Procedural Reasoning on a Group of Adaptive Mobile Robots

Tim Niemueller, Stefan Schiffer, Martin Liebenberg
1 Motivation

2 Logistics League

3 Software Components

4 Lab Project
Goals

For You

- Get familiar with robot software development
- Learn about systems, tools, robot control
- In particular: in-depth contact with reasoning system

For us

- Sparring partner for our robot agent
- Data for comparison of systems

Have fun in the process!
Overview

Software Stack

- Fawkes Robot Software Framework
- OpenPRRS Reasoning System with Behavior Engine

Environment

- RoboCup Logistics League
- Gazebo 3D Simulation

Project Outline

- Warm-up time: get to know software system
- Phase 1: exploration
- Phase 2: production coordination
- Final: competition
Factory Production Logistics

Scenario

Flexible on-demand production in a smart factory setting

4th Industrial Revolution

- Cyber-Physical Systems
- From caged robots to *mobile* co-workers
- From mass to individual production

Requires flexible and automated task coordination
LLSF Challenges in 2014

- Large field, two teams active at the same time
- Robots must account for dynamic varying production schemes
Game Phases

Exploration (4 min)
- Machines show light code specific for machine type
- Robot must recognize and announced this type

Production (15 min)
- Orders are posted dynamically, e.g.
  “Deliver 1x P2 in time window [123, 206] to active gate”
- Robots must complete production chain leading to products
- Coordination is required for effective resource usage
- Machines may fail, other robots on the field
Orders

- Randomized schedule
- Announced by referee box

Machine Productions
Semi-autonomous Referee Box

Game Control
- Maintain game state/score

Communication
- Publish production plans

Data Recording
- Collect benchmarking data

Visualization and Instruction
- Referee/visitor monitoring

Machinery Control
- Instruct field machines
Carologistics RoboCup Team

- Fully integrated base system
- Based on the work of the world champion 2014
1 Motivation

2 Logistics League

3 Software Components

4 Lab Project
Software Stack Overview

Simulation

- Simulate environment of Logistics League close to reality
- Provide sensor data and execute actuation

⇒ Gazebo 3D Simulation Environment

Fawkes

- Robot Software Framework and middleware
- Provides building blocks and connects software components

Behavior System

- Behavior Engine: reactive execution middle layer
- CLIPS agent: existing local agent with coordination
- OpenPRS agent: new component you write to compete
Software Stack Overview

Simulation

- Simulate environment of Logistics League close to reality
- Provide sensor data and execute actuation

⇒ Gazebo 3D Simulation Environment

Fawkes

- Robot Software Framework and middleware
- Provides building blocks and connects software components

Behavior System

- Behavior Engine: reactive execution middle layer
- CLIPS agent: existing local agent with coordination
- OpenPRS agent: new component you write to compete
Gazebo-based Simulation

- Well-known Open Source 3D Simulation
- Easy to adapt, extend (sensors) and integrate
Multi-Level Abstraction

Higher level abstraction
- Bypass acquisition of sensor data
- Allows to run with fewer functional components

Lower level abstraction
- Generate sensor data from simulation
- Run functional processing components
Software Stack Overview

Simulation
- Simulate environment of Logistics League close to reality
- Provide sensor data and execute actuation
- Gazebo 3D Simulation Environment

Fawkes
- Robot Software Framework and middleware
- Provides building blocks and connects software components

Behavior System
- Behavior Engine: reactive execution middle layer
- CLIPS agent: existing local agent with coordination
- *OpenPRS agent*: new component you write to compete
Fawkes at a Glance

Fawkes

- Robot Software Framework providing basic building blocks
- **Component-based** architecture (plugins)
- Hybrid *BlackBoard/messaging* data exchange
- Multi-threaded and distributable
- Aspect-oriented framework feature access
- Structured and synchronized main loop

http://www.fawkesrobotics.org
Run-time Coordination

- Fawkes provides a main loop
- Threads *can* be hooked into the main loop
- Threads *can* also run concurrently
- Main loop is replaceable
- Threads for each hook are woken up concurrently
- Threads sleep during execution of other hooks
Run-time Coordination

- Fawkes provides a main loop
- Threads *can* be hooked into the main loop
- Threads *can* also run concurrently
- Main loop is replaceable
- Threads for each hook are woken up concurrently
- Threads sleep during execution of other hooks
Run-time Coordination

- Fawkes provides a main loop
- Threads *can* be hooked into the main loop
- Threads *can* also run concurrently
- Main loop is replaceable
- Threads for each hook are woken up concurrently
- Threads sleep during execution of other hooks
Run-time Coordination

- Fawkes provides a main loop
- Threads can be hooked into the main loop
- Threads can also run concurrently
- Main loop is replaceable
- Threads for each hook are woken up concurrently
- Threads sleep during execution of other hooks
Run-time Coordination

- Fawkes provides a main loop
- Threads *can* be hooked into the main loop
- Threads *can* also run concurrently
- Main loop is replaceable
- Threads for each hook are woken up concurrently
- Threads sleep during execution of other hooks
Run-time Coordination

- Fawkes provides a main loop
- Threads can be hooked into the main loop
- Threads can also run concurrently
- Main loop is replaceable
- Threads for each hook are woken up concurrently
- Threads sleep during execution of other hooks
Run-time Coordination

- Fawkes provides a main loop
- Threads *can* be hooked into the main loop
- Threads *can* also run concurrently
- Main loop is replaceable
- Threads for each hook are woken up concurrently
- Threads sleep during execution of other hooks
BlackBoard

- BlackBoard created by Fawkes main application
BlackBoard

- BlackBoard created by Fawkes main application
- Interface storage in the BlackBoard memory
- Interface definition via XML (fields/messages)
Fawkes Threads access the BlackBoard via these Interfaces
Fawkes Threads access the BlackBoard via these Interfaces
Fawkes Threads access the BlackBoard via these Interfaces

Remote applications can access BlackBoard via network

Transactions (read/write)
Fawkes Threads access the BlackBoard via these Interfaces

Remote applications can access BlackBoard via network

Transactions (read/write)

Only one writer at a time
Message passing as command channel

Messages can only be sent from reader to writer

Any number of messages in queue
BlackBoard

Fawkes Process

- Remote App
- BlackBoard
- Interface A
- Interface B
- Thread 1
- Thread 2

- Message passing as command channel
- Messages can only be sent from reader to writer
- Any number of messages in queue
Software Stack Overview

Simulation

- Simulate environment of Logistics League close to reality
- Provide sensor data and execute actuation

⇒ Gazebo 3D Simulation Environment

Fawkes

- Robot Software Framework and middleware
- Provides building blocks and connects software components

Behavior System

- Behavior Engine: reactive execution middle layer
- CLIPS agent: existing local agent with coordination
- OpenPRS agent: new component you write to compete
Behavioral Architecture

Agent

- Deliberation
  - Decision making/planning

- Reactive Behaviors
  - Skill execution/monitoring

Behavior Engine

Components

- Actuators/Sensors
- Data Processing

Localization

Motion

Vision

...
Behavioral Architecture

Agent

Behavior Engine

Localization  Motion

Vision  ...

Blackboard or ROS Topics

Niemueller, Schiffer, Liebenberg
Lua-based Behavior Engine

- Basic actions for reasoning layer
- Emphasize description over programming
- Allow programming where necessary
- Modeled using Hybrid State Machines
- Abstract low-level system
- Implemented for Fawkes and ROS
- Written in the Lua scripting language

Variable table

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5.2</td>
</tr>
<tr>
<td>y</td>
<td>4.3</td>
</tr>
<tr>
<td>error</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

www.fawkesrobotics.org/p/behavior-engine
CLIPS

(Rule-based) Production Systems

- First-Order Logic forward chaining systems
- Productions: condition-action rules
- Working memory holds facts ("short-term memory")
- Rules encode heuristic knowledge ("long-term memory")

C Language Integrated Production System – CLIPS

- Graph-based Rete-Algorithm
- Typically large rule bodies and relatively small number of facts
- Integrates nicely with C/C++
CLIPS – Terminology

**Facts**  Information in Working Memory

(machine (name M1) (mtype T2))

**Incomplete**

- Explicitly, e.g. (mtype UNKNOWN)

**Knowledge**

- Non-existence of facts
CLIPS – Terminology

Facts Information in Working Memory

(machine (name M1) (mtype T2))

Incomplete

Explicitly, e.g. (mtype UNKNOWN)

Knowledge

Non-existence of facts

Functions Procedural Knowledge

(deffunction timeout (?now ?tm ?tout)
  (return (> (diff-sec ?now ?tm) ?tout)))
CLIPS – Terminology

Facts  Information in Working Memory

(machine (name M1) (mtype T2))

Incomplete  ■ Explicitly, e.g. (mtype UNKNOWN)

Knowledge  ■ Non-existence of facts

Functions  Procedural Knowledge

(deffunction timeout (?now ?tm ?tout)
  (return (> (diff-sec ?now ?tm) ?tout)))

Rules  Heuristic Knowledge

(defrule rule-name
  ?m <- (machine (name M1))
  =>
  (modify ?m (mtype T2))
)
CLIPS – Terminology

Facts  Information in Working Memory

(machine (name M1) (mtype T2))

Incomplete  Explicitly, e.g. (mtype UNKNOWN)

Knowledge  Non-existence of facts

Functions  Procedural Knowledge

(deffunction timeout (?now ?tm ?tout)
  (return (> (diff-sec ?now ?tm) ?tout)))

Rules  Heuristic Knowledge

(defrule rule-name
  ?m <- (machine (name M1)) =>
  (modify ?m (mtype T2))
)

Agenda  Currently active rules (conflict resolution)
World Model Update

```lisp
(defrule wm-determine-t23-s0-or-s1-now-s2
    (declare (salience ?*PRIORITY_WM*))
    ?w <- (wm-eval (machine ?name) (junk ?junk)
                (was-holding S0|S1) (now-holding S2))
    ?m <- (machine (name ?name) (mtype T2_3)
                    (loaded-with $?lw) (productions ?p))
    =>
    (retract ?w)
    (modify ?m (mtype T2) (loaded-with)
            (junk (+ ?junk (length$ ?lw)))
            (productions (+ ?p 1)))
)
```
World Model Update

(defrule wm-determine-t23-s0-or-s1-now-s2
  (declare (salience ?*PRIORITY_WM*))
  ?w <- (wm-eval (machine ?name) (junk ?junk)
    (was-holding S0|S1) (now-holding S2))
  ?m <- (machine (name ?name) (mtype T2_3)
    (loaded-with $?lw) (productions ?p))
  =>
    (retract ?w)
    (modify ?m (mtype T2) (loaded-with)
      (junk (+ ?junk (length$ ?lw)))
      (productions (+ ?p 1)))
)

- Basic rule syntax: antecedent ⇒ consequent
World Model Update

(defrule wm-determine-t23-s0-or-s1-now-s2
  (declare (salience ?*PRIORITY_WM*))
  ?w <- (wm-eval (machine ?name) (junk ?junk)
     (was-holding S0|S1) (now-holding S2))
  ?m <- (machine (name ?name) (mtype T2_3)
     (loaded-with $?lw) (productions ?p))
  =>
  (retract ?w)
  (modify ?m (mtype T2) (loaded-with)
    (junk (+ ?junk (length$ ?lw)))
    (productions (+ ?p 1)))
)

- Basic rule syntax: antecedent $\Rightarrow$ consequent
- New information becomes available
World Model Update

(defrule wm-determine-t23-s0-or-s1-now-s2
  (declare (salience ?*PRIORITY_WM*))
  ?w <- (wm-eval (machine ?name) (junk ?junk)
      (was-holding S0|S1) (now-holding S2))
  ?m <- (machine (name ?name) (mtype T2_3)
      (loaded-with $?lw) (productions ?p))
  =>
    (retract ?w)
    (modify ?m (mtype T2) (loaded-with)
      (junk (+ ?junk (length$ $?lw)))
      (productions (+ ?p 1)))
)

- Basic rule syntax: antecedent $\Rightarrow$ consequent
- New information becomes available
- Infer: completed production at T2/3 with S2 $\Rightarrow$ Type T2
OpenPRS

Procedural Reasoning System
- Implementation of Belief-Desire-Intention model
- Based in beliefs determine goals and achieve them
- Supports open and closed world reasoning

Building Blocks
- **Database** Store facts about the world
- **OPs** Plan/procedure library to achieve goals
- **Tasks Graph** Graph of currently active OPs

Program Flow
- Facts are collected in the database
- Determine activated OPs (based on DB)
- Execute no, one, or more OPs
OpenPRS – Database

Predicates and Facts (Manual Sec. 5.5ff)
- Name for specific piece of information
- Variety: regular, basic event, closed world, functional fact etc.
- Facts are instances of predicates in the database

Database (Manual Sec. 5.2ff)
- The database is a collection of known facts
- Can be queried for information and decisions
- Is used to determine activation of OPs
- $(bb\text{-}data\ldots)$ facts for blackboard data
OPs (Manual Chapter 4)

- Procedure that can be executed
- Invocation pattern activation
- Execution graph
- Graphical and textual definition

Actions (Manual Sec 4.3, 7.7)

- Basic level actions
- Mostly for us: execution of skills (skill-call ...)
Lab Course: Procedural Reasoning on a Group of Adaptive Mobile Robots

Text OP Example

```
(defop llsf2014-init
  :invocation (agent-init)
  :context ()
  :setting (& (setting "idle-looptime-sec" $ilt_sec)
             (setting "idle-looptime-usec" $ilt_usec))
  :body ( ! (pb-setup))
          ;(! (bb-open "SkillerInterface" "Skiller" BB-READ))
          (! (time-set-idle-looptime $ilt_sec $ilt_usec))
          (! (navgraph-load))
          (! (say "Agent loaded and ready to run"))
          (! (print "LLSF2014 agent successfully initialized"))
)
  :effects ((~> (agent-init)) (=> (run)))
)
```
<table>
<thead>
<tr>
<th></th>
<th>Motivation</th>
<th>Logistics League</th>
<th>Software Components</th>
<th>Lab Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Motivation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Logistics League</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Software Components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Lab Project</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Project Outline

**Goal**

Develop and document an executive using OpenPRS in Fawkes to play the Logistics League in the Gazebo simulation.

**Milestones**

- **October**: Get acquainted to the overall system, call skills
- **November**: Exploration phase running as an OpenPRS agent
- **December**: Basic production phase running – single robot
- **January**: Production with (some) coordination
- **February**: Conclusion and Tournament
Project Notes

Documentation

- *Documentation* is crucial – and grading relevant!
- Includes documenting code and OPs
- Use wiki to describe how to use the system and your code

(Human) Communication

- Be competitive – but in particular cooperative!
- Talk and discuss your progress and ideas, help others
- Use the mailing list so all can benefit – even us
- Give credit where credit is due!
- Ask advisers – but try hard and prepare first

Side Projects

- You may need to hack skills and C++ code
- There may be a bug or two...
Rescission Policy

Up to three weeks from now on you are allowed to recede from the seminar without any consequences. A later rescission will be graded as a failed attempt!

Accounts

- Group accounts for lab machines
- Register in Trac if you have not done so, yet
- Send your SSH public key (RSA, 2048 bits min)

Working Environment

- You can use our machines available as long as there is someone to lock the door
- You can use your own machine, but we provide support only for Fedora 20 – and only minimal
## Conclusion and Questions

Develop and document an executive using OpenPRS in Fawkes to play the Logistics League in Gazebo simulation.

- Logistics League Simulation based on Gazebo
- Executive in OpenPRS with multi-robot coordination
- Document thoroughly and precisely
- Have fun and join efforts

https://trac.fawkesrobotics.org/wiki/Projects/LabPRoGrAMR2014