ASEBA, an event-based middleware for distributed robot control

Stéphane Magnenat*, Valentin Longchamp†, and Francesco Mondada†
LSRO - École Polytechnique Fédérale de Lausanne (EPFL)
*stephane at magnenat dot net, †surname dot name at epfl dot ch

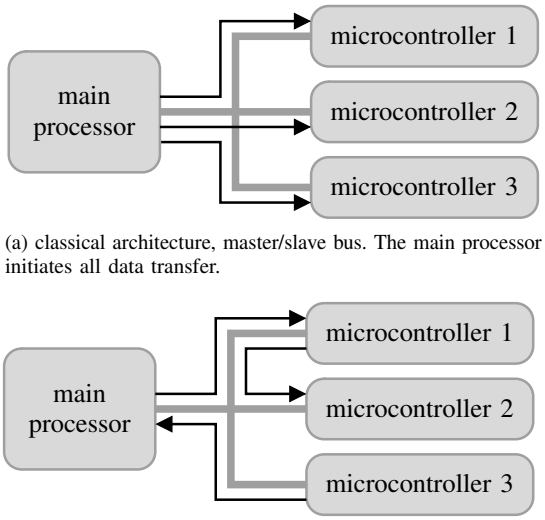
***Abstract*— Computing infrastructures of mobile robots have grown in complexity in the last decades; They have evolved from single processor systems to networks of microcontrollers communicating through a shared bus. This has induced additional architectural constraints that do not fit well with the traditional polling-based sensors and actuators control. To address this issue, we have developed ASEBA, an event-based middleware that allows distributed control and efficient resources exploitation of multi-microcontrollers robots. ASEBA provides hardware modularity, better efficiency, and improved scalability by embedding a lightweight virtual machine in each microcontroller and providing an IDE to develop and debug the whole robot reactive control from a single place.**

I. INTRODUCTION

Computing infrastructures of mobile robots have grown in complexity in the last decades; Robots evolved from single processor platforms, directly connected to sensors and actuators [7] to distributed systems [11]. Mobile robots now contain several processors that communicate through a shared bus: Peripheral microcontrollers, close to sensors and actuators, read sensors and set actuators in real-time while a main processor attends to cpu-intensive tasks such as vision and higher-level control [12].

Traditionally, robot control code reads sensors, process the data, and sets actuators at regular intervals. This functions well for single-processor robots, when the control code has direct access to the hardware. Recently, this control structure has been adapted to multi-processors robots. In most systems, the control loop is running on the main processor: it sequentially polls the sensors, process the data, and sets the actuators. This sequential mode of operation is not well suited for multi-processors robots:

- A *synchronous* control loop is not efficient when several processors are connected through a bus in a network. The bus is under load during short periods of time, when the main processor reads the sensors or sets the actuators, but rests idle otherwise. Also, the main processor has to read sensors at each control loop iteration, whether or not the state of the world has changed. This consumes more bus bandwidth than strictly required to implement the robot behaviour.
- The topology of the network must be predefined so that the main processor reads and sets the data from the correct microcontroller. This reduces flexibility: For example, a better sensor cannot be transparently added.



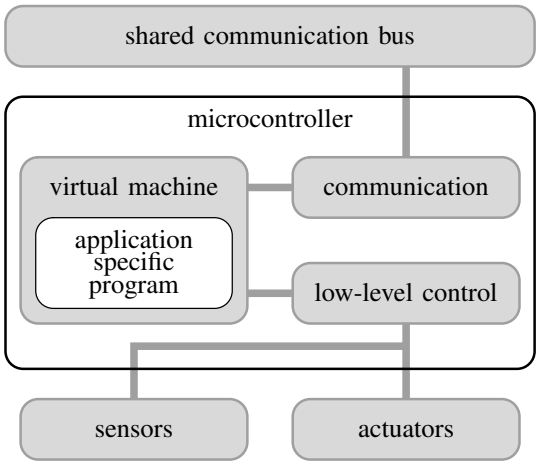
(a) classical architecture, master/slave bus. The main processor initiates all data transfer.

(b) event-based architecture, multi-master bus. Any node can initiate a data transfer.

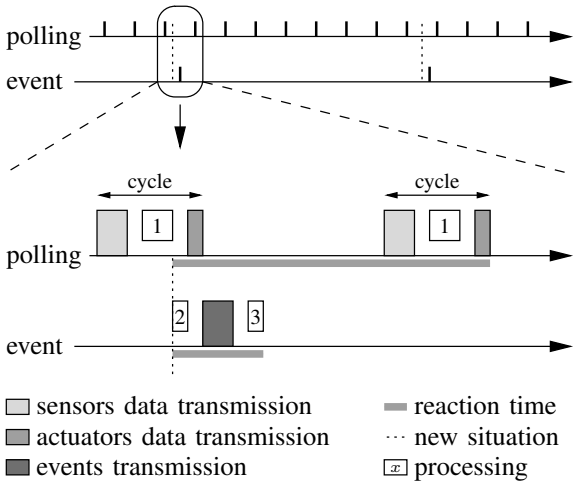
1: Comparison of dataflow between classical and event-based architectures.

- Modern sensors, such as cameras or laser scanners, typically produce vectorial data, such as pictures or arrays of distances. The amount of data generated is orders of magnitudes larger than the more traditional sensors such as bumpers or infraread sensors; Yet the switch to multi-processors robots has decreased the bandwidth between the hardware and the control code. Communication buses used in mobile robots provide a bandwidth several orders of magnitude smaller than direct connections to microcontrollers. This severely limits the performances of the robots.

We experienced the aforementioned limitations while developing and using the s-bot mobile robot [12]. To improve the efficiency and the scalability of low level control, we propose a new *event-based architecture* called ASEBA. It runs on robots with several microcontrollers connected through a shared communication bus. On that bus, asynchronous messages sent by any microcontroller, called *events*, replace periodic sensors readings and actuators commands from the main processor. Events consist of an identifier and optional payload data. Any microcontroller can communicate with any microcontroller asynchronously by sending events (Figure 1). This requires the communication bus to be multi-master. However this feature is relatively common; for example the CAN [14] bus we use



2: A microcontroller in an ASEBA network.



3: A time oriented comparison of polling versus events-based systems. 1) main processor processing all sensors, 2) microcontroller processing its local sensors, 3) microcontroller processing incoming event and setting actuator. Because processing is done locally and only useful data is transmitted and the transfer occurs asynchronously, bus load and reaction time are both reduced when using events.

in the robots we are currently developing provides this feature.

The microcontrollers send outgoing events and react to incoming events. The event emission policy depends on external or internal conditions, including the result of local pre-processing of sensors data. For instance, a microcontroller driving a distance scanner can send an event when the mean scanned distance is lower than a given threshold. Inter-microcontroller communication is possible: the main processor does not need to react to all events. For example, the motor microcontroller can slightly change trajectory in order to avoid a small obstacle. In such case, the main processor needs not be interrupted.

The decisions of sending different outgoing events change with robot behaviours: the events exchanged by the microcontrollers will not be the same in a robot engaged in obstacle avoidance than in a robot following walls. Therefore, the event control code has to be easily modifiable by the robot user and not only by the robot developer. In ASEBA, this flexibility is implemented by splitting the microcontroller code in two parts (Figure 2):

- Sensors readings (for example generating the timings for an infraread sensor), actuators low level control (for example the PID controller of a motor), and the communication layer are implemented in native code on the microcontrollers. This allows real-time, interrupt driven handling of low level resources.
- Application specific programs that control the events emission and reception policy run in a *virtual machine* on the microcontrollers. They are compiled out of a simple scripting language called AESL (ASEBA Event Scripting Language). It provides the necessary flexibility to allow non-specialist users to develop event-based behaviours. AESL scripts do not perform complex computations, they depend upon native functions to do so. The overhead of the virtual machine is thus not a problem for modern microcontrollers (See footprints in Subsection II-D)

In this paper, we present ASEBA in the perspective of a comparison with others robots middlewares. We describe the specificities of ASEBA in Section II and provide a comparison with the state of the art in Section III.

II. ASEBA

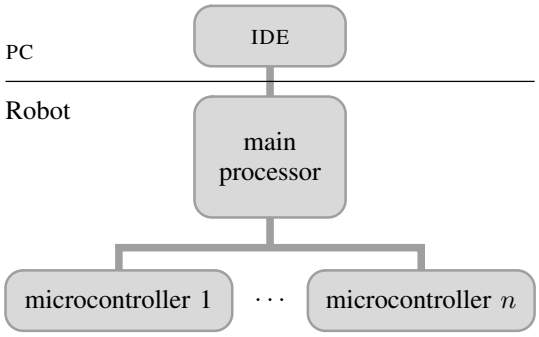
ASEBA improves the modularity and efficiency of robot middleware by distributing some of the software tasks to all the microcontrollers and communicating only the relevant data to the main processor. In ASEBA, microcontrollers are not restricted to reading sensors or driving actuators, but are also responsible for sensor data pre-processing and some low level control. This is possible because modern microcontrollers are powerful and often provide optimized DSP instructions for signal processing. As they are directly connected to the sensors and actuators, they can efficiently use those instructions to only transmit relevant data to the main processor.

Each microcontroller with its sensors and actuators running ASEBA is a module emitting events and reacting to the received events. This distribution of tasks improves performances but also increases complexity. ASEBA copes with that complexity by providing:

- events as the data abstraction mechanism,
- a simple and easy to learn language,
- an Integrated Development Environment (IDE) to develop and debug the whole robot reactive control from a single place.

A. Event-based architecture

An event-based architecture only transmits data when relevant information is sensed (Figure 3). This reduces the load on the bus, because less data is transmitted. For



4: Block diagram of an ASEBA network.

example, a bumper can send events only when it touches something that was not sensed previously. Furthermore, when compared to polling with a fixed frequency, asynchronous events also decrease the latency which improves the robot reaction time. Figure 1 shows how dataflows differ between classical and event-based architectures. Event-based architectures reduce the computation complexity at higher level by distributing the sensor pre-processing on the microcontrollers that have direct accesses to the sensors. Events in a multi-processor system can be compared to interrupts in a microcontroller.

To support an event-based architecture, the communication bus must allow any node to transmit data at any time. In the robots we are currently developing, we use the CAN [14] bus that natively provides this feature. On the main processor, a software switch extends this bus over TCP/IP. Programs such as the IDE can connect to this switch to exchange events with the rest of the robot (Figure 4).

The development of a behaviour in an event based architecture begins by defining all required events. Those events abstract the details of the implementations on each microcontroller. Event-driven programs are easy to derive from specification documents like use case diagrams and state machines. This close mapping improves the maintainability of the robot software.

B. AESL *Language*

In ASEBA, event emission and reception policy is described in a simple language, AESL (ASEBA Event Scripting Language). Syntactically, AESL resembles matlab scripts; semantically, it is a simple imperative programming language with a single basic type (16 bits signed integers) and arrays. It has some specific features:

- *Events*. Scripts can send events. Events can trigger the execution of scripts. As events can take any script variable as argument, they can be considered as function calls. Such calls can be local or remote, but they are asynchronous: the event-related script will be executed once the current script is completed. It is thus not suitable for recursive calls.
- *When conditional*. In addition to the usual *if* conditional, AESL provides the *when* conditional that is true when the actual evaluation of the condition is true and the last was false. For example, this can be used to execute some script at the moment a bumper detects an obstacle.
- *Native functions*. Native functions are written in C or assembler and are well suited to do computing-intensive tasks such as signal processing.

```
# declare and initialize variables
var bumperLimit = 10
var bumperFilter[4] = 0, 0, 0, 0

# to execute periodically
ontimer:
# call native function meanFilter
call meanFilter(bumper, bumperFilter)
# when conditional
when bumper > bumperLimit do
    # send event with variable bumper
    emit ObstacleDetected bumper
end

# to execute on event SetLimit
onevent SetLimit:
# args contains the arguments of events
bumperLimit = args[0]
```

Listing 1: Example of AESL script. Lines beginning with # are comments.

An example of AESL script illustrating those features is given in Listing 1.

C. Development Environment

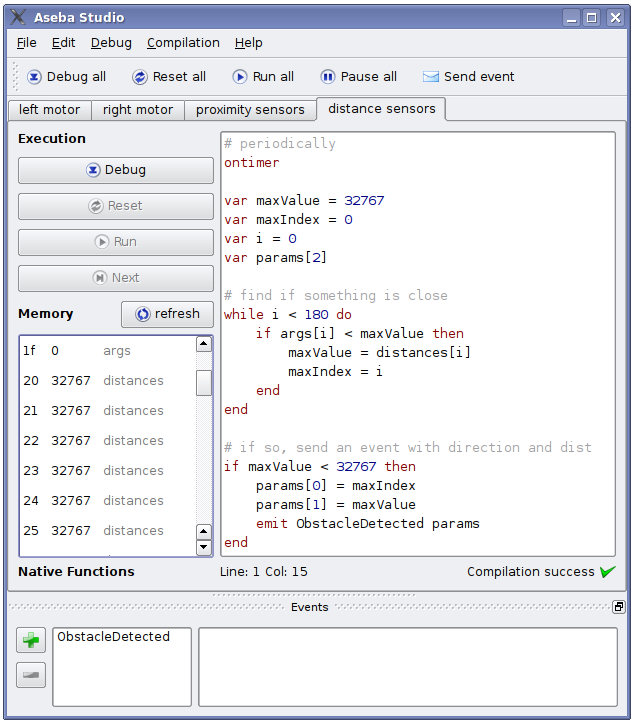
An event-based framework is not a common way of implementing robots control. To facilitate the use of this asynchronous and parallel paradigm, and to ensure an efficient development process when using ASEBA, we provide an Integrated Development Environment (IDE) with the following features:

- a script editor with syntax highlighting,
- a compiler that recompiles script while typing and visually marks errors inside the editor,
- a distributed debugger; the developer can visually set breakpoints and control the execution state of each microcontroller from inside the IDE while each microcontroller runs one separate debugger core.

This IDE (Figure 5) allows seamless development and debugging of the whole network of microcontrollers from a single place.

D. Virtual machines

AESL scripts are compiled into bytecode by the IDE or, if the robot runs autonomously, by the main processor. The bytecode is then loaded to the microcontrollers through the bus. On the microcontrollers, it runs in lightweight virtual machines, with the following characteristics:



5: Screenshot of ASEBA IDE.

- Stack based.
- Less than 1000 lines of C, including debugging logic.
- RAM footprint: 16 bytes + user defined amount of bytecode, variable and stack size,
- Flash footprint: less than 500 DsPIC instructions (1.5 KB flash)
- No external library requirement, excepted the implementation of bus communication.

Because virtual machines embed the debugging logic, the user has full control of the remote execution directly from the IDE. If a script produces an exception, such as a division by zero, the error is broadcasted on the bus and the state of the faulty virtual machine is reset. Bytecode is not deleted and the virtual machine is ready to continue operation.

III. Related work

A. Comparison criteria

Table I compares ASEBA with other robots control architectures, using the following criteria:

- *Modularity*: Is the architecture built of modules:
  - full: each sensor and actuator, or each microcontroller, is an exchangeable module. The modularity is present both at the software and at the hardware level.
  - limited: modules exist in the system, but they are limited to main processors and do not extend to the microcontrollers. In this class, modularity is essentially software.
  - none: all control is done at a single place.

	Modularity	Event-based	Footprint	IDE	Specification	Implementation
CLARAty [13]	none	n/a	med.	ext.	open	open
Player [5]	none	n/a	large	ext.	open	open
Urbi [2]	none	n/a	med.	no	open	open
OrIN [10]	limited	yes	large	ext.	open	open
Miro [15]	limited	yes	large	ext.	open	open
RT Middlew. [1]	limited	yes	large	ext.	open	open
Orocos [3]	limited	yes	medium	ext.	open	open
Orca [8]	limited	yes	large	ext.	open	open
Microsoft RS [9]	limited	yes	large	yes	closed	closed
OpenR [4]	yes	no	small	n/a	open	closed
ASEBA	yes	yes	small	yes	open	open

I: Comparison of the features of several robots control middlewares. A detailed description of the comparison criteria is given in Subsection III-A.

- *Event-based*: Can modules exchange events asynchronously.
- *Footprint*: How much ressource does the implementation requires:
  - large: an operating system with TCP/IP capabilities, memory in the order of megabytes.
  - medium (med.): a small processor board, no operating system, memory in the order of hundreds of kilobytes.
  - small: a microcontroller, memory in the order of kilobytes.
- IDE : Does the architecture comes with an Integrated Development Environment. Ext. means that external ones can work with the architecture.
- *Specification*: Is the specification open or closed.
- *Implementation*: Is the implementation open or closed.

We only consider architectures that abstract differents sensors and actuators to present them under a unified interface.

B. Classes of architectures

From our comparison (Table I), we classify robotics middlewares in three main groups. Our analysis consider the use of those middlewares in small mobile robots, in the order of one kilogramme and one cubic decimeter. In such robots, the computational power is limited and standard PC laptops cannot be used.

The first group consists of non modular architectures. They are essentially software libraries, but not components, that wrap accesses to physical devices. We review three noteworthy frameworks in this category:

- CLARAty [13] is a library to access robot sensors and actuators and provides additional features useful for robot control, such as a vision library.
- Player [5] is a robot device server that provides network transparent robot control. It provides a sensors

and actuators abstraction that allows client softwares to control the robot.
- Urbi [2] provides the same features as Player. In addition, it defines a scripting language to access sensors and actuators. Script developers can attach temporal profiles to variables.

The second group consists of modular architectures at software level, through software components. We consider this modularity to be limited, because the connections of sensors and actuators with components are not dynamic. A new sensor cannot be transparently added. The core of such architectures are typically TCP/IP-based interprocess communication services, such as CORBA [16] or HTTP. Several well developed frameworks fall in this category:

- OrIN [10] is based on HTTP and other web technologies and targets industrial robots.
- Miro [15] and RT Middleware [1] are based on CORBA.
- Orca [8] uses Ice [6], which is lighter than CORBA.
- Orocos [3] has its own component architecture, lighter than CORBA. Orocos also provides support to do Bayesian filtering, kinematics and dynamics computation.
- Microsoft Robotics Studio [9] uses HTTP to send events between components. Components run on the .NET framework with a Concurrency and Coordination Runtime (CCR).

While such frameworks are well suited to build higher-level robots controllers and to do complex computations such as path-planning, Bayesian processing, and reasoning; they are not modular at the level of physical devices because they are too heavy to run on microcontrollers.

The third group consists of architectures that are fully modular, both at software and hardware level. Such architectures provide physical modules that can be freely interchanged. Each module can describe itself and the services it provides. The most noteworthy example is OpenR [4] from Sony, used in the Aibo pet robot. In OpenR, modules can even describe their physical properties, such as their masses or their possible attach points on the robot. However, since Sony closed its robotics department, OpenR seems unmaintained. ASEBA falls in the same category as OpenR. Simpler than the latter, it provides full hardware modularity: each module can be added or removed at any time. Furthermore, because it is event driven, application can be designed to scale with additional modules. For instance, a faster distance sensor can be added to an existing robot and will decrease the reaction latency.

C. ASEBA

Compared to most robots middlewares, ASEBA has the following advantages:

- Distributed, event-based: small bus load, fast reaction time. By pre-processing sensor data directly on the microcontrollers, ASEBA reduces bus and main processor load. By sending events asynchronously, ASEBA decreases reaction time.
- Simple scripting language and user friendly development tools. Sensors and actuators are seen as normal script variables. Furthermore, once used to it, event-driven programs are easy to derive from specification documents.
- For selected modules, robustness to failure. For example, in an obstacle avoidance scenario where a long range distance sensor provides early reaction and short range distance sensors act as virtual bumpers; if the long range sensor fails, the robot can still avoid obstacles with the short range sensors but in a less intelligent way. In a polling architecture, the robot would be blocked or get erroneous data while trying to read the faulty sensor.

The cost of those features is more software layers that require more powerful microcontrollers and a multi-master shared communication bus. Yet we think that the advantages surpass the drawbacks: the overhead of code due to the virtual machine is reasonable and multi-master buses are now available to small robots.

The source code repository and bug tracker of ASEBA are available at `http://gna.org/projects/aseba`. The current implementation is working with the Enki simulator (`http://gna.org/projects/enki`). Future work include the development of behaviours and the port to the robots we are currently developing, in particular to the successor of the s-bot [12].

IV. CONCLUSION

ASEBA improves the modularity and efficiency of robot middleware by distributing some of the software tasks to all the microcontrollers and communicating only the relevant data to the main processor. It brings the modularity and dynamics of CORBA-based architectures into the microcontrollers. It achieves that by embedding a lightweight virtual machine in each microcontroller and providing an IDE to develop and debug the whole robot reactive control from a single place. The resulting architecture has hardware modularity, better efficiency, and improved scalability.

V. ACKNOWLEDGMENTS

We thank Cyrille Dunant, Michael Bonani, Daniel Burnier, Daniel Roggen, Yves Piguet, Martin Voelkle and Sebastian Gerlach for their insightful comments on ASEBA and for reviewing draft versions of this paper.

This work was supported by the Swarmanoid project and a CSEM research grant. The CSEM is the Centre Suisse d'Electronique et de Microtechnique. The Swarmanoid project is funded by the Future and Emerging Technologies programme (IST-FET) of the European Community, under grant IST-022888. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

REFERENCES

[1] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and Woo-Keun Yoon. Rt-middleware: distributed component middleware for rt (robot technology). In *International Conference on Intelligent Robots and Systems (IROS)*, pages 3933–3938. IEEE Press, 2005.

[2] J.-C. Baillie. Urbi: towards a universal robotic low-level programming language. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 820–825. IEEE Press, 2005.

[3] H. Bruyninckx. Open robot control software: the orocos project. In *International Conference on Robotics and Automation (ICRA)*, pages 2523–2528. IEEE Press, 2001.

[4] Masahiro Fujita and Koji Kageyama. An open architecture for robot entertainment. In *International Conference on Autonomous Agents*, pages 435–442. ACM Press, 1997.

[5] B. Gerkey, R.T. Vaughan, and A. Howard. The playerstage project: Tools for multi-robot and distributed sensor systems. In *International Conference on Advanced Robotics (ICAR)*, pages 317–323. IEEE Press, 2003.

[6] M. Henning. A new approach to object-oriented middleware. *Internet Computing, IEEE*, 8(1):66–75, Jan-Feb 2004.

[7] Joseph L. Jones and Anita M. Flynn. *Mobile Robots: Inspiration to implementation*. A.K. Peters, 1993.

[8] A. Makarenko, A. Brooks, and T. Kaupp. Orca: Components for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 163–168. IEEE Press, 2006.

[9] Microsoft. Microsoft robotics studio. `http://msdn2.microsoft.com/en-us/robotics/default.aspx`.

[10] Makoto Mizukawa, Hideo Matsuka, Toshihiko Koyama, Toshihiro Inukai, Akio Nodad, Hirohisa Tezuka, Yasuhiko Noguch, and Nobuyuki Otera. Orin: Open robot interface for the network. In *SICE*, pages 925–928. IEEE Press, 2002.

[11] F. Mondada, E. Franzi, and P. Ienne. Mobile robot miniaturisation: A tool for investigation in control algorithms. In *Experimental Robotics III*, pages 501–513, Kyoto, 1994. Springer-Verlag.

[12] F. Mondada, A. Guignard, M. Bonani, D. Bär, M. Lauria, and D. Floreano. SWARM-BOT: From Concept to Implementation. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1626–1631. IEEE Press, 2003.

[13] I.A. Nesnas. *The CLARAty Project: Coping with Hardware and Software Heterogeneity*, volume 30/2007 Software Engineering for Experimental Robotics of *Springer Tracts in Advanced Robotics*, pages 31–70. Springer, 2006.

[14] ISO Standards. *Road Vehicles Interchange of Digital Information - Controller Area Network - ISO 11898*. International Organization for Standardization, 1993.

[15] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar. Miro - middleware for mobile robot applications. *Robotics and Automation, IEEE Transactions on*, 18(4):493–497, Aug 2002.

[16] S. Vinoski. Corba: integrating diverse applications within distributedheterogeneous environments. *Communications Magazine, IEEE*, 35(2):46–55, Feb 1997.