ASEBA
Open-Source Low-Level Robot Programming

Stéphane Magnenat ¹
stephane at magnenat.net

Philippe Rétornaz ²

¹Autonomous Systems Lab
ETH Zürich

²Mobots group - Laboratory of robotics Systems
EPFL

February 5, 2012
Outline

Motivation

Current Use

Technical Description

Performances

Wrap-up
Outline

Motivation

Current Use

Technical Description

Performances

Wrap-up
Motivation: Multi-Microcontrollers Robots

Modern integrated mobile robots have

- lots of degrees of freedom
- many and various sensors

Their computing infrastructure consists of

- a main processor running Linux (ex. Gumstix)
- multiple microcontrollers
- a common communication bus (ex. I2C or CAN)
Motivation: Usual Approach is Centralized Polling...

- **main processor**
  - continuously
  - read sensors
  - process data
  - set actuators

- **microcontrollers**
  - interface to physical devices

- **bus**
  - I2C
Motivation: ...but Distributed Events are Better!

- main processor
  - react to events
  - send events

- microcontrollers
  - interface to physical devices
  - preprocess data
  - send events
  - react to events

- bus
  - CAN
Motivation: An Example of Distributed Events

- proximity sensors
- left motor
- right motor

(time)
Motivation: An Example of Distributed Events

proximity sensors

left motor
right motor

no obstacle

time
Motivation: An Example of Distributed Events
Motivation: An Example of Distributed Events

- Time
  - Obstacle detected
  - No obstacle

- Proximity sensors
  - Left motor
  - Right motor

- No obstacle
Motivation: How to Implement Distributed Events?

- Contrary to centralized polling, with events the emission policy must be distributed as well.
- Microcontrollers must take decisions about what event to send when, and how to link incoming events to actuators.
- Therefore they must be programmable, but flashing is slow.
- A virtual machine is the solution...
- And so Aseba was born!
Motivation: How to Implement Distributed Events?

- Contrary to centralized polling, with events the emission policy must be distributed as well.
- Microcontrollers must take decisions about what event to send when, and how to link incoming events to actuators.
- Therefore they must be programmable, but flashing is slow.
- A virtual machine is the solution...
- And so Aseba was born!

- Aseba stands for Actuator and Sensor Event-Based Architecture.
- Aseba puts virtual machines inside microcontrollers; enabling their programming through a user-friendly language and IDE.
Outline

Motivation

Current Use

Technical Description

Performances

Wrap-up
Swarm Robotics: hand-bot

6 Aseba VM, real-time unwind control, grasping, and state machines implemented in Aseba
Swarm Robotics: hand-bot

hand-bot video at
http://www.youtube.com/watch?v=92bLgE6D02g

In this video, the hand-bot climbs a shelf and retrieves a book solely using Aseba.
5 Aseba VM, state machines coded in Aseba localisation, mapping, planning running on Linux ARM
Autonomous Construction: Lonelybuilder

Lonelybuilder video at
http://www.youtube.com/watch?v=h865RHbT9Ms

In this video, Lonelybuilder constructs a tower by manipulating cubes. The manipulation state machines are implemented with Aseba.
Brick for Building Robot: Smartrob

- 2 motor and 8 servo drivers
- 8 infrared-sensor drivers
- additional I/O and A/D
- CAN bus, stackable

- single 3 to 25 V input
- shipped with Aseba
- ROS and D-Bus integration
- soon available for buying
Educational Robotics: Thymio II

Full-featured open-hardware programmable mobile robot for 100 CHF ($≈ 80 €), see http://aseba.wikidot.com/en:thymio
Educational Robotics: Thymio II
Educational Robotics: Thymio II

- Li-Po battery level
- Speaker
- Microphone
- Infrared remote control receiver
- 3 axis accelerometer
- 5 proximity sensors (obstacle detection)
- 2 ground sensors (line following)
- 39 LED to visualize sensors and interaction
- Reset button
- 5 capacitive touch buttons with activity display and ON-OFF function
- Pencil support
- USB connection (programming and recharging)
- Memory card slot
- Hook for trailer
- 2 proximity sensors
- Mechanic fixation
- 2 wheels with speed control
- Temperature sensor
Educational Robotics: Thymio II

Enables children to discover programming with a mobile robot.
Simulated Robotics: Challenge
Outline

Motivation

Current Use

Technical Description

Performances

Wrap-up
Technical Overview

Aseba:

- allows fast prototyping of the behaviour of microcontrollers connected through a network,
- provides an IDE for edition, debugging and inspection of the values of variables (including sensor and actuators),
- compiles scripts into bytecode,
- executes bytecode on microcontrollers in a virtual machine,
- safe execution,
- dynamic enumeration of microcontrollers’ variables, native functions and events,
- dynamic reprogramming of the microcontrollers,
- asynchronous code execution upon events,
- open source, LGPL.
Software Architecture

desktop computer

IDE

central embedded computer running Linux

software switch

ROS or D-Bus or TCP

high-level control programs

robot

optional

microcontroller

CANTCP

microcontroller

microcontroller

microcontroller

microcontroller
Microcontrollers

- shared communication bus
- microcontroller
  - virtual machine
    - application specific program
  - communication
  - native functions
  - low-level control
- sensors
- actuators
Virtual Machine

- targets 16-bit microcontrollers and better,
- stack based, 16-bit integers,
- executes bytecode
  (4-bit opcode, 12-bit payload + optional trailing 16-bit words)
- $\approx 1000$ lines of C, including debugging logic,
- RAM: 22 bytes + user defined amount of bytecode, variable, stack, and breakpoints.
- flash: 7.5 kB flash (dsPIC30, e-puck),
- no external library requirement, excepted the implementation of bus communication.
Language

Simple imperative scripting language, octave-like syntax.

- blocks of code executed upon events,
- 16-bit integer variables and arrays,
- common mathematical expressions and arrays access,
- if and when conditionals,
- while and for loops,
- native functions for complex processing,
- subroutines.
Live Demonstration using Thymio II
Outline

Motivation

Current Use

Technical Description

Performances

Wrap-up
Evaluation conducted using the marXbot base.
Aseba vs Polling: Bus Bandwidth

- Data transmitted for 1 minute:
  - 100 B
  - 1 kB
  - 10 kB
  - 100 kB
  - 1 MB

- Event-based polling

- Empty, Boxes, Walls
  - Empty
  - Boxes
  - Walls
  - 10 Hz
  - 25 Hz
  - 67 Hz
Aseba vs Polling: Bandwidth Use w.r.t. Location

100 bytes per second
Aseba vs Polling: Latency

initially

once stopped

polling 10 Hz

polling 25 Hz

polling 67 Hz

event–based
Aseba vs Native

On a 40 MHz dsPIC:

- Ratio of about 70 dsPIC instructions for 1 Aseba instruction
- Rate of 600’000 instructions per second
- Event round trip lasts 25 $\mu s$ on idle VM
- Native functions in DSP assembly
- Native functions use dsPIC’s DSP, event round trip for mean of 100 values is 60 $\mu s$, faster than bare C code
Lessons Learnt on Performance

- Events save bandwidth compared to polling.
- Events allow for lower latencies than polling.
- Virtual machines are suitable for embedded, provided the critical path is optimised.
Outline

Motivation

Current Use

Technical Description

Performances

Wrap-up
Future Directions

Improve IDE

- store program as abstract syntax tree, textual surface form for editor
- contextualised help/errors
- intelligent completion
- statistics and machine learning for common mistakes, automatic tutoring

Lower entry curve

- improve tutorial
- robotics course
- more translations

More platform support

- new firmware for Thymio II
- affordable service robot based on Smartrob
- add your favourite platform!

On the long run

- no more arithmetic/logic operation in the VM, use only native functions
- complete type system
- more proving in the compiler
- JIT on some platforms
- use standard language?
Take-Home Message

The combination of VM and custom-tailored IDE enables efficient embedded development, in particular for robotic applications.

Aseba is a mature and robust implementation of this idea. Yet there is much room for improvement and innovative ideas.

You are welcome to join us in this endeavour!
Thank you

Thank you for your attention, your questions are welcome.

The Aseba community awaits you: http://aseba.wikidot.com

Thanks to: Michael Bonani, Florian Vaussard, Fanny Riedo, Valentin Longchamp, Basilio Noris, Sandra Moser and Francesco Mondada
Program Memory Layout

<table>
<thead>
<tr>
<th>addresses (in 16-bit words)</th>
<th>content</th>
</tr>
</thead>
<tbody>
<tr>
<td>bytecodeSize −1</td>
<td>unused bytecode</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>bytecode for last managed event</td>
</tr>
<tr>
<td>evLastAddr</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>bytecode for first managed event</td>
</tr>
<tr>
<td>ev0Addr</td>
<td></td>
</tr>
<tr>
<td>evVectSize −1</td>
<td>evLastAddr</td>
</tr>
<tr>
<td>evVectSize −2</td>
<td>evLastId</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x0002</td>
<td>ev0Addr</td>
</tr>
<tr>
<td>0x0001</td>
<td>ev0Id</td>
</tr>
<tr>
<td>0x0000</td>
<td>evVectSize</td>
</tr>
<tr>
<td>addresses (in 16-bit words)</td>
<td>content</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>variablesSize −1</td>
<td>temporary variables to pass constants to native calls</td>
</tr>
<tr>
<td>...</td>
<td>unused variables</td>
</tr>
<tr>
<td>...</td>
<td>user-defined variables</td>
</tr>
<tr>
<td>exportedVarsLength</td>
<td>exported variables</td>
</tr>
<tr>
<td>...</td>
<td>0x0000</td>
</tr>
</tbody>
</table>
# Types of Bytecodes - 1/2

<table>
<thead>
<tr>
<th>name</th>
<th>w.c.</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>1</td>
<td>stop execution</td>
</tr>
<tr>
<td>small immediate</td>
<td>1</td>
<td>push a constant onto the stack</td>
</tr>
<tr>
<td>large immediate</td>
<td>2</td>
<td>push a constant onto the stack</td>
</tr>
<tr>
<td>load</td>
<td>1</td>
<td>push data from memory onto the stack</td>
</tr>
<tr>
<td>store</td>
<td>1</td>
<td>pop data from the stack into the memory</td>
</tr>
<tr>
<td>load indirect</td>
<td>2</td>
<td>push data from memory onto the stack using an offset from the stack</td>
</tr>
<tr>
<td>store indirect</td>
<td>2</td>
<td>pop data from the stack into the memory using an offset from the stack</td>
</tr>
<tr>
<td>unary arithmetic</td>
<td>1</td>
<td>unary arithmetic operation on the stack</td>
</tr>
<tr>
<td>binary arithmetic</td>
<td>1</td>
<td>binary arithmetic operation on the stack</td>
</tr>
<tr>
<td>name</td>
<td>w.c.</td>
<td>function</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jump</td>
<td>1</td>
<td>jump to another execution address</td>
</tr>
<tr>
<td>conditional branch</td>
<td>2</td>
<td>check a condition on the stack and jump depending on the result</td>
</tr>
<tr>
<td>emit</td>
<td>3</td>
<td>send an event</td>
</tr>
<tr>
<td>native call</td>
<td>1</td>
<td>call a native function</td>
</tr>
<tr>
<td>sub call</td>
<td>1</td>
<td>jump into a subroutine, store return address on the stack</td>
</tr>
<tr>
<td>sub ret</td>
<td>1</td>
<td>return from a subroutine, using return address from the stack</td>
</tr>
</tbody>
</table>