High Performance Relaying of C++11 Objects across Processes and Logic-Labeled Finite-State Machines

V. ESTIVILL-CASTRO*

**Griffith University, Nathan Campus,*
Brisbane, Australia.
v.estivill-castrol@griffith.edu.au,

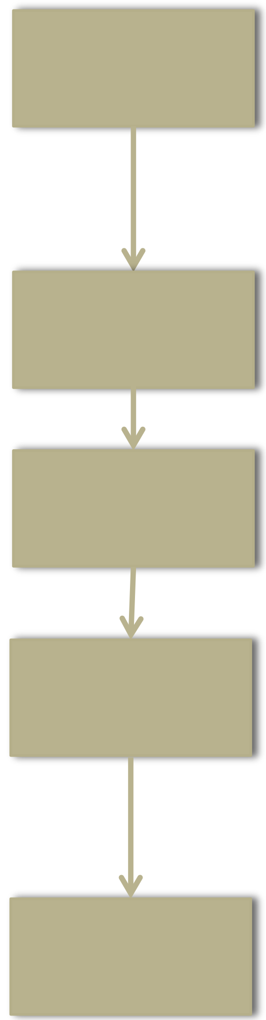
In collaboration with Rene Hexel, Carl Lusty and many other members of MiPal

Outline

- Two tools
  - `clfsm`
  - `mipal gusimplewhiteboard`
  - What do they do?
- Finite-State Machines (FSM)
  - Logic-labeled FSMs
- Examples

- What have they enabled
  - software architectures /middleware
  - Model-driven development
  - Formal verification

- Conclusions
  - What can I do so you would use them?

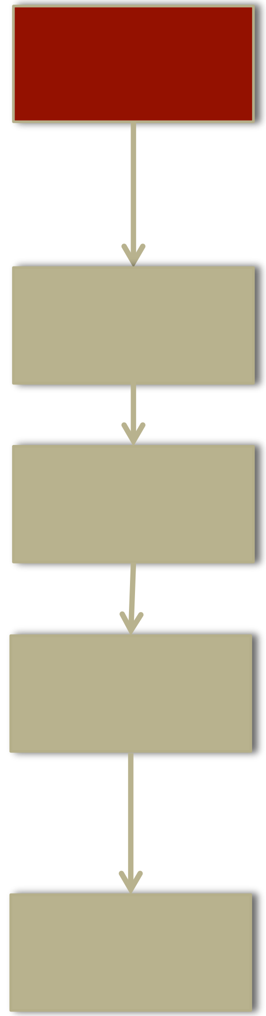


Outline

- Two tools
  - `clfsm`
  - `mipal gusimplewhiteboard`
  - What do they do?
- Finite-State Machines (FSM)
  - Logic-labeled FSMs
- Examples

- What have they enabled
  - software architectures /middleware
  - Model-driven development
  - Formal verification

- Conclusions
  - What can I do so you would use them?



`clfsm`: compiled logic-labeled finite-state machines

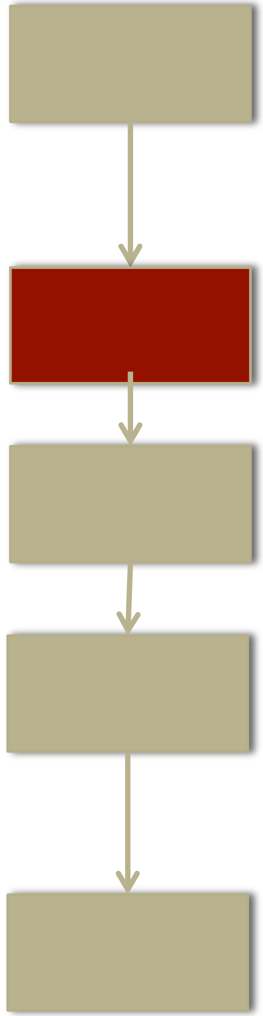
- Complete **`POSIX`** and **`C++11`** compliance.
- Open source **`catkin ROS`** package release (**`mipal.net.au/downloads.php`**).
- Transitions are labeled by Boolean expressions (not events), facilitating formal verification and eliminating all need for concerns about event queues.
- Transition labels are arbitrary **`C++11` Boolean expressions**, enabling reasoning into what may otherwise seem a purely reactive architecture.
- Handling of machines constructed with states that have UML 2.0 (or SCXML) **`OnEntry`**, **`OnExit`**, and **`Internal`** sections with clear semantics.
- Guaranteed **sequential ringlet schedule** for the concurrent execution of FSMs (removing the need for critical sections and synchronization points).
- Efficient execution as the entire arrangement runs as **compiled code** without thread switching.
- Being agnostic to communication mechanisms between machines, allowing, for example use with **`ROS:services`** and **`ROS:messages`** – however, we recommend the use of our class-oriented **`gusimplewhiteboard`**.
- Mechanisms for sub-machine hierarchies and introspection to implement complex behaviors. FSMs can **be suspended, resumed,** or **restarted**, as well as queried as to whether they are running or not.
- Formal semantics that enables **simulation, validation**, and **formal verification**.
- Associated tools such as (**`MiEditLLFSM`** and **`MiCASE`**) that enable rapid development of FSM arrangements.
- Tested in 64-bit, 32-bit CPUs and *even 8-bit* controllers like the Atmel AVR.

Outline

- Two tools
  - `clfsm`
  - `mipal gusimplewhiteboard`
  - What do they do?
- Finite-State Machines (FSM)
  - Logic-labeled FSMs
- Examples

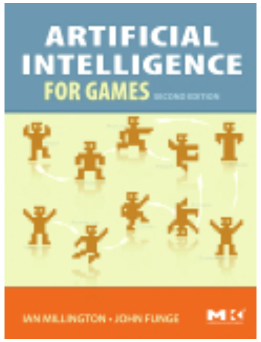
- What have they enabled
  - software architectures /middleware
  - Model-driven development
  - Formal verification

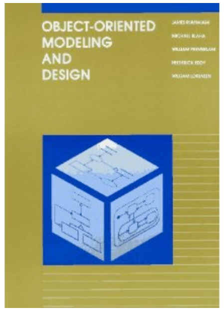
- Conclusions
  - What can I do so you would use them?

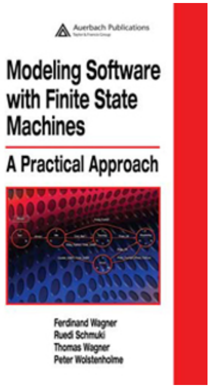


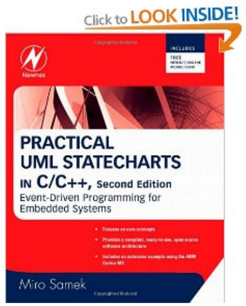
Finite-State Machines (FSM)

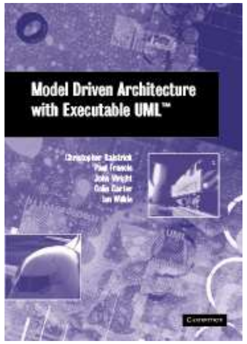
- Widely used model of behavior in embedded systems
  - *QP* (Samek, 2008), *Bot- Studio* (Michel, 2004) *StateWORKS* (Wagner et al., 2006) and *MathWorks*◯R *StateFlow*. The UML form of FSMs derives from OMT (Rumbaugh et al., 1991, Chapter 5), and the MDD initiatives of Executable UML (Mellor and Balcer, 2002).

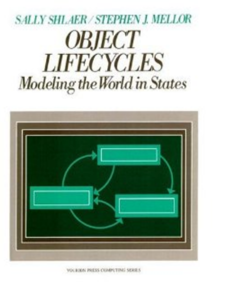






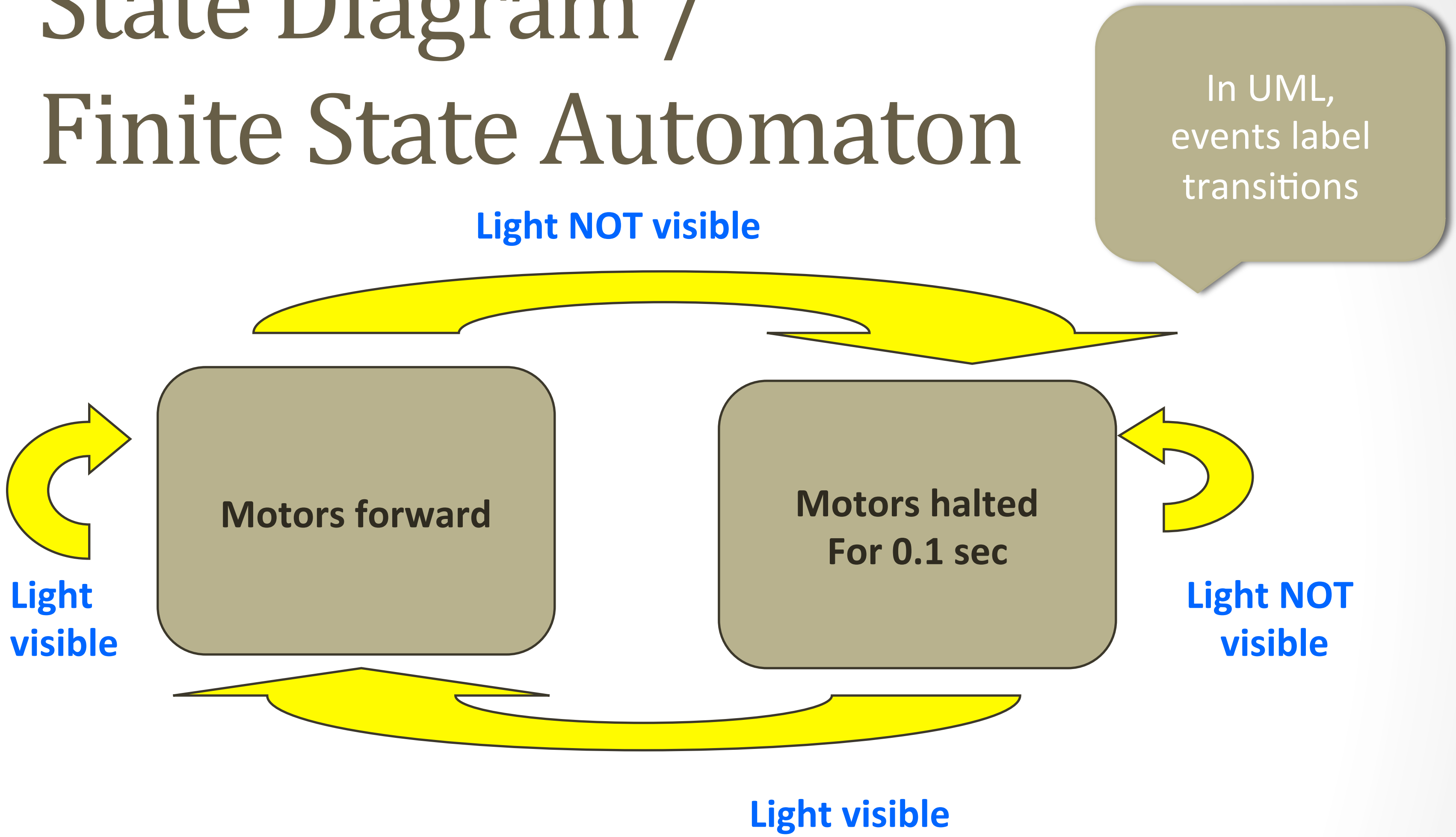




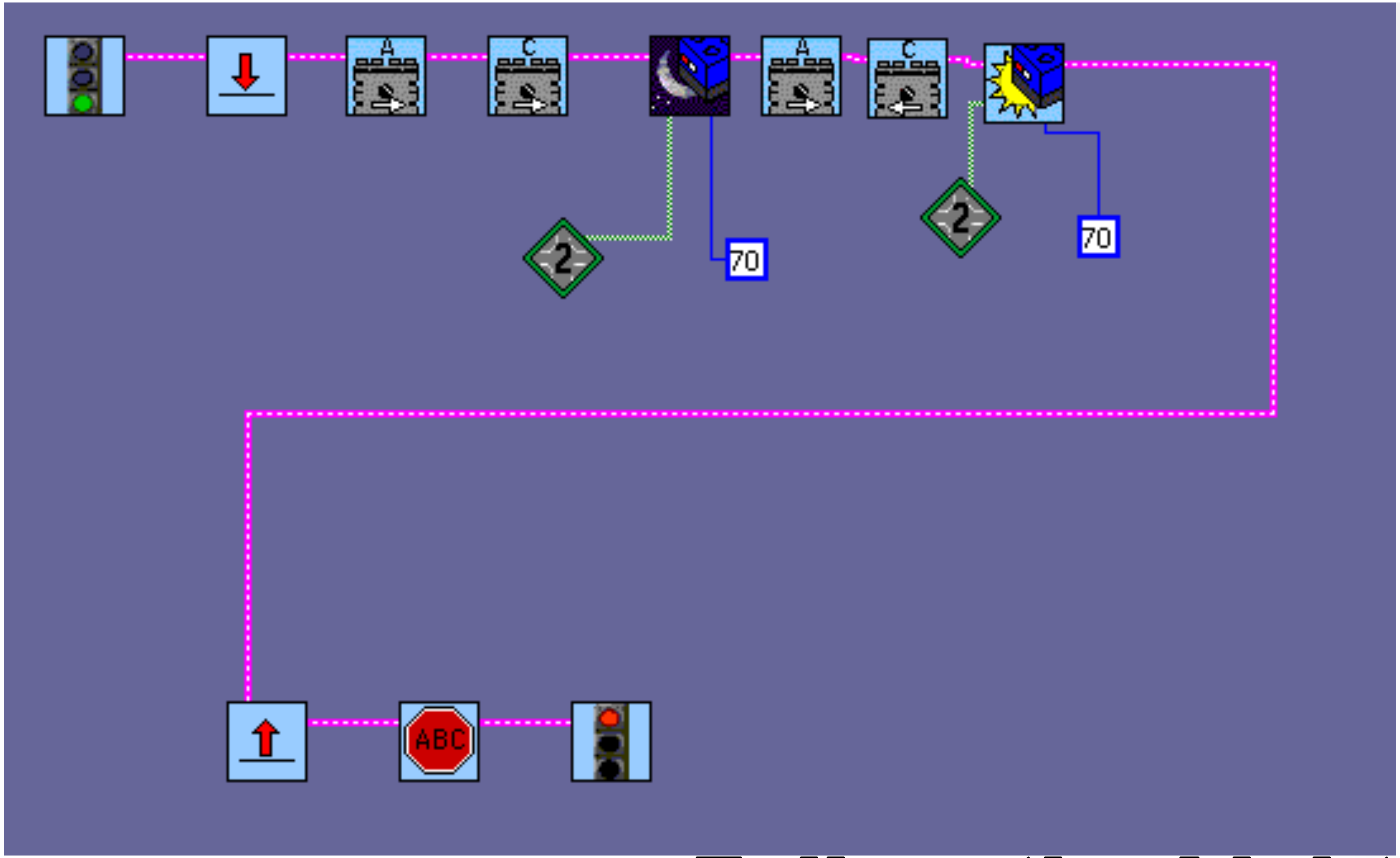


- The original Subsumption Architecture was implemented using the <u>Subsumption Language</u>
- It was based on <u>**finite state machines**</u> (FSMs) augmented with timers (AFSMs)
- AFSMs were implemented in Lisp

State Diagram / Finite State Automaton



Follow the Light



Follow the Light

LabVIEW (short for Laboratory Virtual Instrument Engineering Workbench)
LEGO RoboLab

Robot control (philosophies)

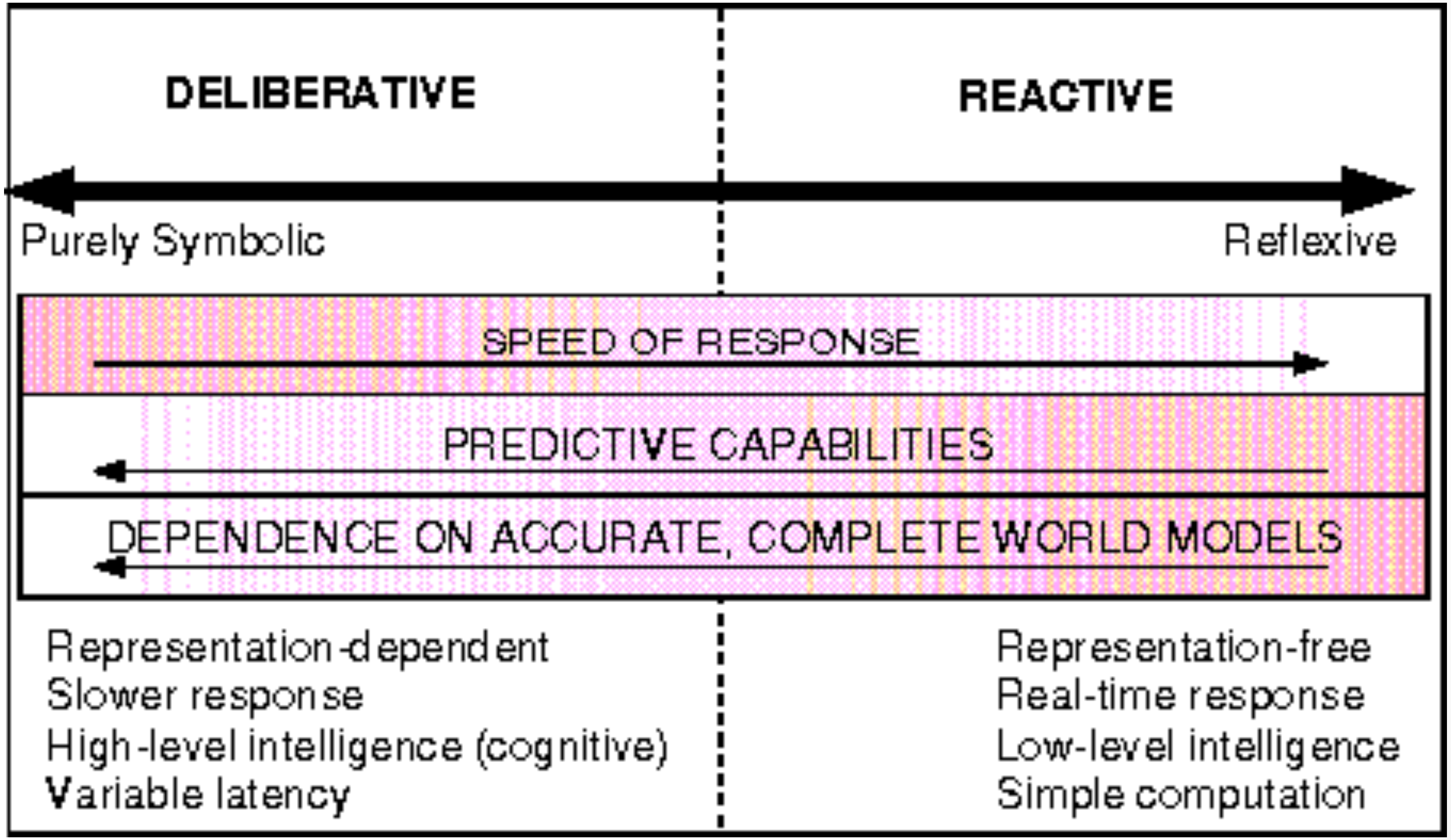
- Open Loop Control
  - Just carry on, don't look at the environment
- Feedback control
  - Minimize the error to the desired state
- Reactive Control
  - Don' t think, (re)act.
- Deliberative (Planner-based/Logic -based) Control
  - Think hard, act later.
- Hybrid Control
  - Think and act separately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

No use of logic

no use of common sense

no intelligence?

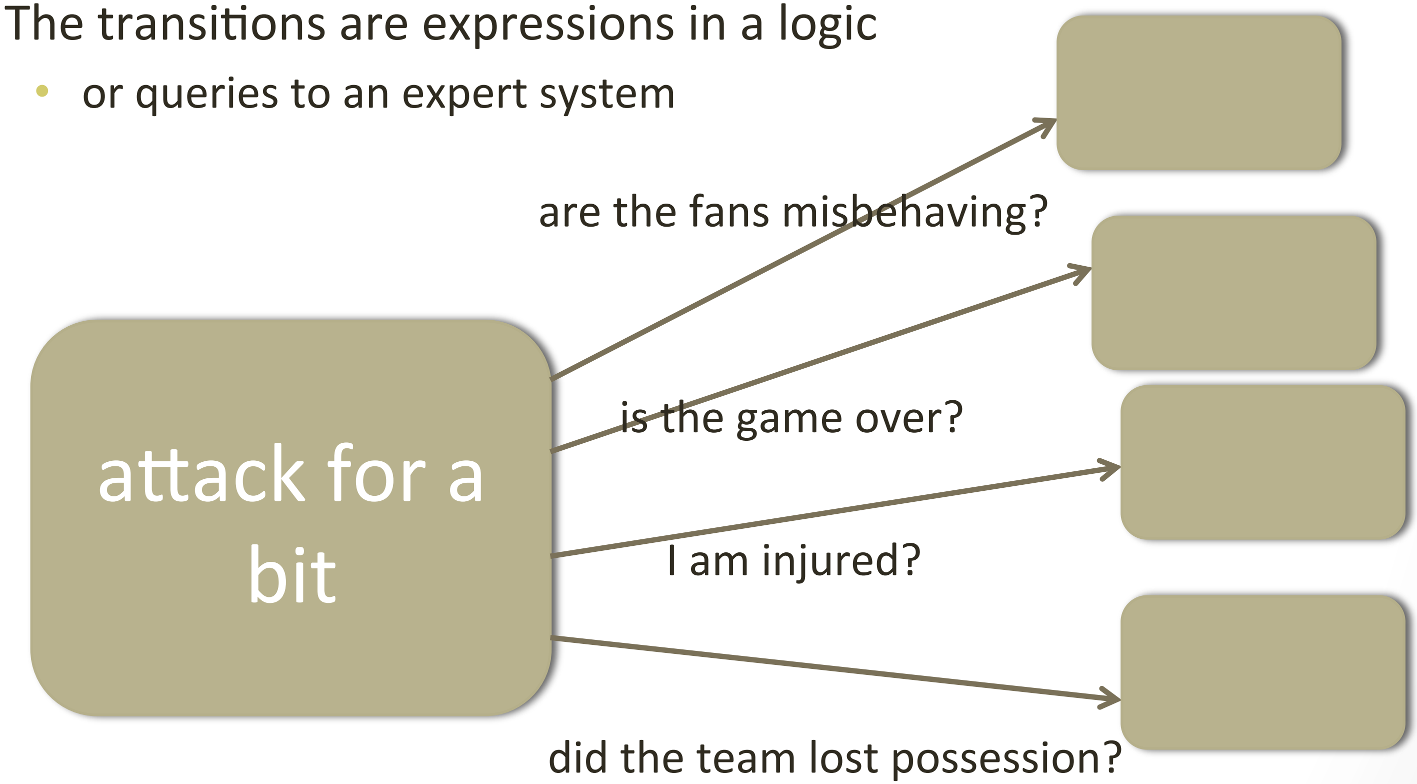
How is a robot architecture organized



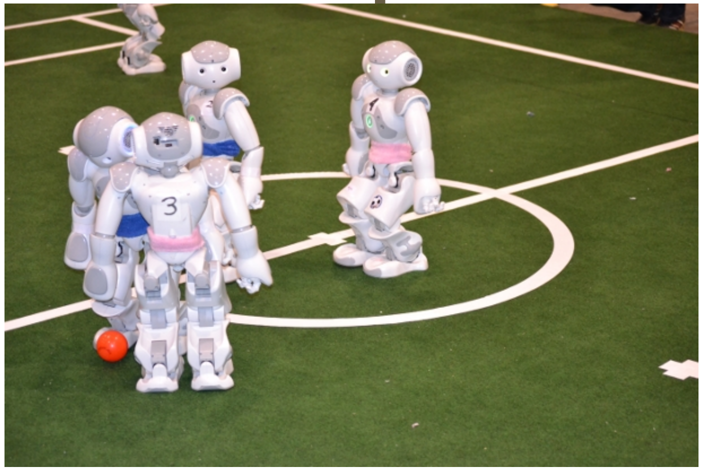
From "Behavior-Based Robotics" by R. Arkin, MIT Press, 1998

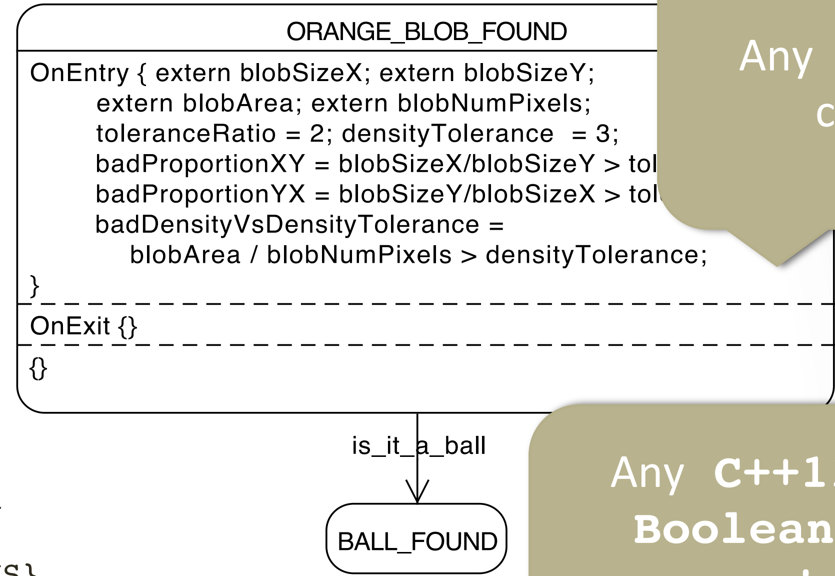
Logic-labeled FSMs

- 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



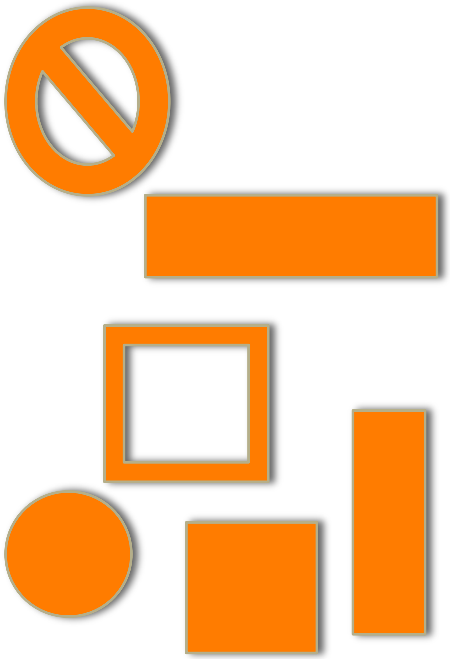
Example from robotic soccer





Any **C++11** code

Any **C++11 Boolean expression** (code)



```
% 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"}.
```

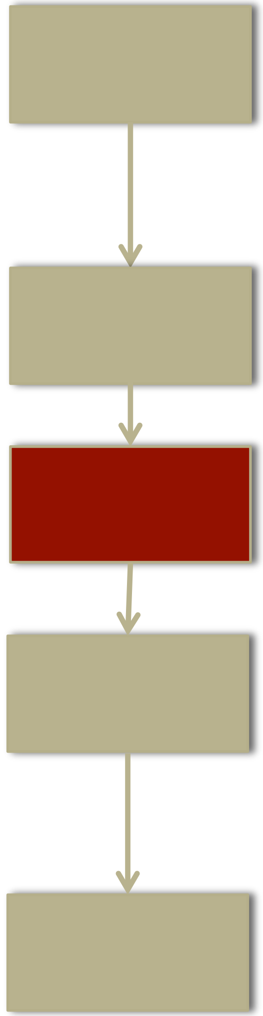
Logic labeled FSMs provide deliverative control

Outline

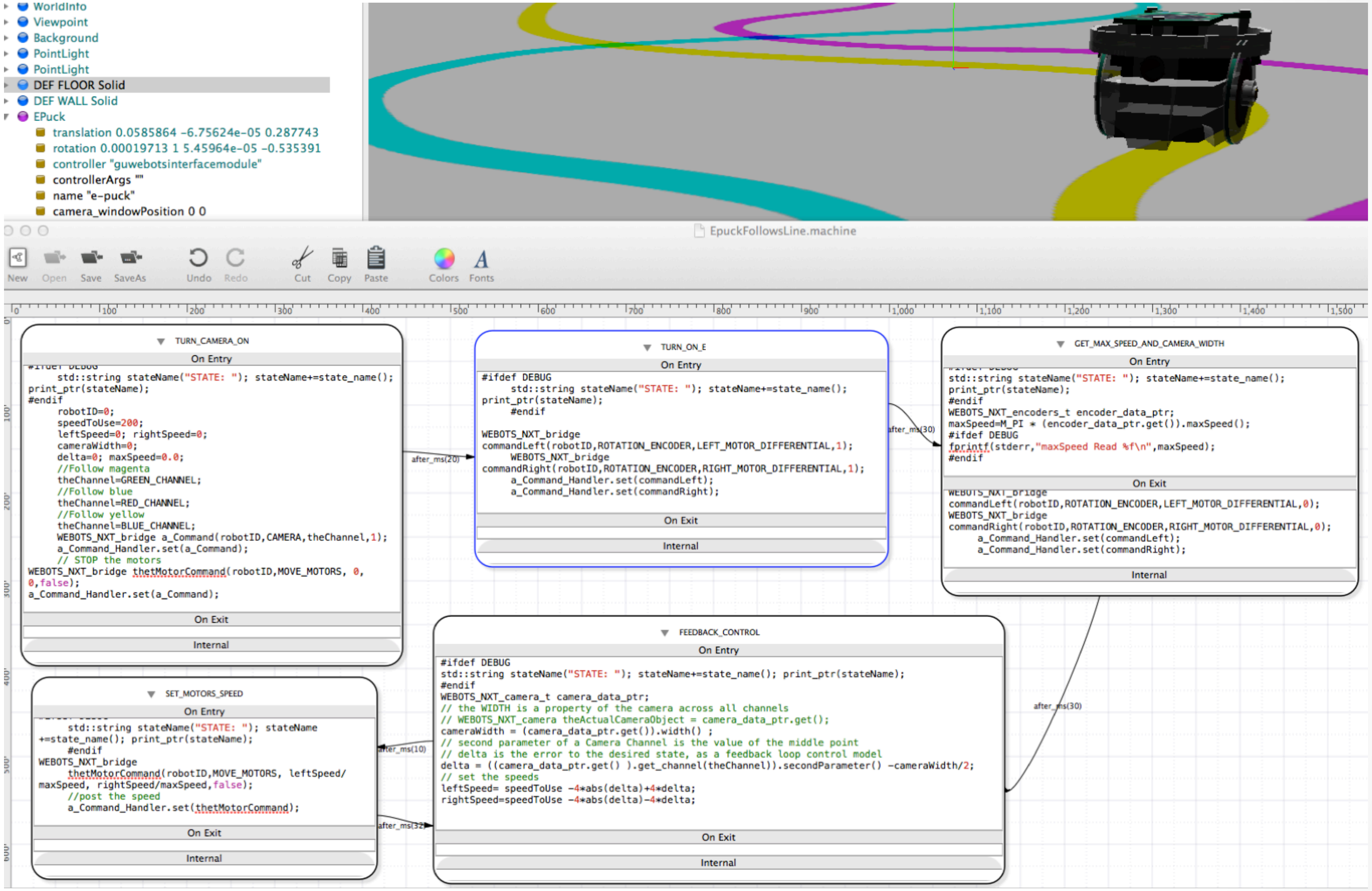
- Two tools
  - `clfsm`
  - `mipal gusimplewhiteboard`
  - What do they do?
- Finite-State Machines (FSM)
  - Logic-labeled FSMs
- Examples

- What have they enabled
  - software architectures /middleware
  - Model-driven development
  - Formal verification

- Conclusions
  - What can I do so you would use them?

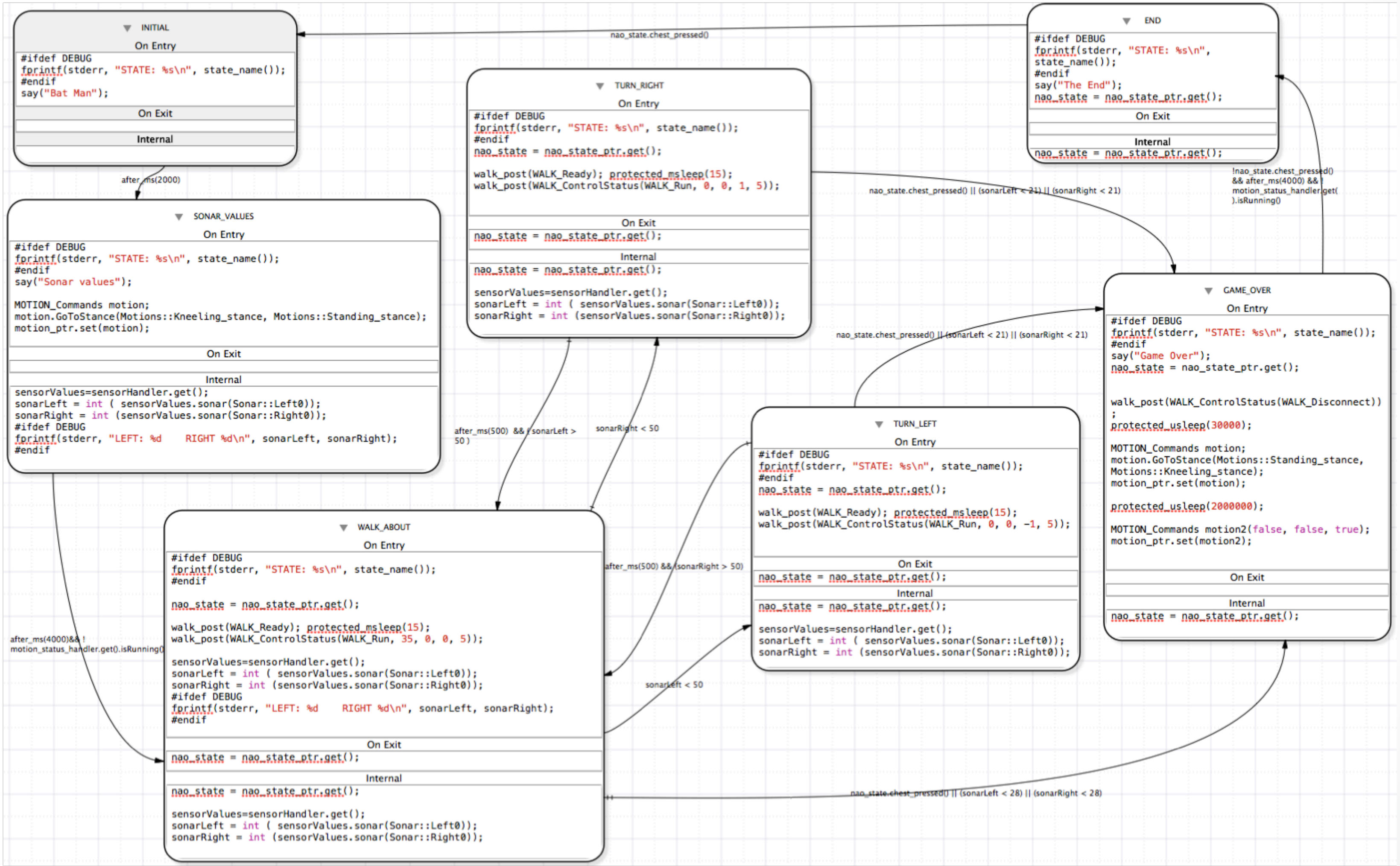


Example 1: Pure reactive control



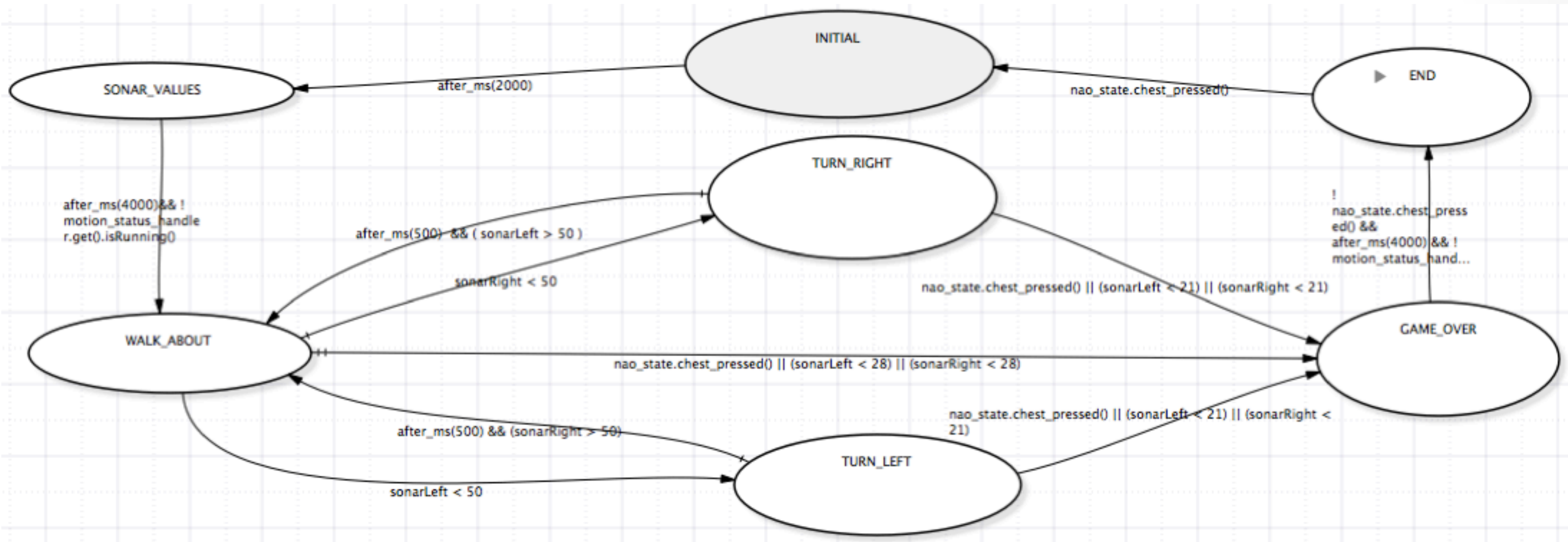
- https://www.youtube.com/watch?v=F8K4V78vUbk&feature=youtu.be

Example 2: BatMan moves (reactive control on a Nao)



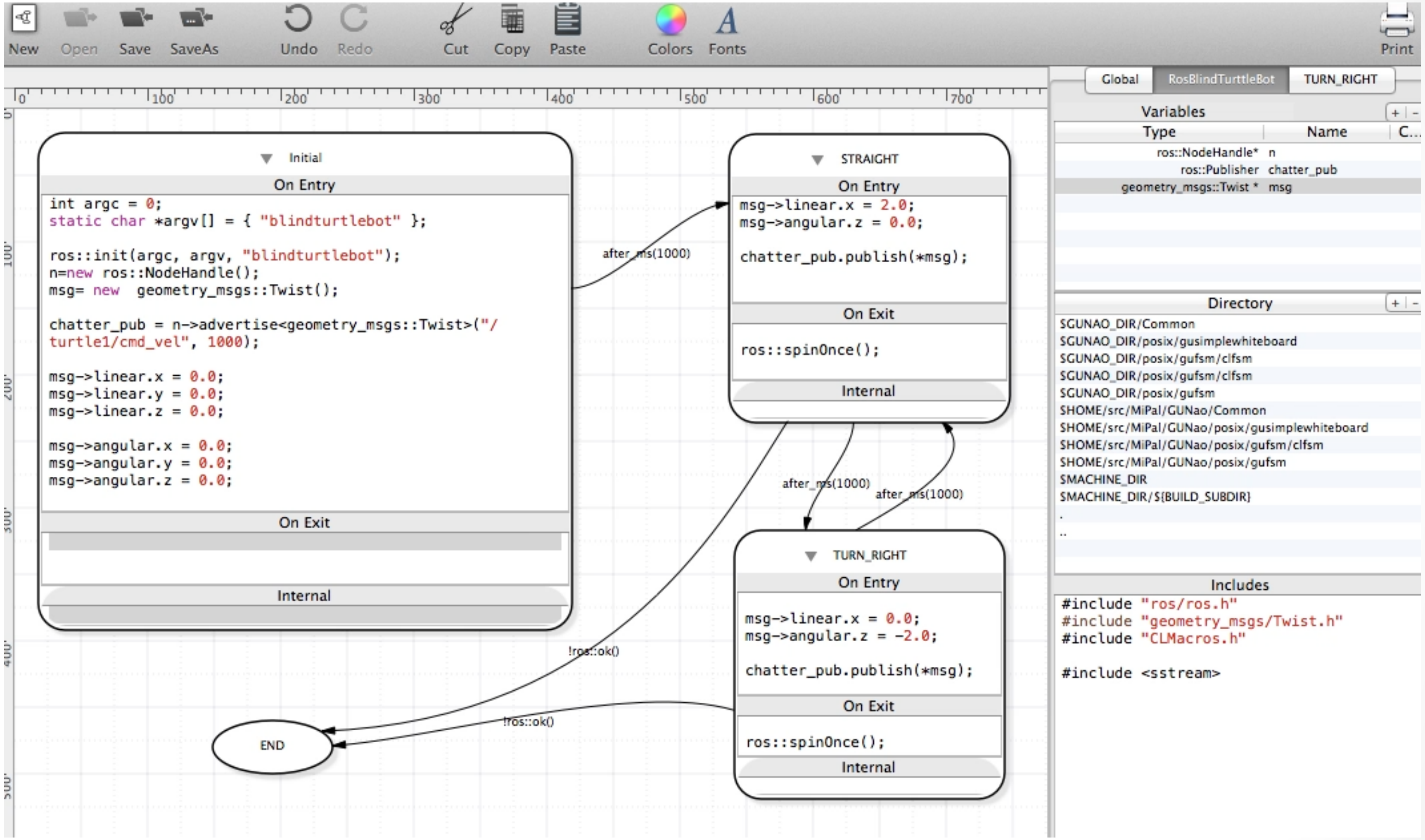
- https://www.youtube.com/watch?v=gN6rIveCWNk&feature=youtu.be

Example 2: BatMan moves (reactive control on a Nao)



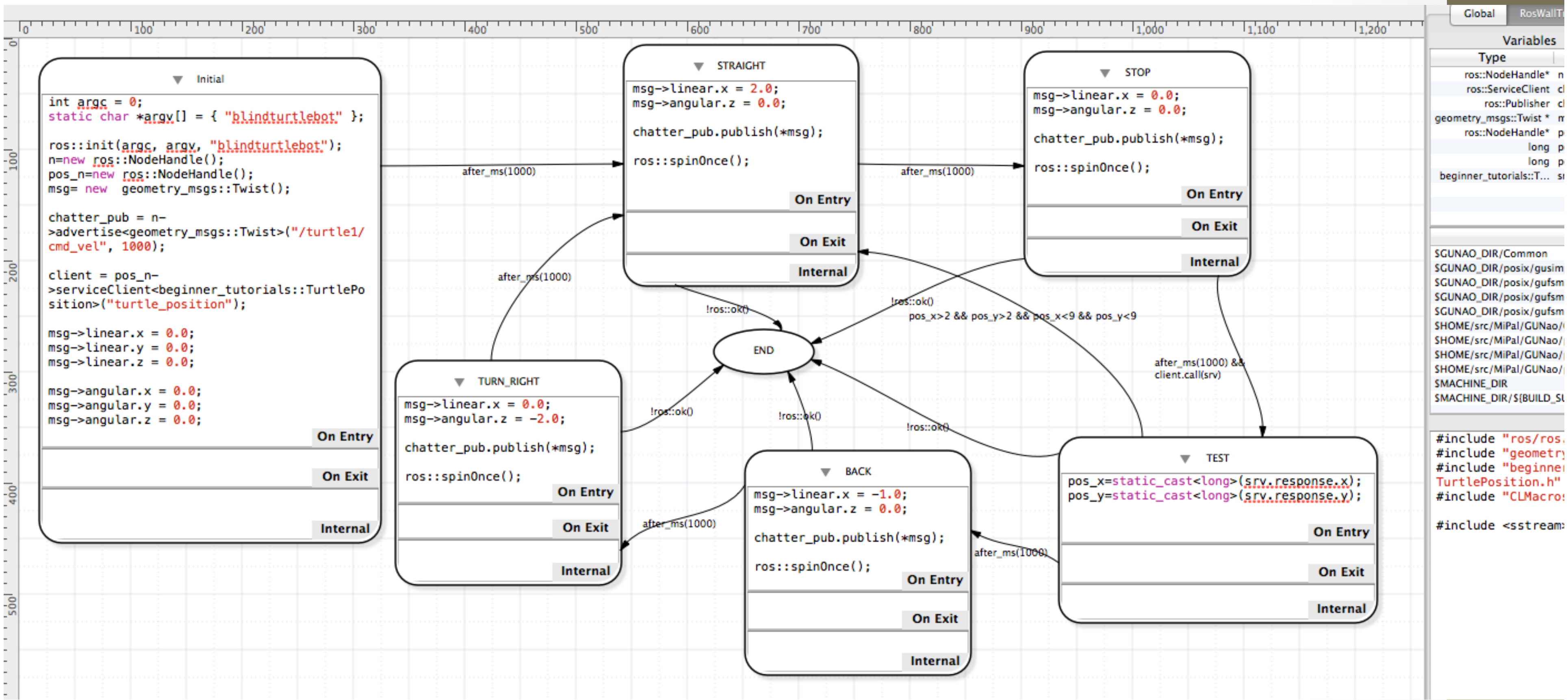
- https://www.youtube.com/watch?v=gN6rIveCWNk&feature=youtu.be

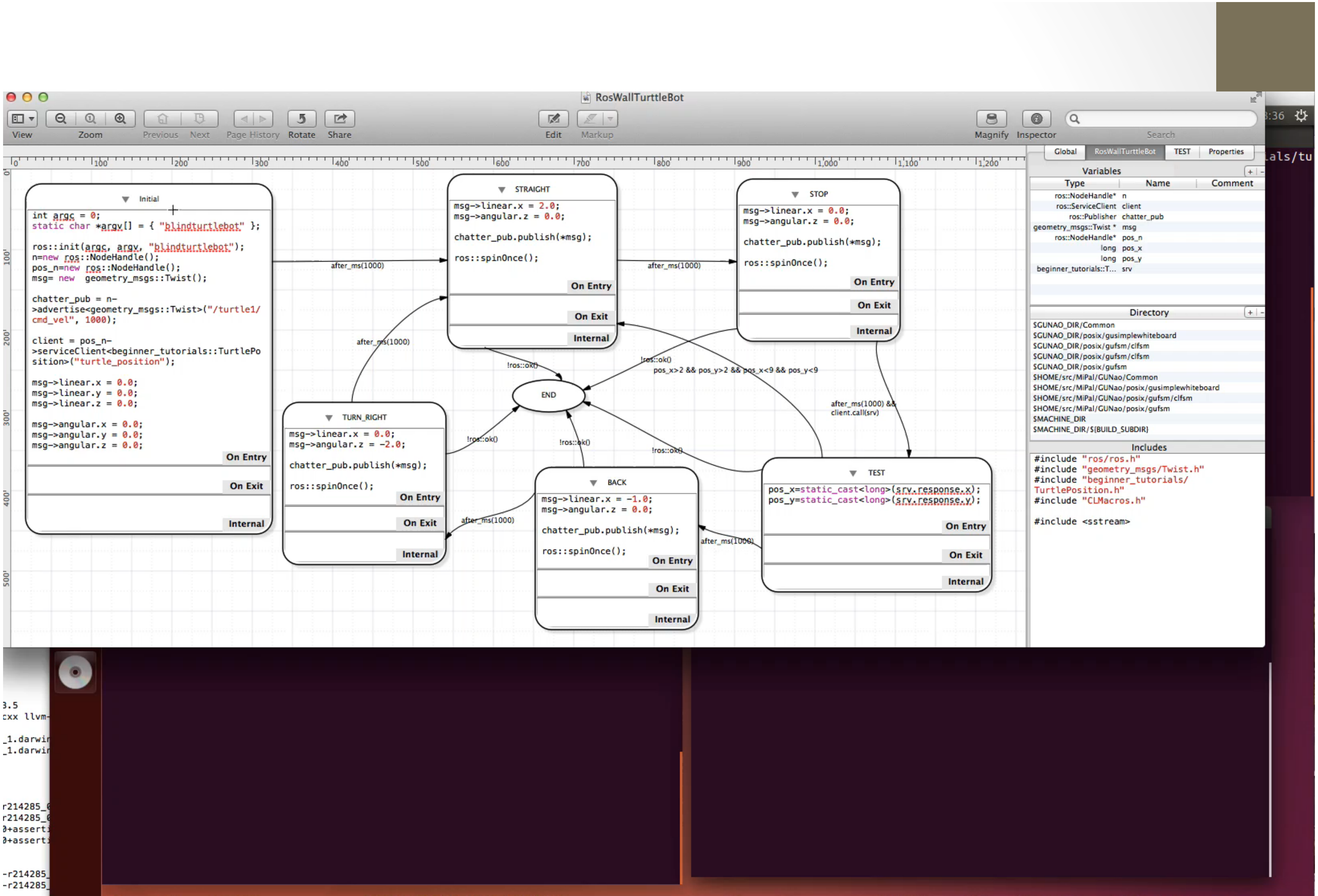
Example 3: Reactive control on ROS



- https://www.youtube.com/watch?v=AJYA2hB4i9U&feature=youtu.be

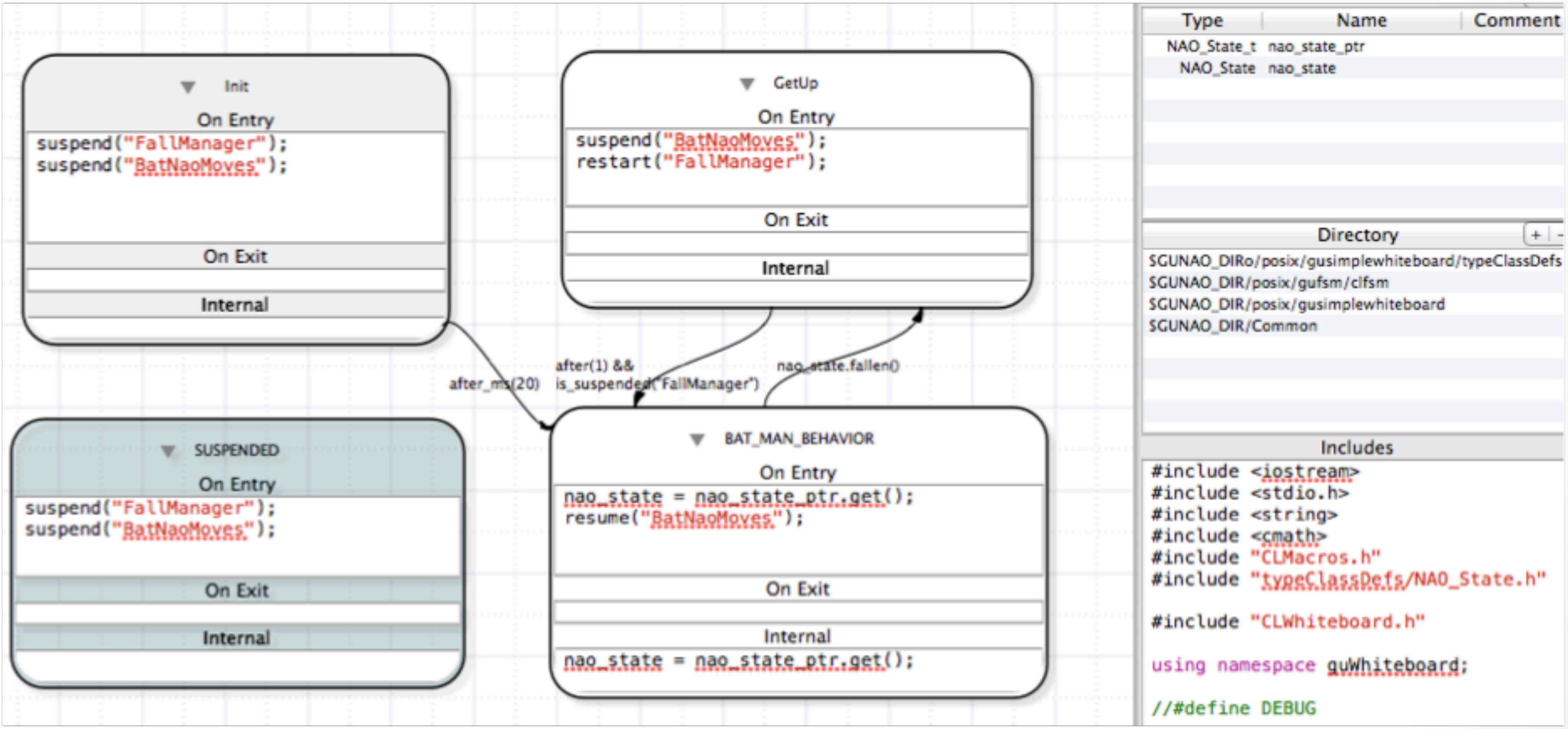
A turtle afraid of the walls





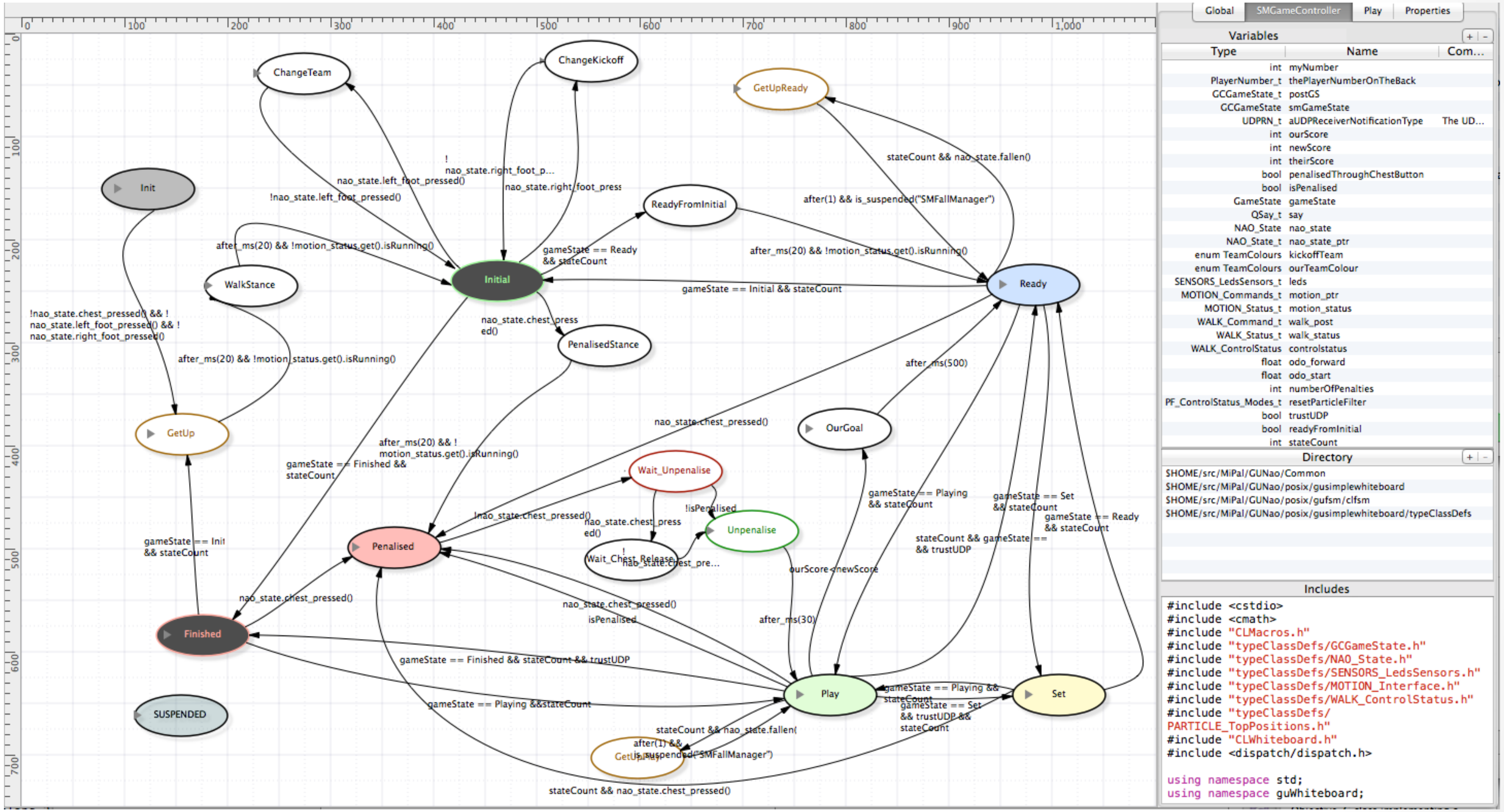
RosWallTurttleBot
View
Zoom
Previous
Next
Page History
Rotate
Share
Edit
Markup
Magnify
Inspector
Search
Initial
int argc = 0;
static char *argv[] = { "blindturtlebot" };
ros::init(argc, argv, "blindturtlebot");
n=new ros::NodeHandle();
pos_n=new ros::NodeHandle();
msg= new geometry_msgs::Twist();
chatter_pub = n->advertise<geometry_msgs::Twist>("/turtle1/cmd_vel", 1000);
client = pos_n->serviceClient<beginner_tutorials::TurtlePosition>("turtle_position");
msg->linear.x = 0.0;
msg->linear.y = 0.0;
msg->linear.z = 0.0;
msg->angular.x = 0.0;
msg->angular.y = 0.0;
msg->angular.z = 0.0;
On Entry
On Exit
Internal
STRAIGHT
msg->linear.x = 2.0;
msg->angular.z = 0.0;
chatter_pub.publish(*msg);
ros::spinOnce();
STOP
msg->linear.x = 0.0;
msg->angular.z = 0.0;
chatter_pub.publish(*msg);
ros::spinOnce();
TURN_RIGHT
msg->linear.x = 0.0;
msg->angular.z = -2.0;
chatter_pub.publish(*msg);
ros::spinOnce();
BACK
msg->linear.x = -1.0;
msg->angular.z = 0.0;
chatter_pub.publish(*msg);
ros::spinOnce();
TEST
pos_x=static_cast<long>(srv.response.x);
pos_y=static_cast<long>(srv.response.y);
END
after_ms(1000)
!ros::ok()
pos_x>2 && pos_y>2 && pos_x<9 && pos_y<9
after_ms(1000) && client.call(srv)
Global
RosWallTurttleBot
TEST
Properties
Variables
Type
Name
Comment
ros::NodeHandle* n
ros::ServiceClient client
ros::Publisher chatter_pub
geometry_msgs::Twist * msg
ros::NodeHandle* pos_n
long pos_x
long pos_y
beginner_tutorials::T... srv
Directory
$GUNAO_DIR/Common
$GUNAO_DIR/posix/gusimplewhiteboard
$GUNAO_DIR/posix/gufsm/clfsm
$GUNAO_DIR/posix/gufsm/clfsm
$GUNAO_DIR/posix/gufsm
$HOME/src/MiPal/GUNao/Common
$HOME/src/MiPal/GUNao/posix/gusimplewhiteboard
$HOME/src/MiPal/GUNao/posix/gufsm/clfsm
$HOME/src/MiPal/GUNao/posix/gufsm
$MACHINE_DIR
$MACHINE_DIR/$(BUILD_SUBDIR)
Includes
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "beginner_tutorials/TurtlePosition.h"
#include "CLMacros.h"
#include <sstream>

Example 4: Behavior Based Control / **Subsumption** Architecture



Mechanisms for sub-machine hierarchies and introspection to implement complex behaviors. FSMs can **be suspended, resumed,** or **restarted**, as well as queried as to whether they are running or not.

Example 5: RoboCup Game Controller



Mechanisms for sub-machine hierarchies and introspection to implement complex behaviors. FSMs can **be suspended, resumed,** or **restarted**, as well as queried as to whether they are running or not.

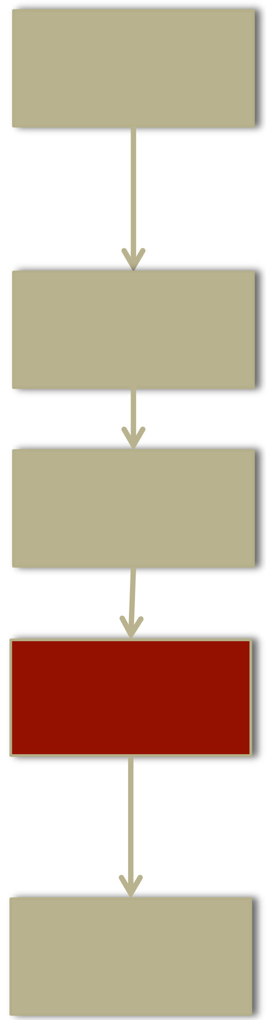
`clfsm`: compiled logic-labeled finite-state machines

SUMMARY

- Complete **`POSIX`** and **`C++11`** compliance.
- Open source **`catkin ROS`** package release (**`mipal.net.au/downloads.php`**).
- Transitions are labeled by Boolean expressions (not events), facilitating formal verification and eliminating all need for concerns about event queues.
- Transition labels are arbitrary **`C++11` Boolean expressions**, enabling reasoning into what may otherwise seem a purely reactive architecture.
- Handling of machines constructed with states that have UML 2.0 (or SCXML) **`OnEntry`**, **`OnExit`**, and **`Internal`** sections with clear semantics.
- Guaranteed **sequential ringlet schedule** for the concurrent execution of FSMs (removing the need for critical sections and synchronization points).
- Efficient execution as the entire arrangement runs as **compiled code** without thread switching.
- Being agnostic to communication mechanisms between machines, allowing, for example use with **`ROS:services`** and **`ROS:messages`** – however, we recommend the use of our class-oriented **`gusimplewhiteboard`**.
- Mechanisms for sub-machine hierarchies and introspection to implement complex behaviors. FSMs can **be suspended, resumed,** or **restarted**, as well as queried as to whether they are running or not.
- Formal semantics that enables **simulation, validation**, and **formal verification**.
- Associated tools such as (**`MiEditLLFSM`** and **`MiCASE`**) that enable rapid development of FSM arrangements.
- Tested in 64-bit, 32-bit CPUs, and *even 8-bit* controllers like the Atmel AVR.

Outline

- Two tools
  - `clfsm`
  - `mipal gusimplewhiteboard`
  - What do they do?
- Finite-State Machines (FSM)
  - Logic-labeled FSMs
- Examples
- What have they enabled
  - software architectures /middleware
  - Model-driven development
  - Formal verification
- Conclusions
  - What can I do so you would use them?

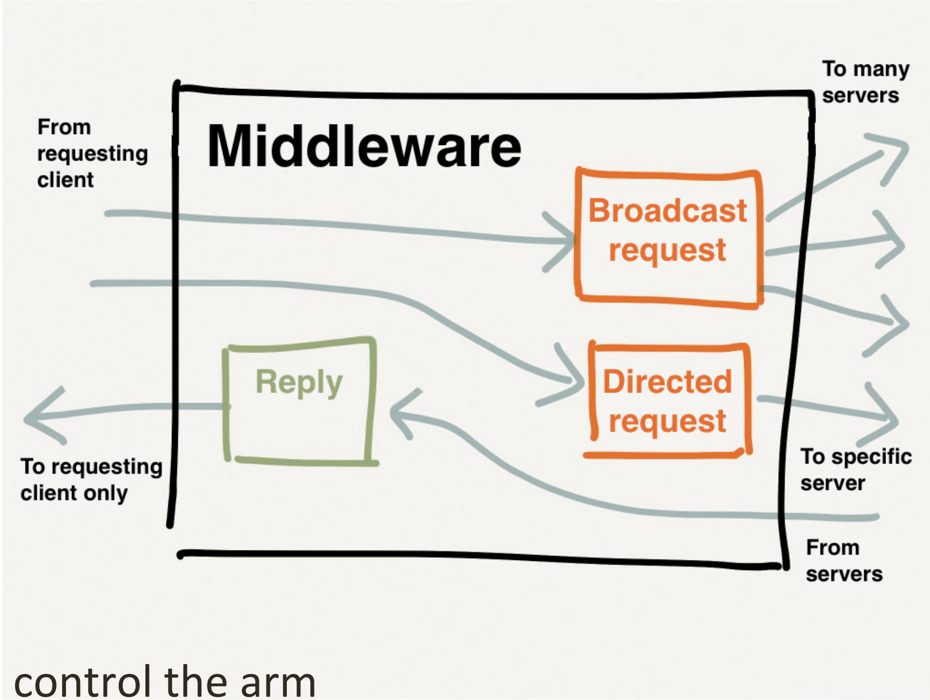


gusimplewhiteboard :In memory OO-messages/classes

- Completely **C++11** and **POSIX** compliant; thus, platform independent: used on Mac OS X (Mountain Lion), LINUX 13.10, Aldebaran Nao 1.14.3, Webots 7.1, the Raspberry Pi (www.raspberrypi.org), and Lego NXT.
- Released as a **ROS:catkin** package (**mipal.net.au/downloads.php**).
- Extremely fast performance for **add_Message** and **get_Message**, intra-process as well as inter-process.
- Completely **OO-compliant**. The classes that can be used are not restricted, the full data-structure mechanisms of **C++11** are available.
- Very **clear semantics** that removes lots of issues of concurrency control.

Middleware - Architecture

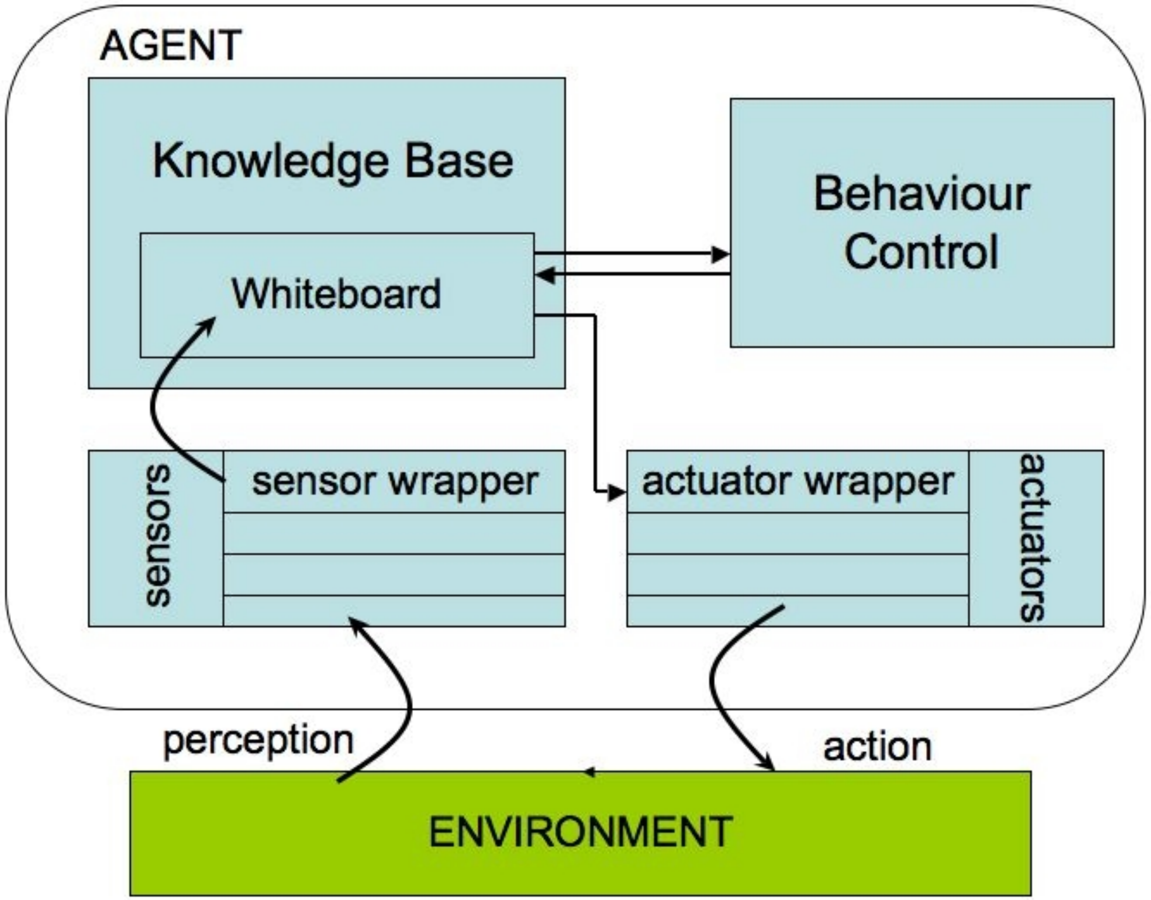
- In robotics we need to integrate many pieces of software in charge of different things
  - Sensors
  - Actuators
  - Filtering the sensors
  - Fusing the sensors
  - Coordinating the actuators
    - making the motors in an arm control the arm
  - Perform tasks, make decision, plan, learn
  - Communicate with others



Software Engineering concerns

- Modularity
- Integration
- Reliability/ Testing
- Development cycle
  - Simulations
  - Monitoring

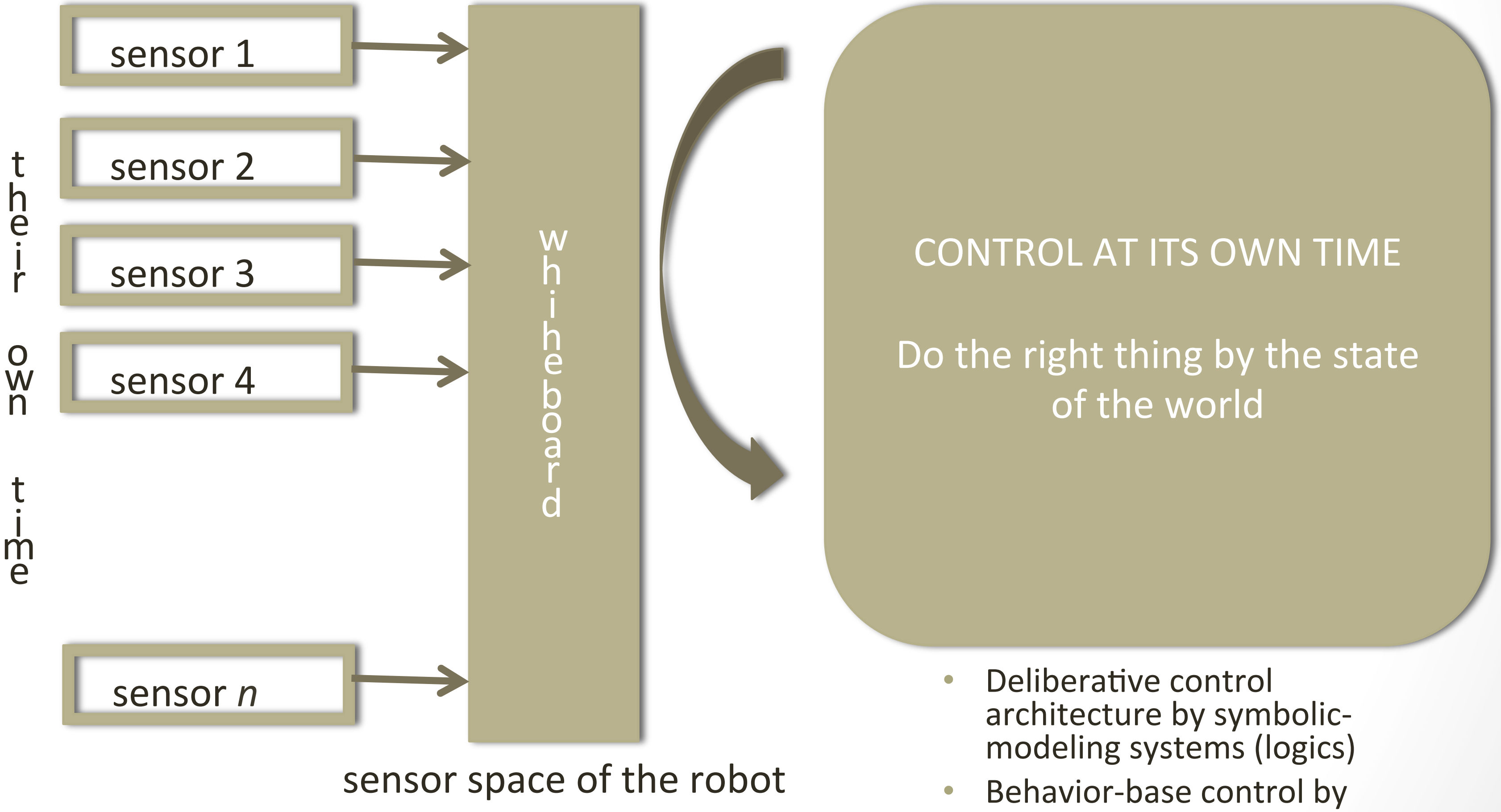
Whiteboard/Blackboard architecture



Reduce the number of APIs

Conceptual cycle

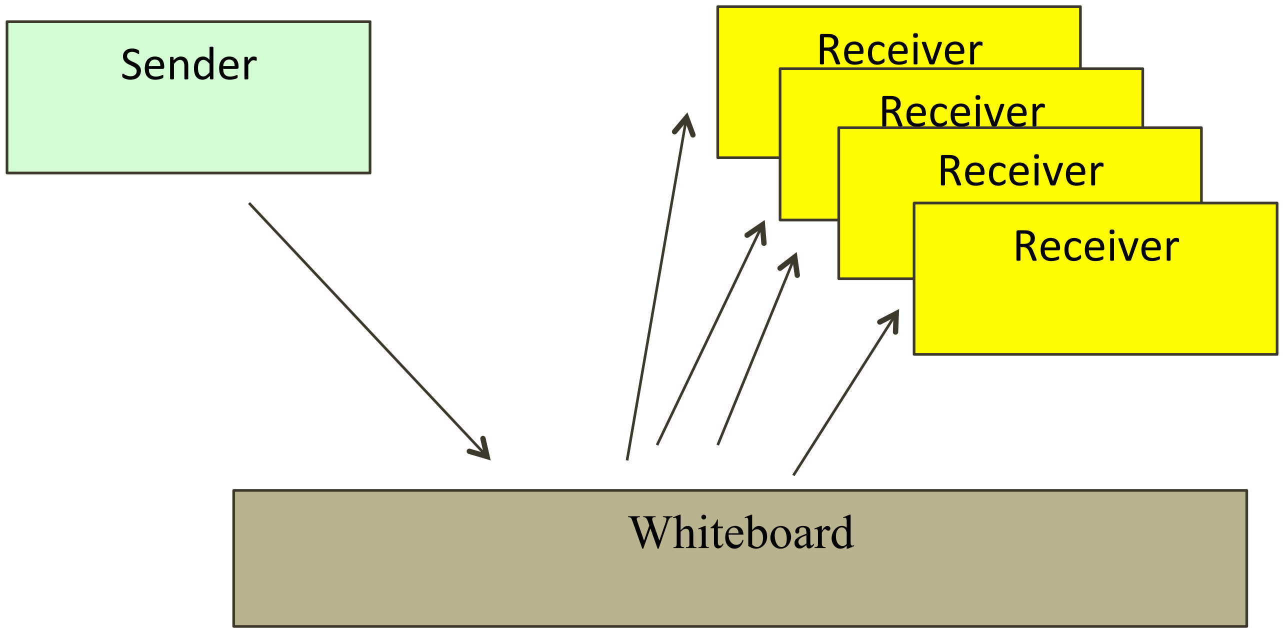
- Similar to a 'reactive-architecture'
- Similar to a whiteboard architecture



- Deliberative control architecture by symbolic-modeling systems (logics)
- Behavior-base control by arrangements of FSMs

Modes of communication

- PULL (closer to time-triggered)
  - receivers query the whiteboard for the latest from the sender
  - own thread for the receiver
  - sender just does and add message
- PUSH (closer to event-driven)
  - the receivers subscribe a call-back in the whiteboard
  - add message by sender spans new threads in the receivers



add_Message

- Includes

```
#include "gugenericwhiteboardobject.h"
#include "guwhiteboardtypelist_generated.h”
```

- Declare a handler

```
Ball_Belief_t wb_ball;
```

- Construct you objects (with the constructor of the OO-class)

```
Ball_Belief a_ball(50,30);
```

- Use the setter to actually post to the whiteboard

```
wb_ball.set(a_ball);
```

get_Message

- Includes

```
#include "gugenericwhiteboardobject.h"
#include "guwhiteboardtypelist_generated.h”
```

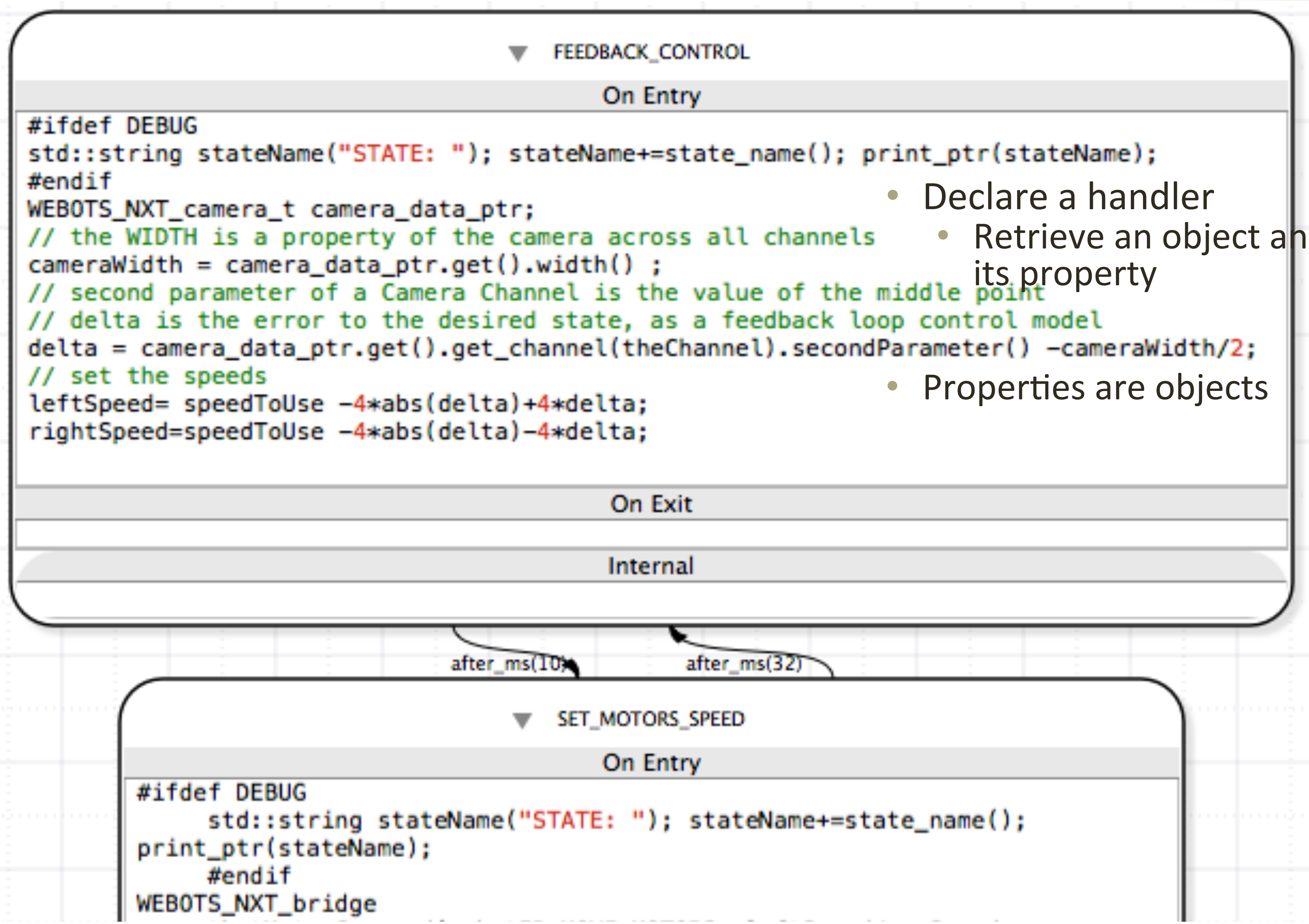
- Declare a handler

```
Ball_Belief_t wb_ball;
```

- Retrieve your object

```
Ball_Belief ball = wb_ball.get();
// or alternatively: ball = wb_ball();
```

Illustration of OO facility



- Declare a handler
  - Retrieve an object and its property
- Properties are objects

Speed

- Of the order of 50 times faster than ROS
- 2013 Mac Pro, 3 GHz 8-Core Intel Xeon E5, 32 GB memory 1867 MHz DDR3 ECC RAM
- Identical compiler flags (compiled with **`catkin_make`**)

gusimplewhiteboard		ROSmacports *Hydro*	
get_Message	0.0024 μs	ROS:subscribe()	20.14 μs
add_Message	0.0120 μs	ROS:publish()	20.87 μs

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

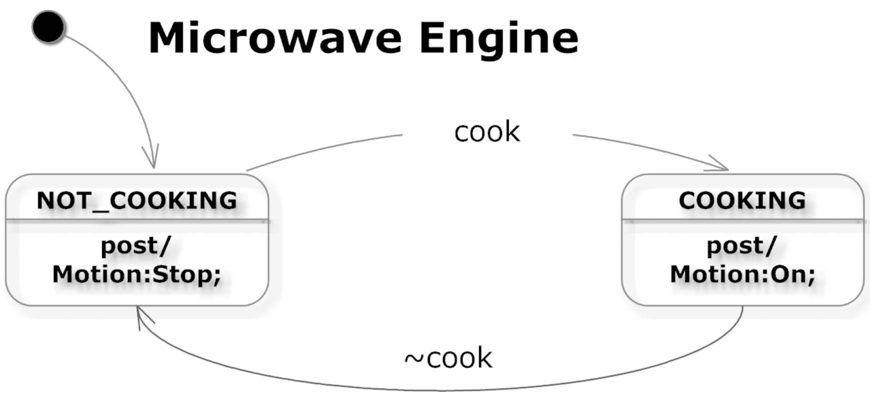
- Important SAFETY feature
  - OPENING THE DOOR SHALL STOP THE COOKING

Requirements

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.

One of the FSMs

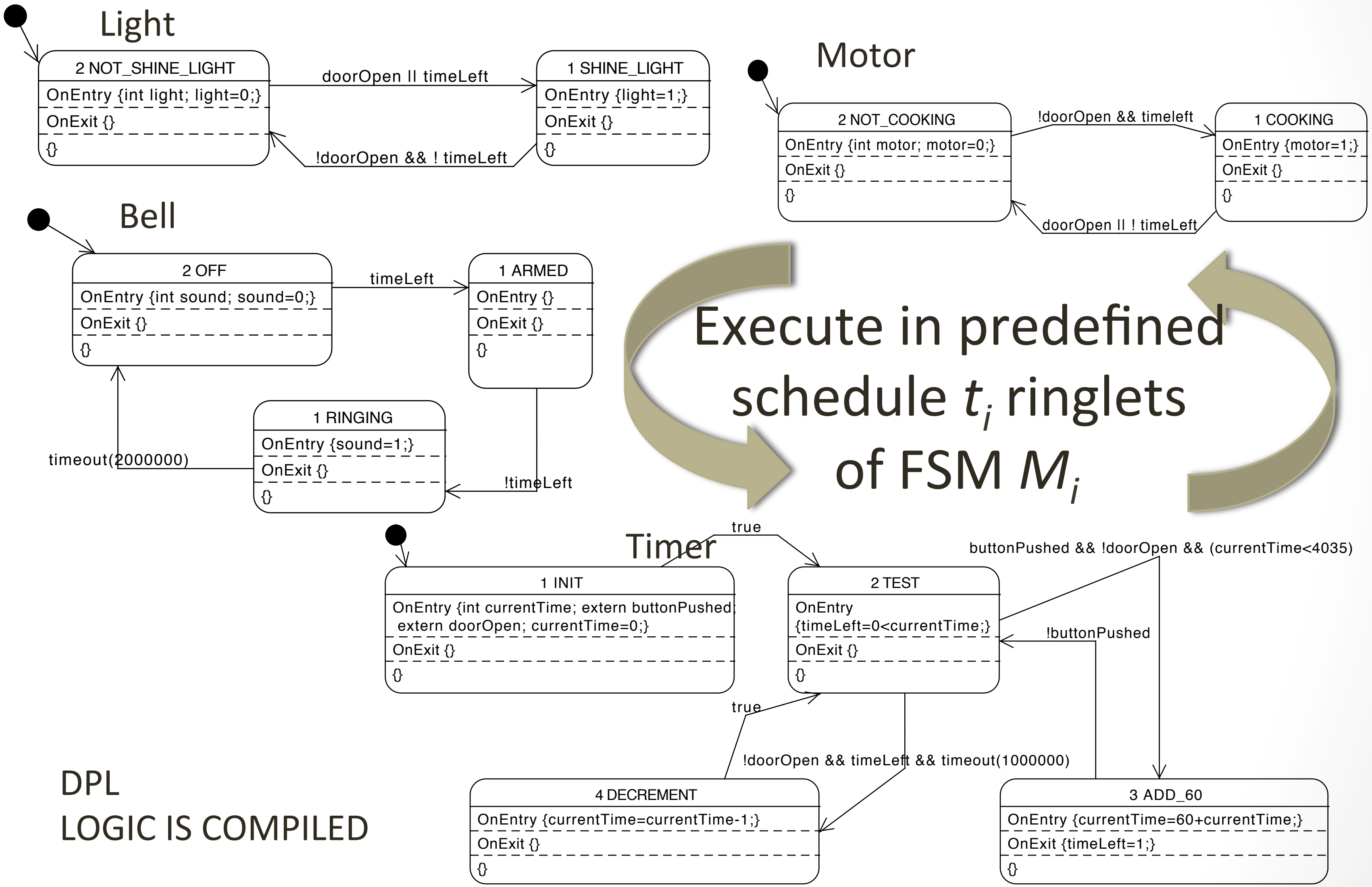
```
% MicrowaveCook.d

name{MicrowaveCook}.

input{timeLeft}.
input{doorOpen}.

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

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



Embedded systems are performing several things

- The models is made of several finite state-machines
  - Behavior-based control
- With a rich language of logic, the modeling aspect is decomposed
  - the action /reaction part of the system
    - the states and transitions of the finite-state machine
  - the declarative knowledge of the world
    - the logic system

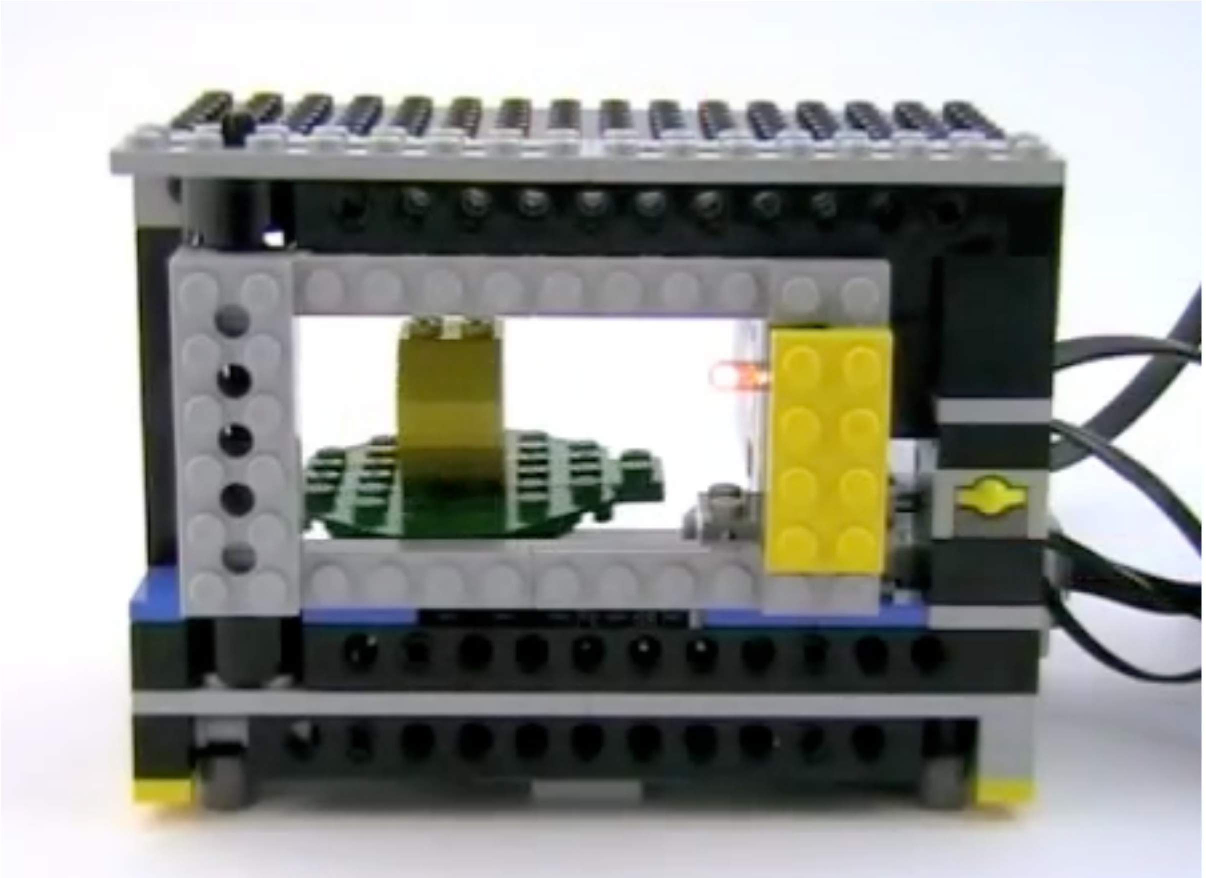
The complete arrangement



DPL

LOGIC IS COMPILED

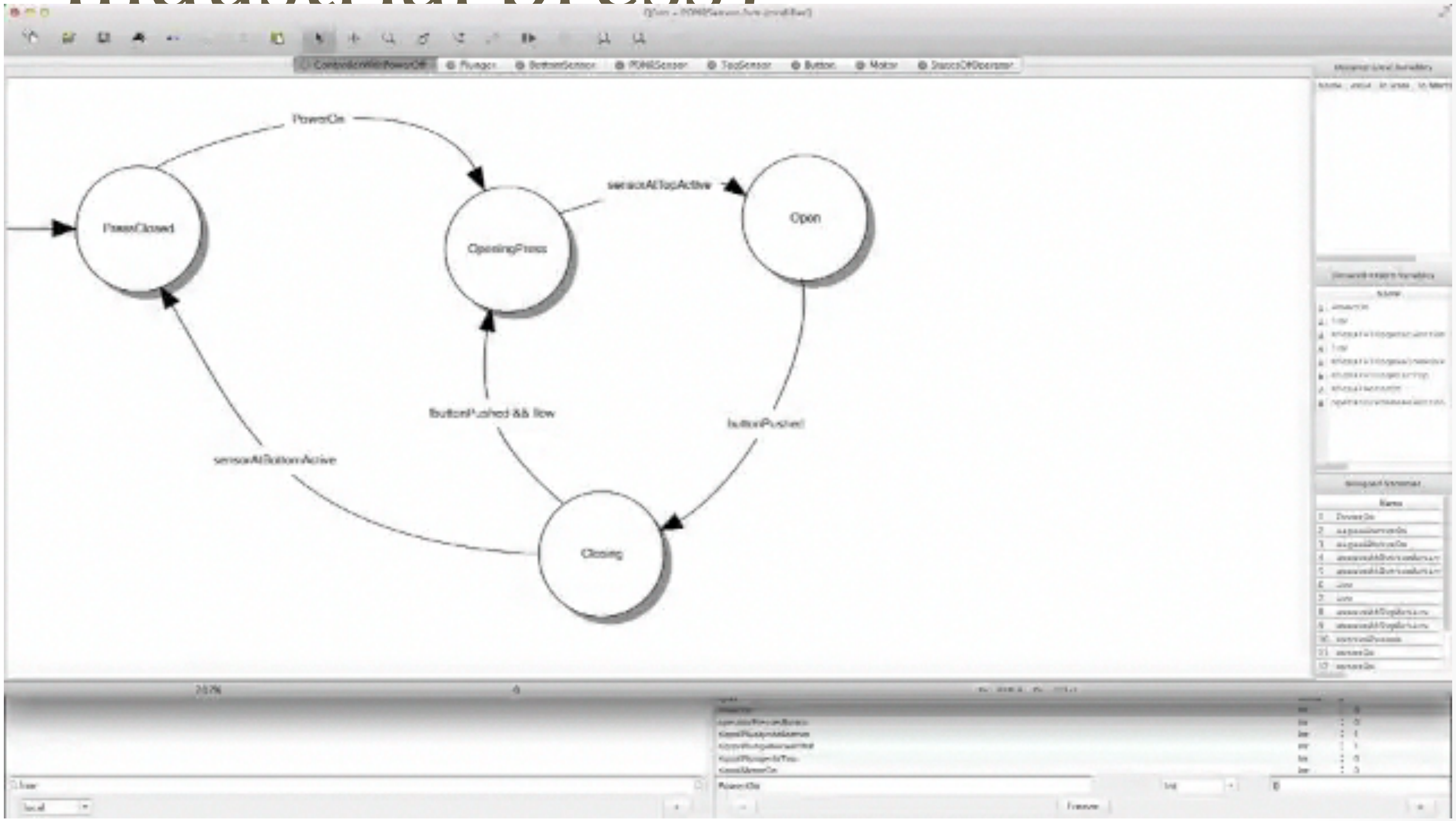
That is all folks!



Demo video

http://www.youtube.com/watch?v=t4ueI1o67Xk&feature=relmfu

Simulator (embedded system: Industrial press)



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

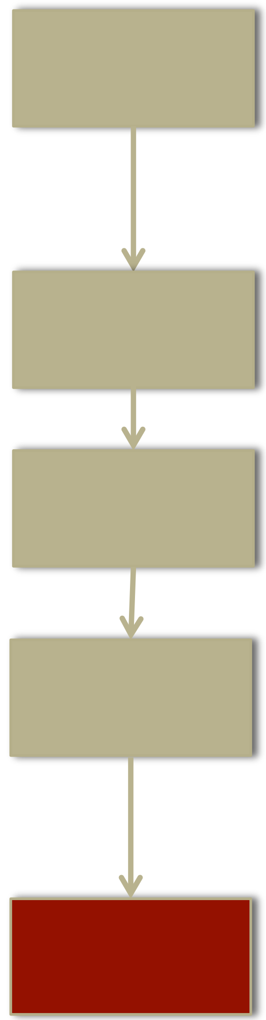
On-line debugging and simulation

Outline

- Two tools
  - `clfsm`
  - `mipal gusimplewhiteboard`
  - What do they do?
- Finite-State Machines (FSM)
  - Logic-labeled FSMs
- Examples

- What have they enabled
  - software architectures /middleware
  - Model-driven development
  - Formal verification

- Conclusions
  - What can I do so you would use them?



Regulate the number of threads

```
clfsm SMGameController Safety_BatteryMonitor
SMFallManager SMButtonChest SMButtonLeftFoot
SMButtonRightFoot SMRobotPosition SMSayIP SMShutdown
```

One thread

```
clfsm SMSoundStartStop SMSoundWhistle SMSoundDemo
SMGetUp SMPlayer SMBallFollower SMKicker SMH
SMBallSeeker SMReadyFromInitial SMReadyFromA
SMHeadBallTracker SMWalkScanner SMSeeker Col
SMHeadScannerGoal SMHeadGoalTracker SMGetClo
SMSet SMFindGoalOnSpot SMGoalieSaver SMFindG
SMLeapController SMTeleoperationController
SMTeleoperation SMTeleoperationHea
StopMotionRecorder SMYouCannotCat
```

Second thread

```
clfsm gukalmanfilter
```

Third thread

```
clfsm guUDPreceiver
```

Fourth thread

Very quick development of behaviors

- Very rapidly produces results
- Very rapidly we can trace the observed behavior to the code
- Very rapidly we have building blocks that add sophistication
  - All the behaviors in one go

The two paradigms

- Event-triggered
  - optimistic
    - best-case, response time
  - can't handle event-showers
  - not predictable
  - not scalable
    - repeat the verification

- Time-triggered
  - pessimistic
    - regular response time
  - predictable
  - scalable

Kopetz, H.: "Should Responsive Systems be Event-Triggered or Time- Triggered?"
IEIC*E Transactions on Information and Systems* **76**(11), 1325 (November 1993)

Check out **clfsm**

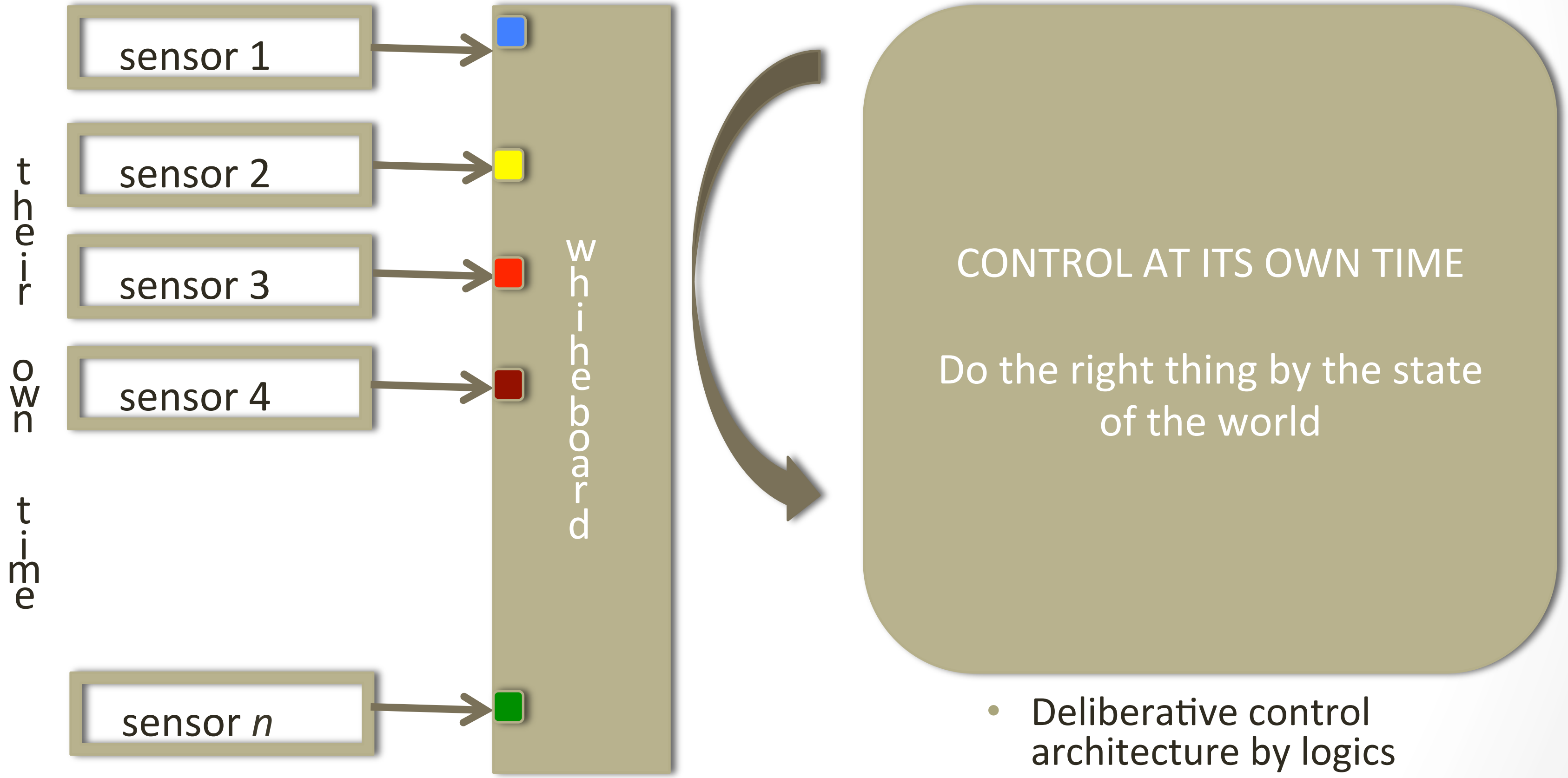
Let us know what you think

THANK YOU

Conceptual cycle

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

under one CPU rate for the sensors



sensory space of the robot

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

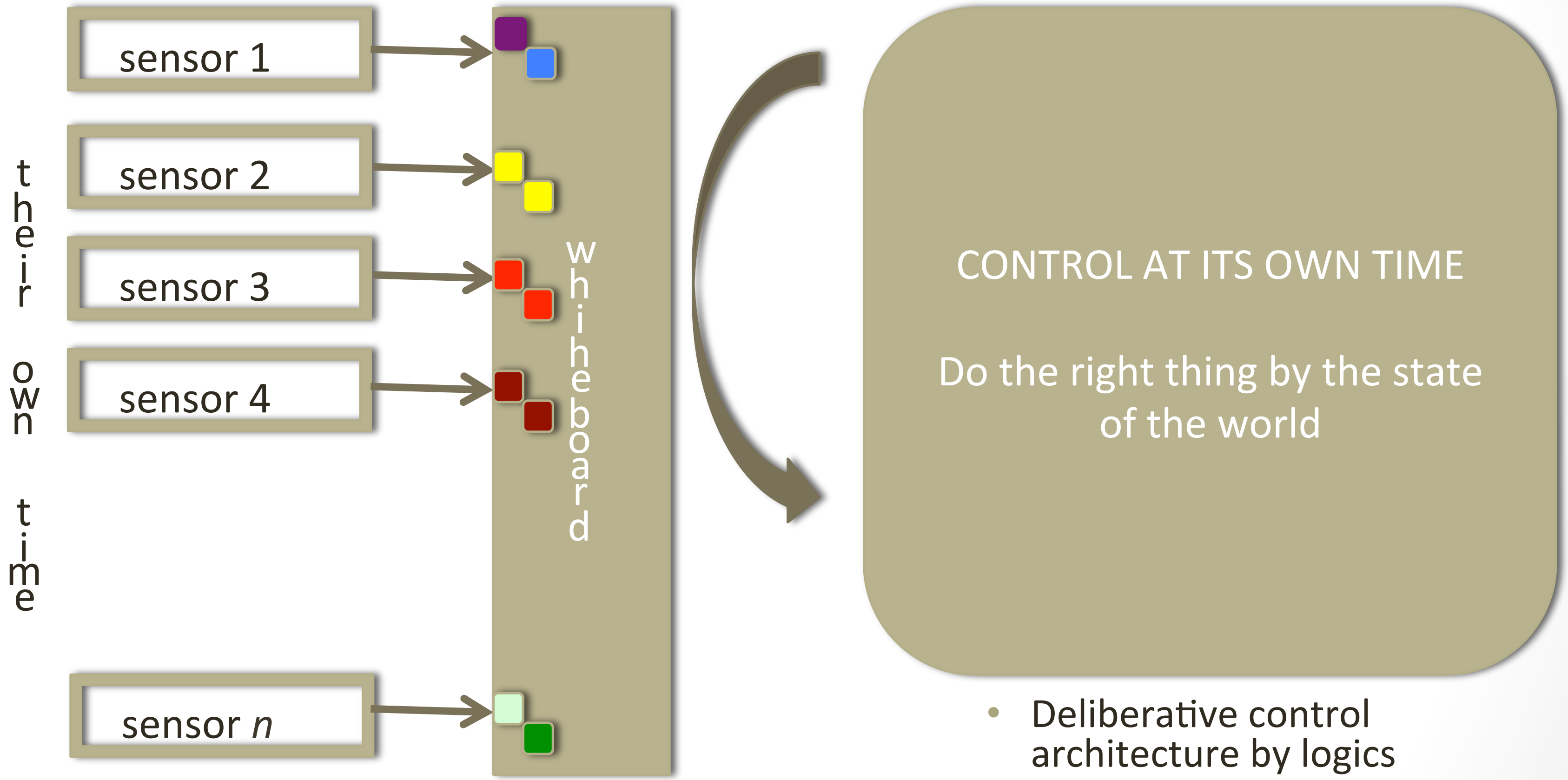
(c) Vlad Estivill-Castro

Conceptual cycle

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

under one CPU rate for the sensors

time t_2



sensory space of the robot

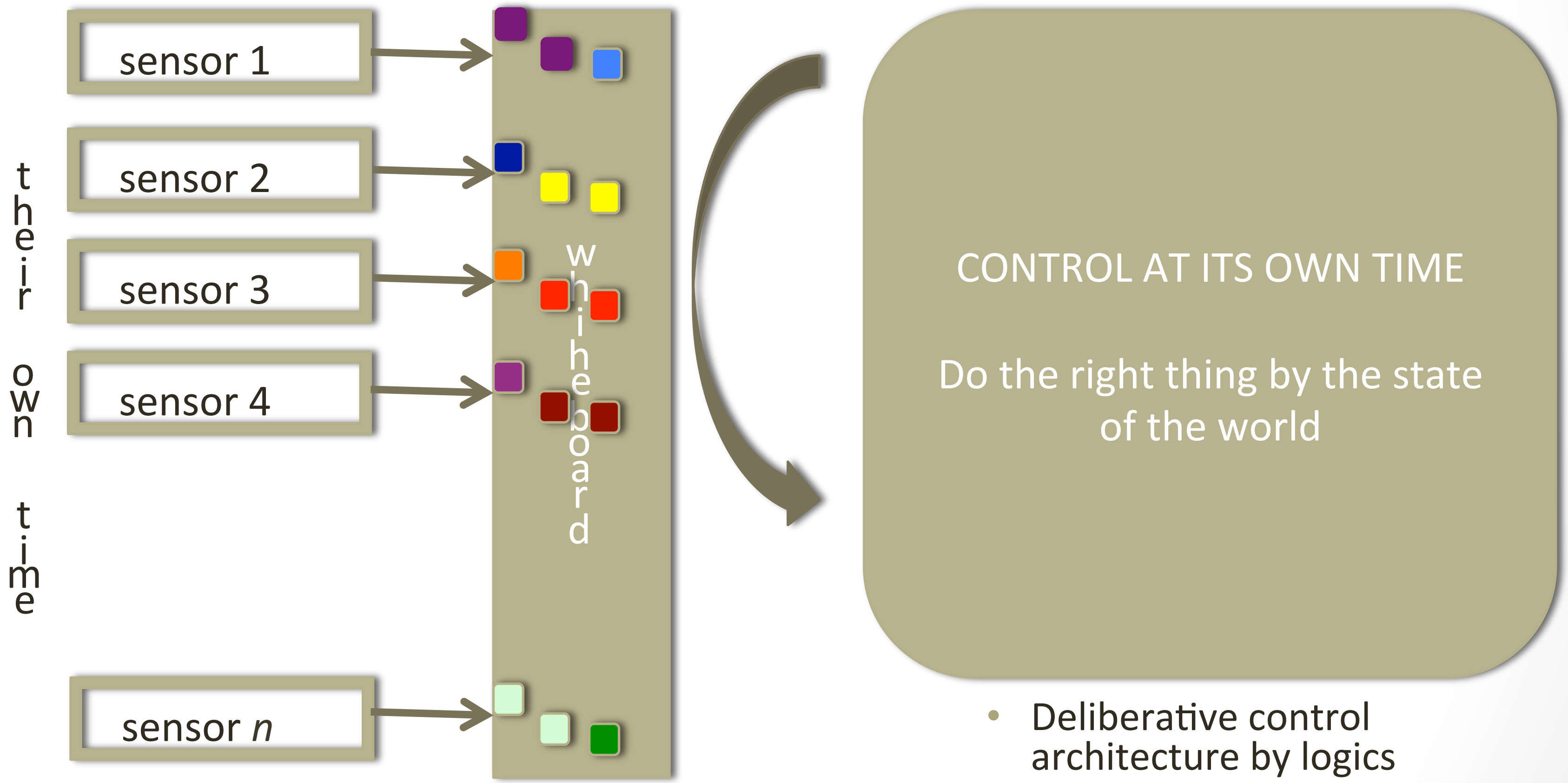
- Deliberative control architecture by logics
- Behavior-base control by vectors of FSMs

Conceptual cycle

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

under one CPU
rate for the sensors

time t_3



sensor space of the robot and memory is **FINITE**

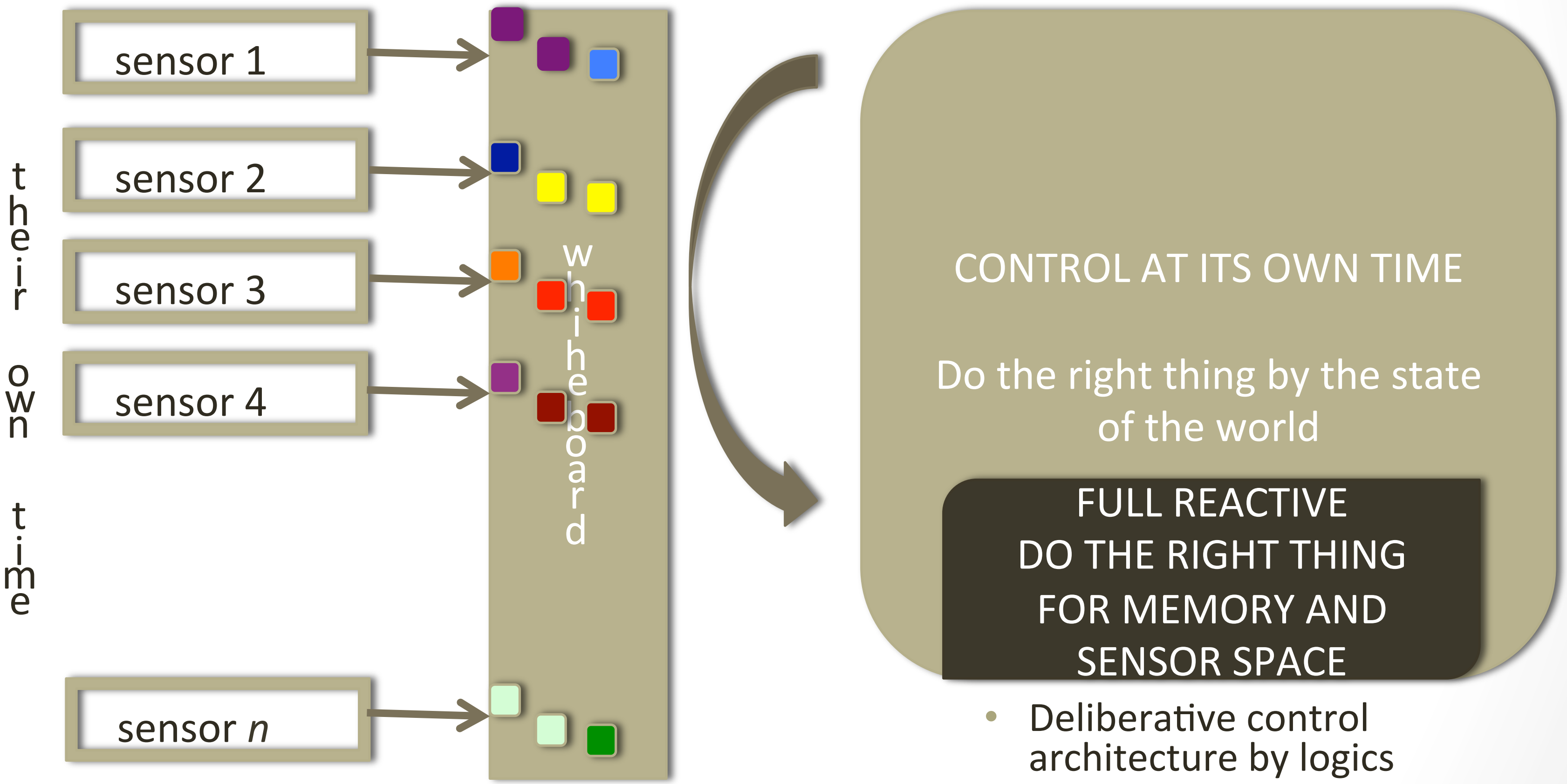
- Deliberative control architecture by logics
- Behavior-base control by vectors of FSMs

Conceptual cycle

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

under one CPU rate for the sensors

time t_3



sensory space of the robot and memory is **FINITE**

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

Conceptual cycle

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

under several CPU rate for the sensors