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® 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 **Subsumption Language**
- It was based on **finite state machines** (FSMs) augmented with timers (AFSMs)
- AFSMs were implemented in Lisp
State Diagram / Finite State Automaton

Motors forward

Light NOT visible

Motors halted For 0.1 sec

Light NOT visible

Light visible

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

- attack for a bit

- are the fans misbehaving?
- is the game over?
- I am injured?
- did the team lost possession?
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"}.

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

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

https://www.youtube.com/watch?v=gN6rlveCWNk&feature=youtu.be
Example 2: BatMan moves (reactive control on a Nao)

- https://www.youtube.com/watch?v=gN6rlveCWNk&feature=youtu.be
Example 3: Reactive control on ROS

- [https://www.youtube.com/watch?v=AJYA2hB4i9U&feature=youtu.be](https://www.youtube.com/watch?v=AJYA2hB4i9U&feature=youtu.be)
A turtle afraid of the walls
A turtle afraid of the walls.
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.
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](http://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?

(c) Vlad Estivill-Castro
23
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

sensor 1

sensor 2

sensor 3

sensor 4

sensor n

sensor space of the robot

CONTROL AT ITS OWN TIME

Do the right thing by the state of the world

• 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
  
  ```c
  #include "gugenericwhiteboardobject.h"
  #include "guwhiteboardtypelist_generated.h"
  ```

- Declare a handler
  ```c
  Ball_Belief_t wb_ball;
  ```

- Construct your objects (with the constructor of the OO-class)
  ```c
  Ball_Belief a_ball(50,30);
  ```

- Use the setter to actually post to the whiteboard
  ```c
  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`)
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

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>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.</td>
</tr>
<tr>
<td>R2</td>
<td>If the button is pushed while the oven is cooking, it will cause the oven to cook for an extra minute.</td>
</tr>
<tr>
<td>R3</td>
<td>Pushing the button when the door is open has no effect.</td>
</tr>
<tr>
<td>R4</td>
<td>Whenever the oven is cooking or the door is open, the light in the oven will be on.</td>
</tr>
<tr>
<td>R5</td>
<td>Opening the door stops the cooking. and stops the timer and does not clear the timer</td>
</tr>
<tr>
<td>R6</td>
<td>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.</td>
</tr>
<tr>
<td>R7</td>
<td>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.</td>
</tr>
</tbody>
</table>
One of the FSMs

% MicrowaveCook.d

name{MicrowaveCook}.

input{timeLeft}.
input{doorOpen}.

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

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

Execute in predefined schedule $t_i$ ringlets of FSM $M_i$
That is all folks!
Demo video

http://www.youtube.com/watch?v=t4uel1o67Xk&feature=relmfu
Simulator (embedded system: Industrial press)

http://www.youtube.com/watch?v=FpVUSrVI0c&feature=relmfu
On-line debugging and simulation

Real-Time Monitoring Tools
FSM Designer & Debugger

Real-Time Monitoring and Debugging of Finite-State Machines running live on the target System (e.g. the Nao Robot)
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

cfsm SMGameController Safety_BatteryMonitor
SMFallManager SMButtonChest SMButtonLeftFoot
SMButtonRightFoot SMRobotPosition SMSayIP SMShutdown

cfsm SMSoundStartStop SMSoundWhistle SMSoundDemo
SMGetUp SMPlayer SMBallFollower SMKicker SMHeadBallScanner
SMBallSeeker SMReadyFromInitial SMReadyFromAny
SMHeadBallTracker SMWalkScanner SMSeeker Collection
SMHeadScannerGoal SMHeadGoalTracker SMGetCloseToGoal
SMSet SMFindGoalOnSpot SMGoalieSaver SMFindGoal
SMLeapController SMTeleoperationController
SMTeleoperation SMTeleoperationHeadControl BatNaoMoves
StopMotionRecorder SMYouCannotCatchMe

cfsm gukalmanfilter

cfsm guUDPreceiver
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?”
IEICE 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

CONTROL AT ITS OWN TIME
Do the right thing by the state of the world

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

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

CONTROL AT ITS OWN TIME
Do the right thing by the state of the world

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

under one CPU rate for the sensors

time $t_2$
Conceptual cycle

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

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

CONTROL AT ITS OWN TIME

Do the right thing by the state of the world

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

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

CONTROL AT ITS OWN TIME
Do the right thing by the state of the world

FULL REACTIVE
DO THE RIGHT THING
FOR MEMORY AND SENSOR SPACE

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

sensor 1
sensor 2
sensor 3
sensor 4
sensor n

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

under one CPU rate for the sensors
time $t_3$
Conceptual cycle

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

under several CPU rate for the sensors