Engineering the behavior of Robots: Simulation and Model-Checking for Embedded Systems and Robotics

Vladimir Estivill-Castro

Griffith University
IIIS

(in collaboration with many others,
particular thanks to members of the MiPAL)

Outline

- Robotics and Software Engineering
- Why State Machines and Why Logic
- Examples
- Comparison
- Model Checking
- Architecture
- Illustrations
- Summary

Outline

- Robotics and Software Engineering
- Why State Machines and Why Logic
- Examples
- Comparison
- Model Checking
- Architecture
- Illustrations
- Summary

Share the Interest for Robots in Human Environments

Implications for:

Safety

Software Engineering for Robots

Reasoning

Human Computer Interaction

Share the Interest

for:

Model driven engineering

Simplicity to program

Keep it simple, stupid (KISS)|80/20 rule

High cohesion, Low Coupling (Distributed Components)

(Data Distribution Service -Publisher/ Subscriber)

Platform Independence

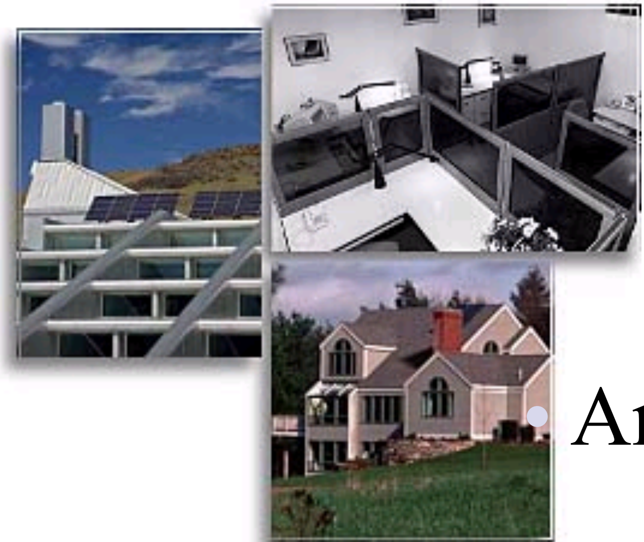
Composability of components

A central project for intelligent integrated systems

- The development of autonomous mobile robots for multi-modal interaction with humans
- leading to
  - useful applications integrating
    - agent technology, HCI, AI, image processing, robotics, vision, planing, problem solving, game theory, machine learning, voice recognition, sensor fusion,
  - emotional reactions and advanced research in areas of intelligent integrated systems
  - participation with prototypes in international benchmarks that have academic and industrial recognition
    - RoboCup Soccer, RoboCup@Home, Agent-Poker, Open Game Play

Hypothesis (1)

- In the not so distant future humans will be surrounded by all sorts of `intelligent machines'



- Intelligent buildings and Sensitive computing
  - Computing environment intended to assist the user for retrieving, organizing and interpreting available information resources by augmenting and extending the sensory as well as the cognitive capabilities of the user
- Ambient Intelligence / Tele-presence

Hypothesis (2)

- The sector of the human population that is to benefit the most from `robots around us' are people with disabilities, sick and rehabilitation patients, the elderly and pupils
  - If technology is to reflect an advance society it should make an impact on improving
    - the life of its weak/disadvantaged/untrained members

Hypothesis (3)

- A convergence is looming on Information and Communication Technologies
  - Mobile phones, PDAs, Wireless/ Internet and Intranets through computer watches
    - “the Cloud”
  - Wearable computers

Hypothesis (4)

- There is a shift from “***accessible computing***” to “***user centered design***” in the Human-Computer Interaction community
  - Accessibility
    - Providing accessibility means removing barriers that prevent people with disabilities from participating in substantial life activities
  - UCD
    - Focusing on the product's potential users from the very beginning, and checking at each step of the way with these users to be sure they will like and be comfortable with the final design.

Hypothesis (5)

- The fields of artificial intelligence, robotics, machine learning, human-computer interaction are advancing in research that

- integrates advances from different fields
- shows deployment of the technology in demonstrable prototypes

Hypothesis (6)
The interface may not be a robot

- The actuators and sensors can be remote
  - Not all of them on board of the robot
  - The control may not be on board of the mobile components
- But the technologies developed will have use in all the applications emerging from this flexibility.

Hypothesis (7)
Agent technology is influencing everyday life

- Computer Games
  - Age of Empires
  - Age of Mythology
- Xbox/PlayStation/ Wii
- Tamagochi/Nintendo DS
- Environments
  - Dofus
  - Runescape
  - Club Penguin
  - 2nd Life

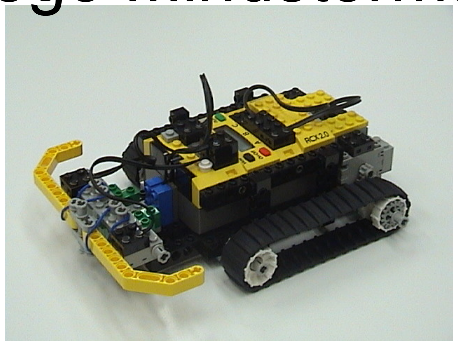
- Automatic assistants
  - eBay

What does robotics provide?

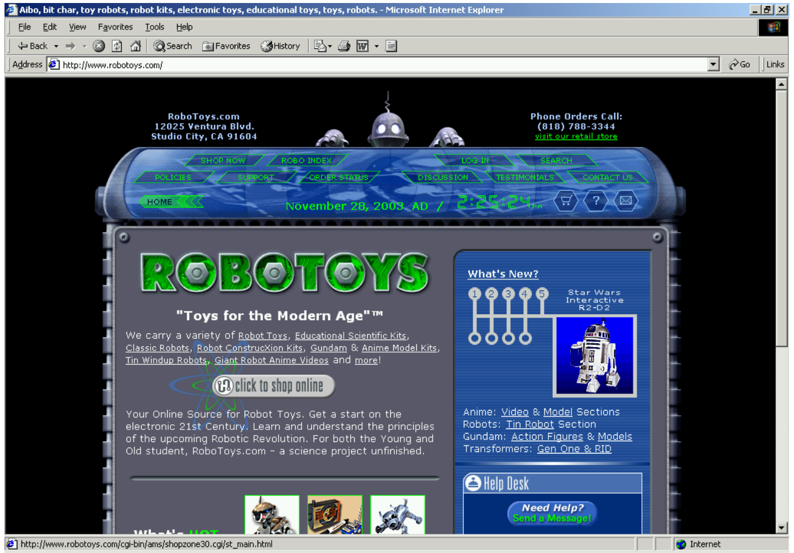
- Mobility/autonomy
- Embodiment
  - Physical presence
- Teams of robots
  - Collective abilities / remote control

Robotics has penetrated the home market

- Toys
  - Lego Mindstorms™



  - Cindy Smart TM



Robots on children's bedrooms

Robotics has penetrated human environments

- Home artifacts
  - The EUREKA Robo Vac^{TM}

  - Electrolux Trilobite ^{TM}

- Guides for
  - visitors in museums and the elderly
  - visitors in airports

Autonomous Vehicles / Robotic Cars are penetrating the Urban Environment

Robots are penetrating the media

- News readers
- Booking agents, traveling agents, eCommerce

- Robotic interfaces are more human-like
  - The uncanny valley
- Environments/Virtual reality/Attractions
  - Opponents are simulated agents
  - Is the matrix possible?
- Movies/Special effects
- Military
- Entertainment parks

Outline

- Robotics and Software Engineering
- Why State Machines and Why Logic
- Examples
- Comparison
- Model Checking
- Architecture
- Illustrations
- Summary

How do you describe the behavior as an everyday person?
(to your robot / companion)

- Humans describe requirements
  - Of the systems
  - Of the capacities of the system

  - Realizing the human description is a common theme between
    - Software Engineering

    - Artificial Intelligence

How do you describe the behavior as an everyday person?
(to your robot / companion)

- In a description
  - There is a **declarative part**
    - a context, a description
      - ontology (?) knowledge representation?
    - If formal (unambiguous), needs a logic
  - There is a state - transition **- action part**
    - Formally, an algorithm in a formal model of computation

© V. Estivill-Castro 22

Specifying a behavior

- It should be natural to the human
  - For the declarative parts, mechanisms used by humans should be provided
    - common sense reasoning
    - non-monotonic logic
- Mechanism should be
  - Simple to learn
  - Formal to remove ambiguity
  - Implementable (interpreter/compiler)

Illustration

- Naturally to develop rules systems where the new rules redefine exception to the previous ones
  - 3 laws of robotics
    1. A robot may not harm a human
    2. A robot must obey a human unless it contradict law 1
    3. A robot must protect itself unless contradicts rule 1 or 2
  - Ripple down rules (Knowledge elicitation)
    - Rules are defined and new rules are subsequently added to revise the cases not covered by the more general rules
    - A tree that is a hierarchy of rules
      - No formal reasoning

Proposal for engineering the behavior

- Using **visual descriptions** of the behaviour that **incorporate formal logic**
- Engineers use diagrams to model artefacts.
  - Iterative refinement
- Software Engineering has traditionally used diagrams to convey characteristics and descriptions of software

- High-level tools
- Observations:
  - Specifying behaviour unambiguously is difficult
  - Interpret human descriptions of behaviour is also difficult

For Requirements Engineering

- Use CASE (Computer Assisted Software Engineering)
  - graphical models
  - code generation
    - Model Driven Engineering

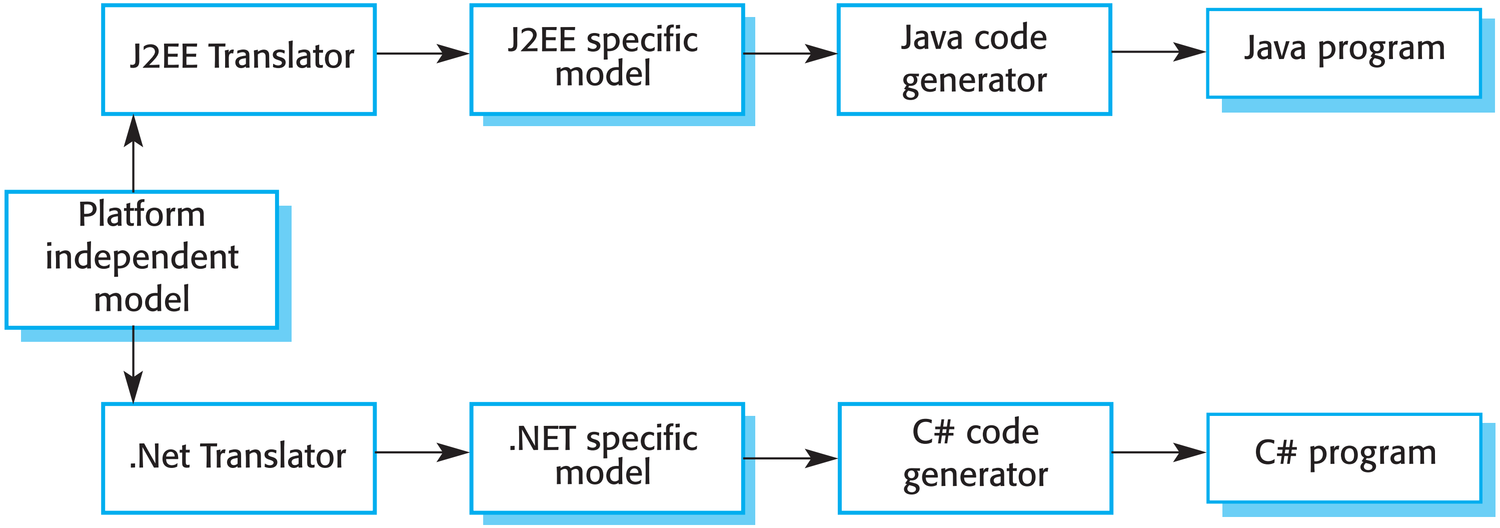
- Bottom-up approach / Iterative refinement
- Elude the very large syntax and semantics of OMG modeling (standard) languages
  - for example : UML [2.0]

Requirements Engineering

- Minimize software faults
  - disambiguate requirements
    - completeness
    - consistency
  - validate requirements
    - correctness
  - model / simulate requirements
    - platform independence
  - traceability of evolution / change in requirements
  - communicate requirements
  - implement requirements (automation)

Model-Driven Engineering

- Approach in Software Engineering
  - Construct software / Safe Software / Quality Software
  - models rather than programs are the principal outputs of the development process (Sommeville, 2009).
  - The programs that execute on a hardware/software platform are then generated automatically from the models.
  - Raises the level of abstraction



Modelling behaviours

- We **introduce diagrams that use logic to describe behaviour.**
- Our proposal extends techniques like
  - Finite State Machines
  - , Petri Nets
  - Object Models for Object Orientation, and
  - Behaviour Trees.
- We model the relationship between several inputs as asserted conditions about the environment that an agent can reason about (using logics) and resolve with respect to knowledge of the environment
- Computer Assisted Software Engineering enables the manipulation of modelling diagrams and the generation of code from the models.

Formal Logics (declarative)

For the description of the behaviour

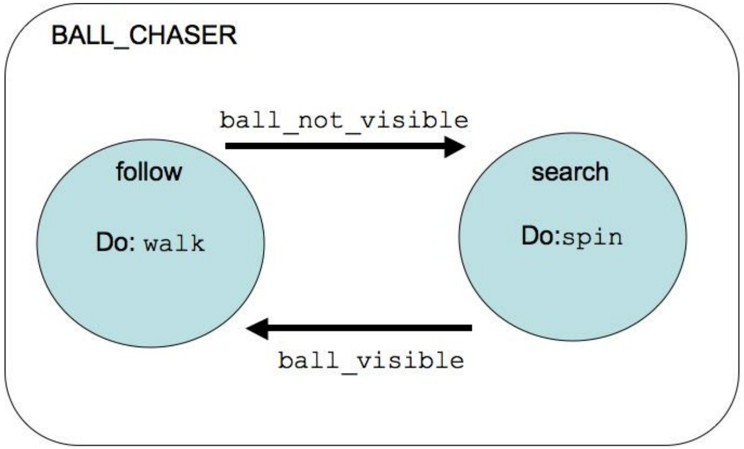
Advantages

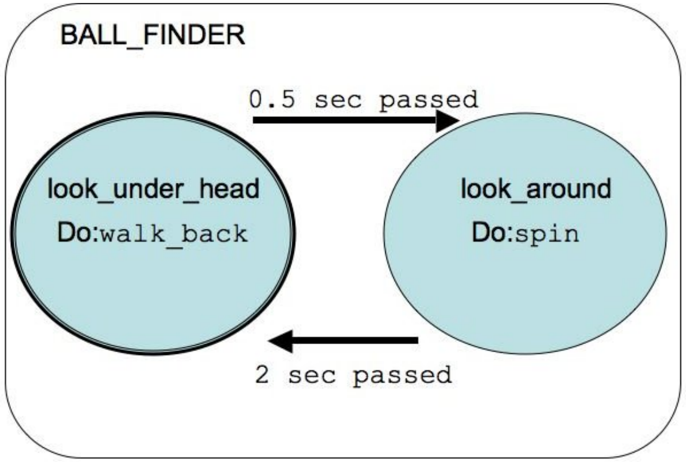
1. Descriptions are unambiguous
   - Descriptions have specific meanings.
2. Ease of description - descriptive
   - Focus is on what the behaviour does, not how it happens
3. Can be translated to implementations in imperative languages like C++, Java
4. Understandable by humans
   - Can be the result of a knowledge engineering exercise
   - Usually humans describe exceptions and laws governing many situations in this way

Disadvantages

1. Can lead to undecidable settings or other difficulties for implementation, like very large and/or inefficient programs

Illustrating state diagrams





- Exclusivity

$c_i \wedge c_j = \textbf{false} \ \forall \ i \neq j$

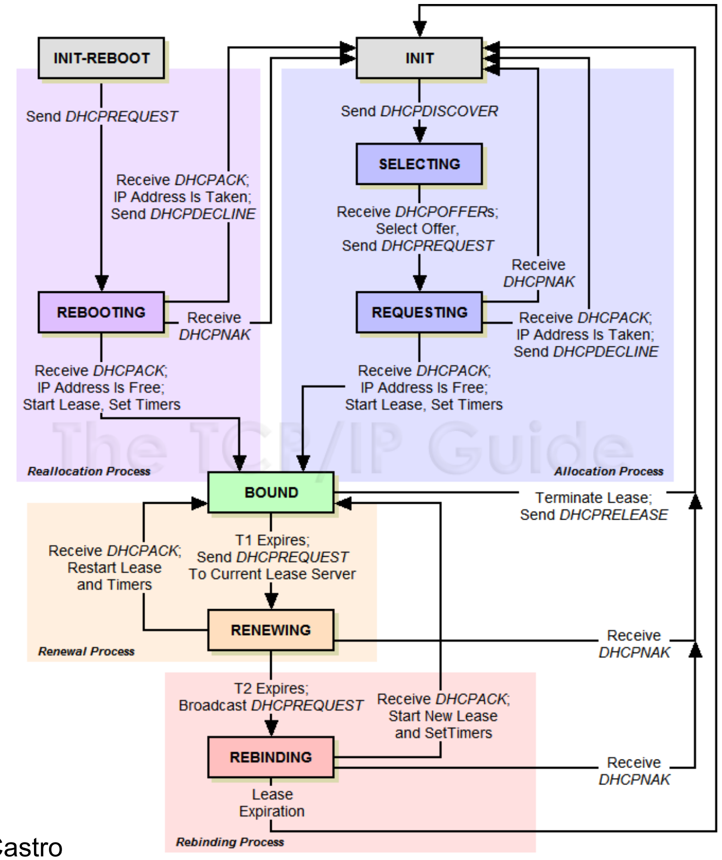
- Exhaustivity

$\bigvee_{i=1}^{n} c_i = \textbf{true}$

s_1	c_1=$event_u$	s_i
s_1	c_2=$event_v$	s_j
s_i	c_t=$event_x$	s_p

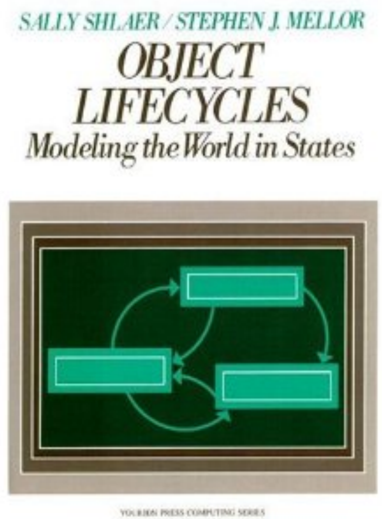
State diagrams (action)

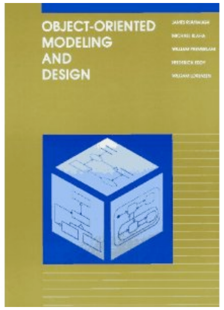
- Correspond naturally to the notion of state machine
- Already very common in many human-computer interfaces
  - elevators/mobile phones/ washing machines
- Formal semantics (formal mathematical object)



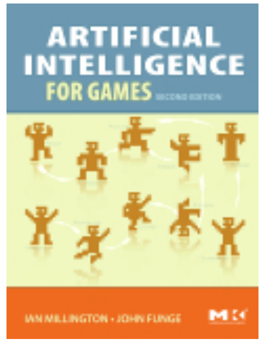
State diagrams (action)

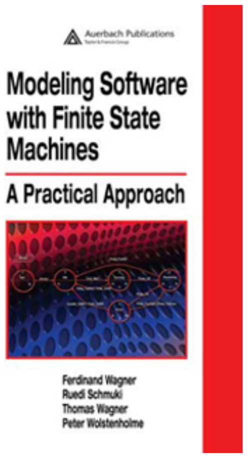
- Widely used in Software Engineering
  - OMT, then UML, Shlaer-Mellor

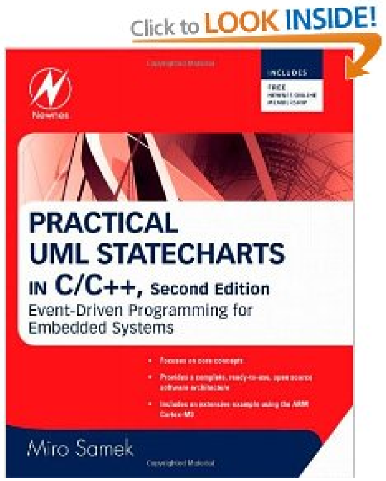


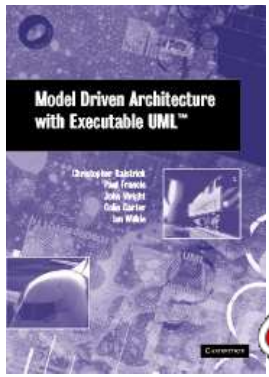


- Widely successful tool in industry
  - StateWorks, executableUML









SimBionic

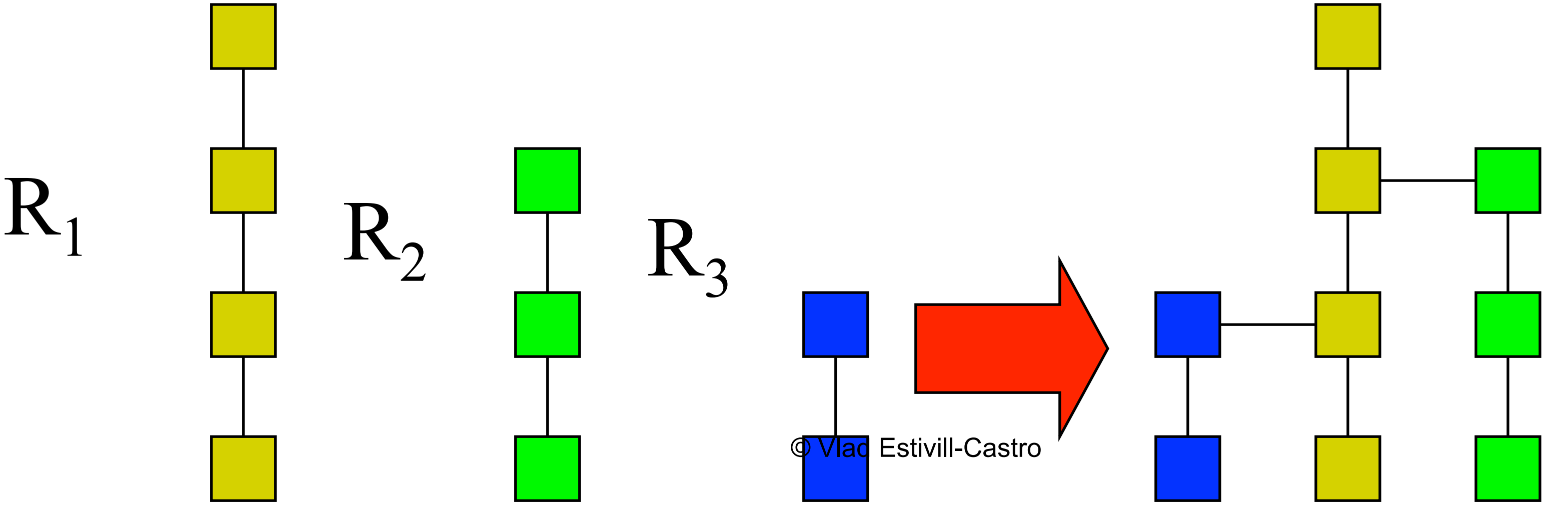
© Vlad Estivill-Castro

State Machines

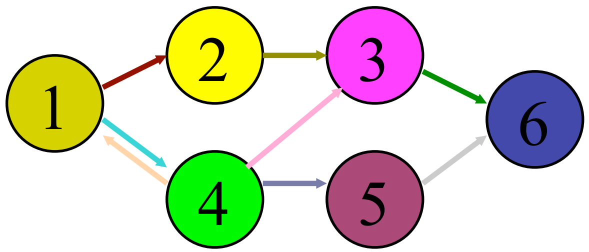
- Some extension and equivalences to other formal models
- Multi-threaded State Machines
- Petri Nets
- Distributed computation
- Team automata
- Security formalisms (verification)

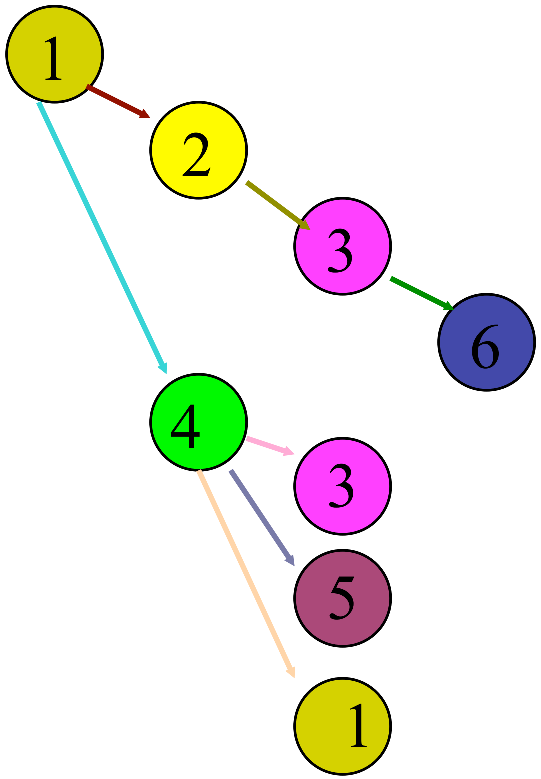
Behavior Trees

- Formalism of requirements engineering
- Similar to ‘Use Case’ Modeling
- Tool for `Behavior Engineering’
  - Capture the threads of behavior from the linear description
  - Textual to formal



Convert State Diagram into Behaviour Tree

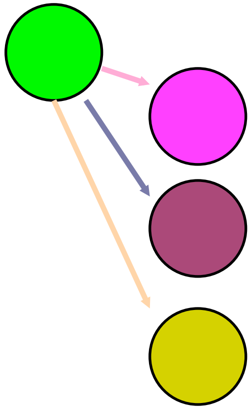




- Draw down by breadth-first search
- Already visited nodes are cloned but not explored again

Potentially equivalent modeling approaches

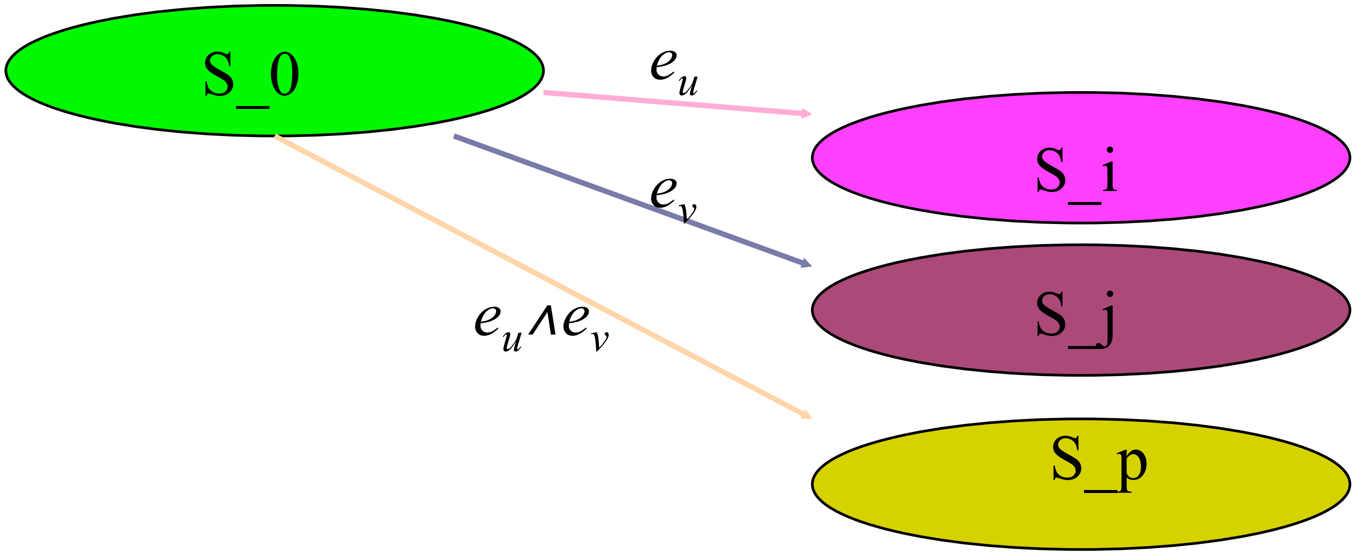
Convert a node in the tree to a module in Plausible Logic



```
1.  name(Node).
2.  type State_Type(S_0, S_1,...,S_k).
3.  v{State(S_0),…,State(S_k)}.
4.  v{¬State(S_i),¬State(S_j)}. (∀ i ≠ j)
5.  input{"e_i"}. (for i=1,…,k}
6.  Default: ⇒ State(S_0).
7.  Switch_S_0_S_i:{"e_i"} ⇒ State(S_i).
    (for i=1,…,k)
8.  Switch_S_0_S_i > Default. (for i=1,…,k)
```

Potentially equivalent modeling approaches

Using the priority relation

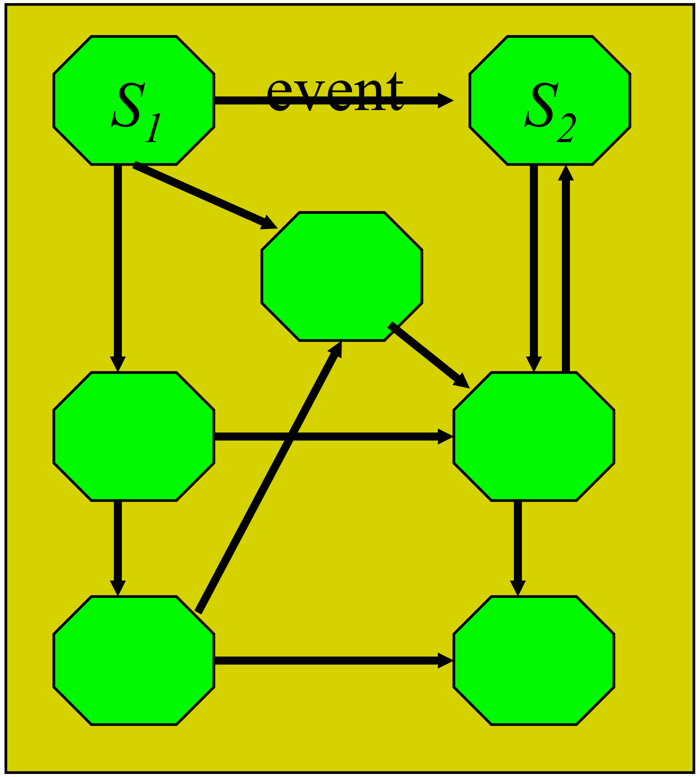


1. `Switch_S_0_S_i:{"e_u"} ⇒ State(S_i).`
2. `Switch_S_0_S_i > Default.`
3. `Switch_S_0_S_j:{"e_v"} ⇒ State(S_j).`
4. `Switch_S_0_S_j > Default.`
5. `Switch_S_0_S_p:{"e_v∧e_u"} ⇒ State(S_p).`
6. `Switch_S_0_S_p > Default.`
7. **`Switch_S_0_S_p > Switch_S_0_S_i.`**
8. **`Switch_S_0_S_p > Switch_S_0_S_i.`**

Flexibility of default reasoning

Hybrid System for Intelligent and Integrated System

- Reactive System
  - State Machine

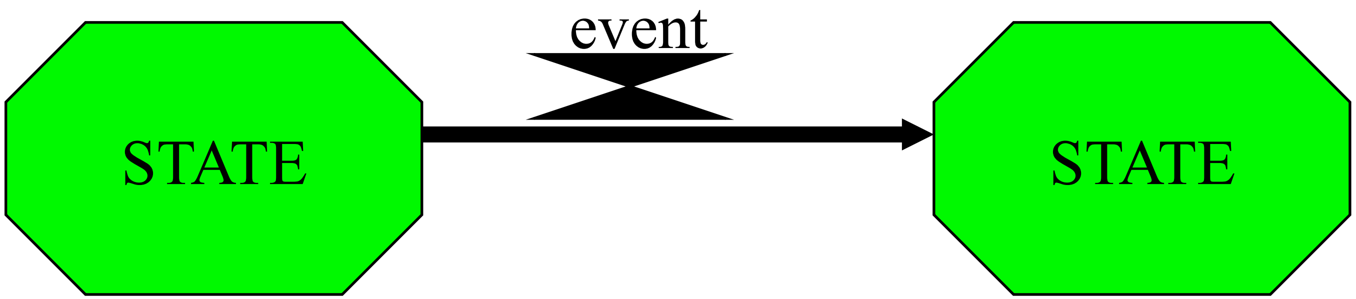


- Reasoning
  - Non-Monotonic Logic

1. name(Node).
2. type State_Type(S_0,..,S_k).
3. ⋁{State(S_0),…,State(S_k)}.
4. ⋁{¬State(S_i), ¬State(S_j)}. (∀ i ≠ j)
5. input{"e_i"}. (for i=1,…,k}
6. Default: ⇒ State(S_0).
7. Switch_S_0_S_i:{"e_i"} ⇒ State(S_i). (for i=1,…,k)
8. Switch_S_0_S_i > Default.

Behaviour Design

- Software Engineering
  - visual models of behaviour



statement from non-monotonic logic

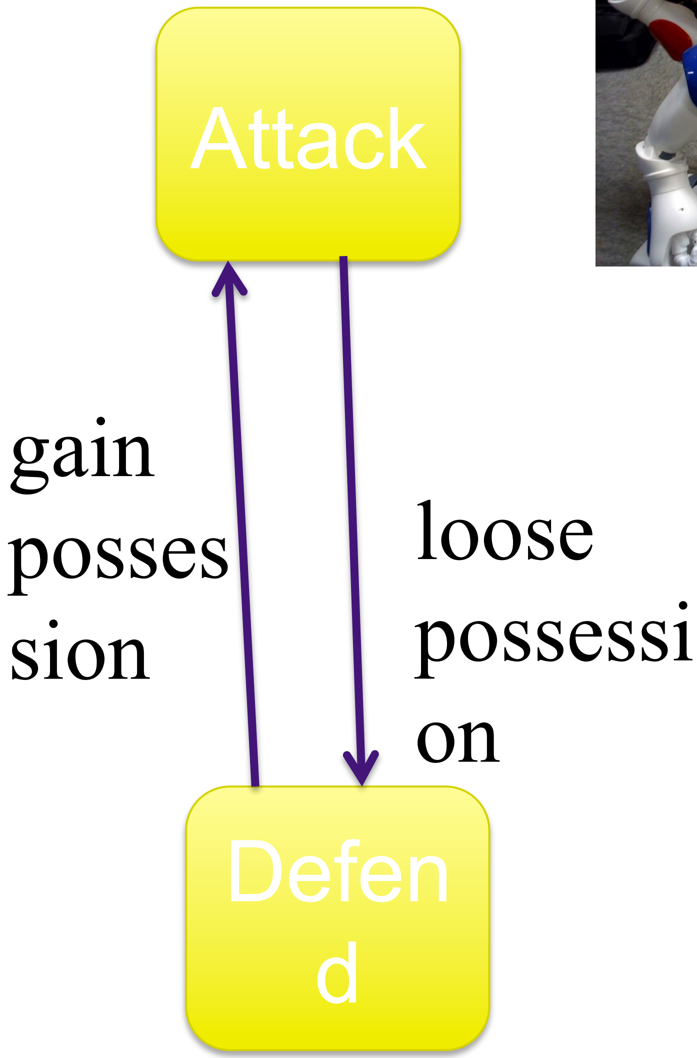
- Behaviour Specification
  - by humans
- Human-Robot Interaction

Human-Robot Collaboration

Event-Driven

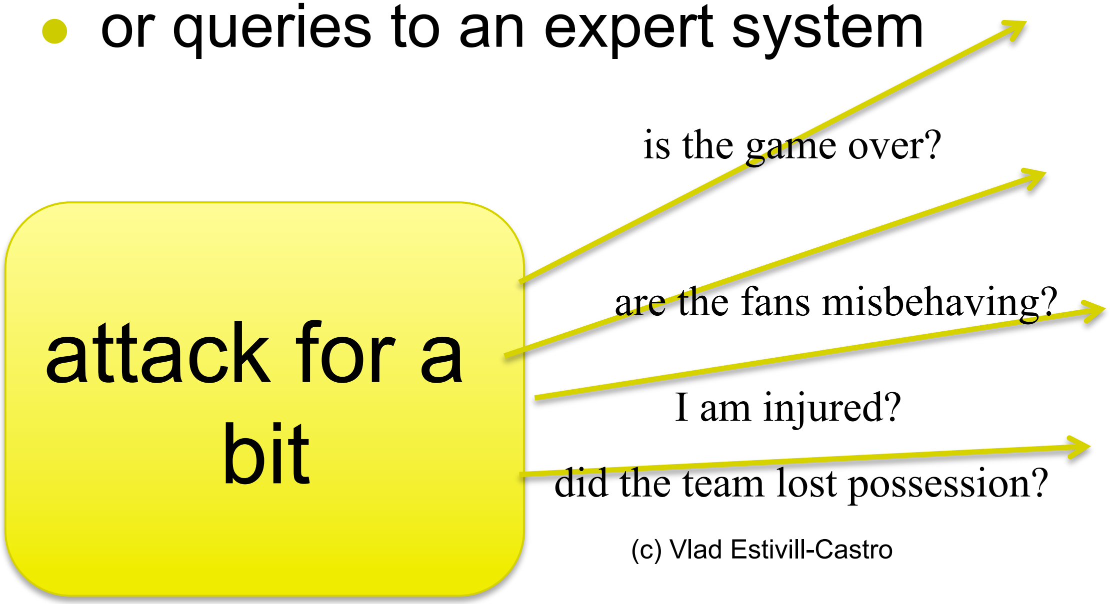
Most common approach

- System is in a state
  - waiting
  - does not change what is
    - doing/happening
  - until event arrives
- Events change the state of the system

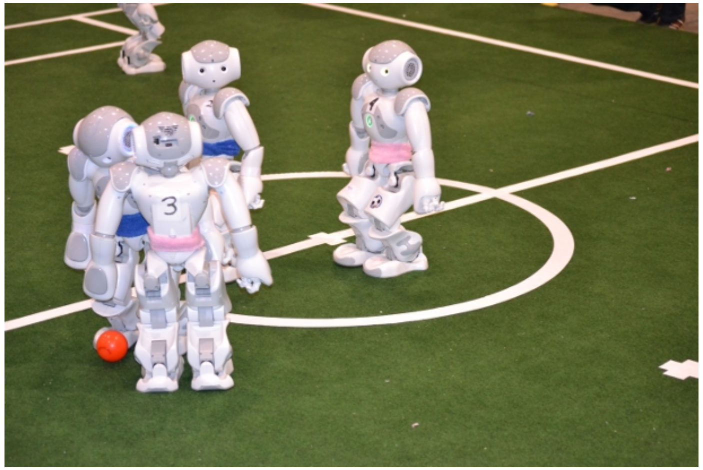


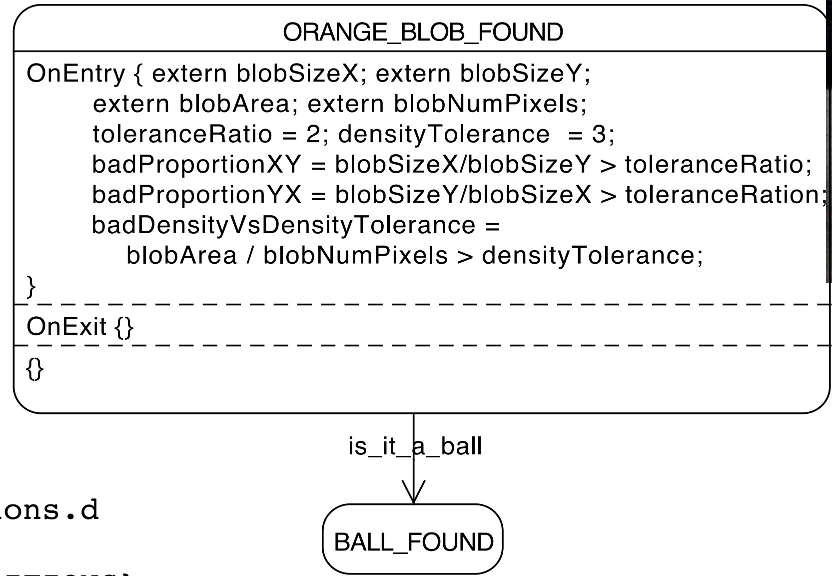
Logic-labeled FSMs

- A second view of time (since Harel's seminal paper)
  - Machines are not waiting in the state for events
  - The machines drive, execute
  - The transitions are expressions in a logic
    - or queries to an expert system



Example from robotic soccer

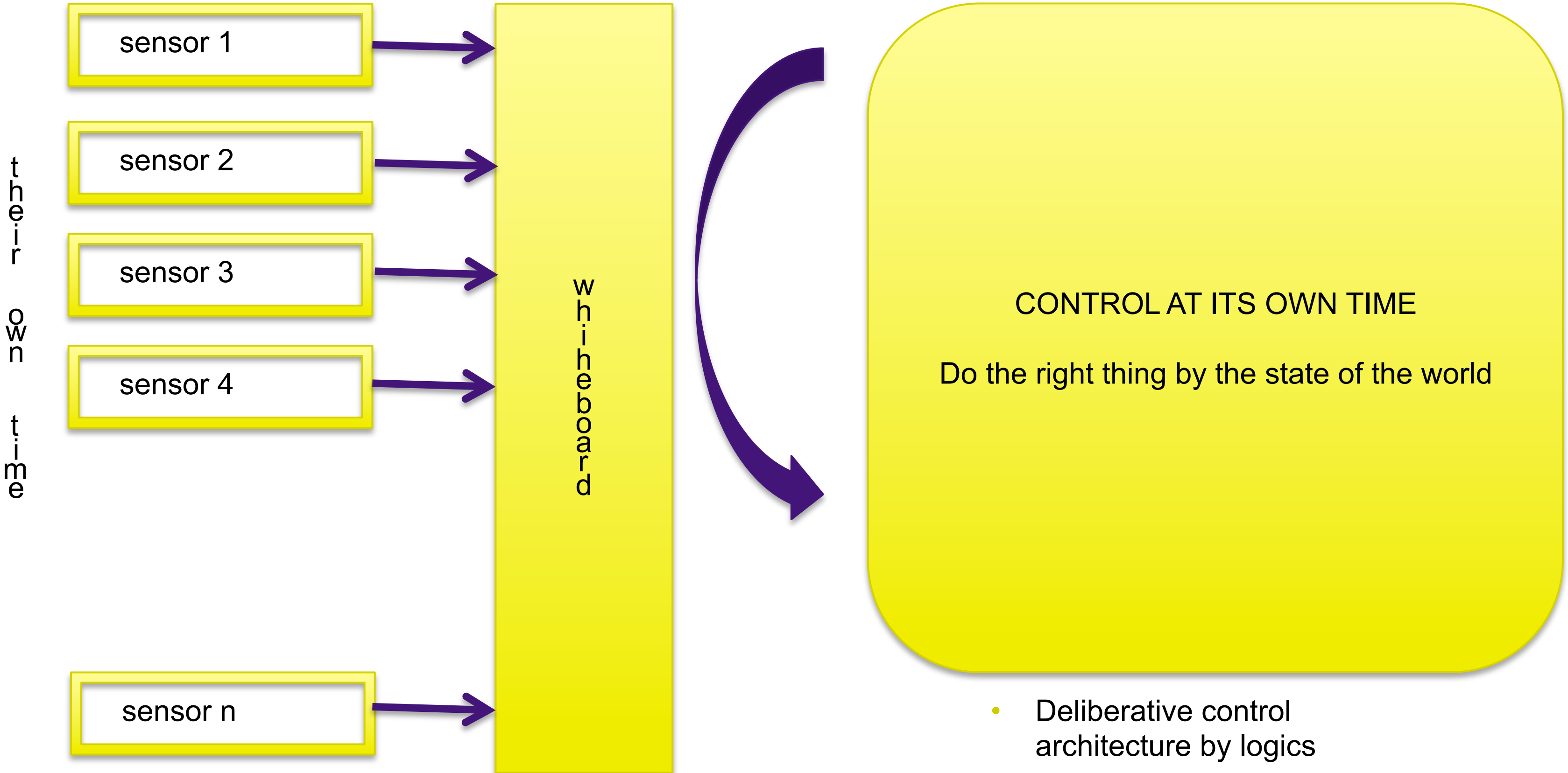




```
% BallConditions.d

name{BALLCONDITIONS}.

input{badProportionXY}.
input{badProportionYX}.
input{badDensityVsDensityTolerance}.

BC0: {}              => is_it_a_ball.
BC1: badProportionXY   =>  ~is_it_a_ball. BC1 > BC0.
BC2: badProportionYX   =>  ~is_it_a_ball. BC2 > BC0.
BC3: badDensityVsDensityTolerance  =>  ~is_it_a_ball. BC3 > BC0.

output{b is_it_a_ball, "is_it_a_ball"}.
```

Conceptual cycle

- Similar to a ‘reactive-architecture’
- Similar to a whiteboard architecture



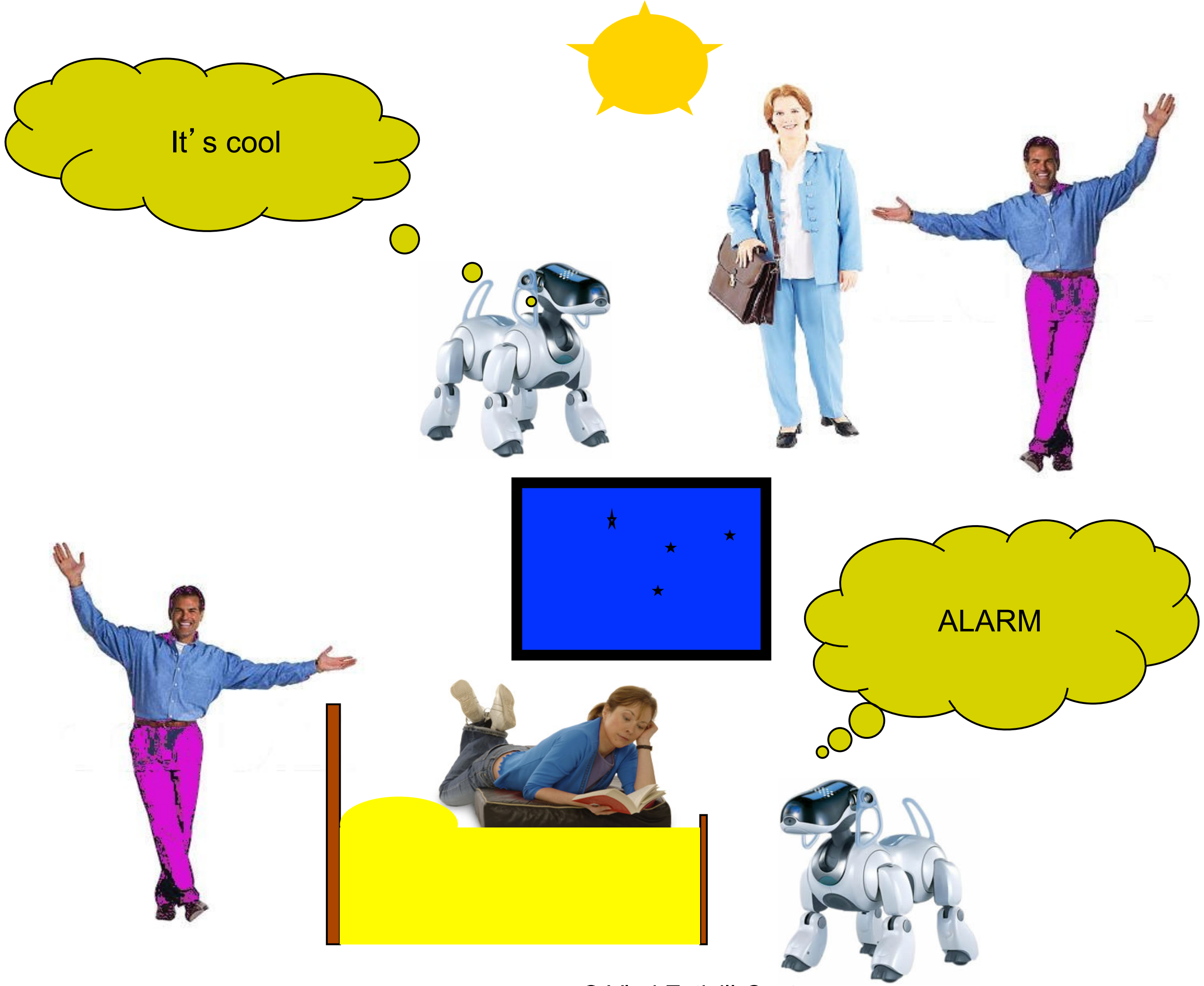
sensor space of the robot

- Deliberative control architecture by logics
- Behavior-base control by vectors of FSMs

Outline

- Motivation
- Robotics and Software Engineering
- Why State Machines and Why Logic
- Examples
- Comparison
- Model Checking
- Architecture
- Illustrations
- Summary

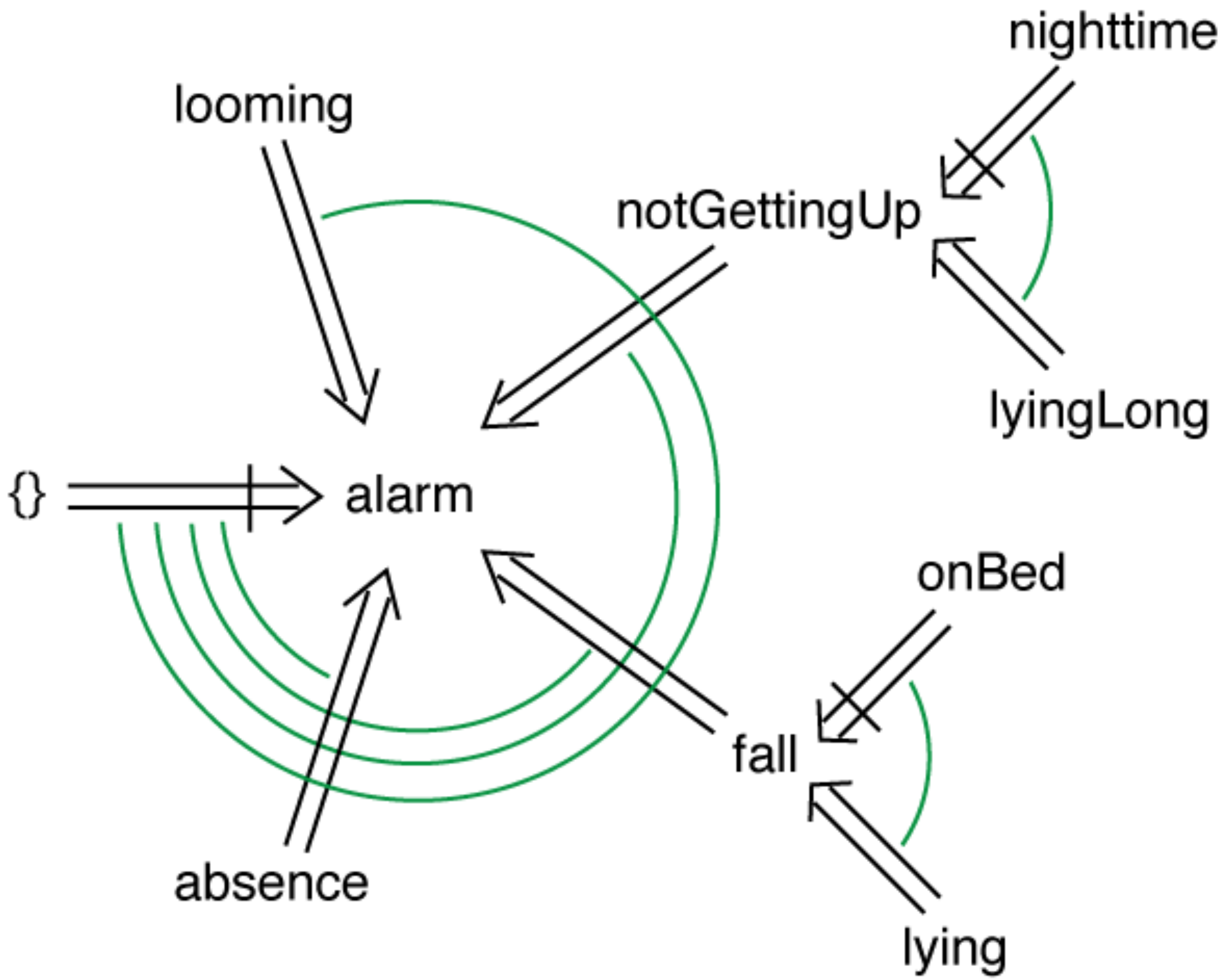
Prototype demonstrated at RoboCup@Home 2007



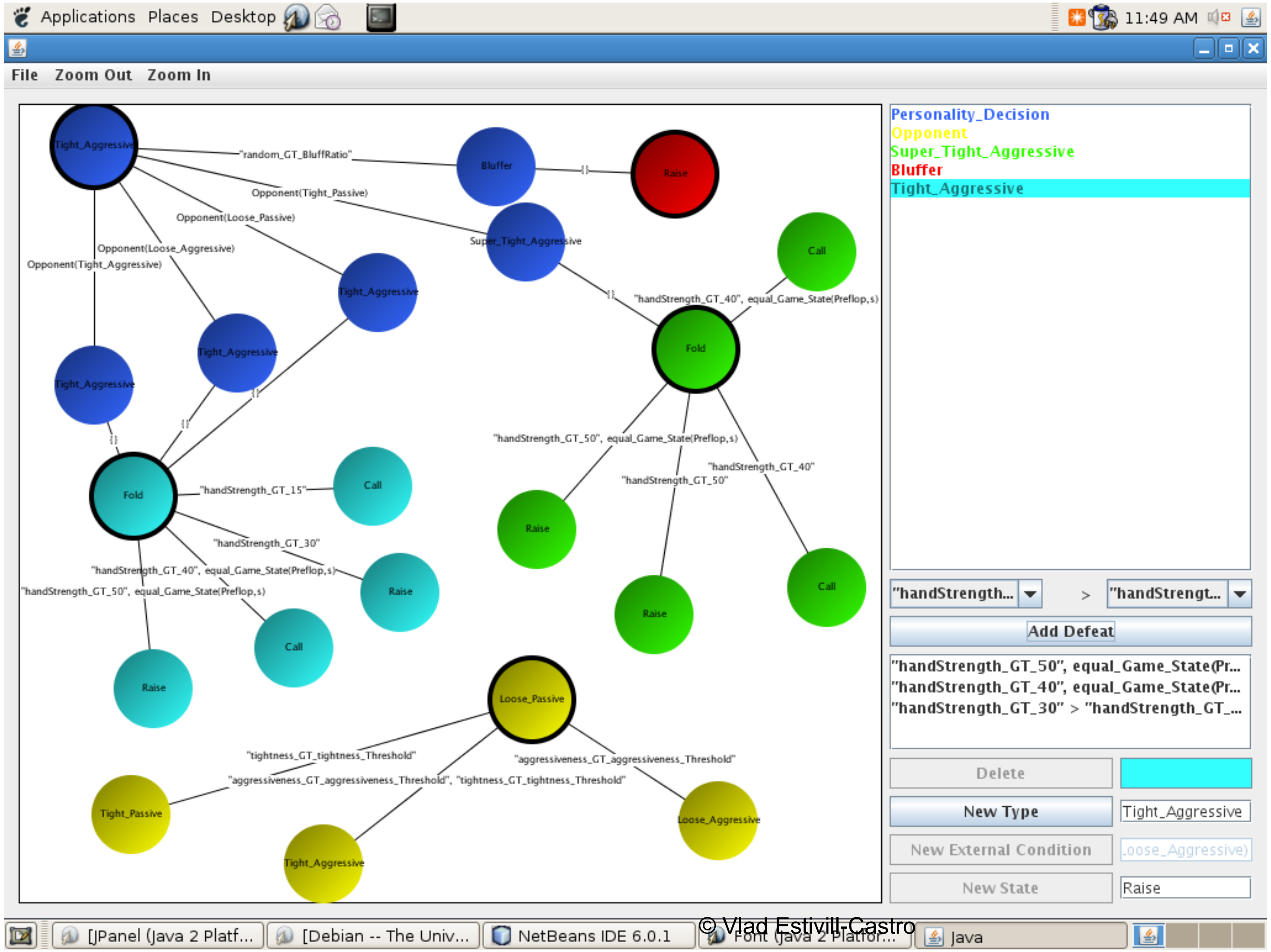
A logic for looking after the lady

1. Usually there is no reason for alarm
2. The absence of owner for a long time is reason for alarm (this takes precedence over rule 1)
3. Lying usually results from a fall
4. A fall is usually a reason for alarm (this takes precedence over rule 1)
5. Being on bed is not a fall (this takes precedence over rule 4)
6. Lying for a long time means owner is not getting up.
7. Not getting up is a reason for alarm (this takes precedence over rule 1)
8. If it is night, it is fine not to get up (this takes precedence over rule 7)
9. If there is a stranger looming over the lady, it is reason for an alarm (takes precedence over rule 1)
10. Owner can't be absent while on bed, or lying or lying for a long time.
11. Owner can't be lying for a long time without lying for a short time.

Diagrams to illustrate rule relations



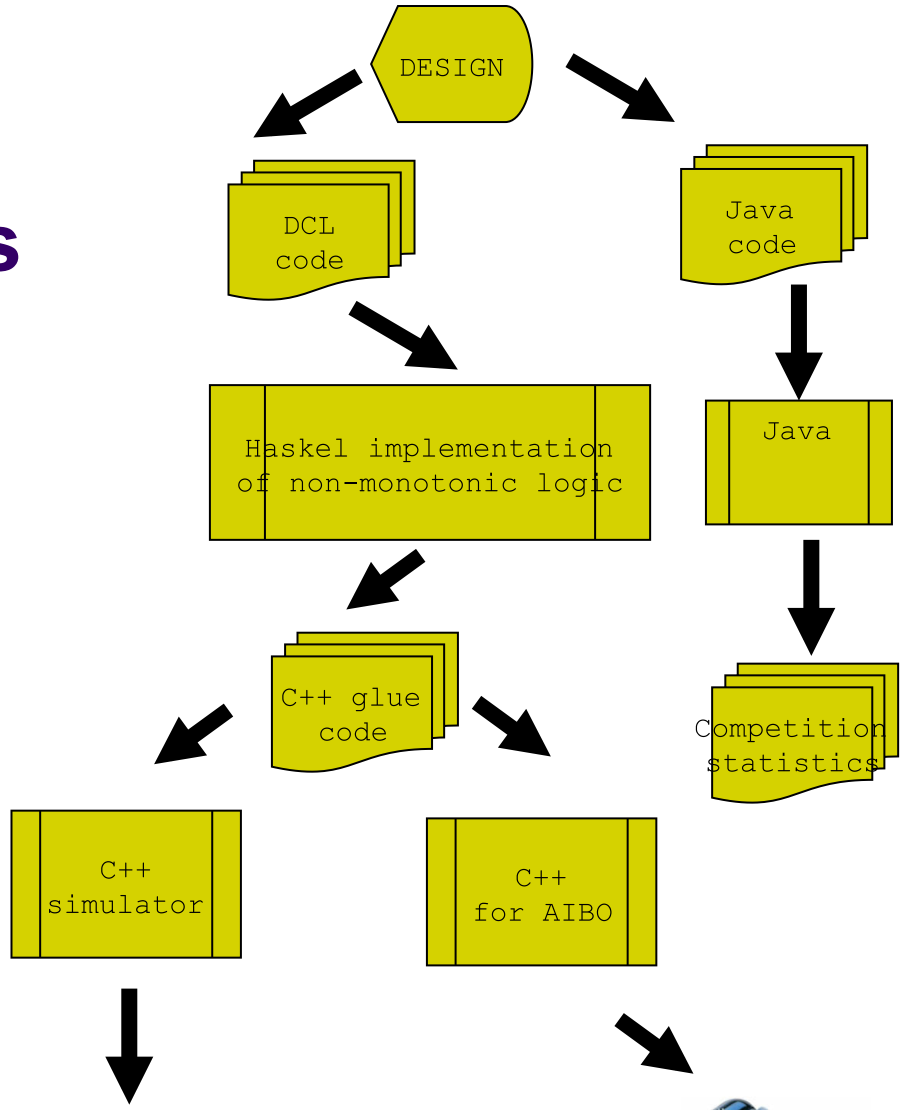
A diagram for a poker player

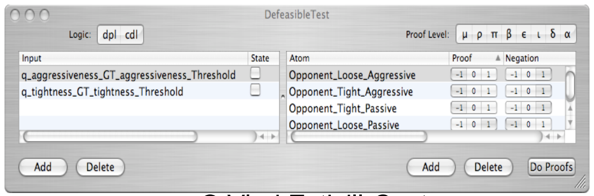


Code generated (example)

```
/* This is code Generated by the DPLGenerator
** This program was made by Mark Johnson 2008 (MiPAL)
** File Opponent.d
*/

name{Opponent}.

type Opponent(x<-Opponent_Type).

type Opponent_Type = {Loose_Passive, Loose_Aggressive, Tight_Passive, Tight_Aggressive}.

\/{Opponent(Loose_Passive), Opponent(Loose_Aggressive), Opponent(Tight_Passive), Opponent(Tight_Aggressive)}.

\/{~Opponent(Loose_Passive),~Opponent(Loose_Aggressive)}.
\/{~Opponent(Loose_Passive),~Opponent(Tight_Passive)}.
\/{~Opponent(Loose_Passive),~Opponent(Tight_Aggressive)}.
\/{~Opponent(Loose_Aggressive),~Opponent(Tight_Passive)}.
\/{~Opponent(Loose_Aggressive),~Opponent(Tight_Aggressive)}.
\/{~Opponent(Tight_Passive),~Opponent(Tight_Aggressive)}.

input{"aggressiveness_GT_aggressiveness_Threshold"}.
input{"tightness_GT_tightness_Threshold"}.

Default_Opponent: {}=>Opponent(Loose_Passive).

Switch_aggressiveness_GT_aggressiveness_Threshold: {"aggressiveness_GT_aggressiveness_Threshold"} => Opponent(Loose_Aggressive).
Switch_aggressiveness_GT_aggressiveness_Threshold > Default_Opponent.

Switch_tightness_GT_tightness_Threshold: {"tightness_GT_tightness_Threshold"} => Opponent(Tight_Passive).
Switch_tightness_GT_tightness_Threshold > Default_Opponent.

Switch_aggressiveness_GT_aggressiveness_Threshold_n_tightness_GT_tightness_Threshold: {"aggressiveness_GT_aggressiveness_Threshold",
        "tightness_GT_tightness_Threshold"} => Opponent(Tight_Aggressive).
Switch_aggressiveness_GT_aggressiveness_Threshold_n_tightness_GT_tightness_Threshold > Default_Opponent.

Switch_aggressiveness_GT_aggressiveness_Threshold_n_tightness_GT_tightness_Threshold > Switch_tightness_GT_tightness_Threshold.
Switch_aggressiveness_GT_aggressiveness_Threshold_n_tightness_GT_tightness_Threshold > Switch_aggressiveness_GT_aggressiveness_Threshold.
```

Earlier Process to Embed Design into the AIBO Robot

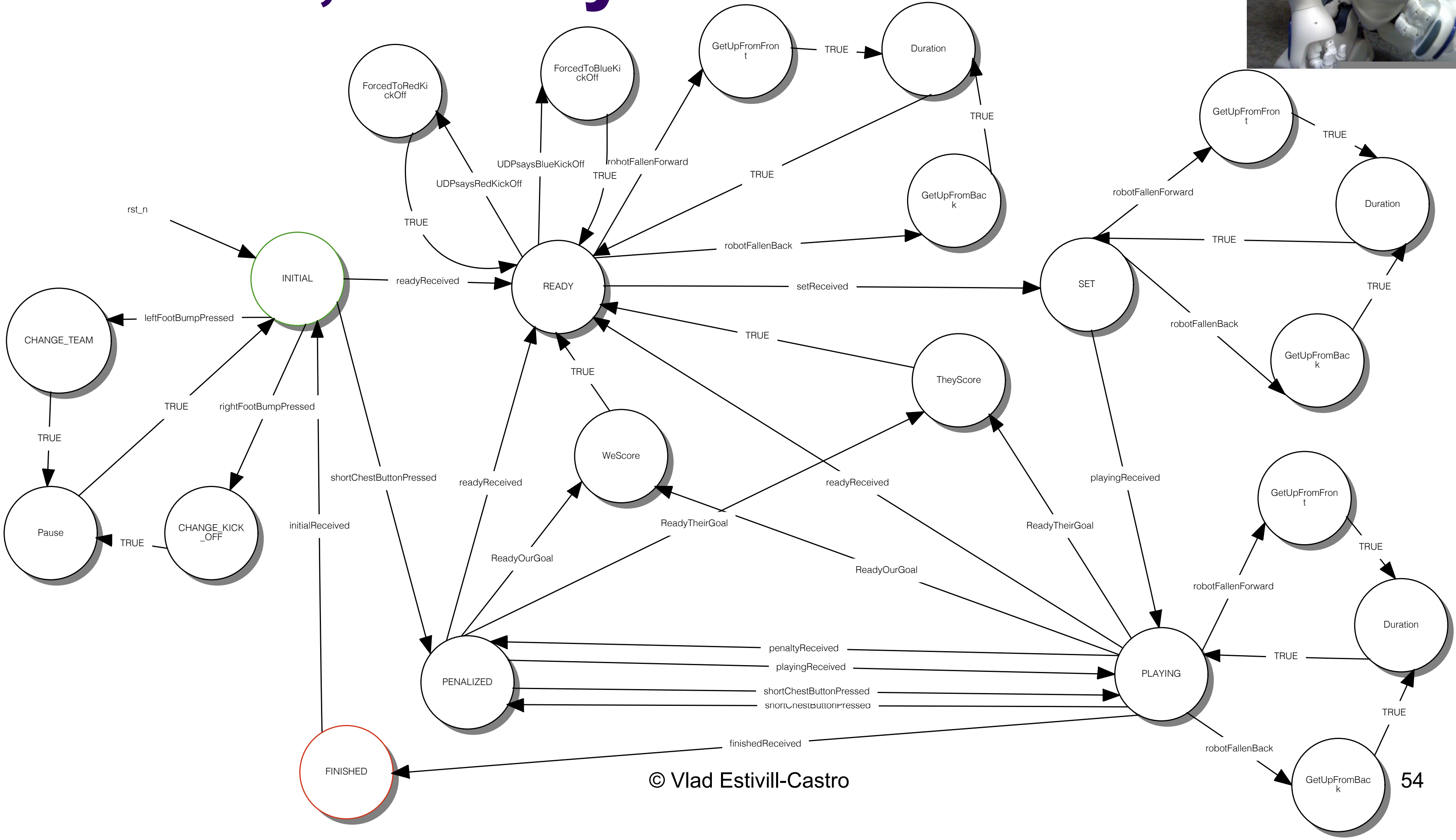




Systems interacting with humans

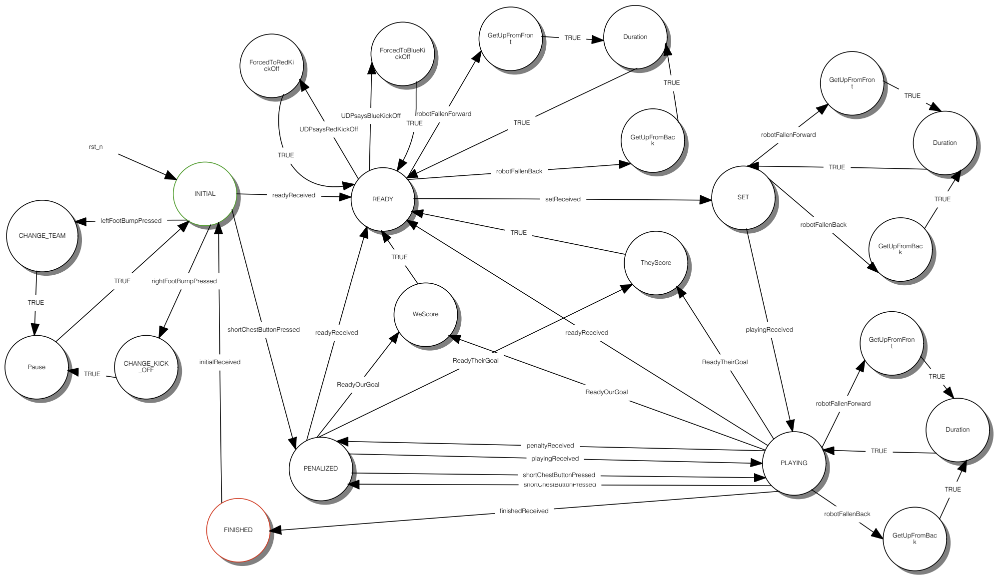
http://www.youtube.com/watch?v=QDxzPzuvFK0&fea

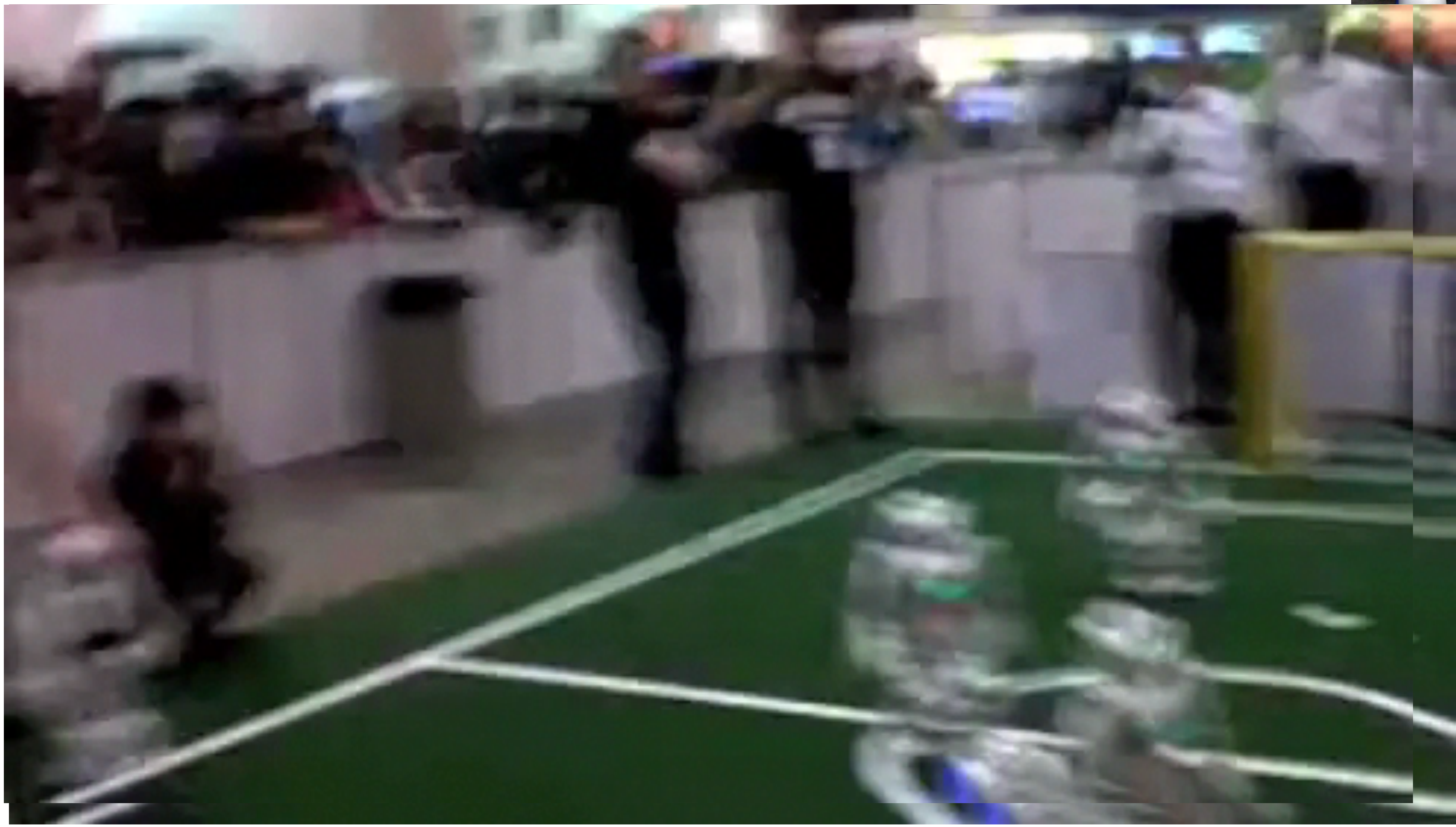
FSM+DPL for top behaviour at competition in RoboCup 2011 Istanbul, Turkey



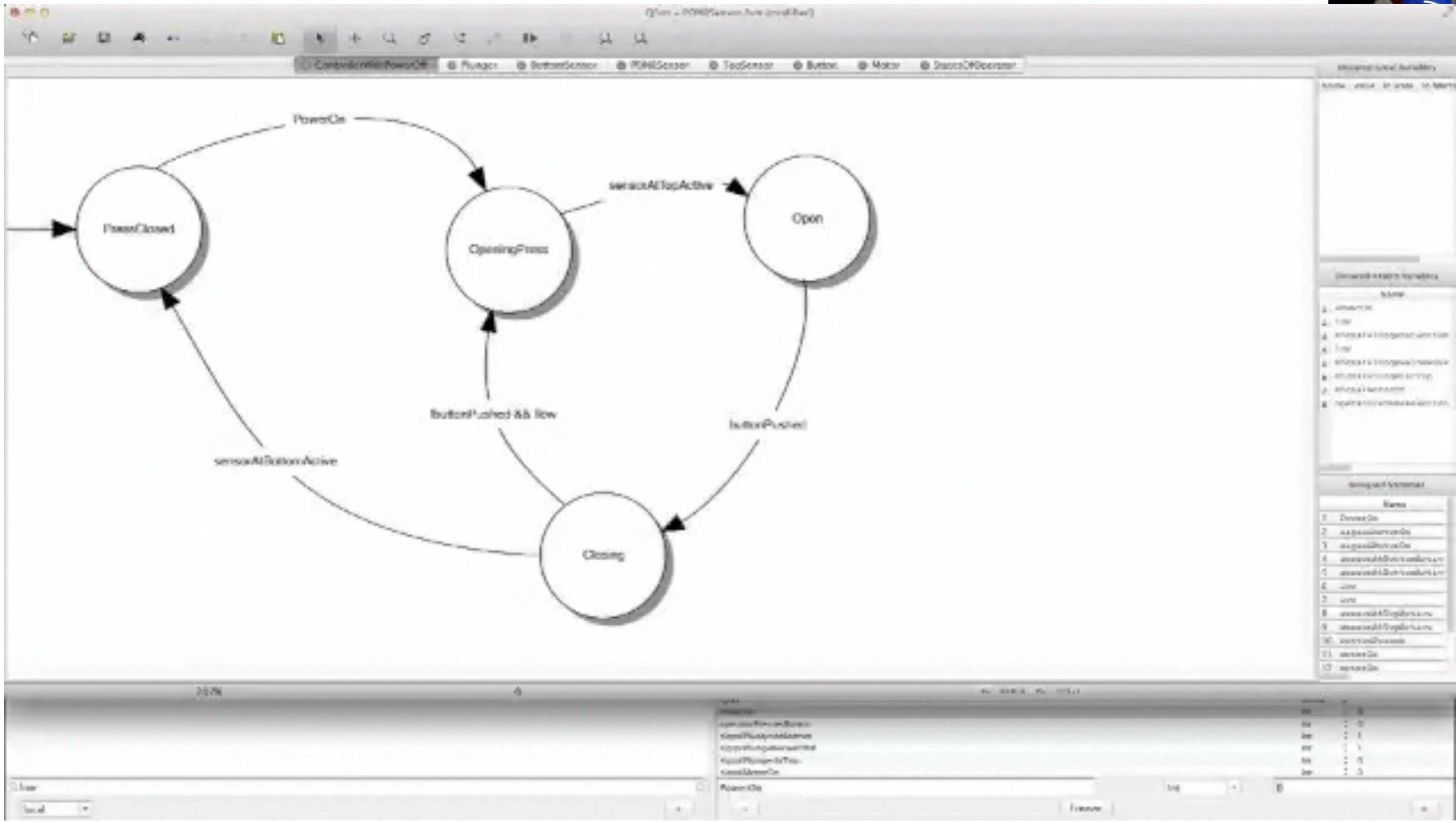
FSM+DPL for top behaviour at competition in RoboCup 2012 Mexico City

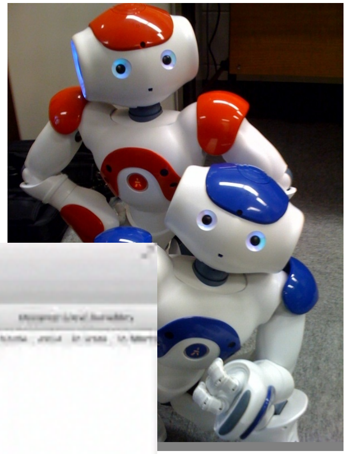
```
export MASTER=StateMachineStarter
export FSMS="SMButtonChest
 SMButtonLeftFoot
 SMButtonRightFoot
SMRobotPosition
SMGetUp
SMGameController
SMPlayer
SMGoalie
SMBallFollower
SMKicker
SMHeadBallFindAndTrack
SMHeadScan
SMRightFootControl
SMLeftFootControl
SMHeadGoalFindAndTrack
SMBallSeeker
SMReady"
```





Simulator



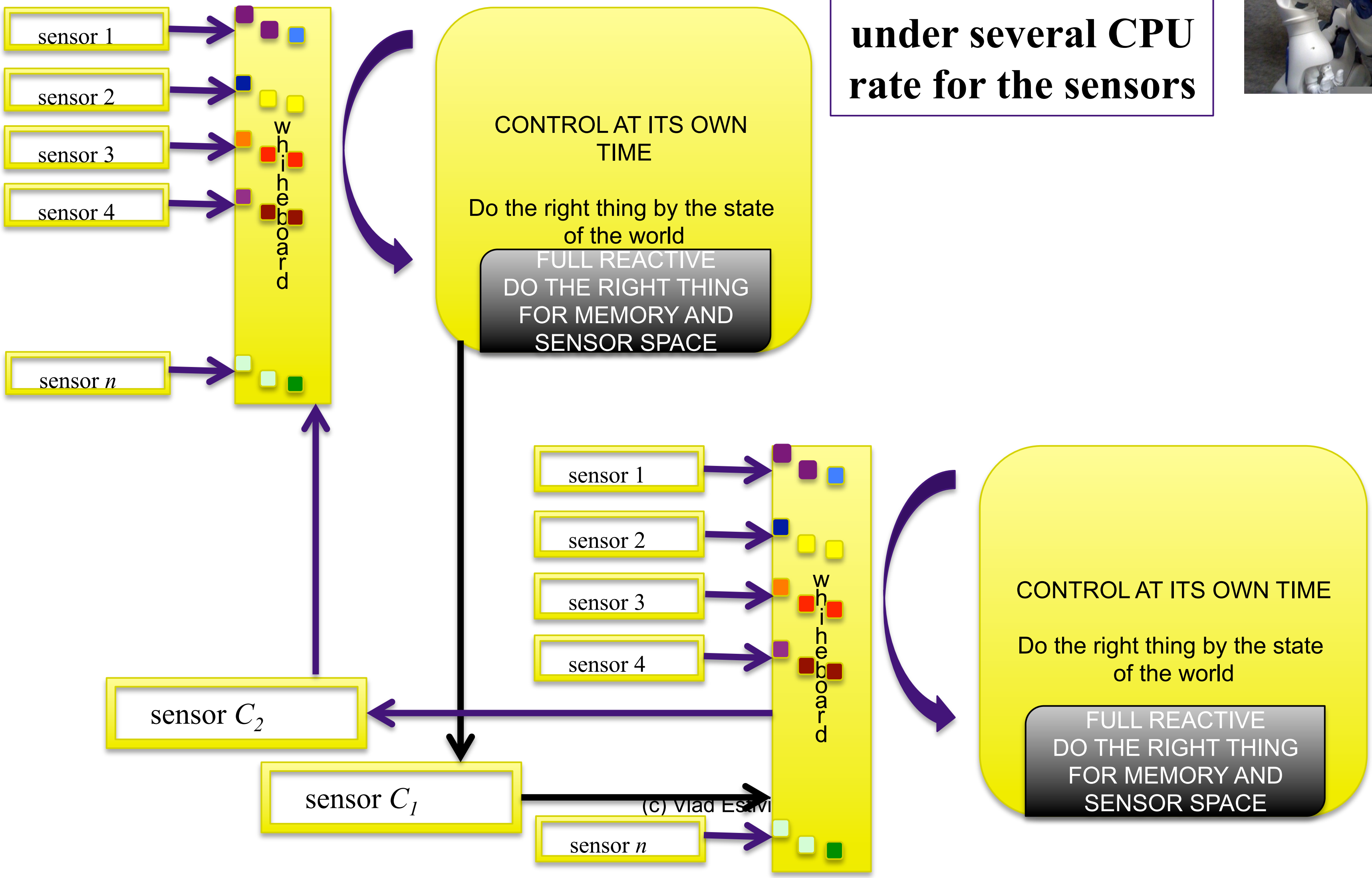


http://www.youtube.com/watch?v=FpVUSrvLI0c&

On-line debugging and simulation

Conceptual cycle

- Similar to a 'reactive-architecture'
- Similar to a whiteboard architecture



under several CPU rate for the sensors

A classical example

- The One-Minute Microwave Oven
  - literature approach
  - behavior specification of all objects of a class
    - Shlaer-Mellor
  - StateWorks
  - Behavior Trees
  - PetriNets
  - SCXML - State Chart XML: State Machine Notation for Control Abstraction
  - Realistic - scaled down version of an X-Ray machine

One Minute Microwave

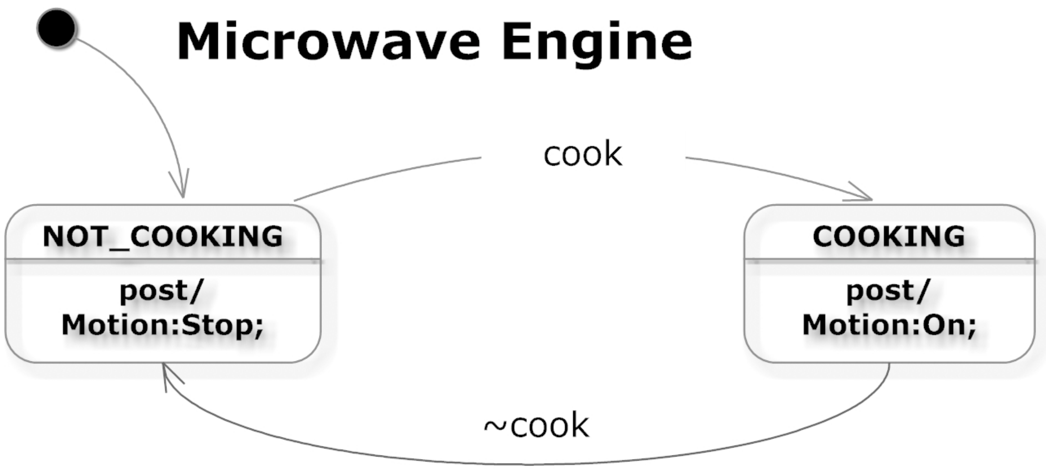
- Widely discussed in the literature of software engineering
- Analogous to the X-Ray machine
  - Therac-25 radiation machine that caused harm to patients
- Important SAFETY feature
  - OPENING THE DOOR SHALL STOP THE COOKING

Requirements (One-Minute Microwave Oven)

Requirements	Description
R1	There is a single control button available for the use of the oven. If the oven is closed and you push the button, the oven will start cooking (that is, energize the power-tube) for one minute
R2	If the button is pushed while the oven is cooking, it will cause the oven to cook for an extra minute.
R3	Pushing the button when the door is open has no effect.
R4	Whenever the oven is cooking or the door is open, the light in the oven will be on.
R5	Opening the door stops the cooking. and stops the timer and does not clear the timer
R6	Closing the door turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.
R7	If the oven times out, the light and the power-tube are turned off and then a beeper emits a warning beep to indicate that the cooking has finished.

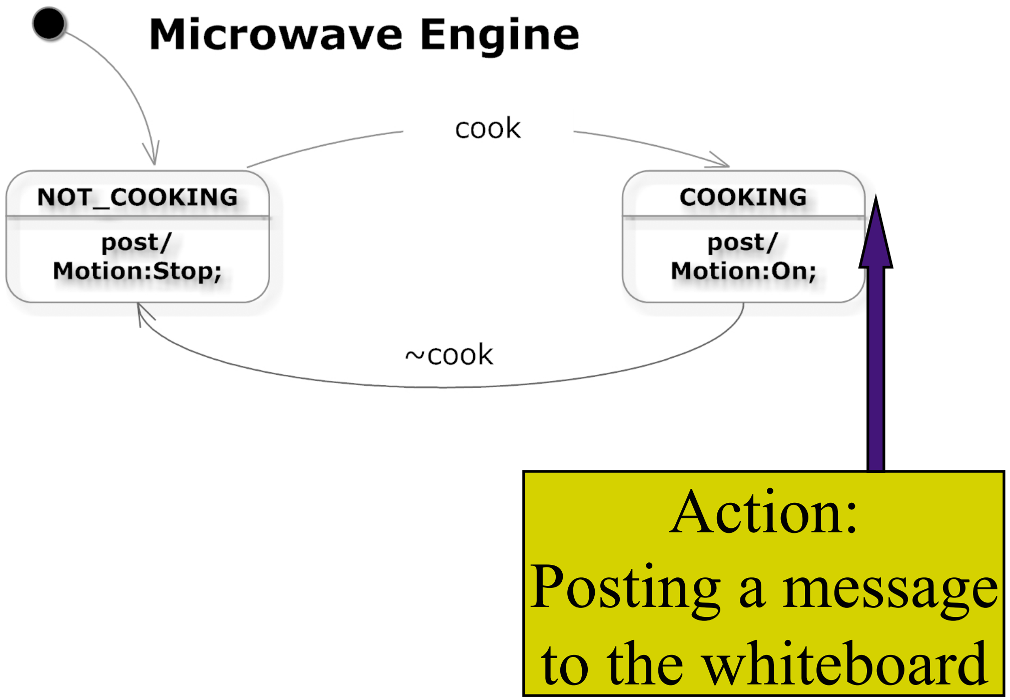
The DPL+State_Machine approach

- Step 1: Consider writing the script of music for an orchestra. Write individual scripts and place together all actuators that behave with the same actions for the same cues
- Example: The control of the tube (energizing), the fan and the spinning plate



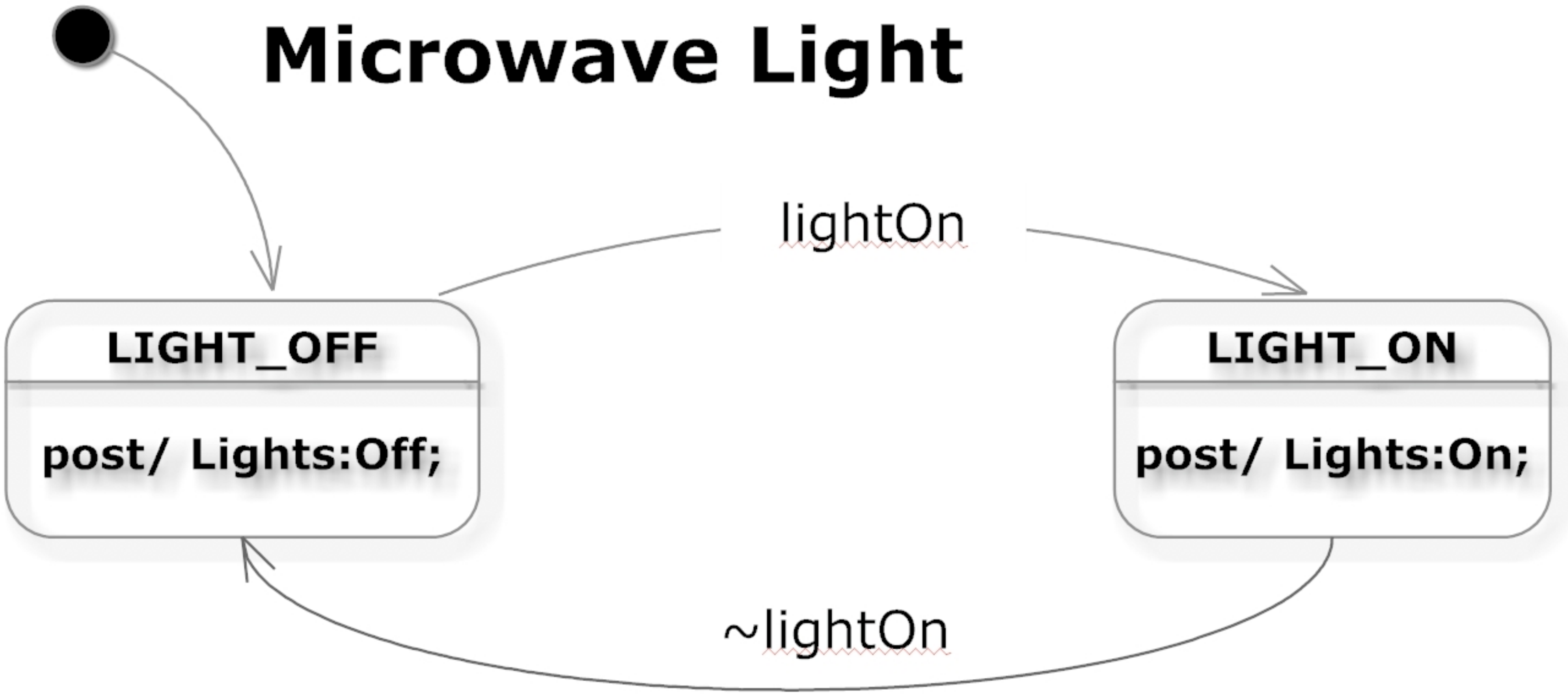
Step 2: Describe the conditions that result in the need to change state

```
% MicrowaveCook.d

name{MicrowaveCook}.

input{timeLeft}.
input{doorOpen}.

C0: {}        => ~cook.
C1: timeLeft =>  cook. C1 > C0.
C2: doorOpen => ~cook. C2 > C1.

output{b cook, "cook"}.
```



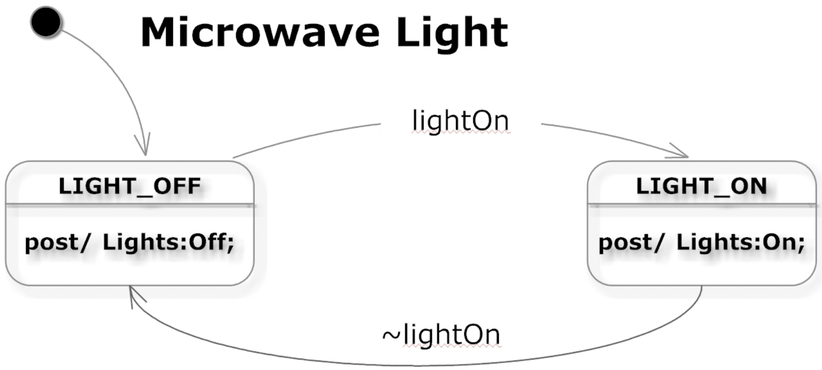
Step 1 (again): Analyze another actuator

- Illustration: The light



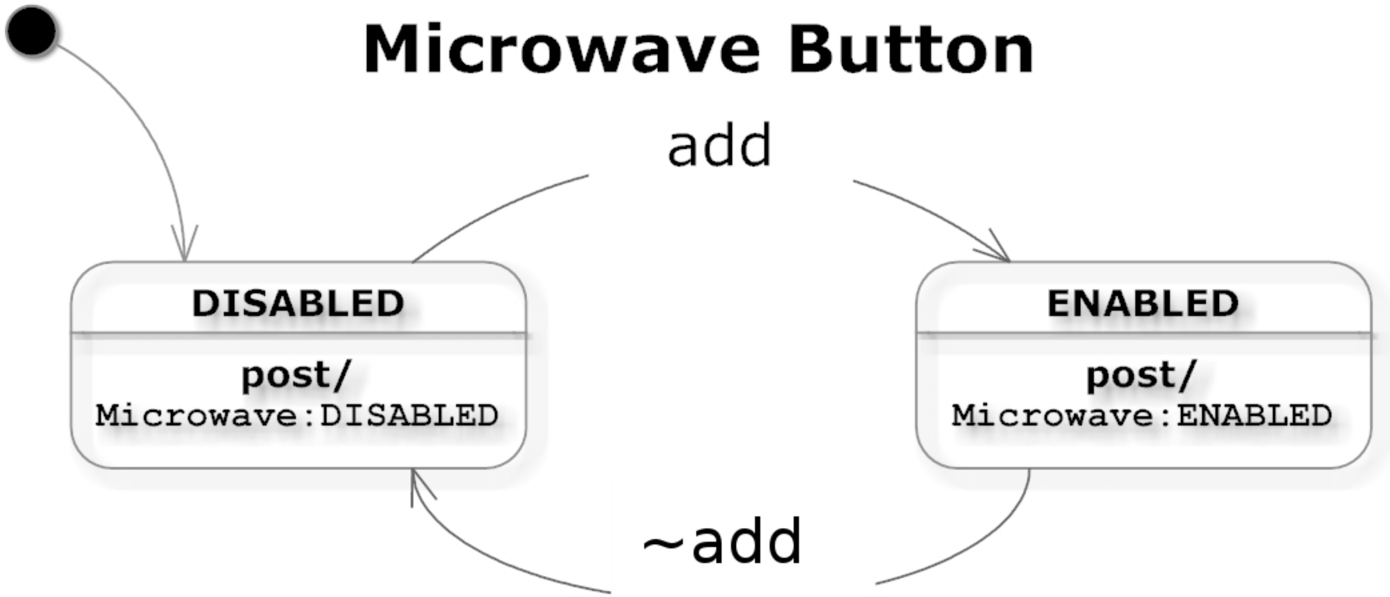
Step 2 (again): Describe the conditions that result in the need to change state

```
% MicrowaveLight.d

name{MicrowaveLight}.

input{timeLeft}.
input{doorOpen}.

L0: {}        => ~lightOn.
L1: timeLeft =>  lightOn. L1 > L0.
L2: doorOpen =>  lightOn. L2 > L0.

output{b lightOn, "lightOn"}.
```



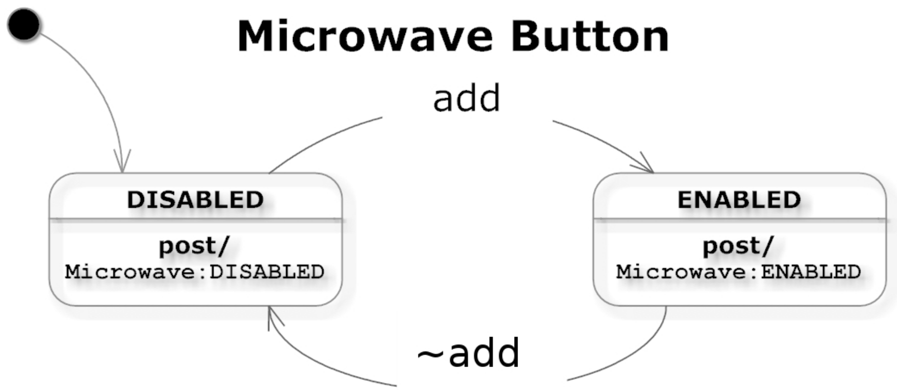
Step 1 (again): Analyze another actuator

- Illustration: The button



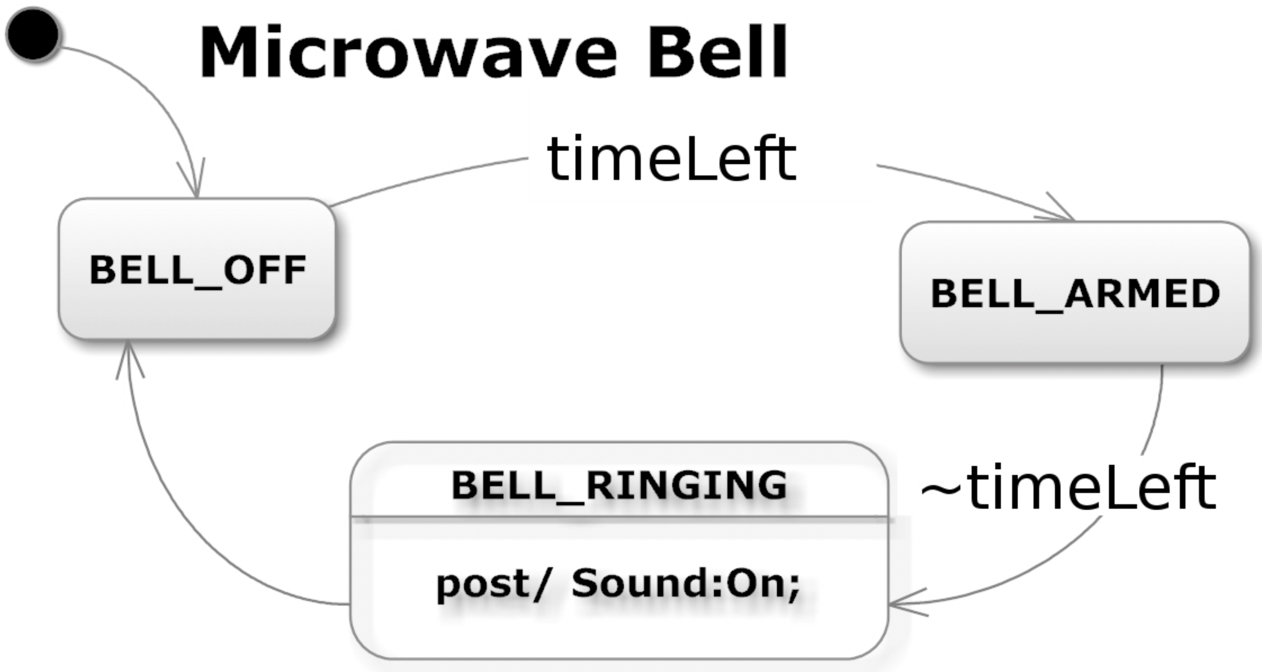
Step 2 (again): Describe the conditions that result in the need to change state

```
% MicrowaveButton.d

name{MicrowaveButton}.

input{doorOpen}.
input{buttonPushed}.


CB0: {}            => ~add.
CB1: buttonPushed =>  add. CB1 > CB0.
CB2: doorOpen      => ~add. CB2 > CB1.


output{b add, "add"}.
```



Step 1 (again): Analyze another actuator

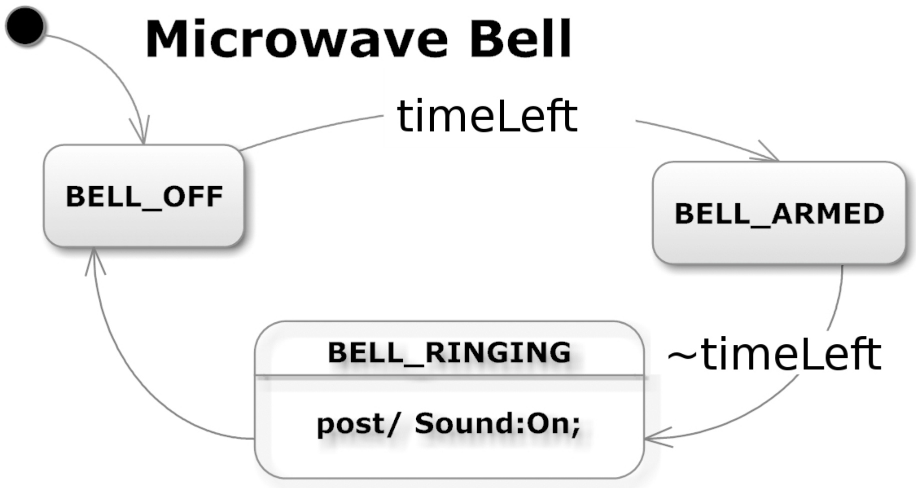
- Illustration: The bell



Step 2 (again): Describe the conditions that result in the need to change state

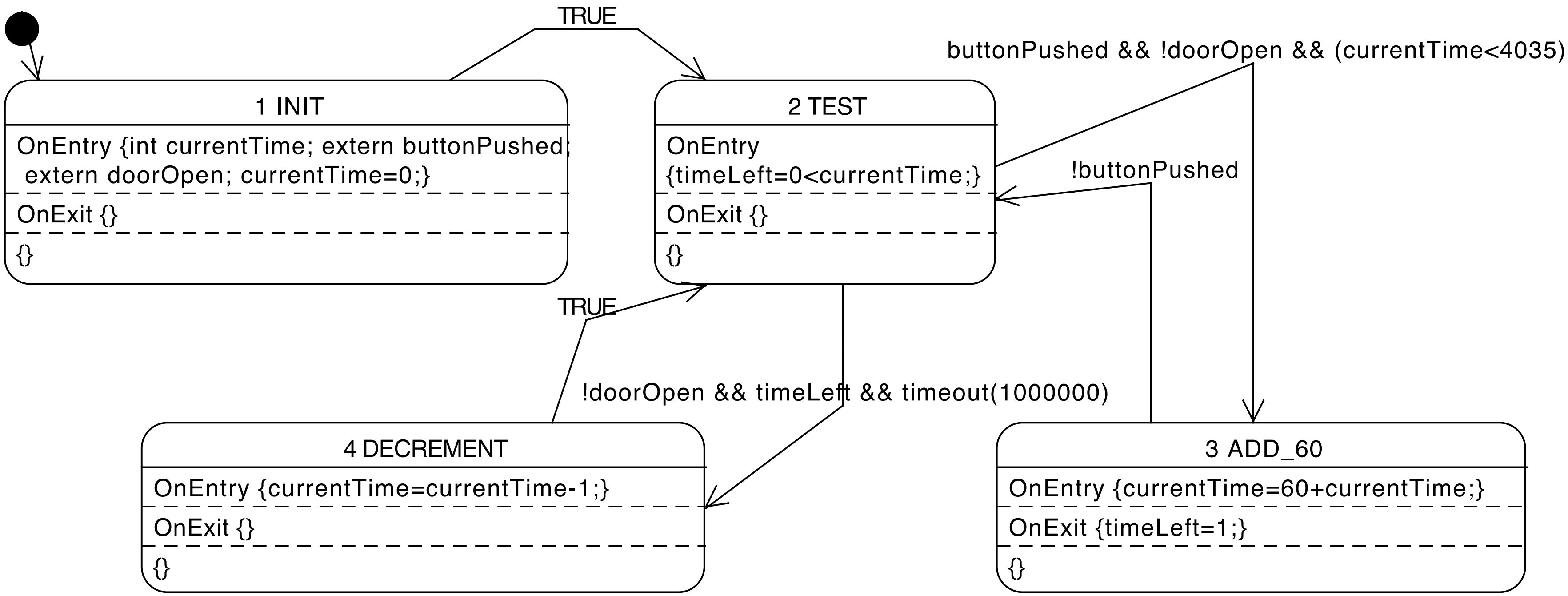
No need for a logic: `timeLeft`

- posted by another module
- does not require a proof

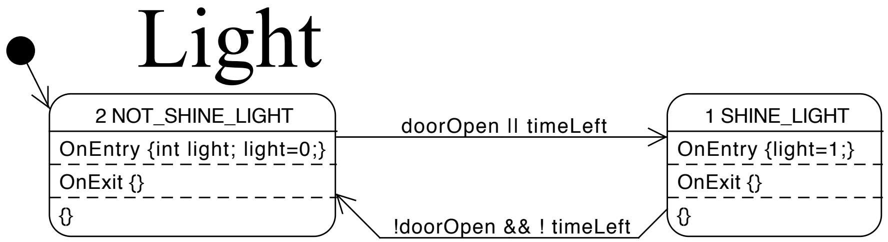


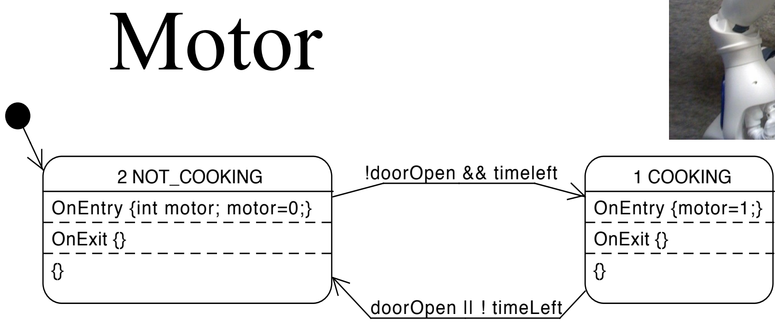
Step 1 (again): Analyze another actuator

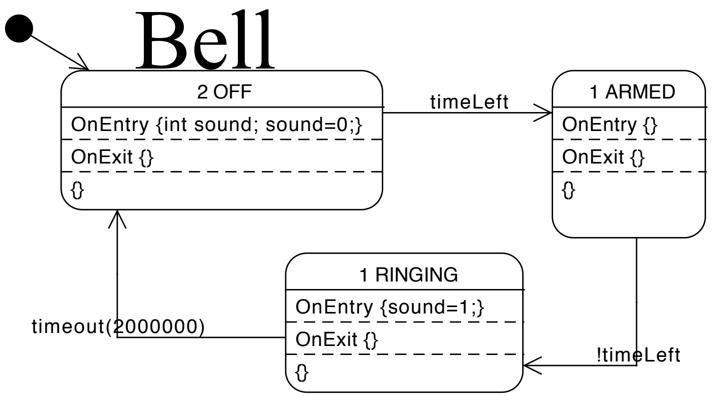
- Illustration: The timer

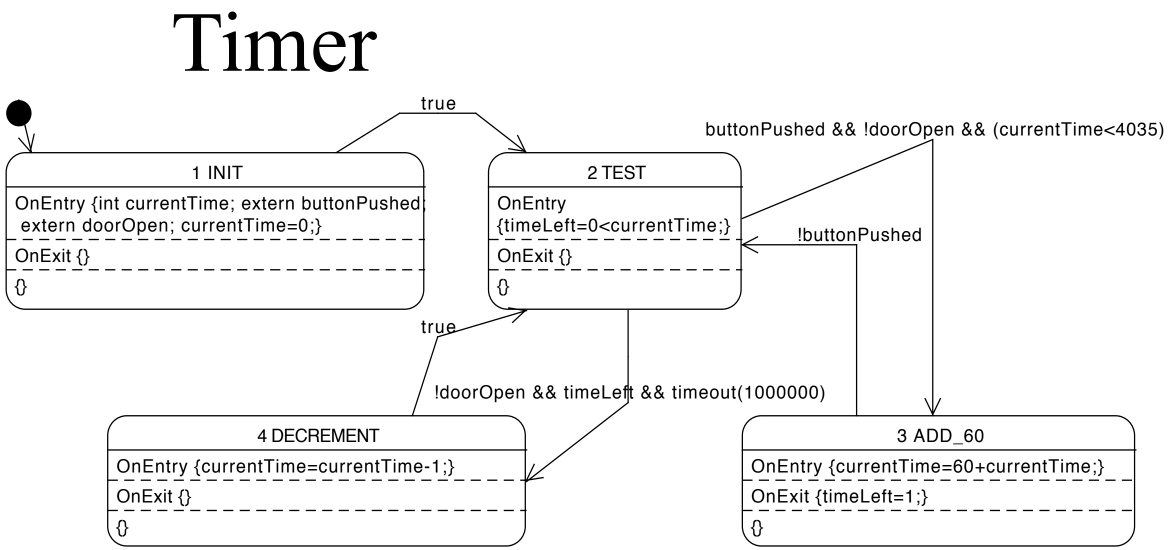


The complete arrangement



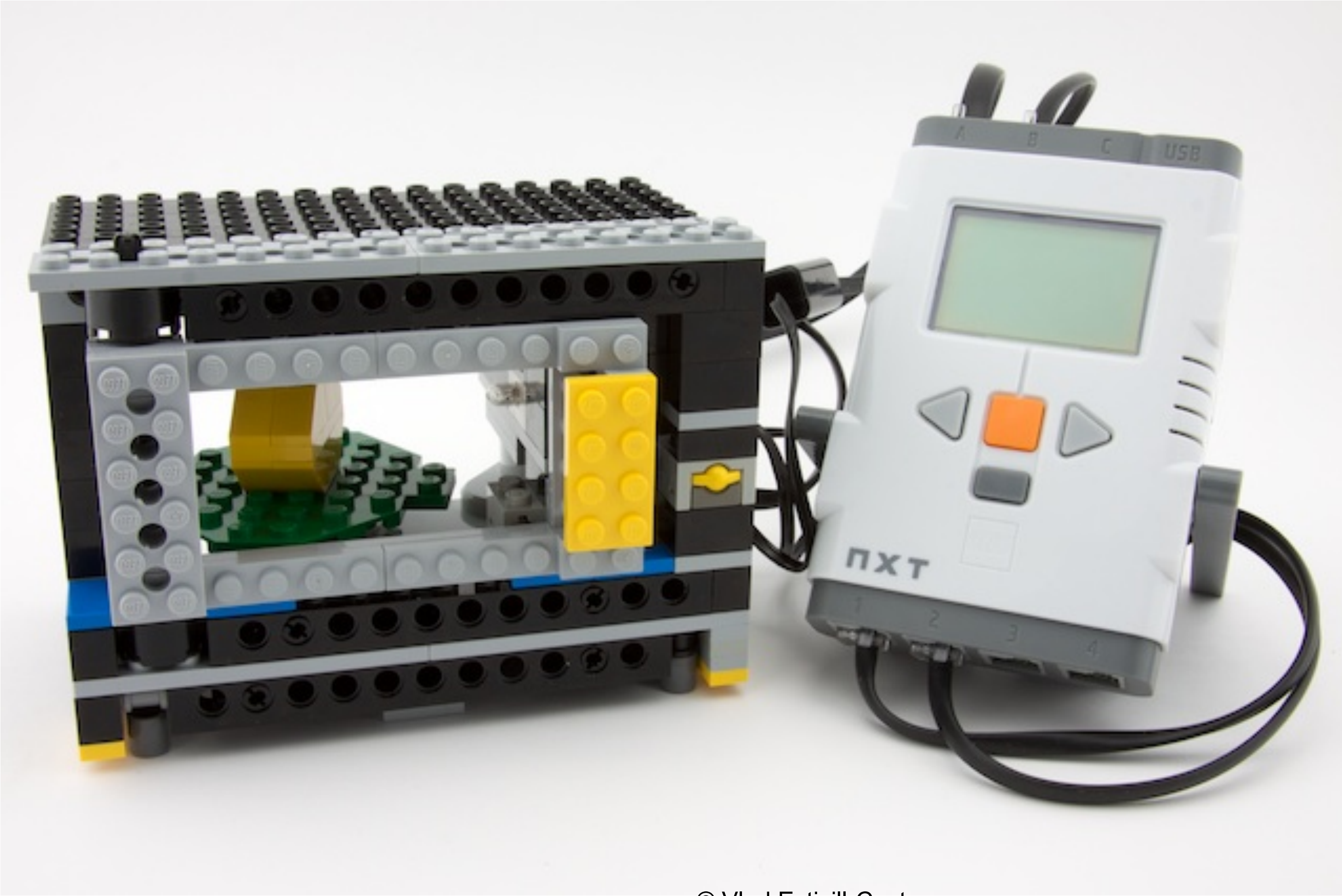






That is all folks!

www.youtube.com/watch?v=iEkCHqSfMco



http://www.youtube.com/watch?v=Dm3SP3q9_VE

© V. Estivill-Castro
75

Outline

- Motivation
- Why State Machines and Why Logic
- Examples
- Comparison
- Architecture

Contrast of sequential execution

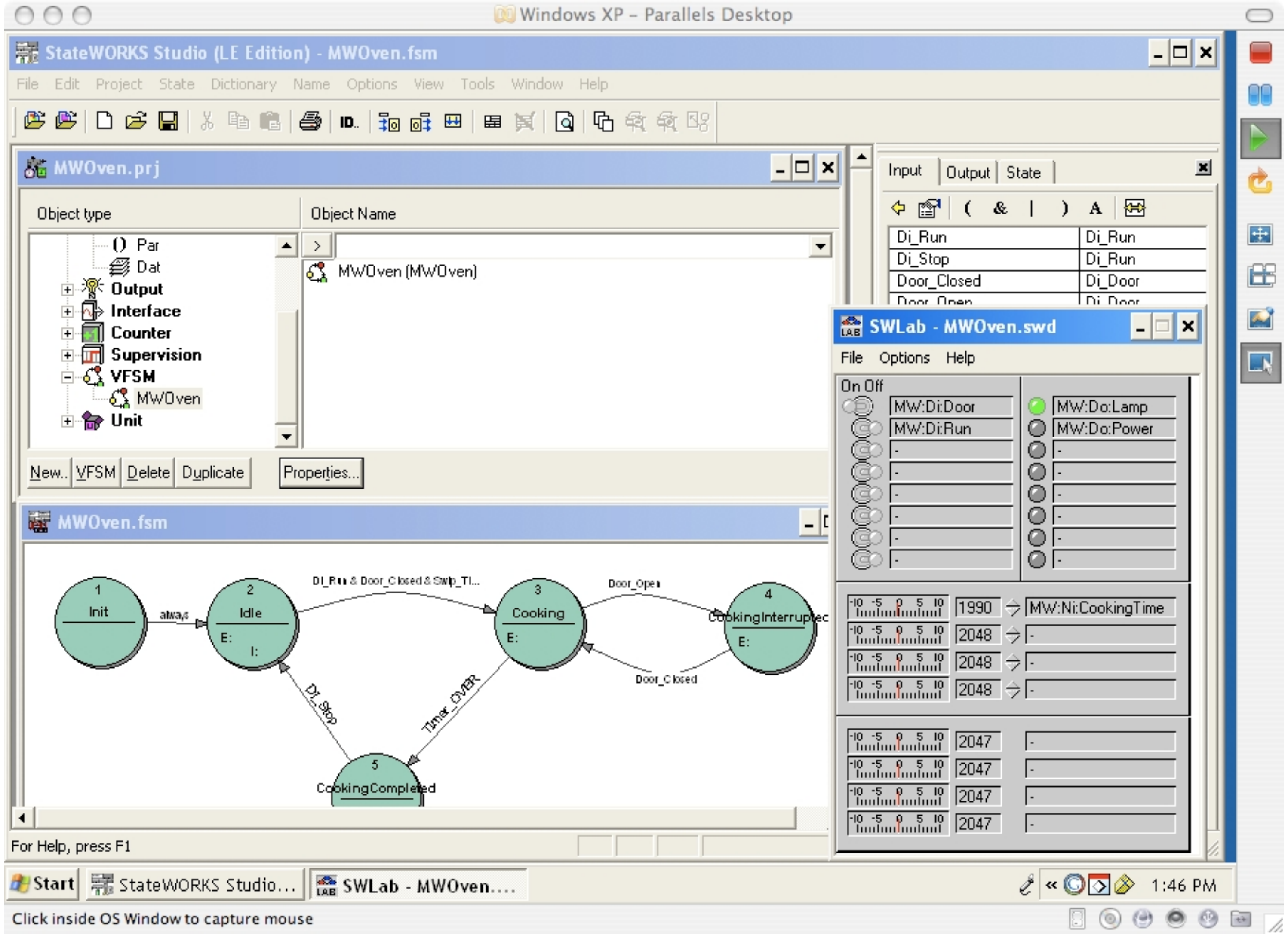
event-driven models

- allow open concurrency
- this means the state of the system are all combinations of states of each thread
- models become complex
  - language constructs for consistency
- model-checking becomes unfeasible
- simulation is not repeatable

time-triggered architecture

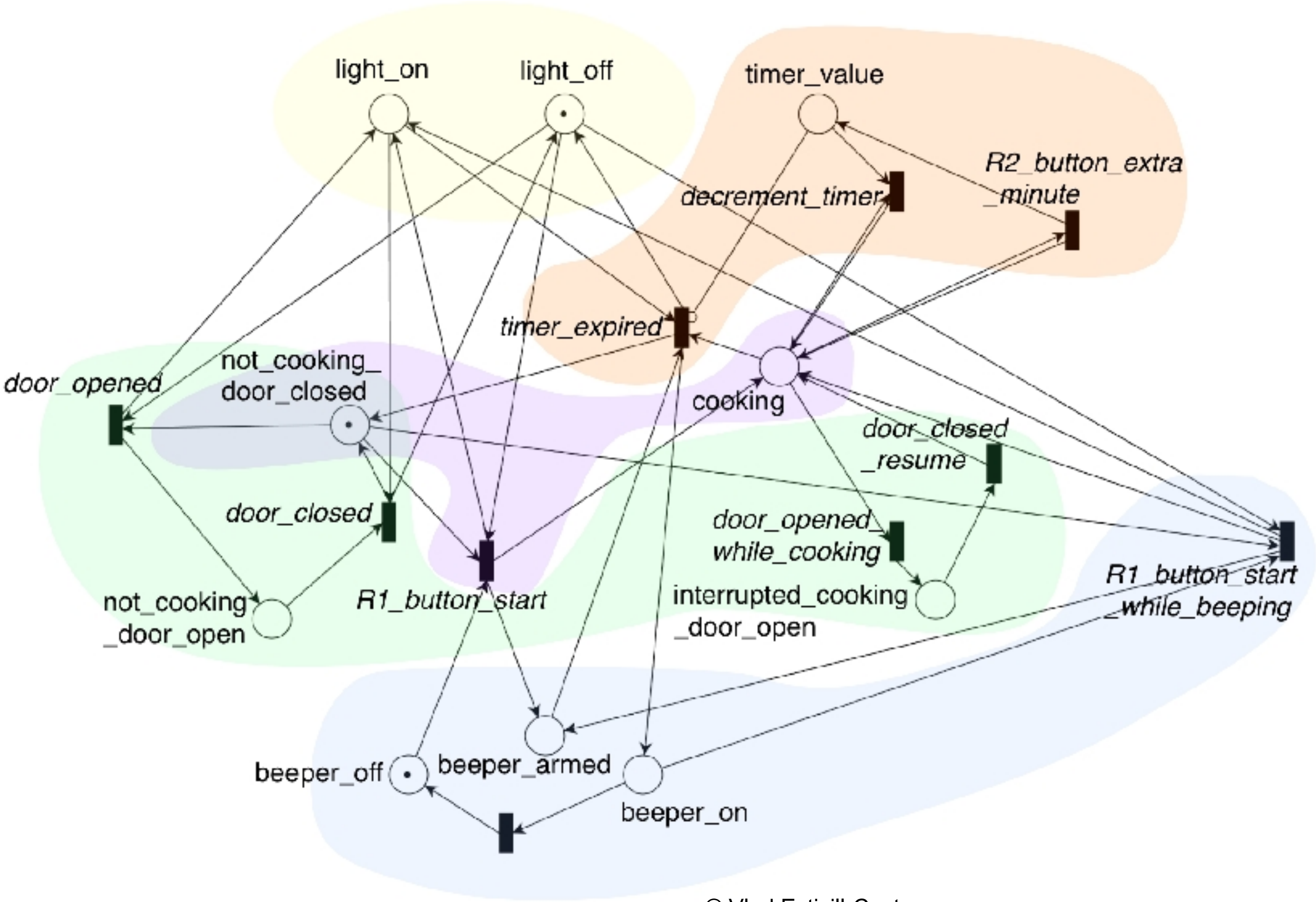
- prescribes the scheduling
- reduced space of states of the system
- models are simpler
- model checking becomes feasible
- SIMULATIONS are repeatable

StateWorks



Windows XP – Parallels Desktop
StateWORKS Studio (LE Edition) - MWOven.fsm
File Edit Project State Dictionary Name Options View Tools Window Help
MWOven.prj
Object type
Object Name
Par
Dat
Output
Interface
Counter
Supervision
VFSM
MWOven
Unit
MWOven (MWOven)
New.. VFSM Delete Duplicate
Properties...
MWOven.fsm
Init
always
Idle
Cooking
Door_Open
CookingInterrupted
Door_Closed
Di_Stop
Timer_OVER
CookingCompleted
Input Output State
Di_Run Di_Run
Di_Stop Di_Run
Door_Closed Di_Door
SWLab - MWOven.swd
File Options Help
On Off
MW:Di:Door
MW:Di:Run
MW:Do:Lamp
MW:Do:Power
1990
MW:Ni:CookingTime
2048
2047
For Help, press F1
Start
StateWORKS Studio...
SWLab - MWOven....
1:46 PM
Click inside OS Window to capture mouse

Petri Nets



light_on
light_off
timer_value
R2_button_extra
_minute
decrement_timer
timer_expired
door_opened
not_cooking_
door_closed
cooking
door_closed
_resume
door_closed
door_opened_
while_cooking
interrupted_cooking
_door_open
R1_button_start
_while_beeping
not_cooking
_door_open
R1_button_start
beeper_off
beeper_armed
beeper_on

Behavior Trees

- Model Behavior Tree

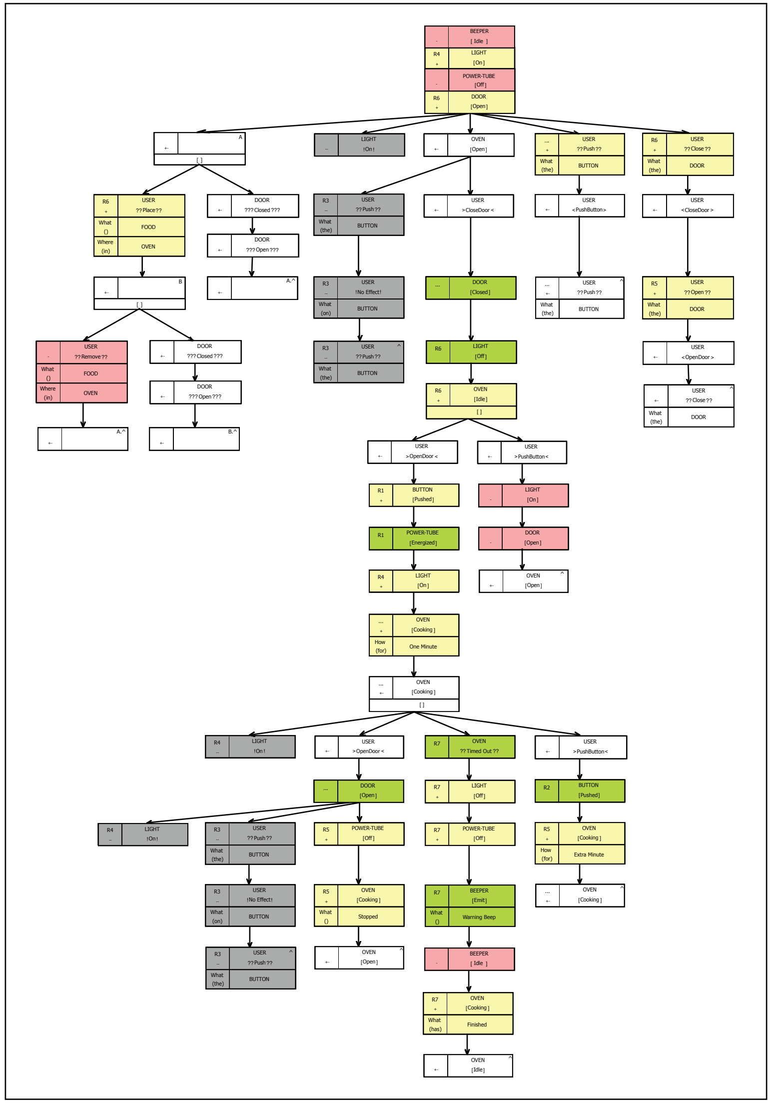


Figure 6: The Model Behavior Tree of the Microwave Oven

Behavior Trees

- Design Behavior Tree

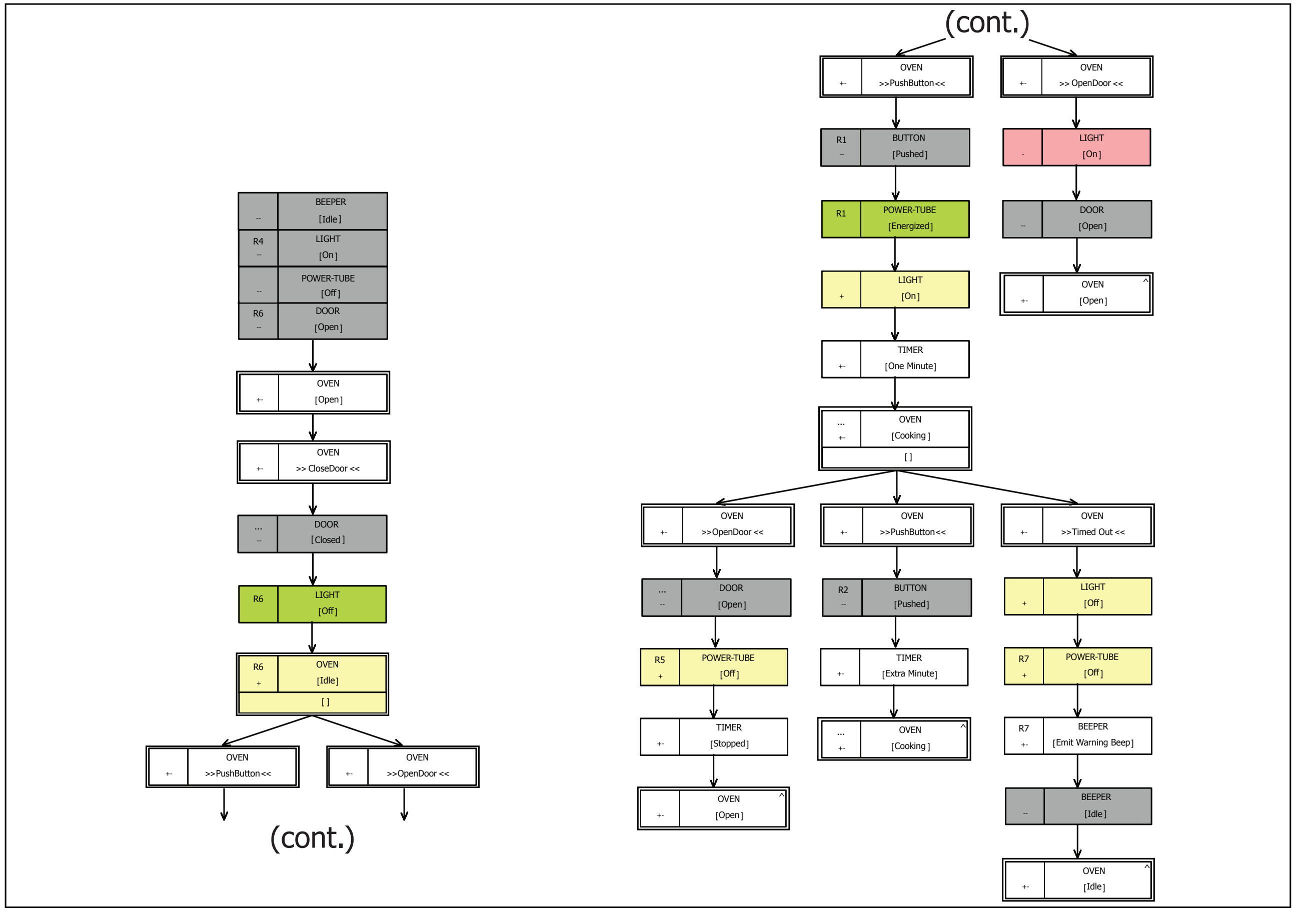


Figure 8. The Design Behavior Tree of the Microwave Oven

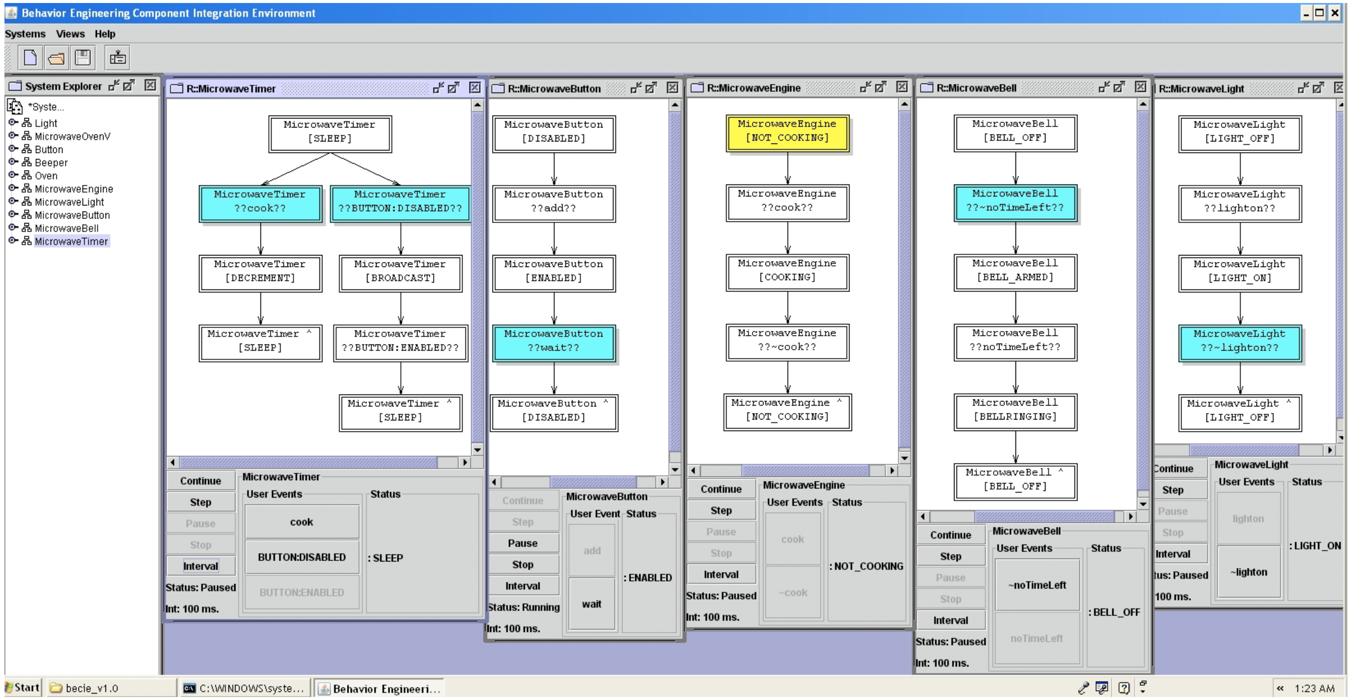
33

Comparison

- Far simpler
  - Less states than
    - StateWorks,
    - Behavior Trees
      - (less boxes and arrows)
  - Far less crossings that Petri nets
- Behavior Trees miss the alarm (beeper).

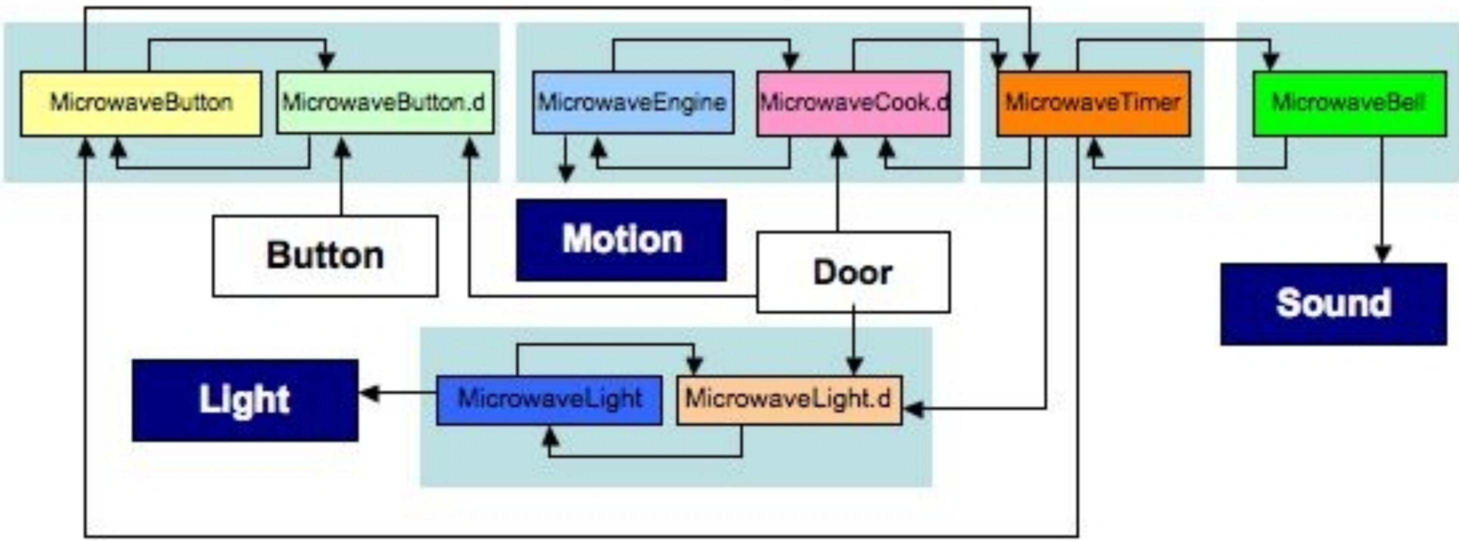
The interaction between modules

- Shows up in the behavior tree.
- But does not happen in BECCIE



Module interaction diagrams

- Perhaps of a global behavior tree



Outline

- Motivation
- Robotics and Software Engineering
- Why State Machines and Why Logic
- Examples
- Comparison
- Model Checking
- Architecture
- Summary

MDD raises the stakes from earlier on

- Importance of Model-Checking
  - Verify the model has correct behavior
- Importance of Failure Modes and Effects Analysis (FMEA)
  - Verify the model is robust and the impact of failures is understood
- NO INTERMIDIATE DEVELOPMENT PHASES
  - WHERE COMMON SENSE OF HUMANS WILL PREVAIL

Sequential finite state machines

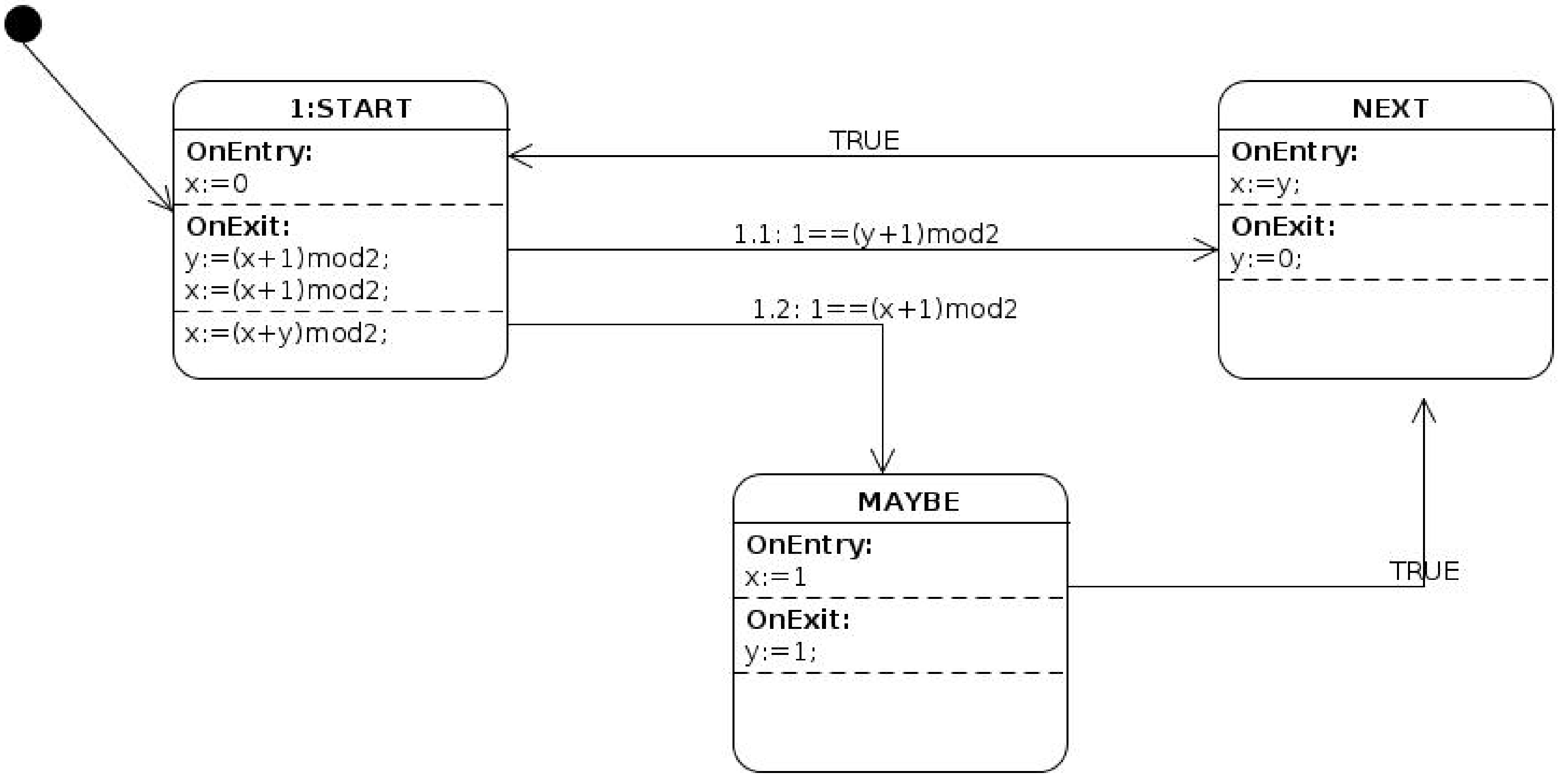


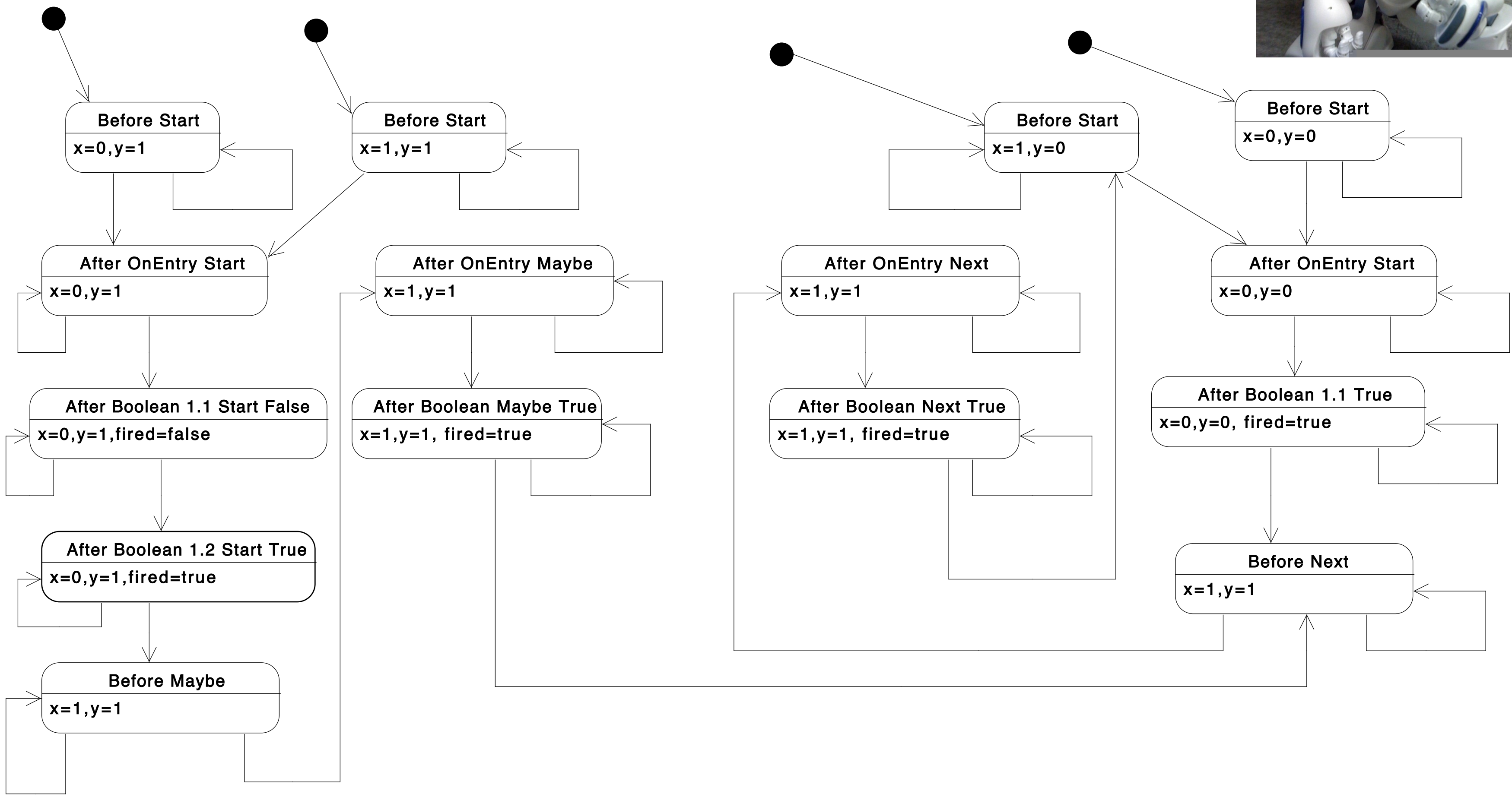
Fig. 1: A sequential finite-state machine is a model of a sequential program.

Operational formal semantics

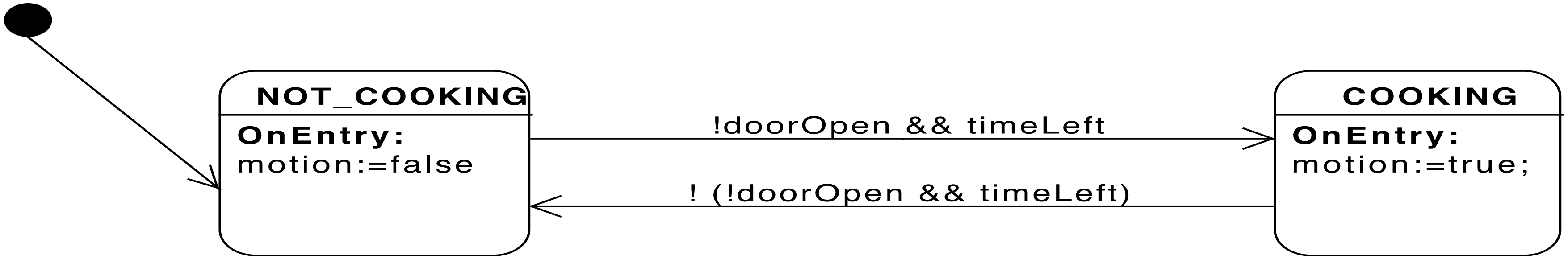
```
{Initial state is set up}
current_state ← s0;
{Default arrival to a state is because a transition fired}
fired ← true ;

{Infinite loop}
while ( true ) do
  {On arrival to a state execute On-Entry activity}
  if ( fired ) then
    execute ( current_state.on_Entry ) ;
  end if

  {If the state has no transitions out halt}
  if ( ∅ == current_state.transition_List ) then
    halt;
  end if

  {Evaluate transitions in order until one fires or end of list}
  out_Transition ← current_state.transition_List.first;
  fired ← false;
  while ( out_Transition ≤ current_state.transition_List.end AND NOT
  fired ) do
    if ( fired ← evaluate (current_state.out_Transition) ) then
      next_state ← current_state.out_Transition.target;
    end if
    out_Transition ← current_state.transition_List.next;
  end while

  {If a transition fired, move to next state, otherwise execute Internal activities}
  if ( fired) then
    execute ( current_state.on_Exit ) ;
    current_state ← next_state;
  else
    execute ( current_state.Internal ) ;
    fired ← false;
  end if
end while
```

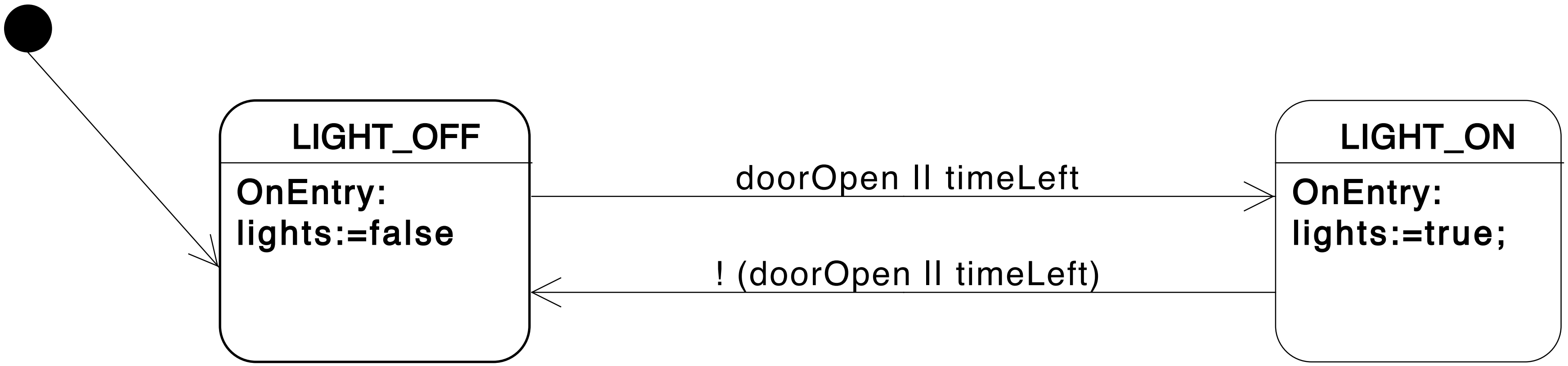
Fig. 2: The interpretation of a sequential finite-state machine.

Translate into a Kripke structure (automatic)



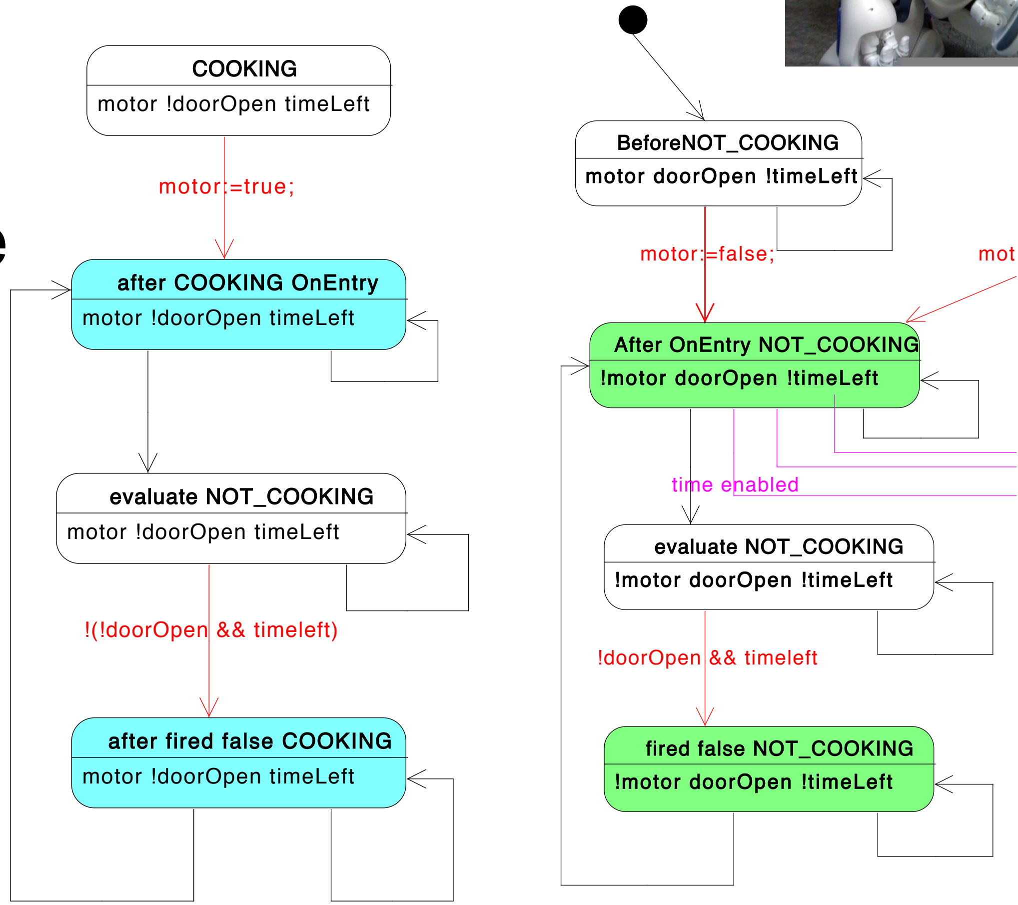
The Microwave example --- We can translate DPL to propositions



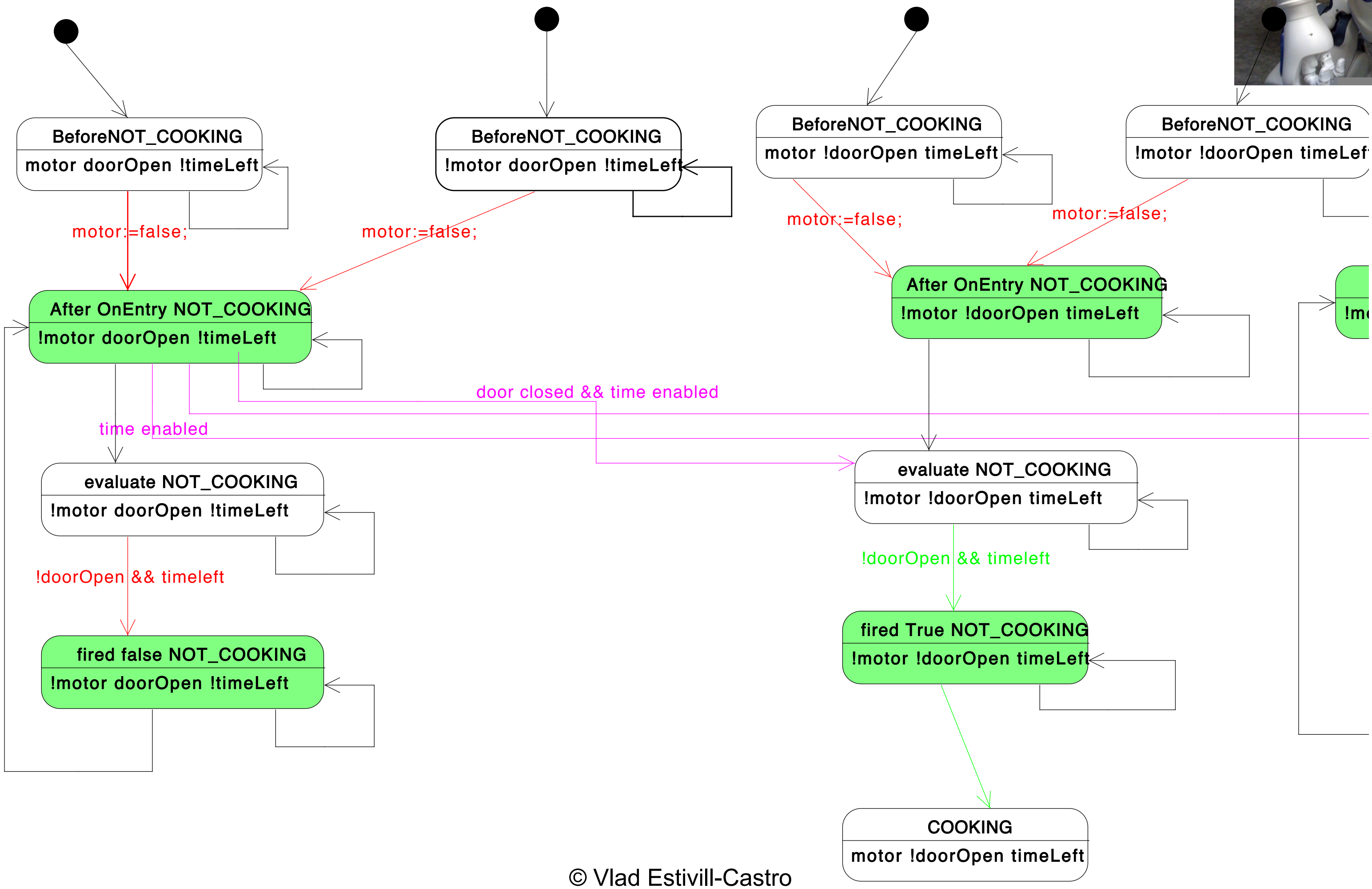


Delicate details
--- external variables

- We convert each sequential FSM state to a ringlet in the Kripke structure (automatic)



Partial view of the overall Kripke structure



Properties we can verify (flex/bison/NuSMV/C++) (antlr/NuSMV/C++)

- Necessarily, the oven stops three transitions (in the Kripke structure) after the door opens
  - `AG( doorOpen=1 & motor =1) -> AX AX AX (motor=0)`
- It is necessary to pass trough a state in which the door is closed to reach a state in which the motor is working and the machine has started.
  - `!E[ !(doorOpen=0) U (motor=1 & !(pc=BeforeNOT_COOKING) )]`
- Necessarily the oven stops three transitions in the Kripke structure after the time elapses
  - `AG ( (timeLeft=0 & motor=1) -> AX AX AX (motor=0)`

Observations

- Kripke structures are efficient
  - Linear states and transitions on the states of FSM
  - Exponential on # of variables
- Can prove properties of necessary atomicity
  - Coordination with the sensor
- Properties given any state at commencement.

Outline

- Robotics and Software Engineering
- Why State Machines and Why Logic
- Examples
- Comparison
- Model Checking
- Architecture
- Summary

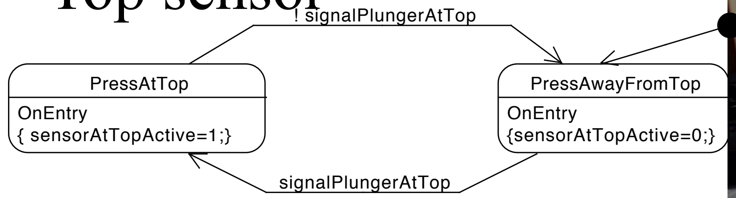
Industrial Press Requirements

Requirements	Description
R1	The plunger is initially resting at the bottom with the motor off.
R2	When power is supplied, the controller shall turn the motor on, causing the plunger to rise.
R3	When at the top, the plunger shall be held there until the operator pushes and holds down the button. This shall cause the controller to turn the motor off and the plunger will begin to fall.
R4	If the operator releases the button while the plunger is falling slowly (above PONR), the controller shall turn the motor on again, causing the plunger to start rising again, without reaching the bottom.
R5	If the plunger is falling fast (below PONR) then the controller shall leave the motor off until the plunger reaches the bottom.
R6	When the plunger is at the bottom the controller shall turn the motor on: the plunger will rise again.

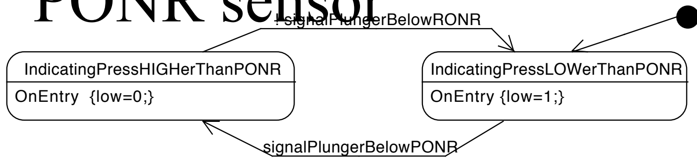
The complete model

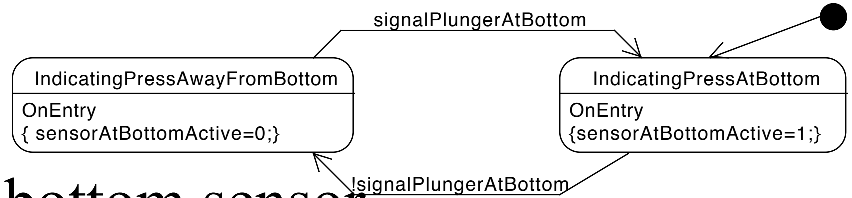
--- with peripherals for model checking and FMEA

Top sensor

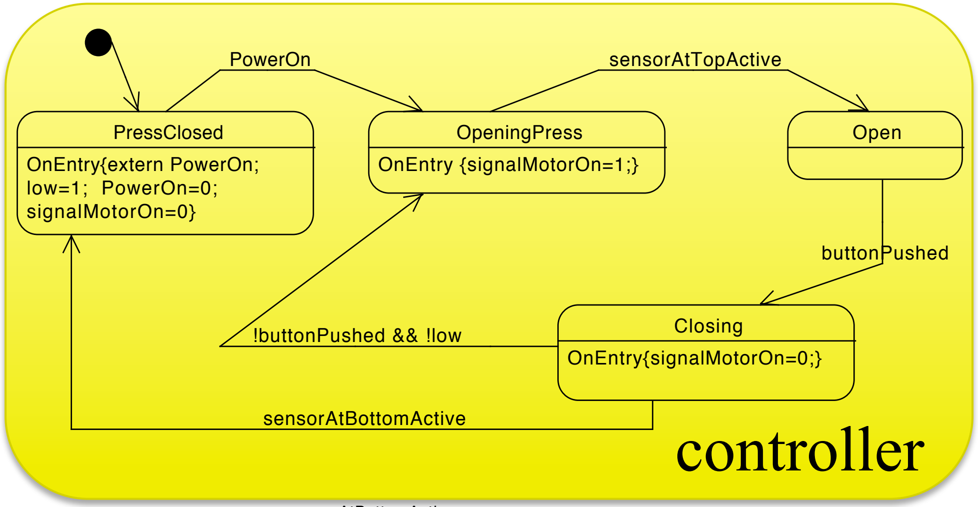


PONR sensor

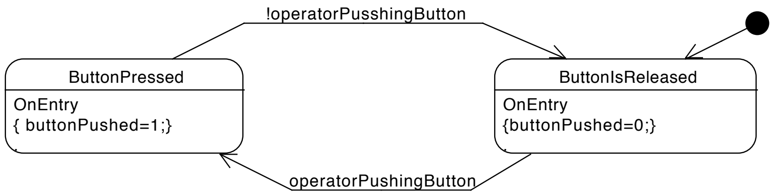




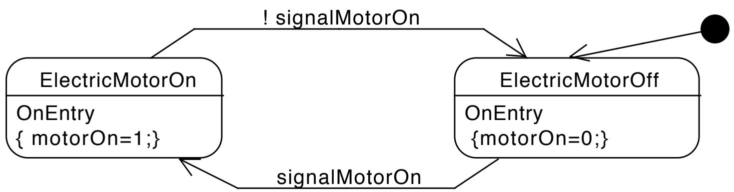
bottom sensor



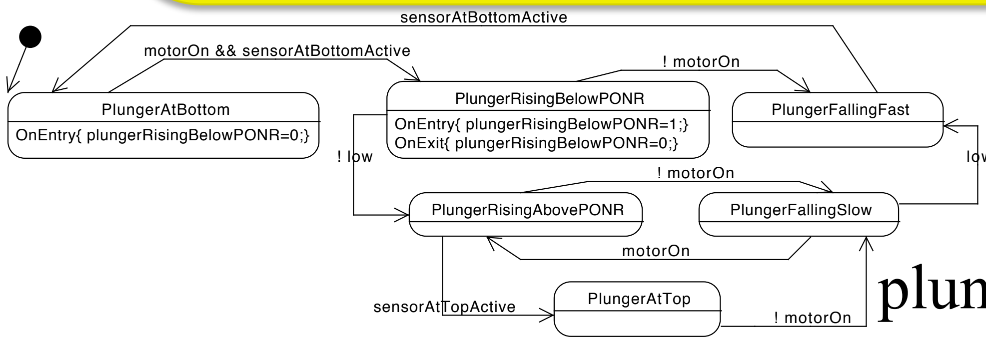
controller



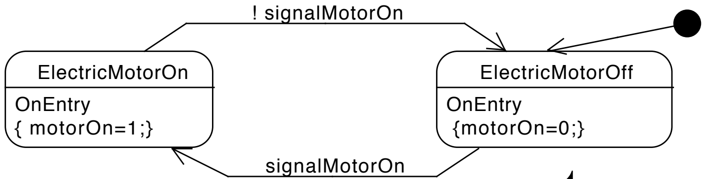
button



operator

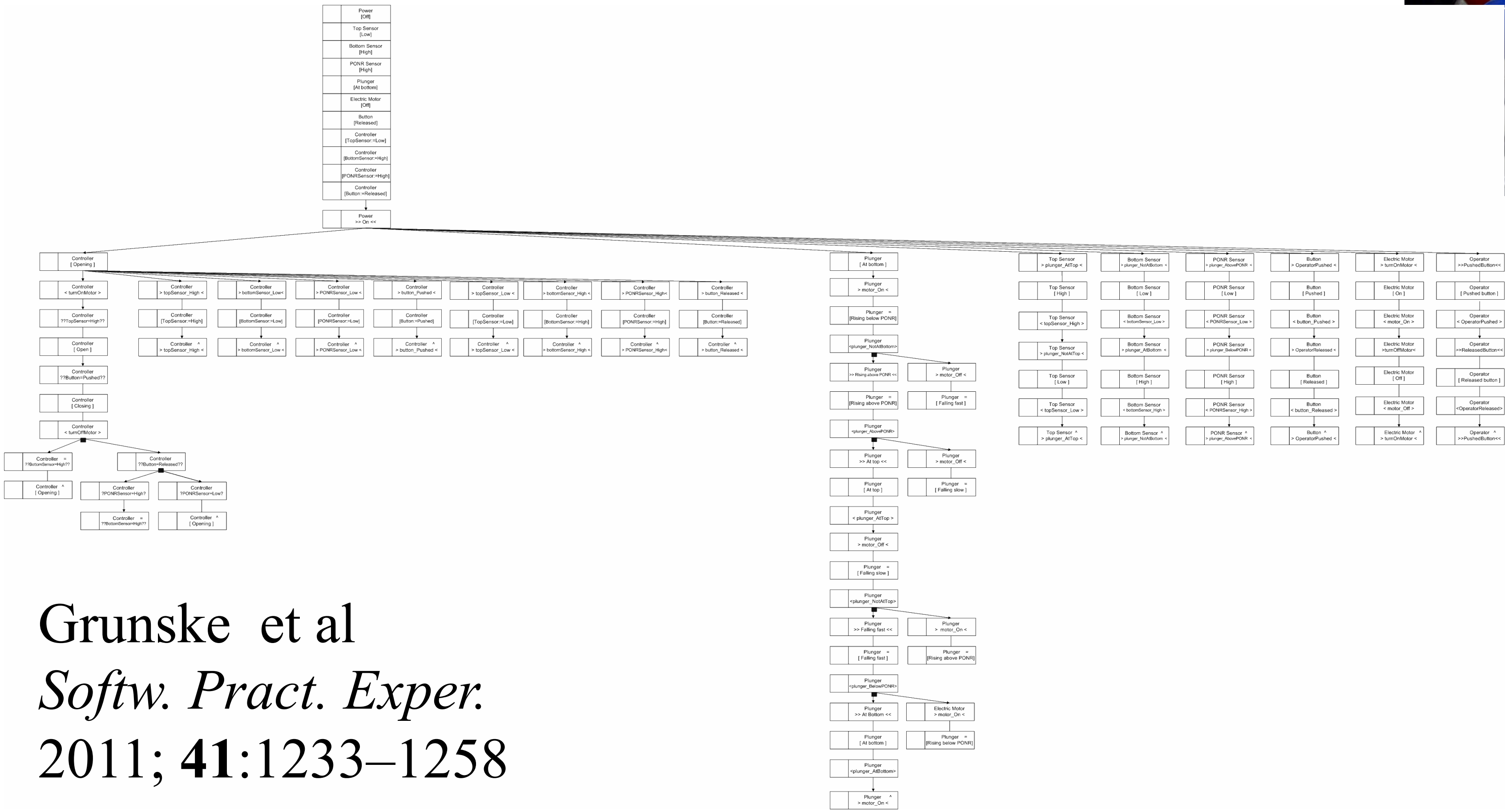


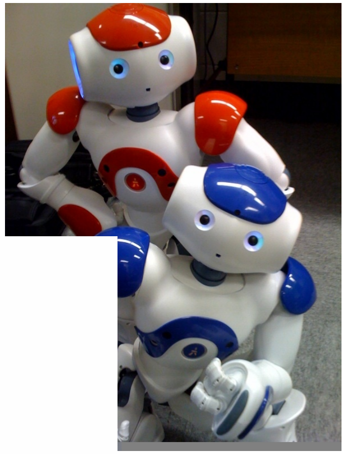
plunger



motor

Contrast with Behavior Trees



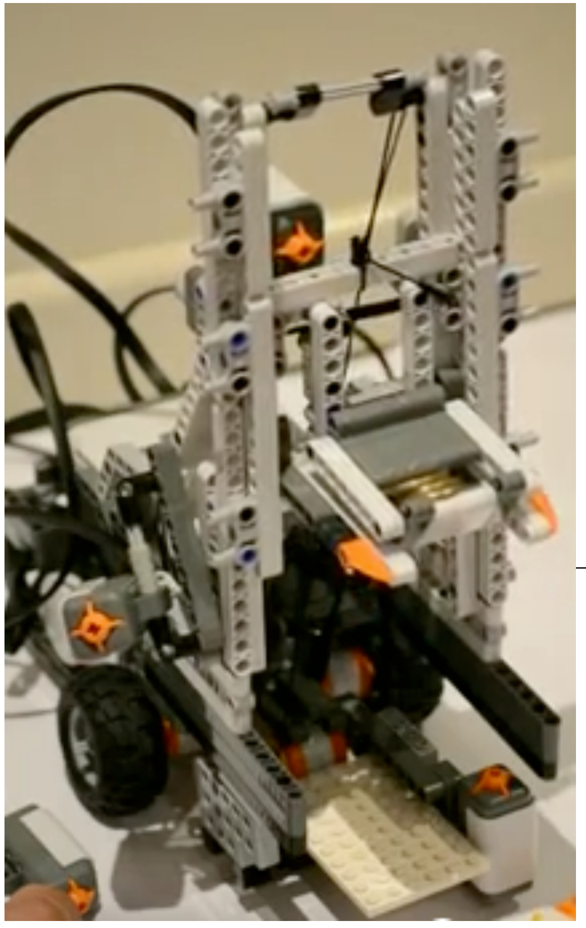


Grunske et al
Softw. Pract. Exper.
2011; **41**:1233–1258

Incorrect modeling of sequence of events after the press falls down

Industrial Press

- Property-1 *“If the operator is not pushing the button and the plunger is at the top, the motor should remain on”.*
  - G((operatorPushingButtom=0 & plunger_state=at_top) -> motorOn=1)
- Property-2 *“If the plunger is falling below the PONR, a state modelled by the plunger falling fast, then the motor should remain off.”*
  - G(plunger_state=falling_fast -> motorOn=0)
- Property-3 *“If the plunger is falling above the PONR, a state modelled by falling slow, and the operator releases the button, the motor should eventually turn on, before the plunger changes state.”*
  - G((plunger_state=falling_slow
& operatorPushingButton=0) ->(plunger_state=falling_slow U motorOn=1))
- Property-4 *“The motor should never turn off while the plunger is rising”.*
  - G(!((plunger_state=rising_below_PONR |plunger_state=rising_above_PONR)
&motorOn=0))



Demo

http://www.youtube.com/watch?v=bIUpMdH14pM

Properties demonstrated by model-checking

Property-1 *“If the operator is not pushing the button and the plunger is at the top, the motor should remain on”.*

Property-2 *“If the plunger is falling below the PONR, a state modeled by the plunger falling fast, then the motor should remain off.”*

Property-3 *“If the plunger is falling above the PONR, a state modeled by falling slow, and the operator releases the button, the motor should turn on, before the plunger changes state.”*

Property-4 *“Once the plunger is down, a new signal is needed to turn the motor on and raise the plunger again.”*

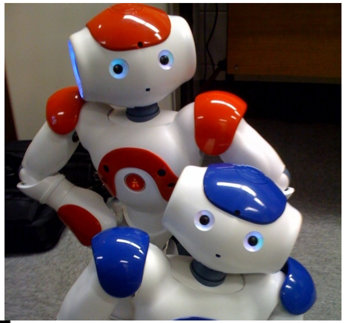


Table level 1

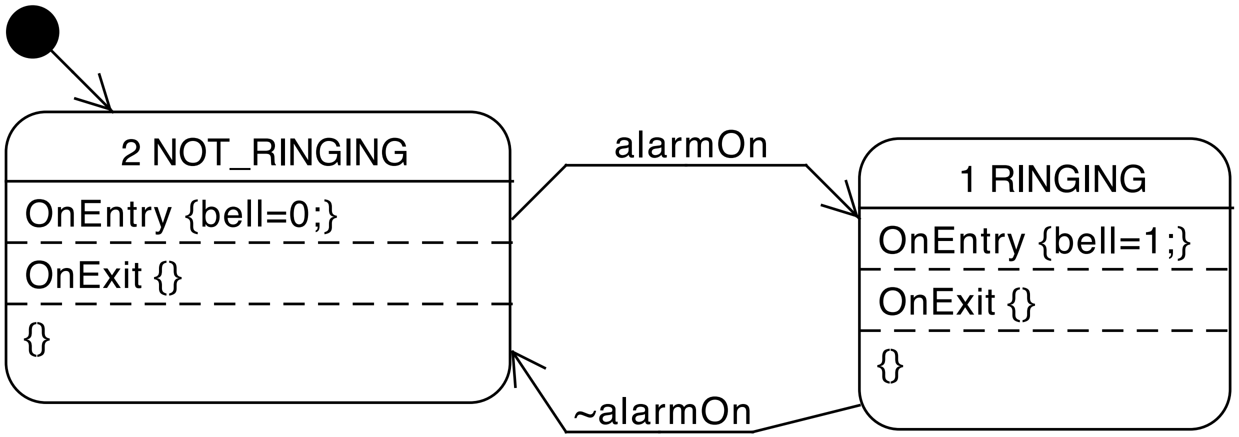
Failures	Consequences			
	Property that fails			
	1	2	3	4
Bottom sensor stuck indicating press away from bottom				X
Bottom sensor stuck indicating press at bottom				
PONR sensor stuck on above PONR		X		
PONR sensor stuck on below PONR			X	
Top sensor stuck indicating press away from top	X			
Top sensor stuck indicating press at top				
Operator button stuck on pressed	X			
Operator button stuck on released				
Motor fails, leaves motor stuck on running				X
Motor fails, leaves motor stuck on off	X			X
Power switch button stuck to supply power				X
Power switch button stuck to no power	X	X	X	X

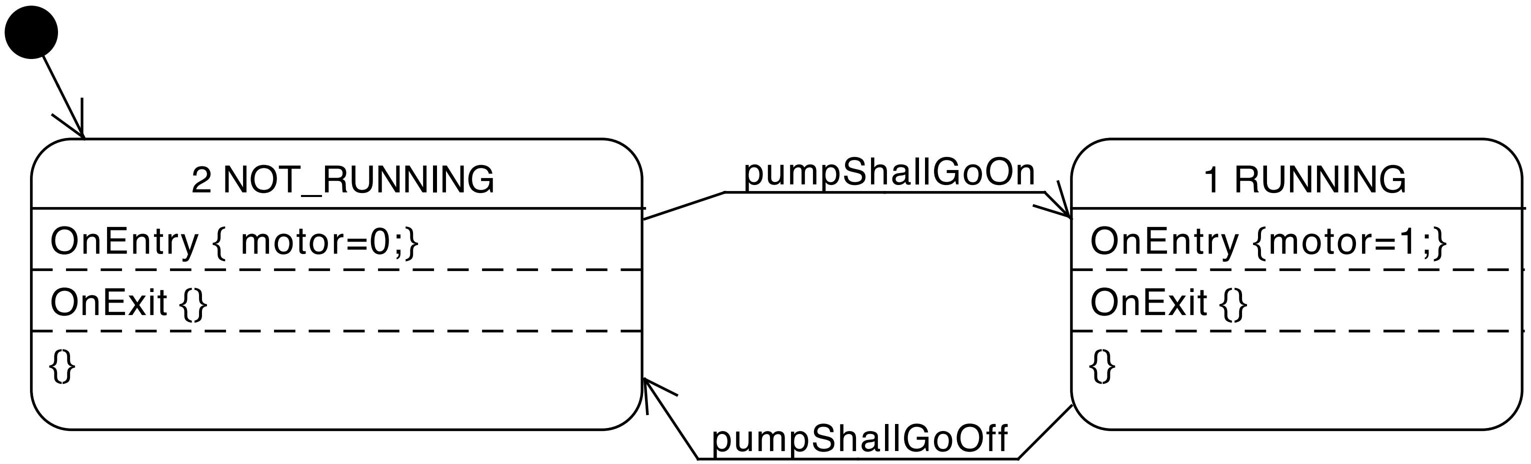
Mine Pump

Requirements	Description
R1	The pump extracts water from a mine shaft. When the water volume has been reduced below the low-water sensor, the pump is switched off. When the water raises above the high-water sensor it shall switch on.
R2	An human operator can switch the pump on and off provided the water level is between the high-water sensor and the low-water sensor.
R3	Another button accessed by a supervisor can switch the pump on and off independently of the water level.
R4	The pump will not turn on if the methane sensor detects a high reading.
R5	There are two other sensors, a carbon monoxide sensor and an air-flow sensor, and if carbon monoxide is high or air-flow is low, and alarm rings to indicate evacuation of the shaft.

Models are two FSMs

- the logic part not illustrated





Mine Pump

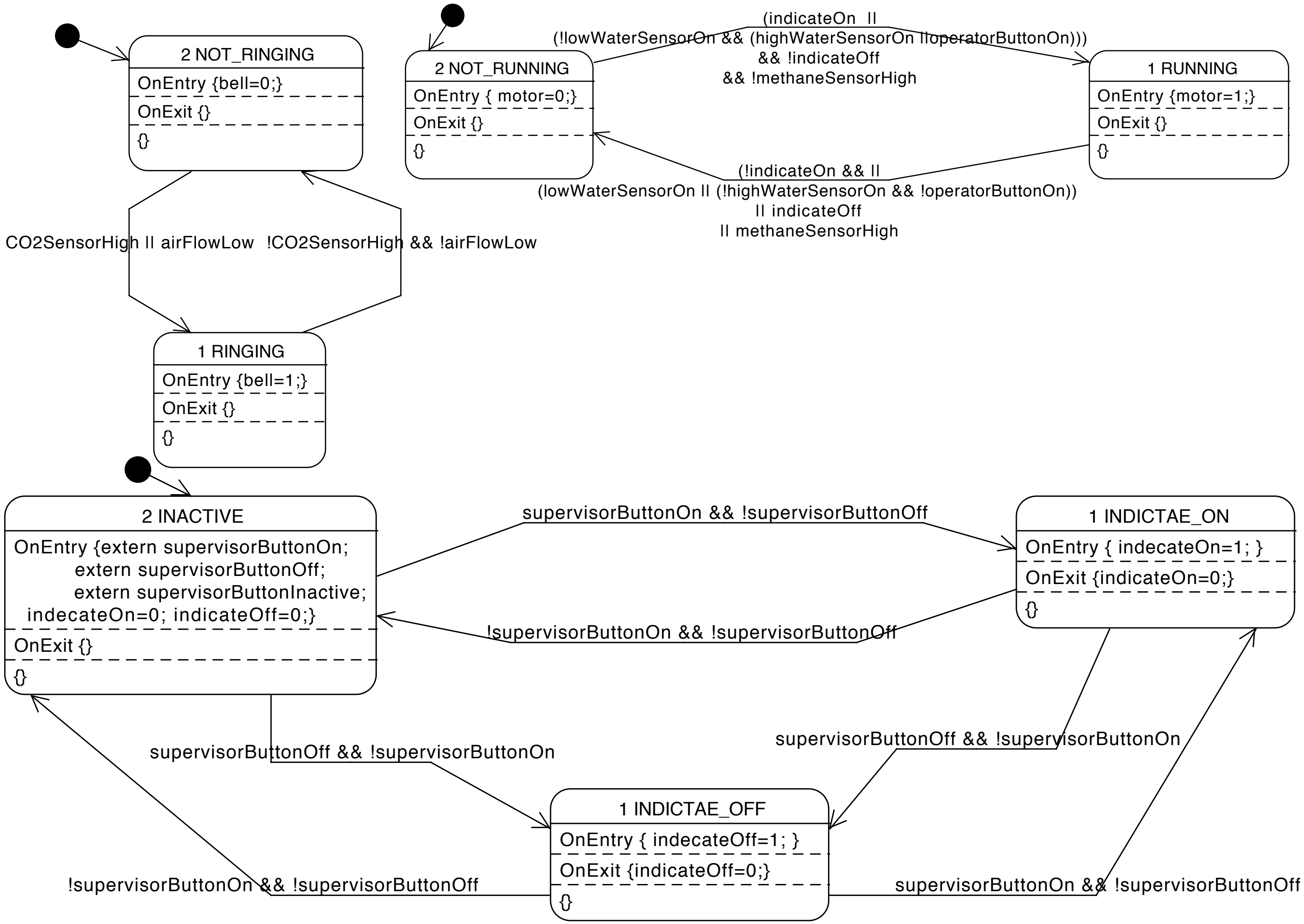
- Property-1 *"If the CO2 is high, the alarm to evacuate personnel must ring."*
- Property-2 *"If the airflow is low, the alarm to evacuate personnel must ring."*
- Property-3 *"If the methane level is high, the pump must be turned off."*
- Property-4 *"If the supervisor turns the pump off when running, the pump will be turned off."*
- Property-5 *"If the operator turns its switch off when the pump is running and the water level is neither low nor high, then the pump motor goes off."*
- Property-6 *"The pump comes on when the water is above the high water sensor (and the low-water sensor's signal is consistent with this), unless the supervisor turn it off or there is high methane."*
- Property-7 *"If the supervisor sets the switch as inactive and the pump is running when the water is not above the high water sensor and the low-water sensor indicates a low level, the pump comes off."*
- Property-8 *"If there is low methane, low water, and the pump is not running, but the supervisor puts the switch to on, then the pump comes on."*
- http://www.youtube.com/watch?v=y4muLP0jA8U&feature=player_embedded

The logic part of the models

```
%Alarm.d
name{ALARM}.

input{CO2SensorHigh}. input{airFlowLow}.

A0: {} =>  ~alarmOn.
A1: CO2SensorHigh =>  alarmOn. A1>A0.
A2: airFlowLow =>  alarmOn. A2>A0.

output{b alarmOn,"alarmOn"}.
```

```
name{MINEPUMP}.
input{lowWaterSensorOn}.  input{highWaterSensorOn}.  input{operatorButtonOn}.
input{methaneSensorHigh}.  input{indicateOn}.  input{indicateOff}.

P0: {} =>  ~pumpShallGoOn.
P1: highWaterSensorOn =>  pumpShallGoOn.                                        P1>P0.
P2: lowWaterSensorOn =>  ~pumpShallGoOn.                                        P2>P1.
P3: {~lowWaterSensorOn,~highWaterSensorOn,operatorButtonOn}=> pumpShallGoOn.    P3>P2. P3>P0.
P4: {~lowWaterSensorOn,~highWaterSensorOn,~operatorButtonOn}=> ~pumpShallGoOn.  P4>P3.
P5: indicateOn => pumpShallGoOn.
          P5>P2. P5>P4. P5>P0.
P6: indicateOff => ~pumpShallGoOn.
          P6>P5.
P7: methaneSensorHigh => ~pumpShallGoOn.                                        P7>P5. P7>P3. P7>P1.

N0: {} =>  ~pumpShallGoOff.
N1: {~indicateOn,lowWaterSensorOn} =>  pumpShallGoOff.                                           N1>N0.
N2: {~indicateOn,~lowWaterSensorOn,~highWaterSensorOn,~operatorButtonOn}=> pumpShallGoOff.       N2>N0.
N3: indicateOff => pumpShallGoOff.                                                               N3>N0.
N4: methaneSensorHigh => pumpShallGoOff.                                                         N4>N0.

output{b pumpShallGoOn,"pumpShallGoOn"}.  output{b pumpShallGoOff,"pumpShallGoOff"}.
```

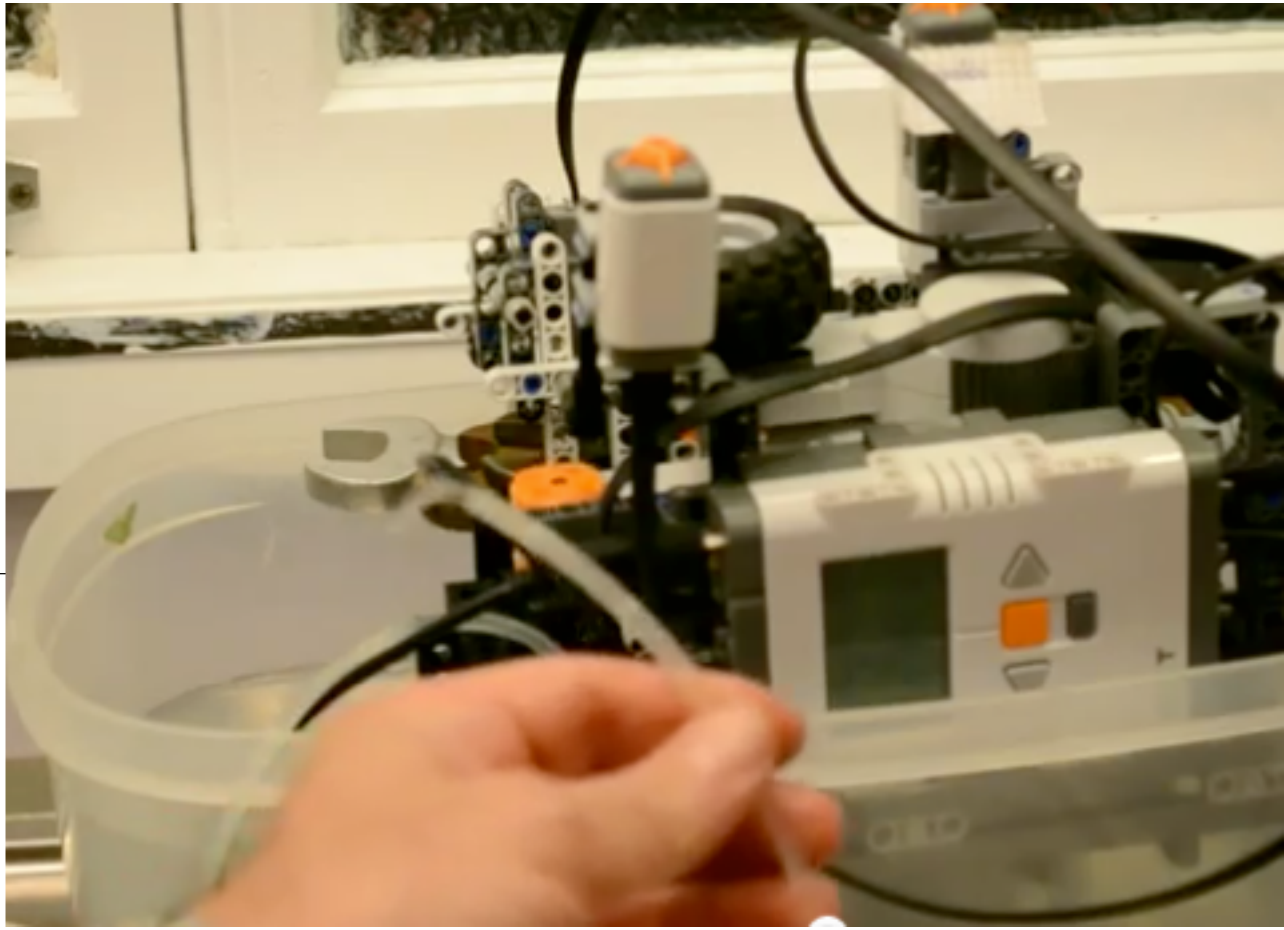
The complete model



Mine Pump

- FMEA-performing *failure modes and effect analysis* (FMEA)

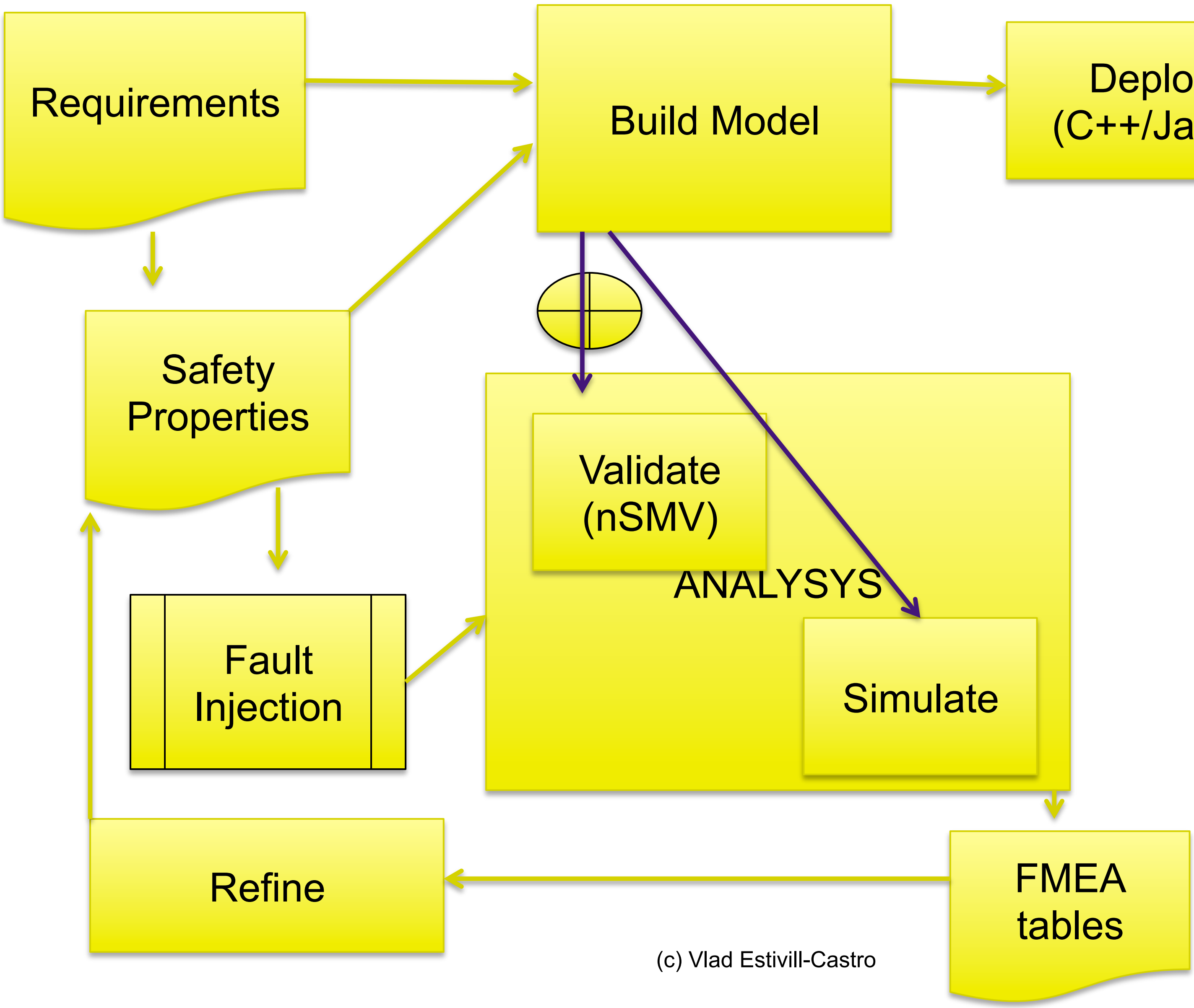
Failures	Consequences							
	Property that fails							
	1	2	3	4	5	6	7	8
CO2-sensor stuck high								
CO2-sensor stuck low	X							
Airflow sensor stuck high		X						
Airflow sensor stuck low								
Bell stuck ringing								
Bell stuck not ringing	X	X						
Supervisor button stuck in on				X			X	
Supervisor button stuck in off						X	X	X
Operator button stuck in on					X			
Operator button stuck in off							X	
Methane sensor stuck in high						X		X
Methane sensor stuck in low			X					
(High water) sensor stuck in on					X		X	
(High water) sensor stuck in off						X	X	
(Low water) sensor stuck in on						X		
(Low water) sensor stuck in off					X	X		X
Motor stuck running			X	X	X		X	
Motor stuck not running						X		X



Demo video

http://www.youtube.com/watch?v=y4muLP0jA8U

The process

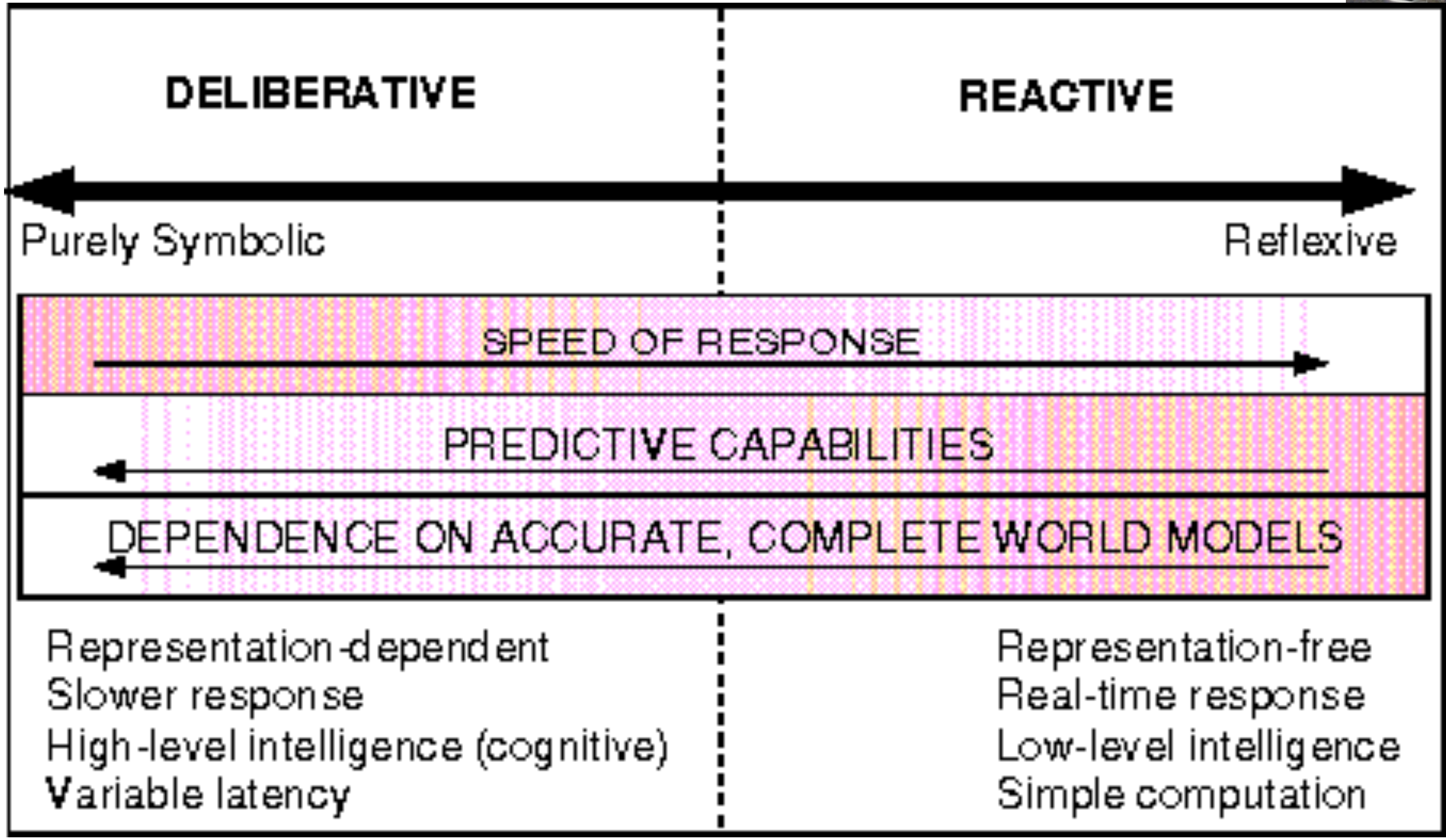


Requirements
Build Model
Deploy
(C++/Java)
Safety
Properties
Validate
(nSMV)
ANALYSYS
Fault
Injection
Simulate
Refine
FMEA
tables

Outline

- Motivation
- Robotics and Software Engineering
- Why State Machines and Why Logic
- Examples
- Comparison
- Model Checking
- Architecture
- Summary

How is a robot architecture organized



From “Behavior-Based Robotics” by R. Arkin, MIT Press, 1998

Robot control (philosophies)

- Open Loop Control
  - Just carry on, don’t look at the environment
- Feedback control
  - Minimize the error to the desired state
- Reactive Control
  - Don’ t think, (re)act.
- Deliberative (Planner-based/Logic -based) Control
  - Think hard, act later.
- Hybrid Control
  - Think and act separately & concurrently.
- Behavior-Based Control (BBC)
  - Think the way you act.

No use of logic

no use of common sense

no intelligence?

Software Architecture

- Agents / Robots

Reactive Systems

Reasoning/ Planning Systems

“Soft-Computing/ Computational Intelligence”

Symbolic AI

Hybrid System Systems

How to integrate?

A hybrid system

- The initial progress on logic and reasoning within AI has largely been discarded from mobile robotics in favour of reactive architectures
- We demonstrate the use of non-monotonic reasoning in the challenging application of RoboCup
- Plausible logic is the only non-monotonic logic with an algorithm that detects loops

Reasoning

- Deriving conclusions from facts
  - Apparently, a fundamental characteristic of intelligence
- An expected aspect of intelligent systems
- Withdrawing conclusions in the light of new evidence is a capability usually referred to as non-monotonic reasoning

Non-Monotonic Reasoning

- **A form of Common Sense**
- **Retract previous conclusions in the light of new evidence**

1. Planes usually leave on time.
2. My flight leaves at 11:00 am.
3. Therefore, I should be at the airport at 9:00am.
4. My flight is cancelled.

5. Makes no sense to take actions for going to the airport early.

Result: Robotic Poker Player

- Integrate
  - Vision
  - Sound recognition
  - Motion Control

- Environment
  - Complex
  - Interactive
  - Unpredictable
  - Competitive
  - Incomplete Information

Previous Work
--- Software architectures for robotics

- Action - Sensor Model [Wooldridge 2002]
  - Solution for control problem
- Golog [Vassos et al 2007]
  - Aim for “Cognitive Robotics”
- Knowledge Middleware [Heintz et al 2007]
  - Bridge low level sensor knowledge
- Robotic Architectures [Liu 2004]
  - Generic Robot [Kim et al 2005]
    - Solution to platform dependence

Global Architecture

- Framework = Software Engineering
  - Solves
    - Module Production / Workload problems
    - Software Development Methodology Problem
- Whiteboard (Blackboard [Hayes-Roth 1988])
  - Solves
    - Knowledge representation problem
      - (facts with timestamp and author)
    - Module Interaction Problem
    - Also called a Data Distribution Service -Publisher/ Subscriber
- Domain Knowledge
  - Logics
    - Belief revision / knowledge elicitation
  - Solves
    - Validation / verification /specification

Our Architecture

- Solution to Control Problem

exclusive

External States

Behaviours (and sub-behaviours)

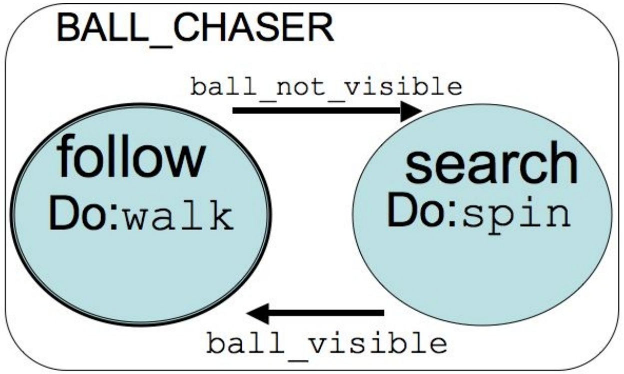
decomposable

priorities
asynchronous
associated with
actuators

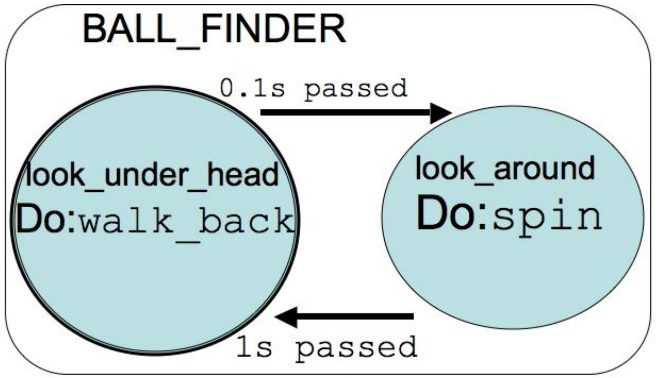
Actions

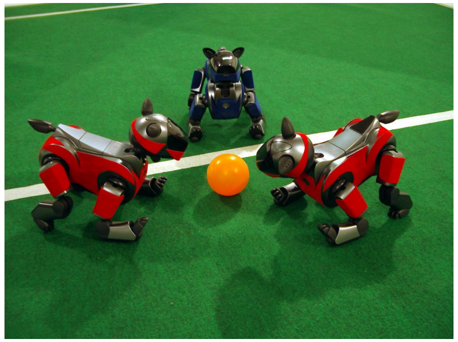
Behaviour Illustration

- Robotic Soccer
  - Simple Behaviour

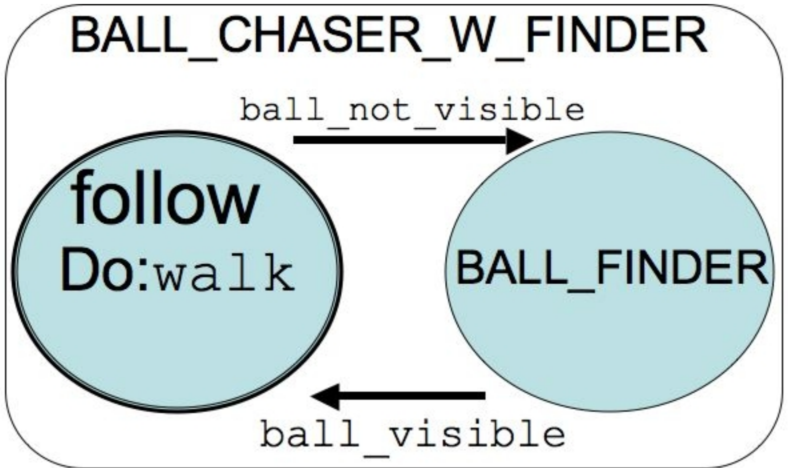


  - Sub-behavior



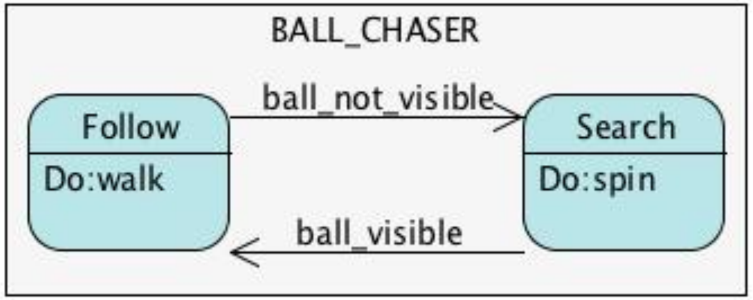


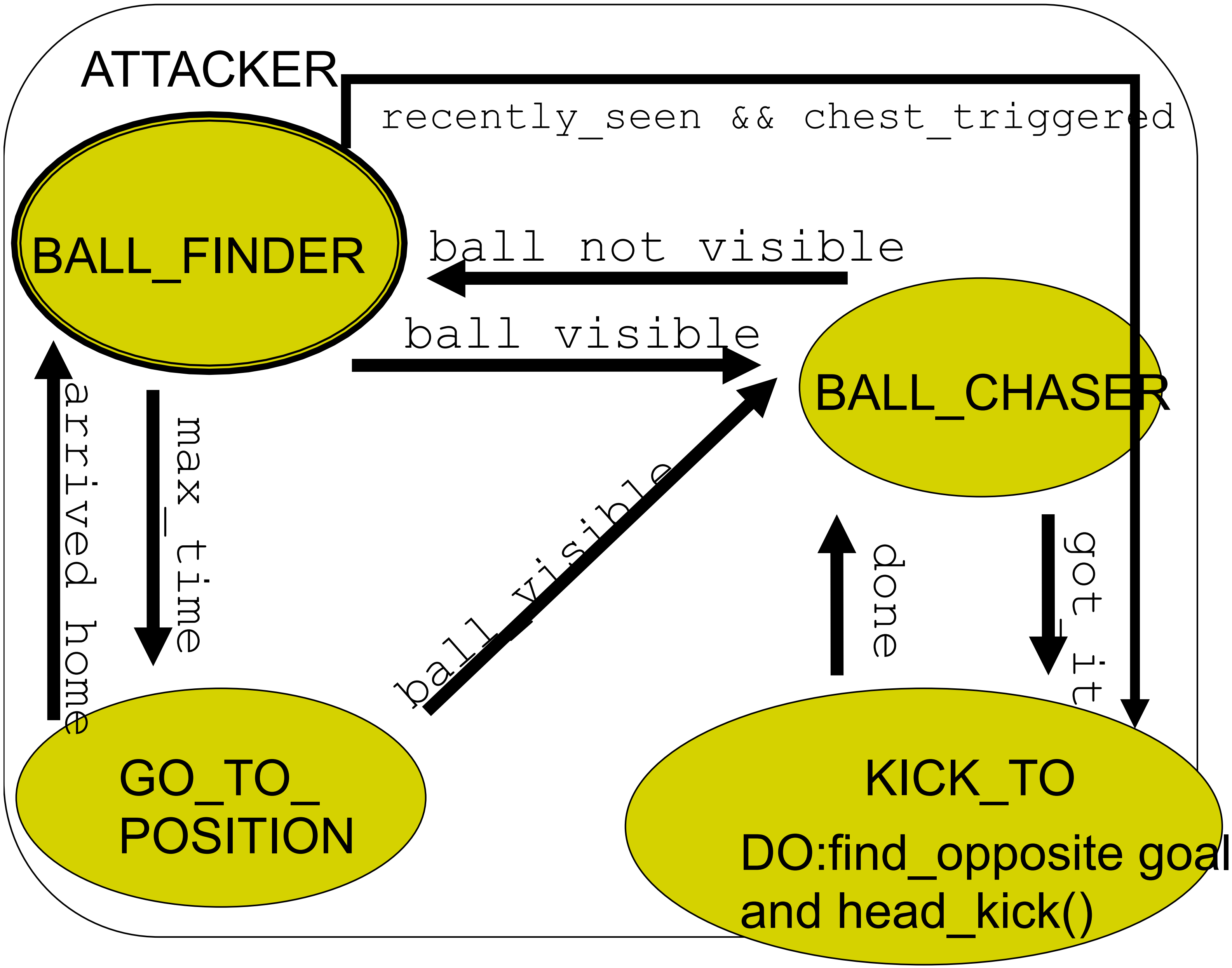
- Robotic Soccer
  - Complex behaviour

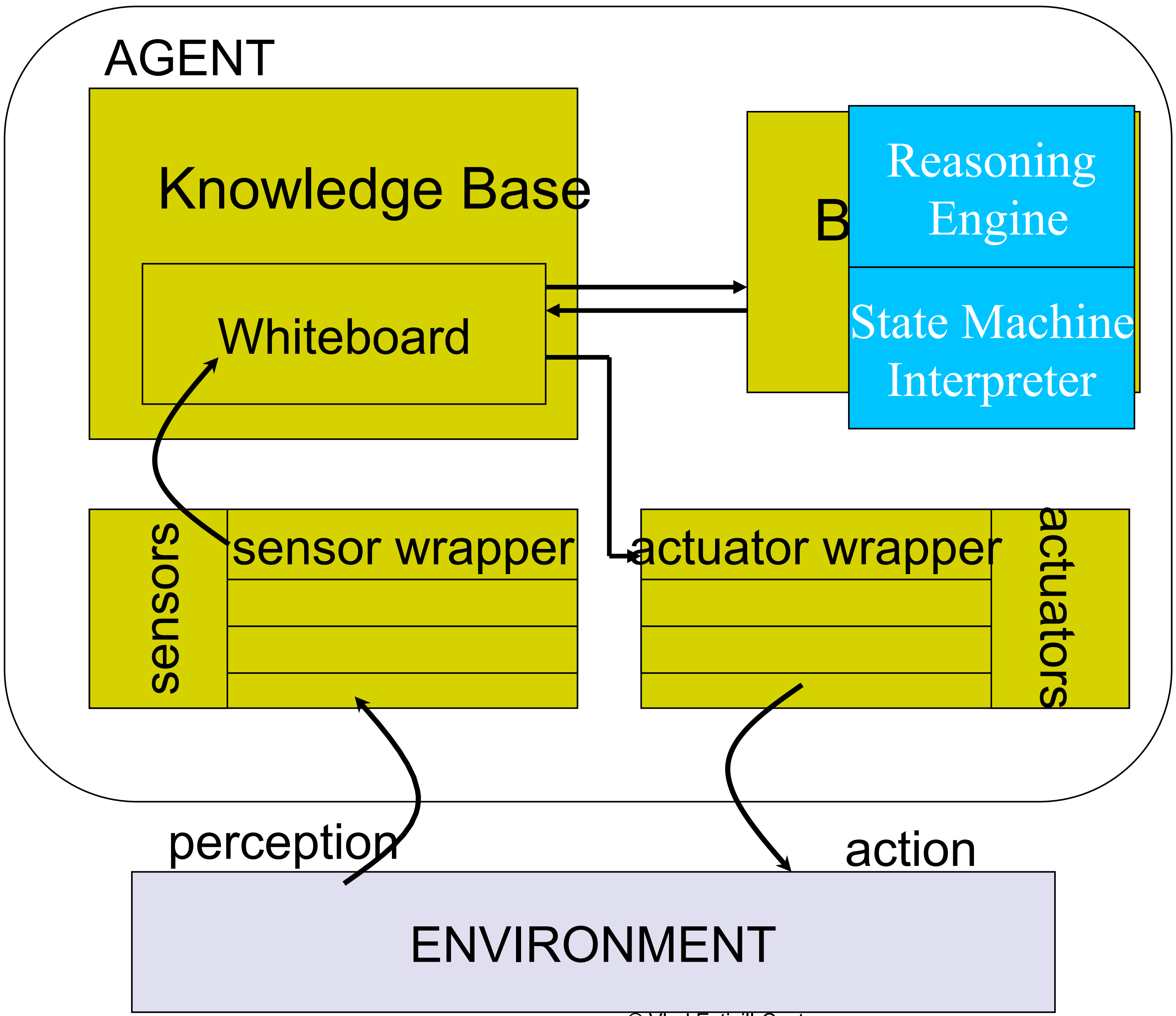


Engineering the behavior

- Using visual descriptions of the behaviour that incorporate formal logic
- Engineers use diagrams to model artefacts.
- Software Engineering has traditionally used diagrams to convey characteristics and descriptions of software

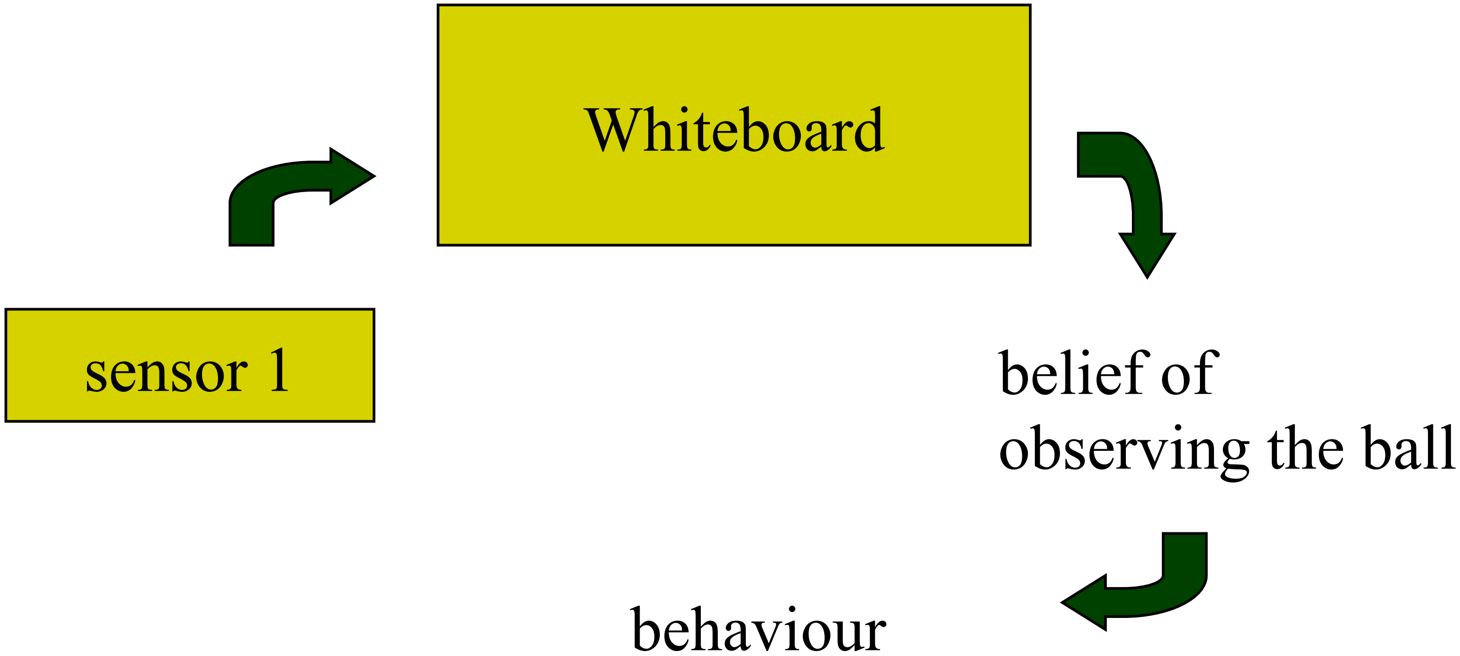



ATTACKER
recently_seen && chest_triggered
BALL_FINDER
ball not visible
ball visible
BALL_CHASER
arrived home
max_time
ball_visible
done
got_it
GO_TO_
POSITION
KICK_TO
DO:find_opposite goal
and head_kick()


AGENT
Knowledge Base
Whiteboard
B
Reasoning Engine
State Machine Interpreter
sensors
sensor wrapper
actuator wrapper
actuators
perception
action
ENVIRONMENT

Wrapping Sensors and Actuators

- Portability
- Simulation / Virtualisation
- Validation



Wrapping Sensors and Actuators

- Portability
- Simulation / Virtualisation
- Validation

Whiteboard

sensor 1

sensor 2

contradictory information about the ball

no behaviour

Alternative
Example: Seeing both goals

Our approach

Vision and
Object Recognition

Non-monotonic
reasoning

Consistency
Module

Sensor fusion

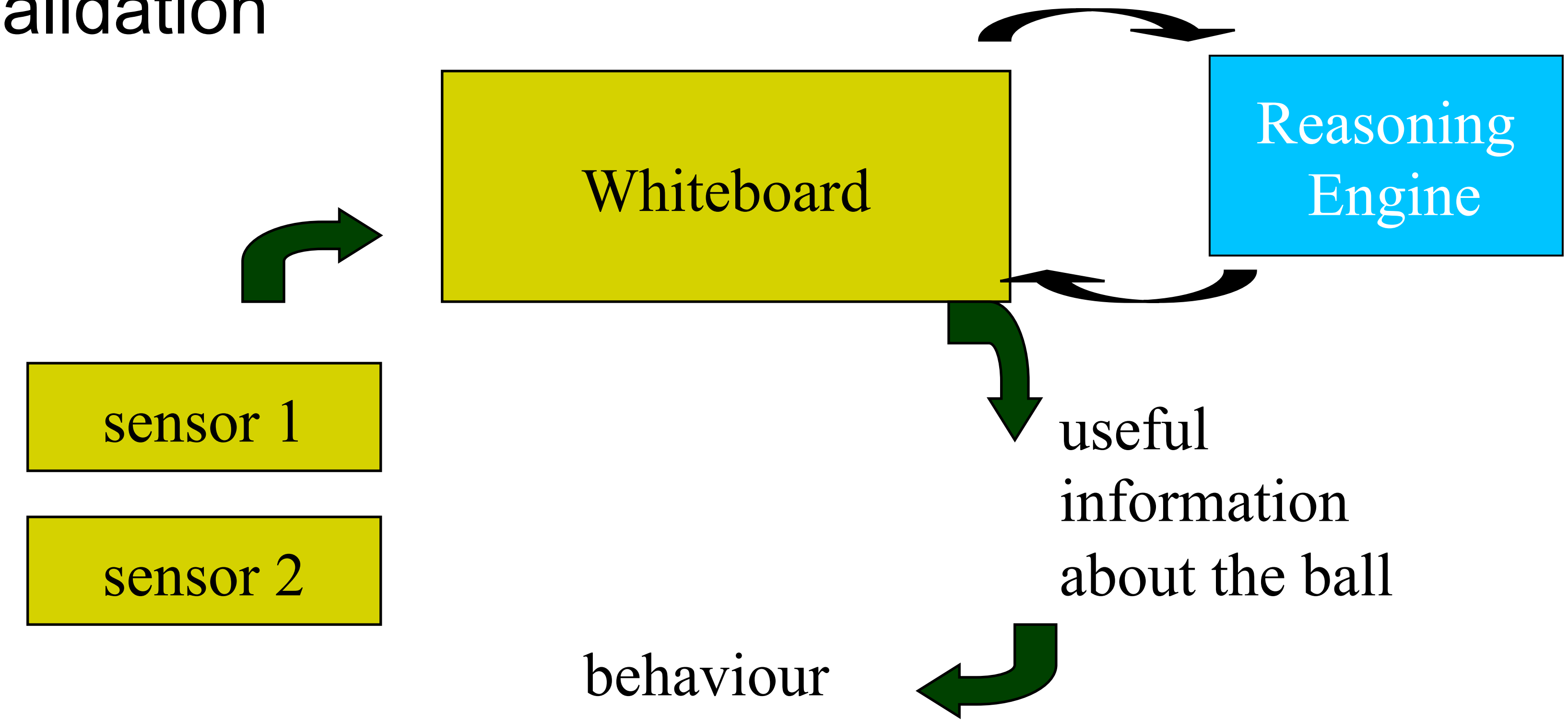
Our approach

Consistency Module

Non-monotonic logic that combines facts known about the environment with what is reported by the sensors

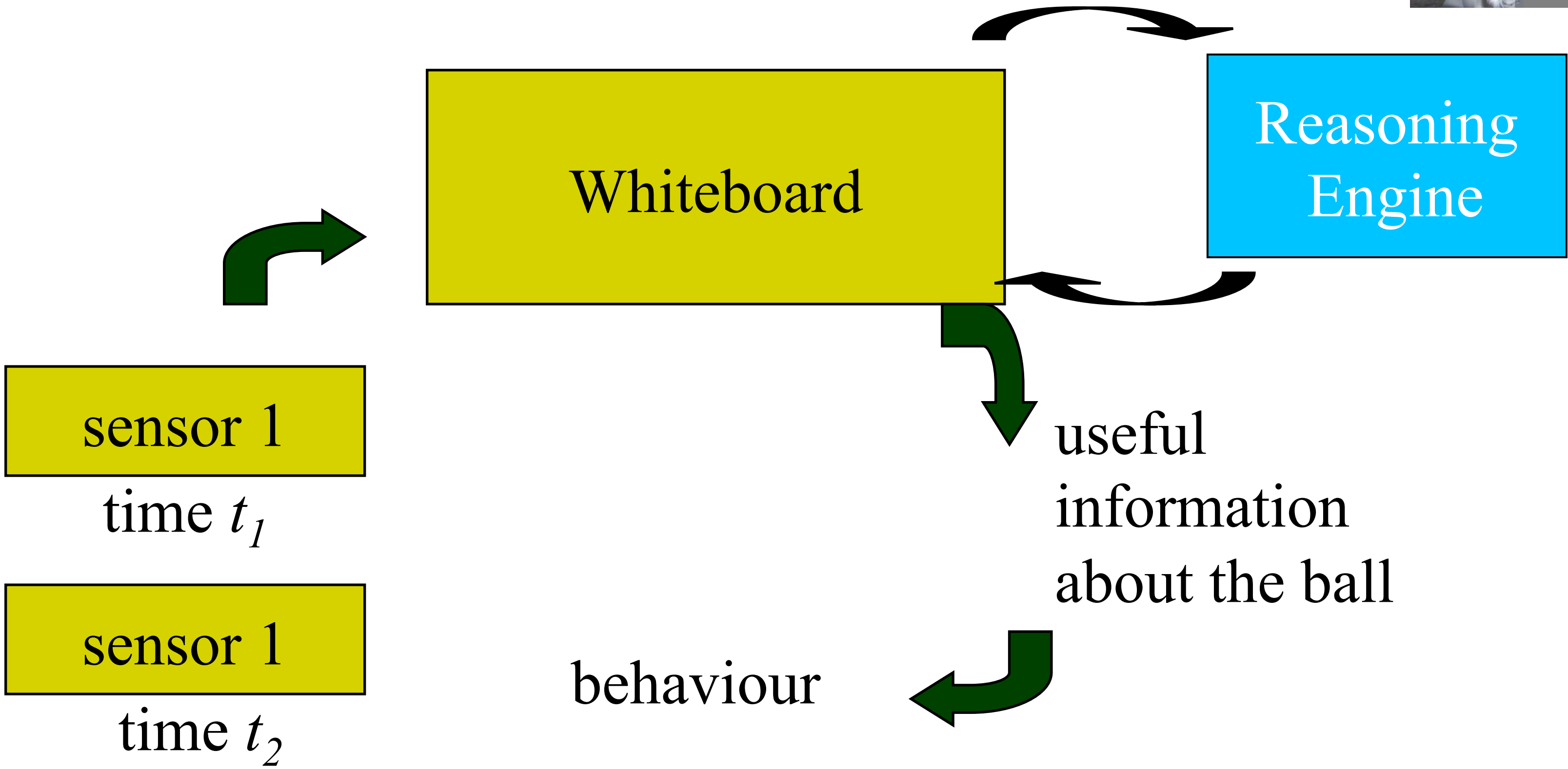
Wrapping Sensors and Actuators

- Portability
- Simulation / Virtualisation
- Validation



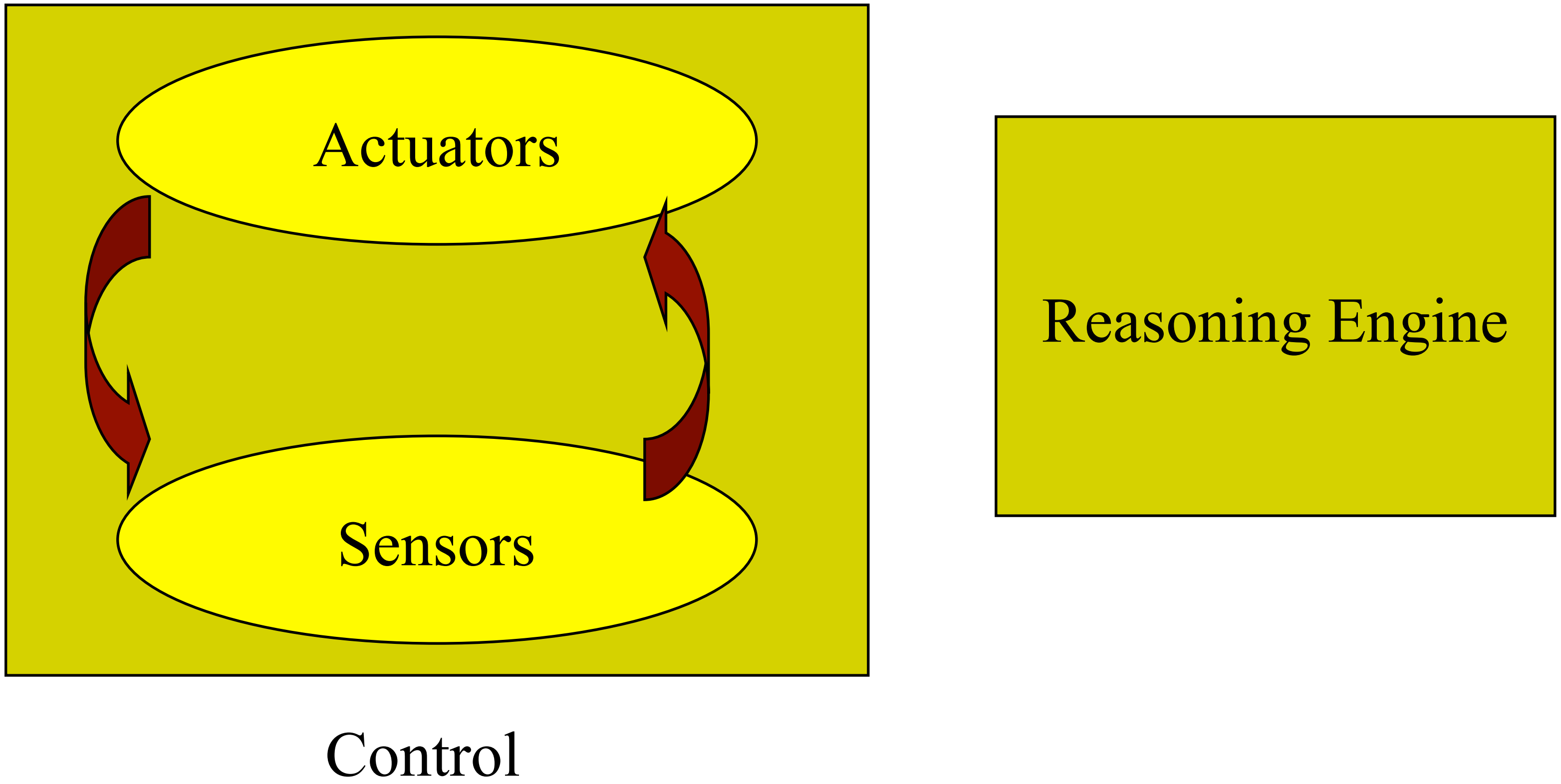
Wrapping Sensors and Actuators

- Fusion in time



Independent and Asynchronous

- Reasoning Engine

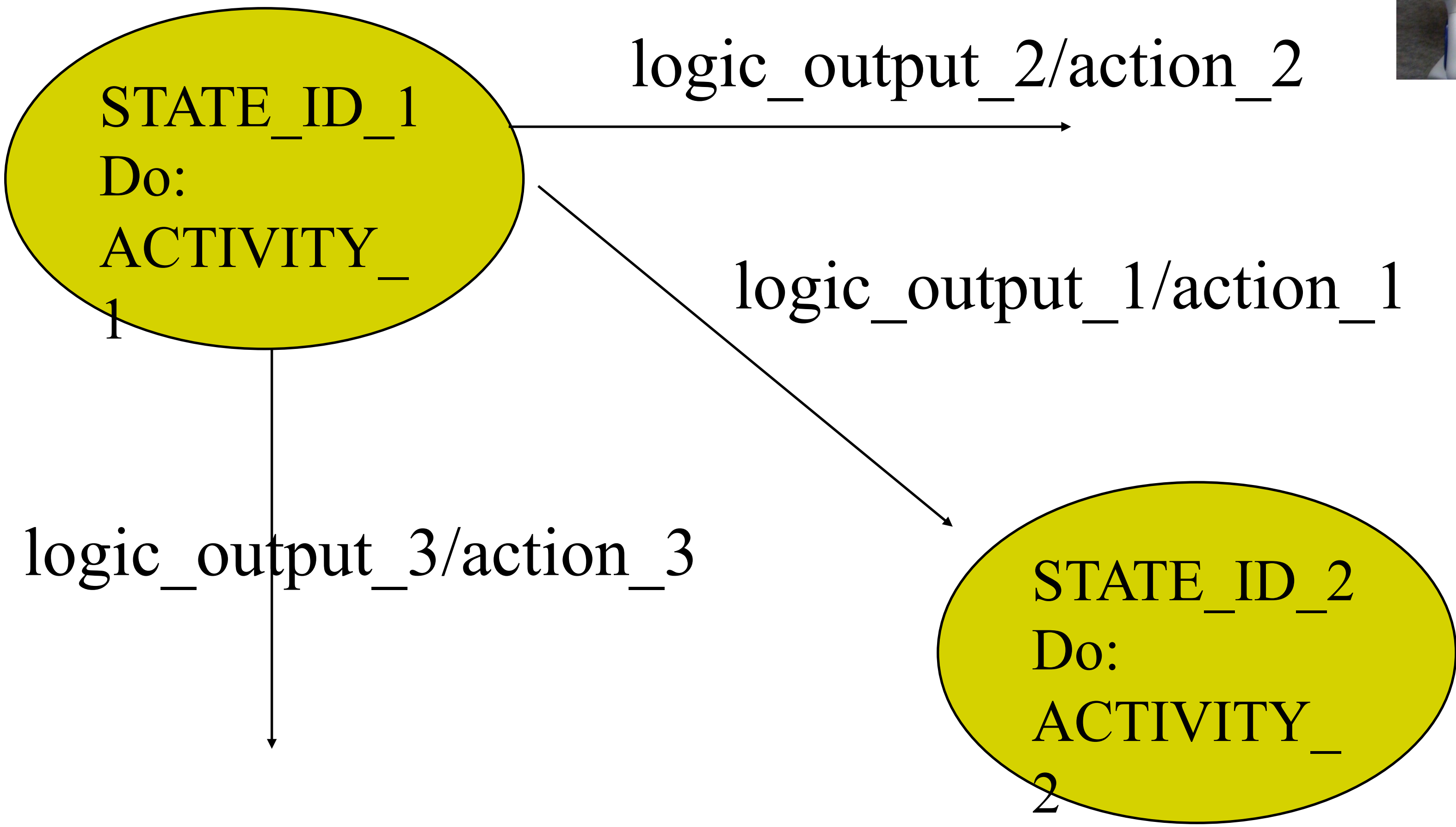


Reasoning Engine

- Template Method

1. All facts are labelled unknown
2. Extract facts from whiteboard
3. Execute predicates that are more efficient in imperative languages
4. Run the necessary queries /proofs on DPL

Interpret a behavior



Behavior Interpreter (version 1)

```
void fsmMachine :: execute ()
{
 vector <fsmState*>::iterator it;
 it=theStates.begin();
 fsmState* current = (*it);
 int currentID = current -> getID();
 cerr << Initial State is State Number " << current->getID()<< "\n";

 while (1) // run for ever
 {// Evaluate labels of transitions going out of current state
  // and may change state
 p_fsmTransition p_itTransitions;
 p_itTransitions = current->theFirstTransition();
 bool transitionFired = false;
  while ((!transitionFired) &&  (NULL!= p_itTransitions))
          {cout << "Evaluate : " <<( (p_itTransitions)->getExpression() ) -> getWhatToEvaluate() << "\n";
           cout << "Does this expression evaluate to true (Y/N)?\n";
           char response;     cin >> response;
           if ('Y'== response) // we need to execute the transition
            {           current= p_itTransitions->getTarget();
                        currentID=current->getID();
                        // break out
                        transitionFired = true;
             }
            else // advance to next transition
              {  p_itTransitions = current ->theNextTransition();
              }
          } // or != NULL
 // send message to Actuators of My Activity
 // by posting to whiteboard
 cout  << " After evalaution the state is : " <<find(current->getID())->getID() << "\n";
 cout << " We are " << ( current->getActivity() )->getWhatToDo() << "\n";
 }
}
```

Get initial state

Always

Get first transition

Evaluate

Move to new state and break if true

Do activity

Robo
Cup
2011

Robo
Cup
2012

Research output derived from RoboCup Standard platform and RoboCup@Home

www.mipal.net.au/publications.php

1. **Estivill-Castro, V. and Lovell, N.** ``Improved Object Recognition – The RoboCup 4-legged league'' *Proc. 2003 IDEAL 4th Int. Conf. on Intelligent Data Engineering and Automated Learning*. Hong Kong Springer-Verlag LNCS. Vol. 2690 p.1123-1130. (2003).
2. **Bartlett, B. Estivill-Castro, V. Seymon, S. and Tourky, A.** ``Robots for Pre-orientation and Interaction of Toddlers and Preschoolers who are Blind" *Proc. 2003 Australasian Conf. on Robotics and Automation*. 2003 CSIRO's QCAT CD-ROM.
3. **Bartlett, B. Estivill-Castro, and V. Seymon, S.** ``Dogs or Robots - Why do we see them as robotic pets rather than canine machines?" *5th Australasian User Interface Conference* (AUIC2004), Dunedin. CRPIT, Vol. 28. Ed. p. 7-14.
4. **Lovell, N.** "Real-Time Embedded Vision System Development using AIBO Vision Workshop 2" *Proc. of the Mexican Int. Conf. on Computer Science* (ENC), IEEE Computer Society Press. 160-167 (2004).
5. **Fenwick, J. and Lovell, N.** "Linear Time Construction of Vectorial Object Boundaries" 6th *IASTED Int. Conf. on Signals and Image Processing* (SIP), August 2004, Hawaii, USA
6. **Estivill-Castro, V. and Lovell, N.** "A Descriptive Language for Flexible and Robust Object Recognition" 8th *International RoboCup Symposium*, July 2004, Lisbon, Portugal
7. **Estivill-Castro, V. and McKenzie B**. "Hierarchical Monte-Carlo Localisation Balances Precision and Speed" *Proc.2004 Australasian Conference on Robotics and Automation*. December 6-8, 2004 in Canberra at Australian National University. CD-ROM.
8. **Lovell, N.** "Illumination independent object recognition." In Noda, I., Jacoff, A., Brendenfeld, A., Takahashi, Y., eds.: *Proc. Robocup 2005 Symposium*, Springer-Verlag . LNCS 4020 (2006), p. 384-395.
9. **Lovell, N.** "Fast Posture and Object Recognition using Symmetries" *Proc. 2005 Australasian Conference on Robotics and Automation*. December 5-7, 2005 in Sydney at University of New South Wales. Claude Sammut (editor) CD-ROM .
10. **Billington, D., Estivill-Castro, V., Hexel, R., and Rock A**., "Non-monotonic Reasoning for Localisation in RoboCup" *Proc. 2005 Australasian Conference on Robotics and Automation*. 2005 in Sydney at University of New South Wales. CD-ROM.
11. **J. Fenwick and V. Estivill-Castro** ``Optimal Paths for Mutually Visible Agents" *The 16th Annual Int. Symposium on Algorithms and Computation*. Deng, X. and D.-Z. (Eds.) 2005, Sanya, Hainan, China. Springer Verlag LNCS 3827, pages 869-881.
12. **V. Estivill-Castro and S. Seymon** "Mobile Robots for an E-mail interface for People who are Blind" *Robocup Int. Symposium 2006*. G. Lakemeyer, E. Sklar , D. G. Sorrenti and T. Takahashi eds. Springer Verlag LNCS. Vol 4434, pages 338-346 2007.
13. **Lovell, N.** "Machine Vision as the Primary Sensory Input for Mobile, Autonomous Robots", PhD thesis, Griffith University, 2007.
14. **D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock,** ``Using Temporal Consistency to Improve Robot Localisation" *Robocup International Symposium 2006*. Springer Verlag LNCS. Vol 4434, p. 232-244, 2007.
15. **J. Fenwick and V. Estivill-Castro.** ``Mutually visible agents in a discrete environment" *The Thirtieth Australasian Computer Science Conference* (ACSC-2007), Ballarat. CRPIT, Vol. 62. p. 141-150.
16. **D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock,** "Non-monotonic Reasoning on Board a Sony AIBO" *Soccer Robotics*. P. Lima editor. IS
17. **N. Lovell, and V. Estivill-Castro "**Color Classification and Object Recognition for Robot Soccer Under Variable Illumination" *Soccer Robotics*. P. Lima editor.

Ramifications

- Motion control
  - pick up the cards / move bricks or chips
- Image processing
  - recognize opponents partners
  - actions / gestures
  - cards / figures
- Agent technology
  - reasoning / game play/ knowledge representation
- Multi-modal
  - sound / motion / speech
- Virtual games / tele-presence

Focusing on a prototype leads to links with other areas

- Research in education
  - Meaningful play is learning

Robots provide to the blind what was lost when text interfaces where replaced by GUIs

- Mobile robots for an E-mail interface for people who are blind
  - Provide a multi-modal mobile interface for ambient intelligence
  - Enable mnemonic commands
  - Allow rapid learning

Summary

- State diagrams
  - Widely used, solid tool to communicate requitements, behaviors
    - The reactive part
- Transitions are labeled by questions to an inference engine
  - Solid tool to model the declarative part
- Simplify the burden by using non-monotonic logics
  - Defaults
  - Iterative refinement
- Loosely coupled architecture
  - Platform independent
  - Simulation / Validation / Model checking

Summary

- Ensure quality and safety
  - software in embedded controllers
- Logic-labeled vectors of FSM, sequentially scheduled
  - provide more succinct models
    - validated
  - with clear semantics
  - that
    - can be simulated
    - can be exported to various platforms
      - (model-driven development)
    - can be model-checked
      - (in a matter of seconds, as opposed to days of CPU time)
    - can be examined with fault-injection

THANK YOU