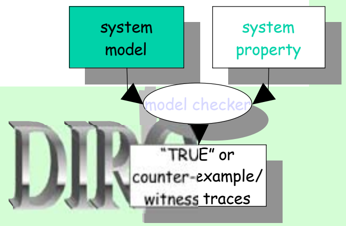
Alternative approach to modelling interactive Systems (Degani)



ri

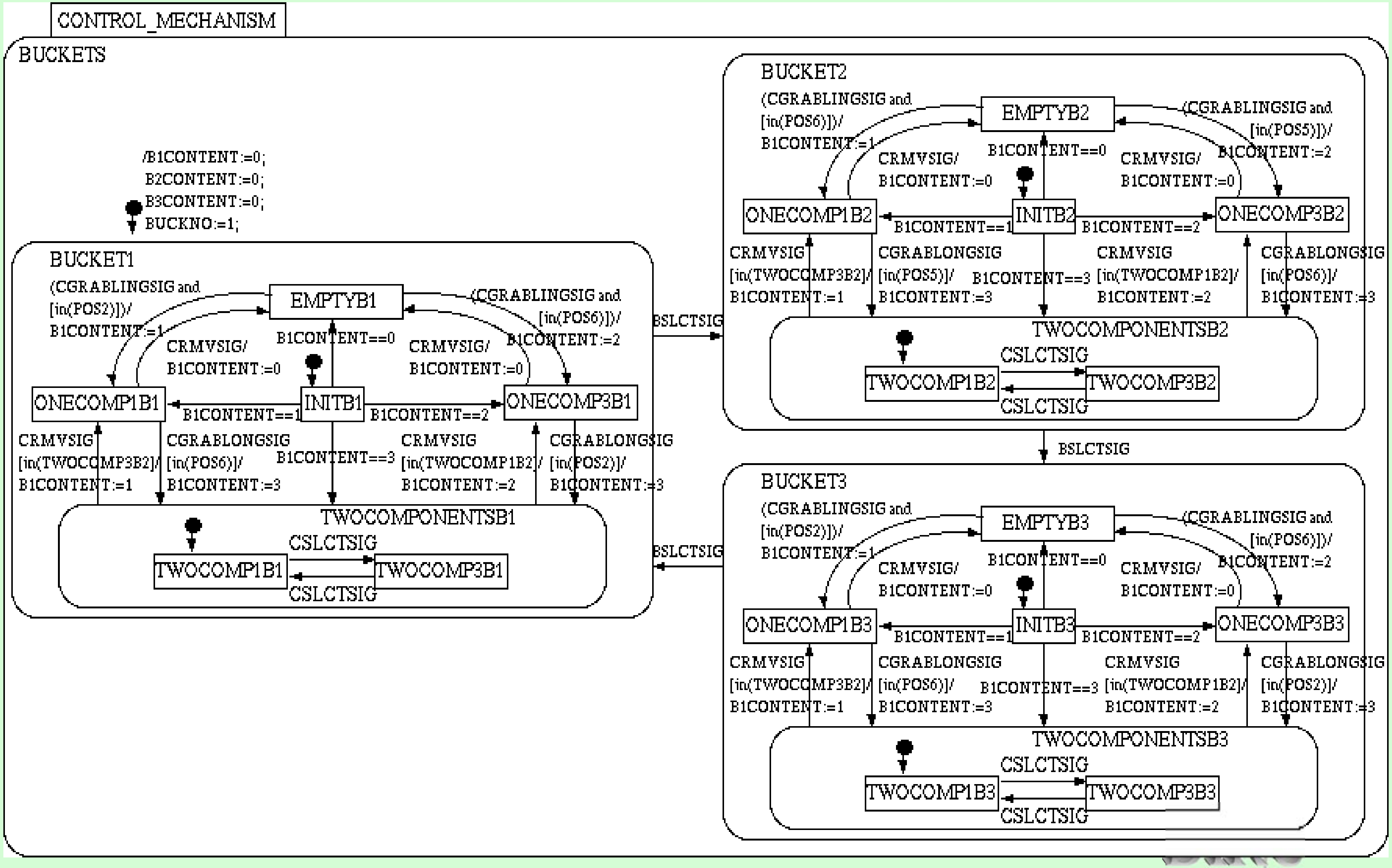
Model 1: controlled devices and environment



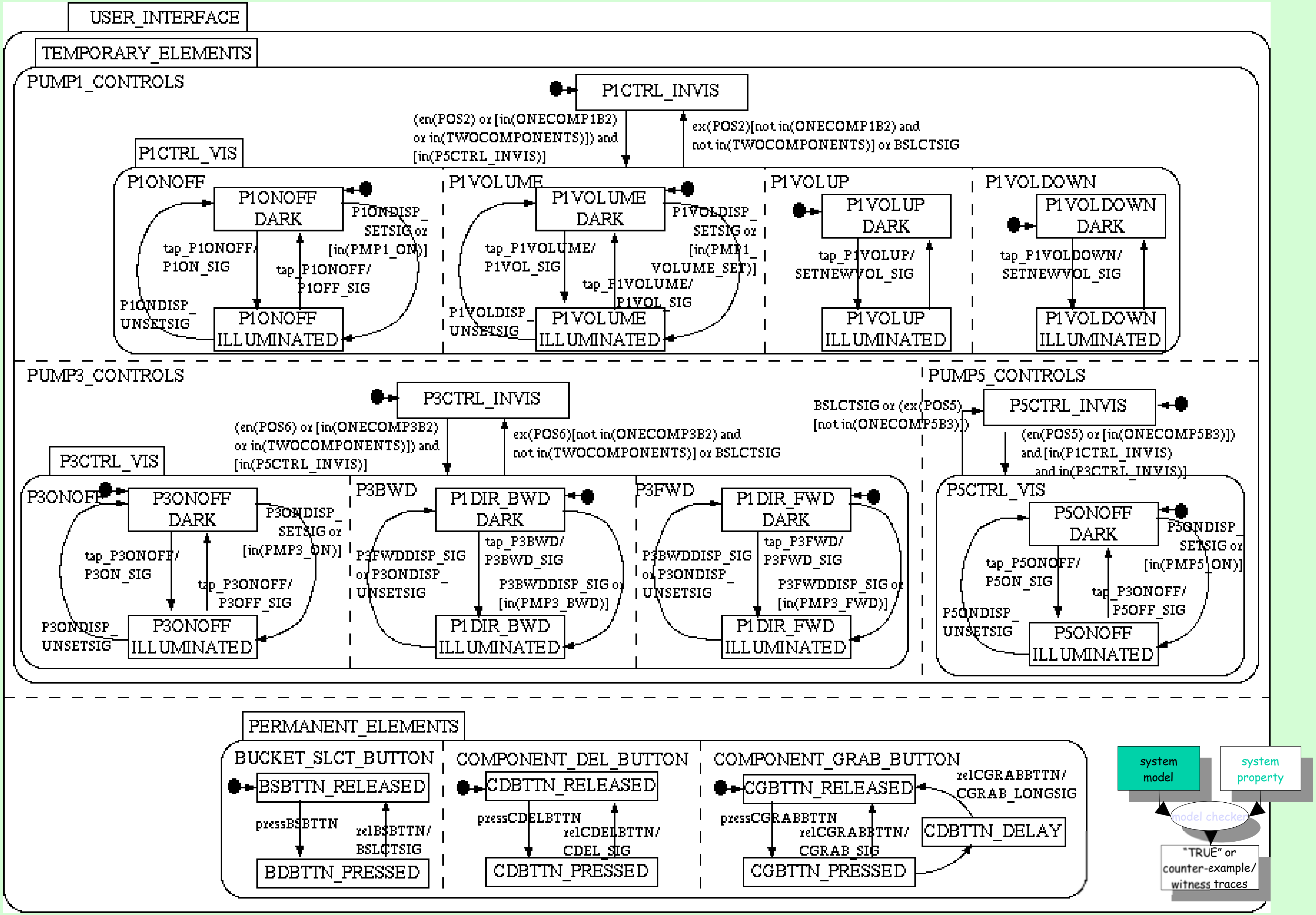


iri

Model 2: Pucketizer “bucket” mechanism



Model 3: Pucketizer device controls

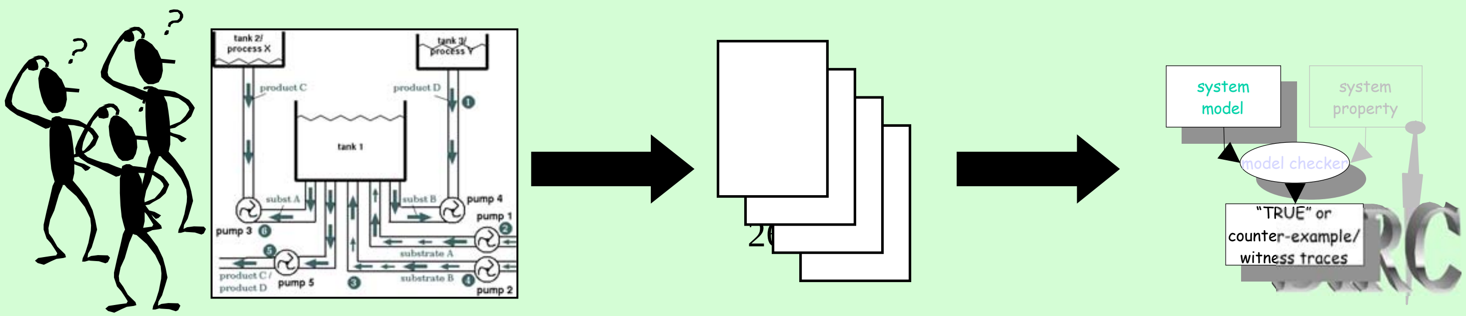


Analysis: Model validity

Does the model behave as intended?

- "sanity": deadlock-freedom, state/event reachability
- "goal reachability":
  - Can product C be produced?
  - What is the easiest way to produce product C?
  - What is the "best" way to produce C under assumptions $a_1 \ldots a_n$?
  - Is it possible to reach unsafe states?

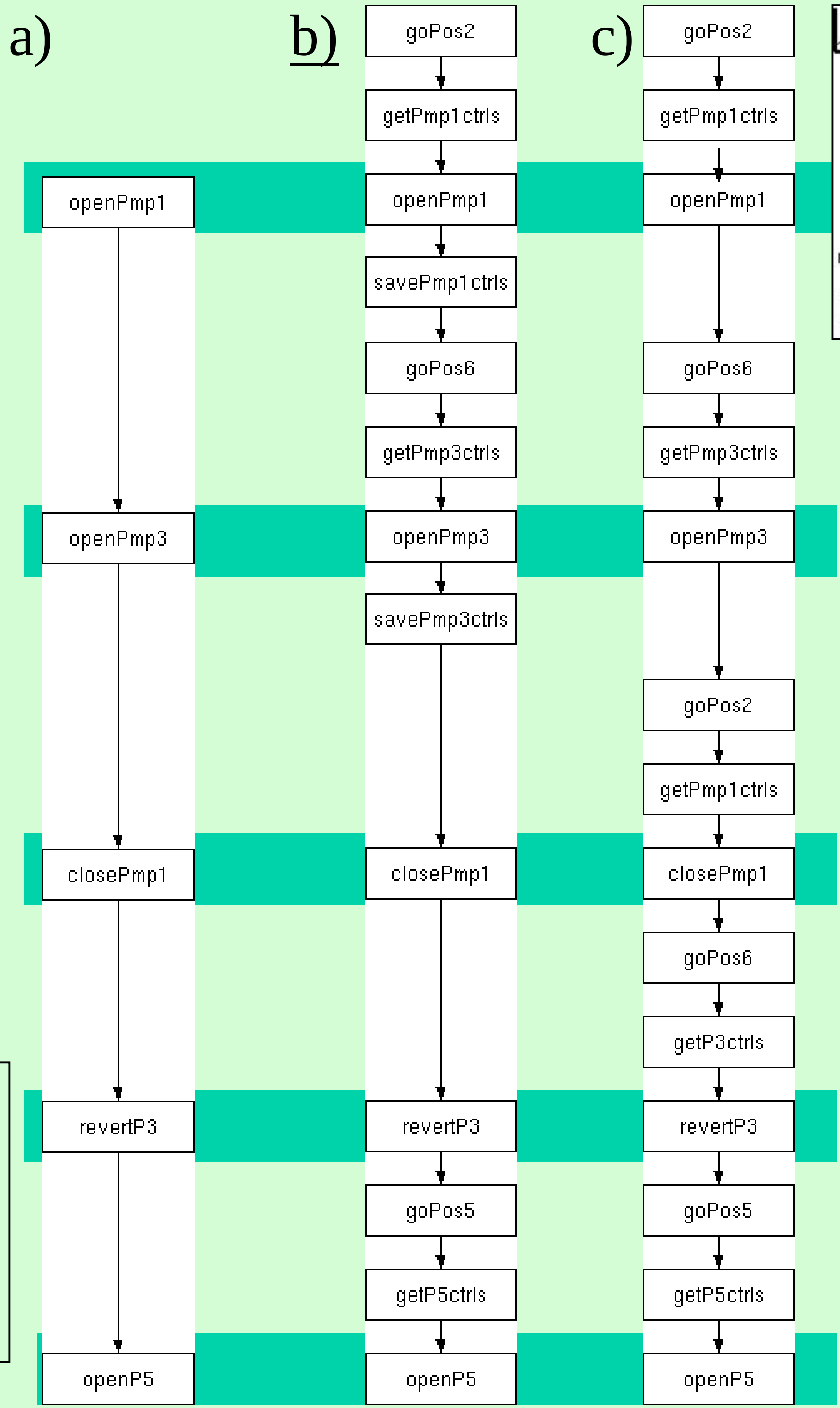
…

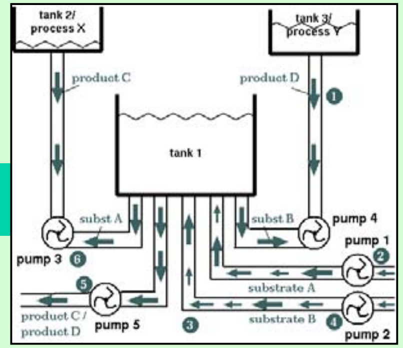


iri

Trace Comparison

Goal/Property:
"Reachability of a state where end product C is released"

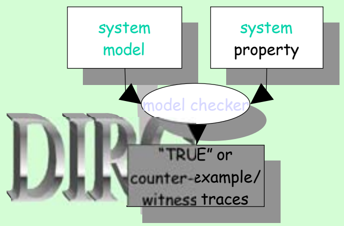




a) Control room interface

b) Pucketizer

c) Pucketizer (forgetful operator)



Allocation of Function

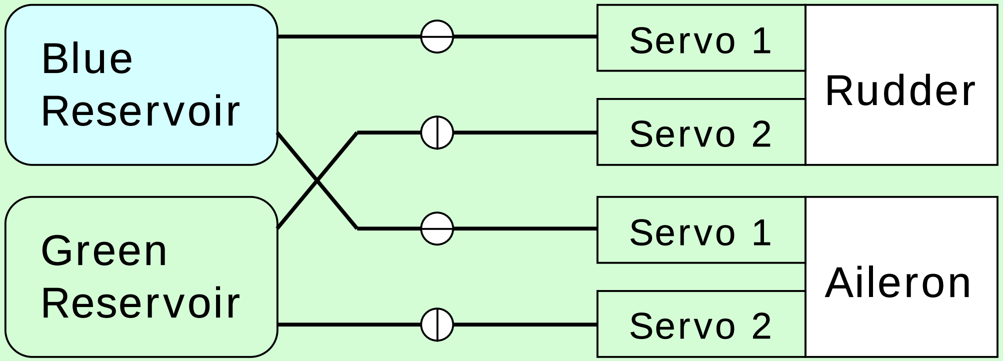
- Aim: To allocate functions amongst the human and machine roles

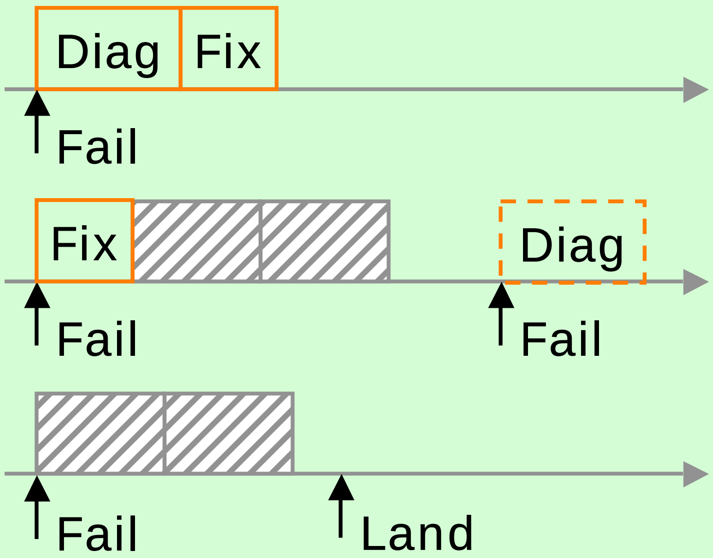
- So that:
  - A coherent set of roles are produced
  - Automation does not interfere with the person's ability to perform the role.
  - Automation supports the person's performance of a role.
  - There are acceptable levels of technical risk.
  - Proposals are capable of satisfying the functional constraints.

Dynamic allocation assumptions

- In the face of a change in circumstances, workload or situation awareness for example, switch level of automation to perform *the same function*
- *In practice, this is a very simplistic view of the way operators need to handle time critical situations*

Hydraulics fault [Fields & Merriam ’99, Doherty, Massink, Faconti ‘01]





Decision parameters:

Performance and time

- Current workload
- Concurrent tasks
- Dynamics of problem [fluid loss]
- Stage of mission [time to land]

Analysis of decision procedure

- Appropriate automation: operator in control?
  - What parameters in the decision
  - What boundaries to the decision
- Initial analyses concerned with extreme conditions
  - based on model checking (similar to Doherty, Massink and Faconti)
- A family of techniques required concerned with typical behaviours, extreme behaviours, experiment
- One concern is how to deal with battery of methods – can we focus experiment using analysis, for example?

PaintShop: Task

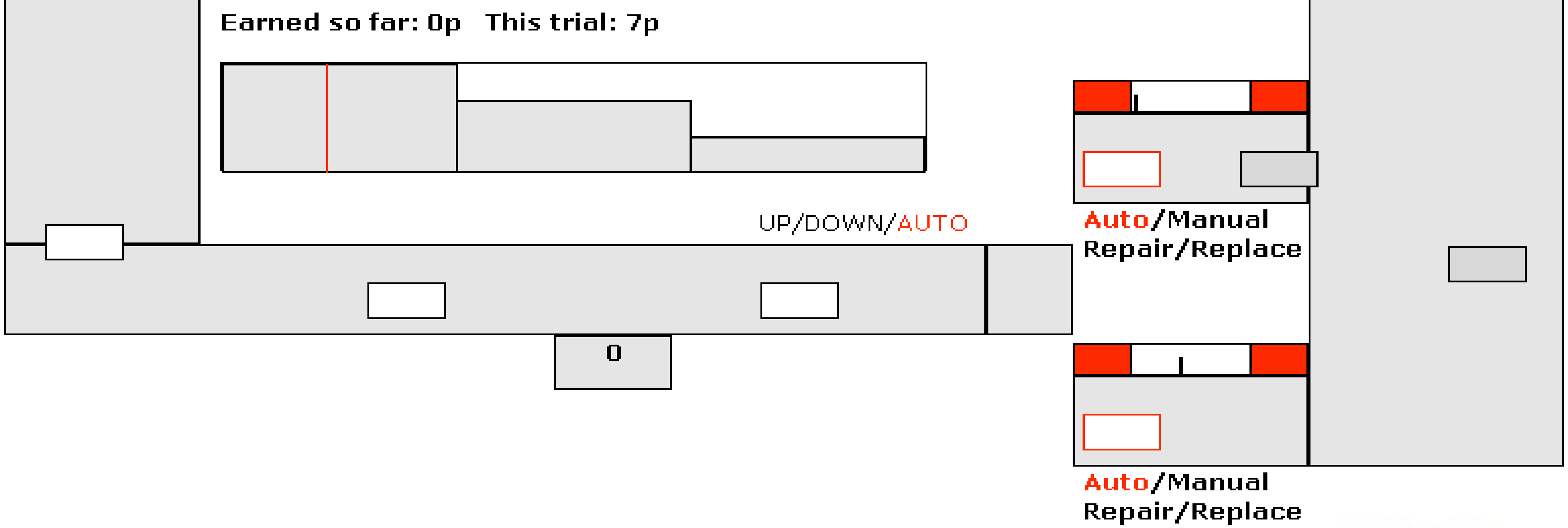
Supervisory control of a dual-line production plant

Money earned per item painted [1p]

Automatic or manual painting [4s vs. 2s]

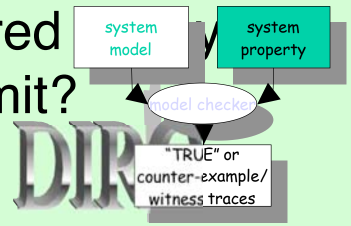
Fault monitoring and servicing: Repair or replace

Repair: No cost, but line unavailable for 24s
Replace: Line available immediately,
but [6, 8, 12]p cost



Temporal properties

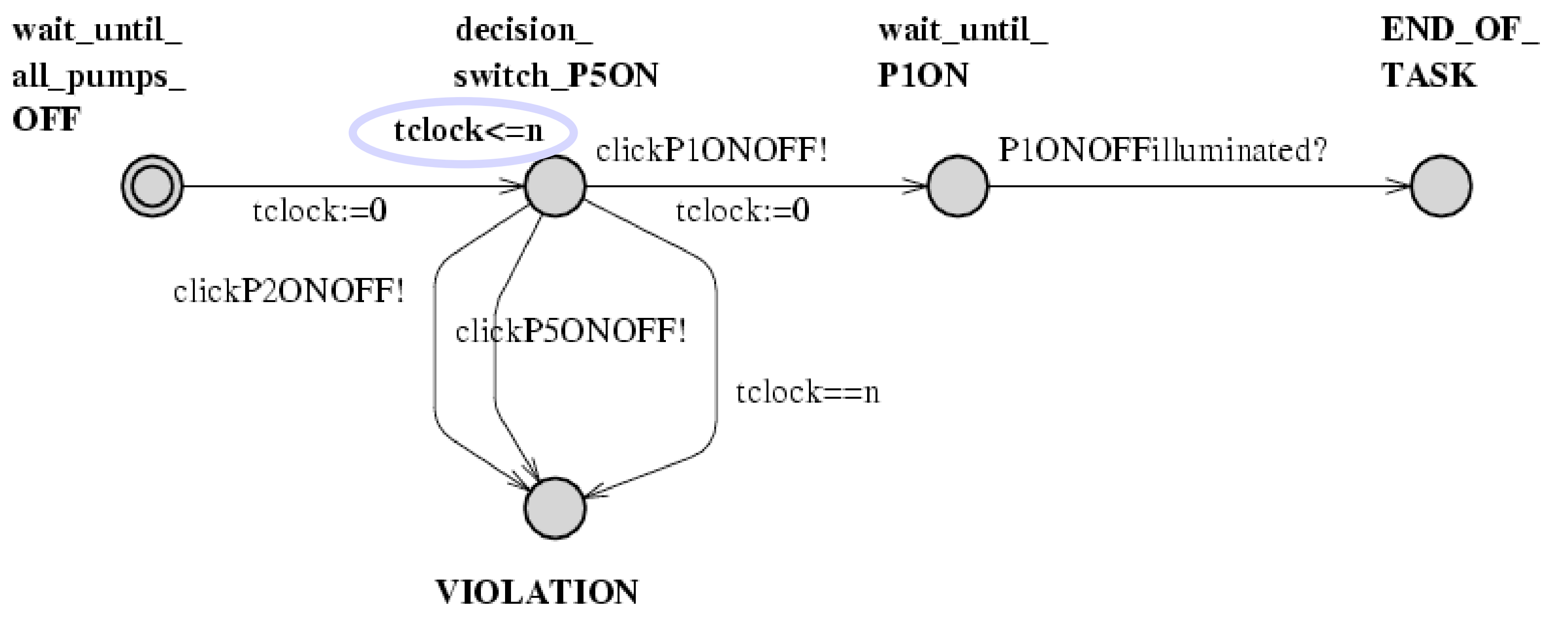
- Sequencing
  - How does the sequence in which actions are performed influence performance?
- Real-time
  - What are best/worst case execution times for a job?
  - How do bcet/wcet vary under different workloads?
- Suitable strategies for decision making:
  - What is the minimal time required to paint all items (regardless of costs or replacing parts)?
  - What costs does the operator need to be prepared in order to paint all items within a certain time limit?



iri

Real-time models

- real-time is explicit element of the model, represented by continuous variables

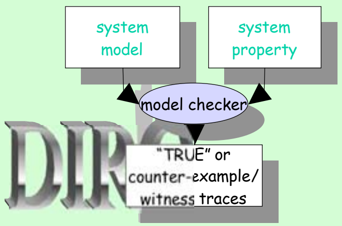


iri

Explorative application of model checking

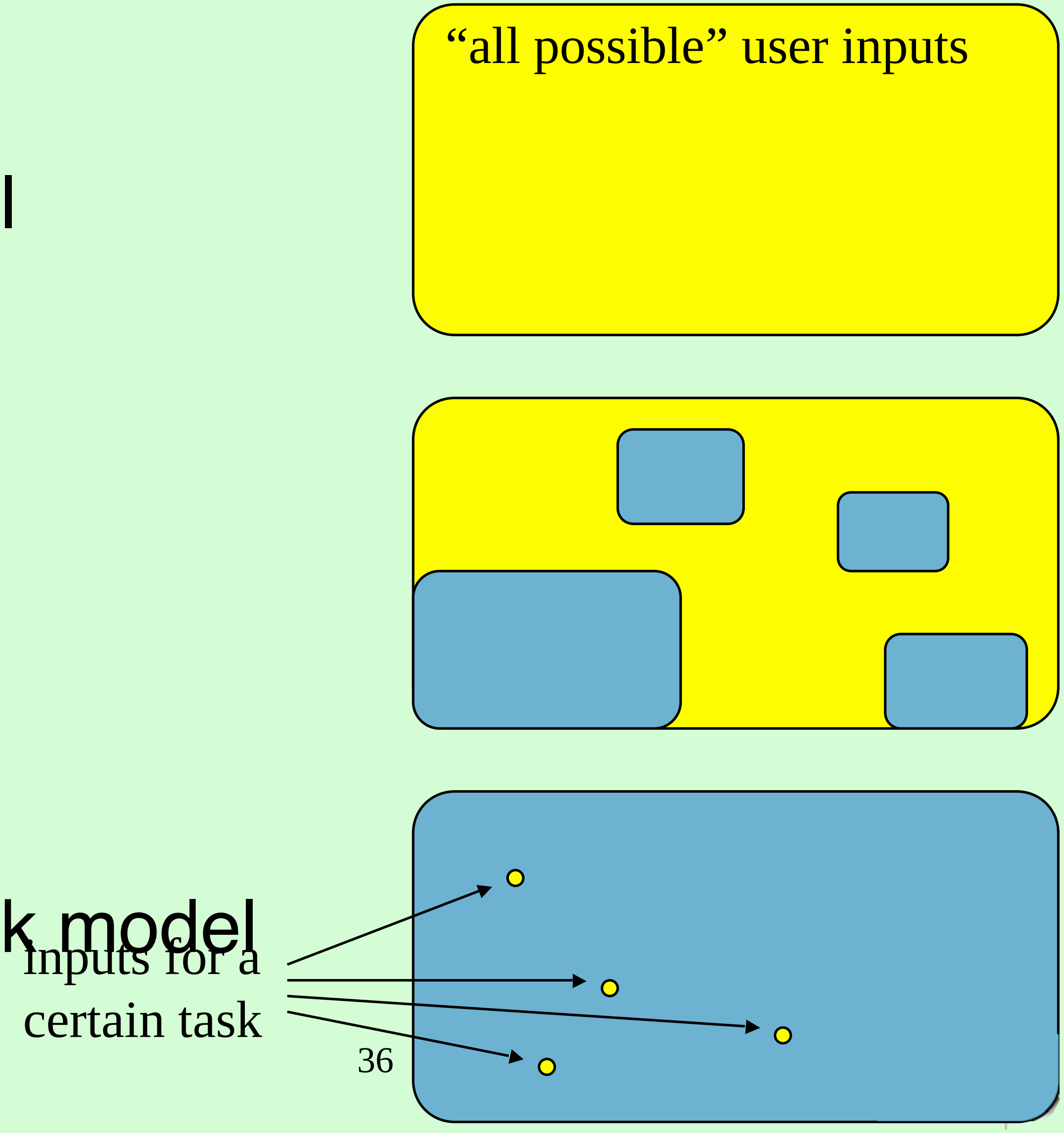
1. starting from a device-centric model
   => all possible user inputs
2. gradually add assumptions about user and environment behaviour
   => sub-set of "sensible" user inputs

- formulation of assumptions:
  1. as part of the property specification
  2. by model enhancements (e.g. observer automata or model decorations)



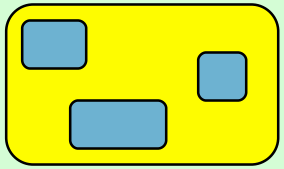
Influence of task models on explored input space

- no task model
- constrained "task space"
- normative task model



iri

“Task space” constraints₁

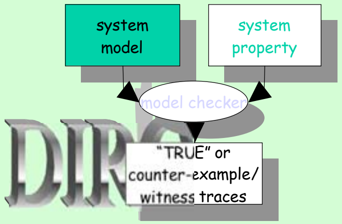


Focus of analysis:

Given:

1. a device specification and
2. a desired target “situation” (= state of the device and environment)

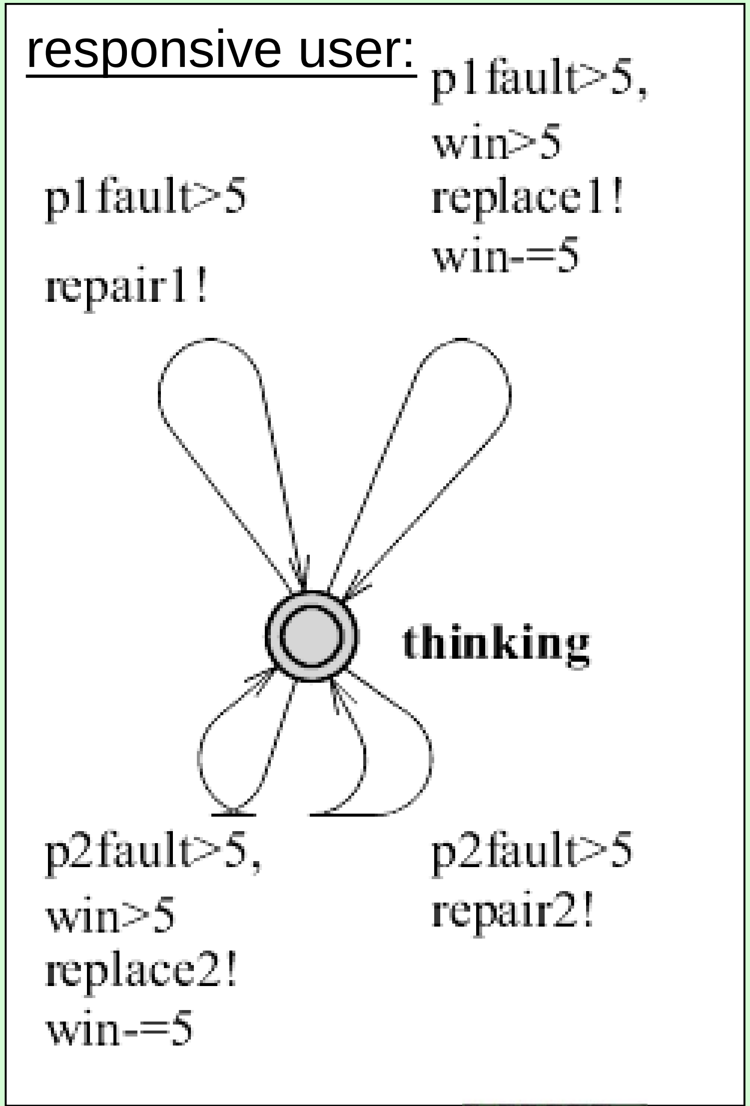
Question: *What assumptions can/need to be made about the user?*



"Task space" constraints₂

- **Goal:**
  Constrain search by adding constraints
  (= set of state machines)
  on the user behaviour

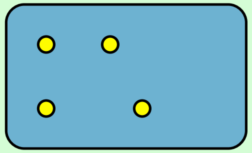
- **Example:**
  "*Whenever the user realises that a nozzle is blocked he/she will opt to either replace or repair the nozzle*"



iri

DIRC

Normative task models

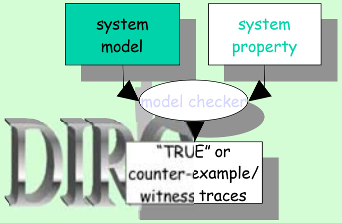


Focus of analysis:

Given: A specification of

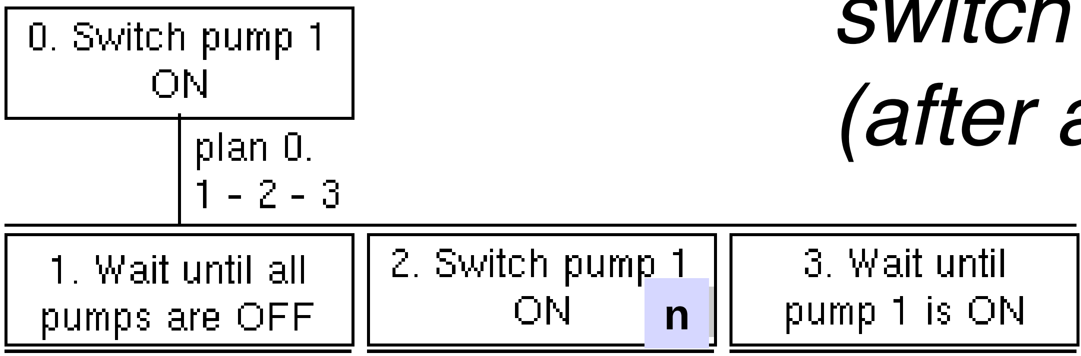
1. the device under development,
2. relevant parts of the environment and
3. a normative task model

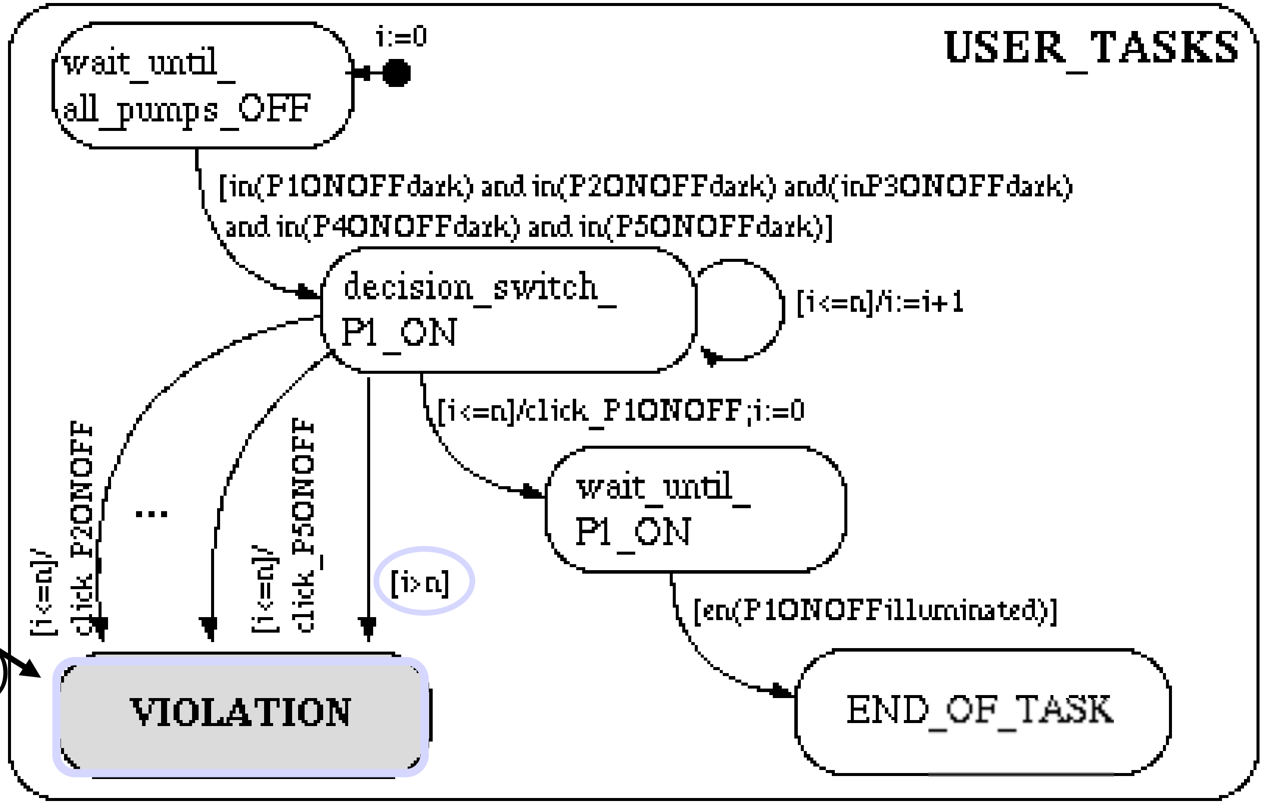
Question: *What states of the environment can be reached?*



Example task:

"Once all pumps are off, switch pump 1 ON (after at most n steps)"

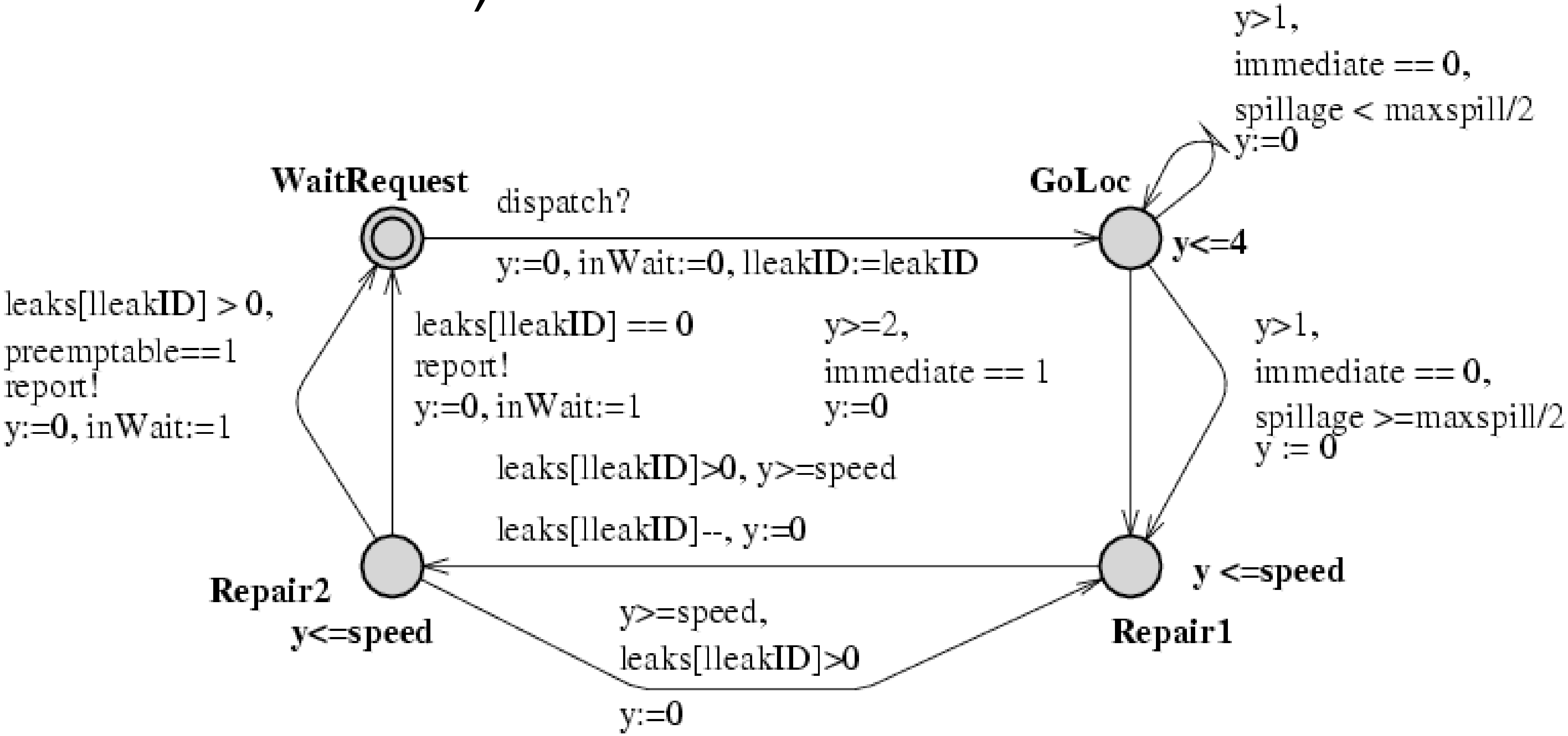




Task violation:
e.g. Hollnagel's error phenotypes
(here: delay and replacement)

Timed user models1

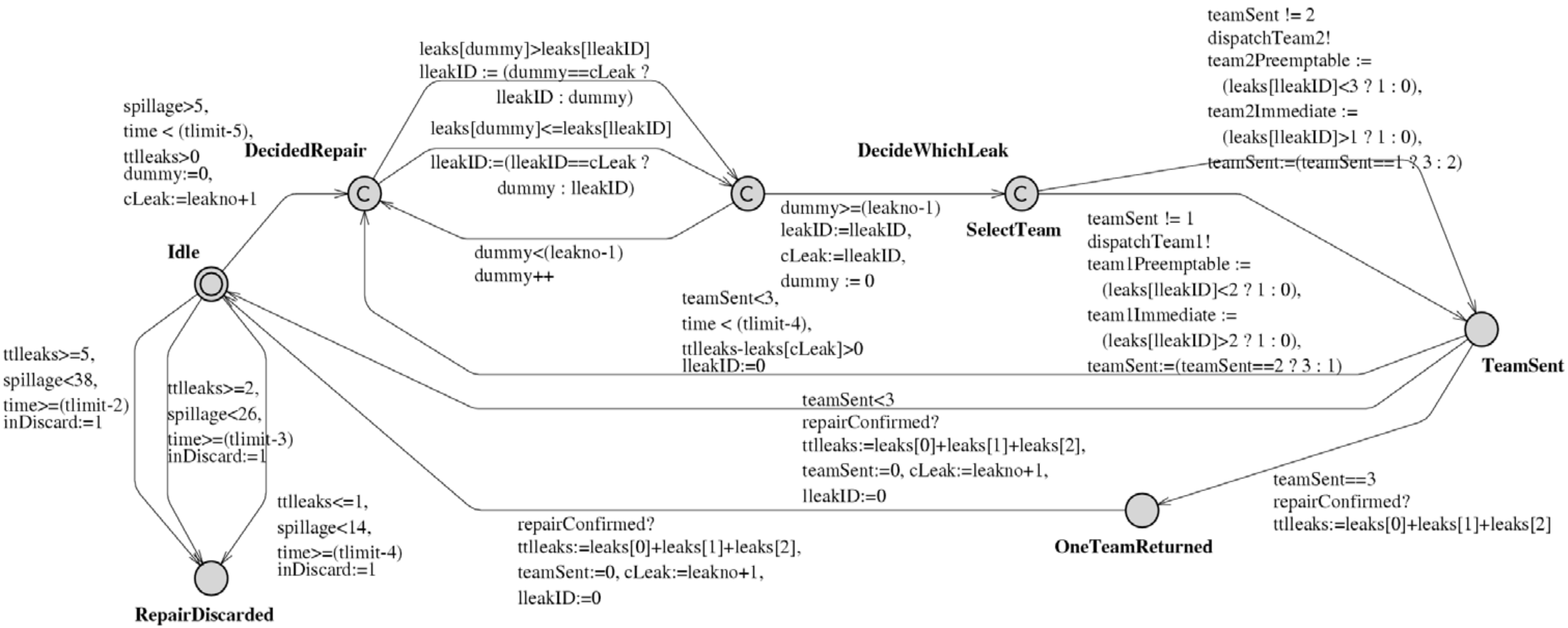
- What is the maximal/minimal time required for a repair (depending on size and location of leak)?



Modelling complex user decisions

- decisions that depend on multiple cost trade-offs

  (time/leakage/monetary costs/ …)



Adding assumptions about operator behaviour

- temporal logic assertions:

  "the operator always forgets to store pump controls"

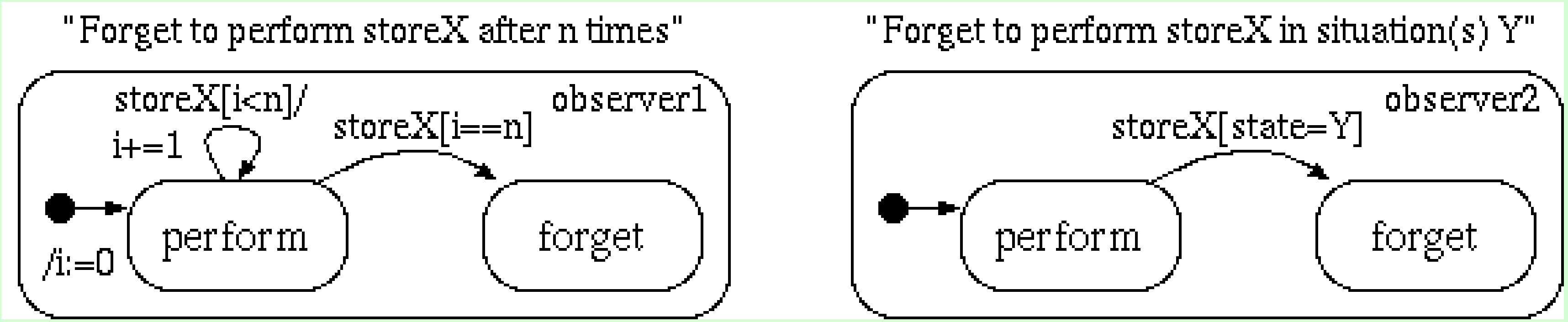
```
assert SAN1:
  F ((PUMP5CTRLM.state=PMP5ON)
     & (TANK1.state=HOLDS_C));
assert alwaysForget:
  G!(savePmp1Controls| … |savePmp5Controls);

assume alwaysForget;

using alwaysForget prove SAN1;
```

Adding assumptions about operator behaviour

- observer automata: the "forgetful" operator



- check properties under the assumption that violation states ("**`forget`**") are absent

Conclusions: Model checkers are good at…

- exhaustive analysis
- “automatic” analysis
  - provided that appropriate input is supplied
- analysis of behavioural reachability properties
  - ordering/sequencing of tasks:
    - e.g. Hollnagel’s error phenotypes:
    - repetition, reversal, omission, delay, premature action, replacement, insertion, and intrusion
  - (physical) timing
  - mode complexity
  - dialogue control:
    - visibility of action effects, visibility of available actions, recoverability, consistency, error prevention, flexibility, efficiency of use

Conclusions:
Model checking has limitations…

- deliver single, sometimes “trivial”, traces
- hard/impossible to determine tendencies, e.g. certain types of user behaviour, characteristics of components that contribute to potential errors …
- technique does not suggest corrections
- difficult/unsuitable to use for analysis of representational properties (layout, direct manipulation etc.)

iri

DIRC