

Applications of a Real-Time Software Framework for Complex Mechatronic Systems

Frederic Pont, Sascha Kolski and Roland Siegwart
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Autonomous Systems Laboratory (ASL)
CH-1015 Lausanne, Switzerland
{frederic.pont, sascha.kolski, roland.siegwart}@epfl.ch

Abstract—As the complexity of the missions to be performed by mechatronic systems grows, so does the amount of embedded software to be produced and the number of involved specialists. System complexity management and software integration become more important. In this paper, we present an initial implementation of a real-time capable software framework for complex mechatronic systems that facilitates embedded software development and integration. The framework also promotes software components reuse across applications, architecture reuse and software portability across hardware platforms. To illustrate the proposed solution, we present two applications of the framework: an autonomous indoor navigation system for mobile robots and a driver assistance system for smart cars.

I. INTRODUCTION

Software integration and complexity management for advanced mechatronic systems like autonomous mobile robots remain open challenges. With the increasing complexity of the missions to be performed (e.g. space exploration), a growing number of disciplines become involved in the conception of a complete system, from mechanical and electronics engineering, to computer and even cognitive sciences. As a result, the amount of software to be produced for controlling such systems gets larger, with contributions from specialists pursuing different goals and with varying software engineering skills.

Autonomous mobile robots are good examples of complex mechatronic systems that require contributions from many specialized roboticists who have little or no knowledge of the complexity of the whole system. Architectures for autonomous robots have been studied for many years, describing how software systems could be organized [1], [2]. Nowadays, a hybrid solution combining reactive behaviors with limited knowledge of the world and higher level reasoning is widely accepted [3], [4], [5]. However, there is no standardized way of implementing this architecture to build a complete software system. As a result, software systems for autonomous robots are usually written from scratch and not built from existing pieces of software, leading to a waste of time and resources. Indeed, students, researchers and developers spend a large amount of their time solving software implementation and integration problems instead of focusing on their specific areas of interest to make valuable contributions. Moreover, software embedded into complex mechatronic systems performing missions in the real world must comply with safety critical requirements. Therefore, the

ability to meet strict timing constraints is mandatory and this adds to the challenge of providing better solutions for embedded software systems development and integration.

A. Related Work

Recently, many robotic research projects started tackling these software reuse, integration and implementation problems. Among them, PlayerStage [6], [7] is widely used. PlayerStage provides a network server for robot control over IP (Internet Protocol). It is designed for off-board control and is therefore not suitable for self-contained autonomous systems, but the proposed hardware abstraction based on devices, drivers and interfaces is promising [7]. ORCA-Robotics [8], [9], [10] promotes a component-based software engineering [11] approach for robotics, where software systems can be composed by mixing existing components with custom in-house developed components. A number of open-source components are already available. Moreover, a number of other projects [12], [13], [14] also investigate how robotic software systems can be improved.

Other research projects focus on how modern software engineering techniques like frameworks or design patterns can be applied to embedded systems in general [15], [16], [17], taking into account the specific requirements of such systems, while providing for easier integration, software reusability across applications or hardware platforms.

B. Approach

In this paper, we present an initial implementation of a real-time capable software framework for self-contained mechatronic systems. It aims at empowering specialists to contribute software components that can be integrated into a complete software system capable of meeting timing constraints. The proposed framework ensures that specialists do not have to deal with the complexity of the whole system and makes the software integration process easier. Moreover, it promotes software reuse and provides for software portability across hardware platforms.

The initial implementation focuses on specialized software components generation and integration into a complete real-time embedded system. Ultimately, the framework will also provide real-time constraints checking and adaptation to events such as missed deadlines. We also present two applications that have been developed and integrated in the

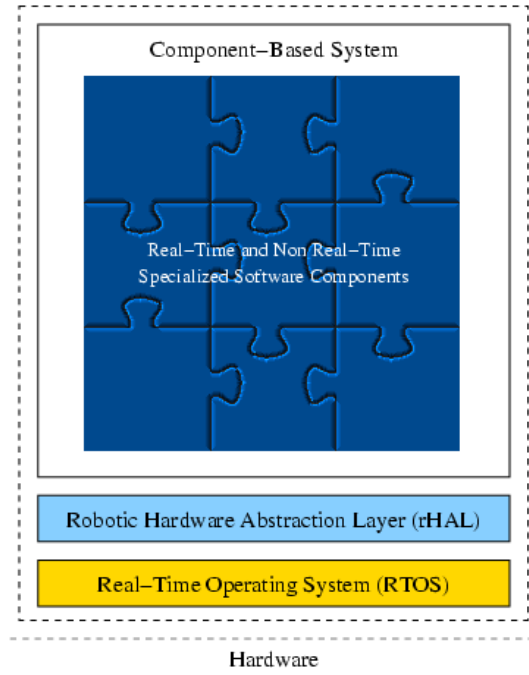


Fig. 1. A real-time component-based software framework for complex mechatronic systems.

context of the proposed software framework. The first is an indoor navigation system for the autonomous robots RoboX and BIBA, and the second is a driver assistance system for smart cars applied to a Smart ForTwo passenger car.

This paper is organized as follows. In the following section, we introduce the real-time software framework. In section III, we present an autonomous indoor navigation system and in section IV we introduce the driver assistance application. Finally, we present concluding remarks and a research outlook in section V.

II. A REAL-TIME SOFTWARE FRAMEWORK

The proposed component-based and real-time software framework is composed of a real-time operating system (RTOS), a robotic hardware abstraction layer (rHAL) and a component-based software system, as represented in figure 1. It targets self-contained, embedded and real-time software systems for controlling complex mechatronic systems, in particular autonomous mobile robots. It aims at empowering specialists to implement software components that can be used to build complete software systems able to meet timing constraints. It also provides for easier integration of components and takes care of inter-component communication.

The current implementation of the framework is based on RTAI Linux [18] as its underlying RTOS. RTAI Linux is a real-time extension to the regular Linux kernel, and has been selected because it provides the usual advantages of the Linux operating system, along with hard-real time capabilities. This choice limits the programming language to C, to allow for component execution in kernel space when hard real-time is required, but extensions to other programming languages are possible for non real-time components.

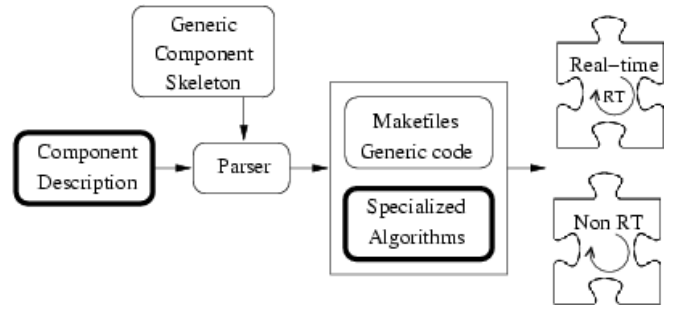


Fig. 2. GenoM-based software component generation: from the component description file to real-time and non real-time executables.

The robotic hardware abstraction layer is a thin real-time capable software layer ensuring that higher level software components are portable across hardware platforms. In some cases, the hardware abstraction is provided by lower level software components and the hardware abstraction layer does not exist as an independent layer. The current implementation of rHAL supports the tour-guiding robot RoboX [19] and provides standardized abstractions for analog and digital inputs and outputs, counters and serial interfaces.

The robotic hardware abstraction layer is implemented as a set of kernel modules for interfacing with real-time components, and as a user-space library for non real-time components. Both user space library and kernel space modules provide the exact same interface to ensure that components accessing hardware through the abstraction layer can be executed transparently in user space or kernel space. For more details about rHAL implementation, see [20].

The component model selected for the proposed software framework is GenoM (**generator of modules**) [21], [22], which is an environment for description and implementation of software components that provides the following:

- **Component Model:** GenoM defines specific interaction between components and composition standards.
- **Component Model Implementation:** GenoM provides the dedicated set of executable software elements required to support the execution of software components that conform to the model.
- **Component Architecture:** GenoM defines the internal architecture of software components, and their structure and functioning.
- **Component Generation Tools:** GenoM provides a set of tools for describing software components and for generating templates.

GenoM-based software component generation is represented in figure 2. The generation process is based on a component description file that is parsed by the tool. Then, templates are generated and can be used by specialists for implementing the value-adding algorithmic part of the software. Finally, binary components are generated, in both real-time and non real-time forms, along with test programs.

The proposed software framework provides for complexity management by integrating modern techniques like component-based software engineering. It also ensures that

real-time capabilities are available when required, and it promotes software reuse by clearly separating value-adding algorithms from utility software. In the following sections, we present two mechatronic software systems that rely on the real-time software framework introduced above.

III. AUTONOMOUS INDOOR NAVIGATION

In this section, we present an autonomous indoor navigation software system developed using the introduced framework and tested with the robot RoboX (figure 3(b)) and the BIBA robot (figure 3(a)). For the software application developed in this indoor navigation project, the following requirements have been defined:

- 1) The complete software system shall be embedded on the autonomous mobile robot, and all processing must be carried out on-board and on-line.
- 2) Localization of the robot shall be performed using both encoders data and laser range finders.
- 3) Localization relies on a known map of the environment.
- 4) Navigation in the environment shall be performed without bumping into humans or fixed obstacles.
- 5) For safety reasons, obstacle avoidance shall be performed in a periodic hard real-time task.
- 6) Requests from a human operator will be in the form of: go to x, y, θ . No path planning is required.

In the remainder of this section, we describe the software layers and components that have been used to build the navigation system represented in figure 3(c).

A. Hardware Platforms

The robot RoboX represented in figure 3(b) was first developed for the Swiss national exposition in 2002. It includes differentially driven wheels located at the center of the robot, allowing on the spot turns. The processing power is provided by an embedded PowerPC 750 clocked at 400MHz enclosed in an industrial rack. A Pentium running at 700MHz is also available, but was used for interaction with humans and is not considered for this indoor navigation application. Two SICK laser range finders are mounted back to back. The BIBA robot embeds the same computer systems and sensors, but does not include the mechanical parts for interaction with humans.

B. Software Components

The indoor navigation software system is made of a mix of reused and newly developed components. Existing components have been developed at LAAS (Laboratory for Analysis and Architecture of Systems) in Toulouse, France, [23], and have been used for many years on LAAS robots. New software components have been developed in collaboration with specialized roboticists from the Autonomous System Laboratory at EPFL. The complete system with data flow between components is shown in figure 3(c).

1) *SICK Laser Range Finder (real-time)*: The *SICK* component is responsible for managing the two SICK laser range finders placed back to back on RoboX. To ensure no data loss on the two high speed serial interfaces RS422 and to meet the hard real-time obstacle avoidance requirement, this component is a periodic real-time task.

- Input: Data from rHAL serial interfaces.
- Output: Scans in polar and cartesian coordinates.

2) *Obstacle Avoidance (real-time)*: The *obstacle avoidance* component is based on nearness diagram navigation (ND) [24], which performs a high level information extraction and interpretation of the environment, producing appropriate linear angular speeds. To ensure safe indoor navigation in a human crowded environment, this component is a periodic real-time task. Note that the real-time requirement on this component implies that all components producing data used by it must also be hard real-time. In the case study, the suggested linear and angular speed produced by this component are read by the *speed control* component.

- Input: Scan in cartesian coordinates from the *SICK* component.
- Output: Suggested linear and angular speeds.

3) *Speed Control (real-time)*: This component is responsible for translating angular and linear speed requests into a specific speed for each wheel. Maximum speeds and acceleration can be configured. This component is a real-time task to ensure smooth movements of the robot, and to comply with the real-time obstacle avoidance requirement. In the indoor navigation scenario, this component reads linear and angular speeds from the *obstacle avoidance* component. However, when the robot is remote controlled or when obstacle avoidance disabled, linear and angular speeds can be set manually.

- Input: Linear speed, angular speed from *obstacle avoidance*.
- Output: Left and right motor values (rHAL analog outputs).

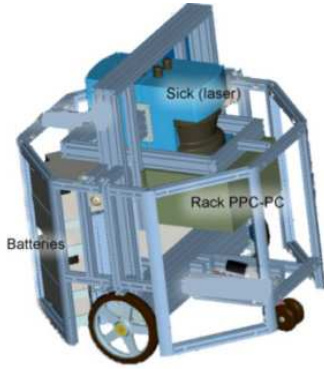
4) *Odometry (real-time)*: The *odometry* component estimates the current robot configuration (x, y, θ) using only wheel sensors. It is well known that position estimation using only odometry is rough and that the error grows with time. Therefore, other kind of position estimation must be introduced for advanced navigation. Odometry configuration estimations are used by the *localization* and *speed control* components.

- Input: Left and right encoder values (rHAL counter).
- Output: Robot configuration (x, y, θ, C) .

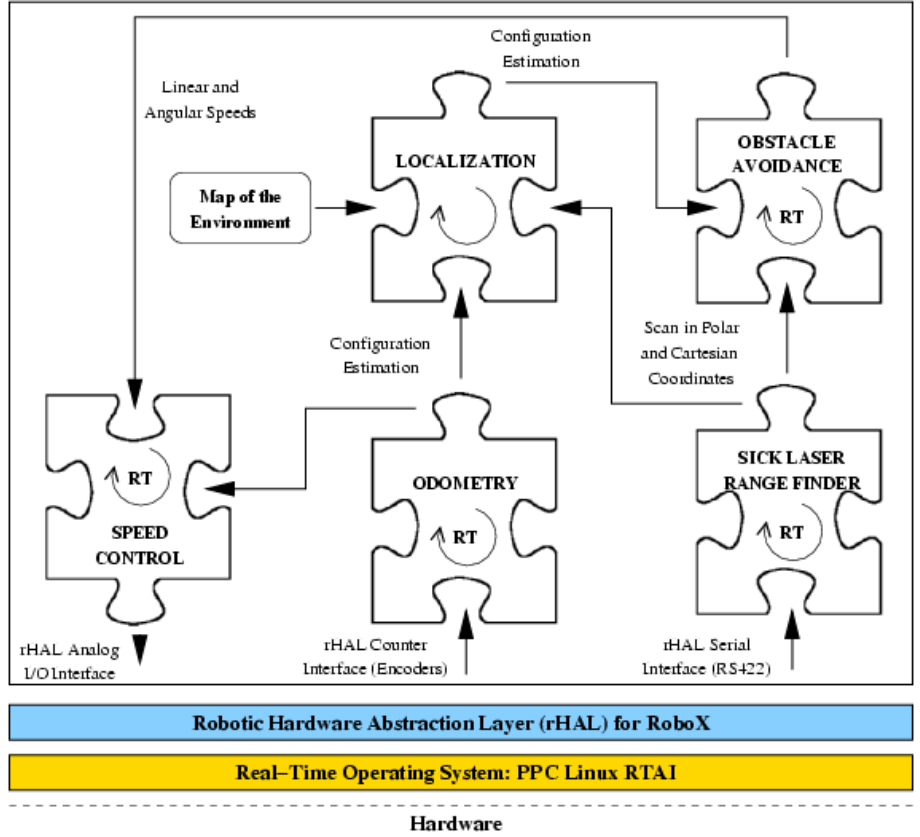
5) *Localization*: The *localization* component relies on the two SICK laser range finders and on an a-priori known map of the environment to evaluate the current configuration of the mobile robot. This component also relies on configuration estimations provided by the *odometry*. The line extraction method used is based on the split and merge approach [25] and the fusion with the configuration estimation produced by the *odometry* is performed using an Extended Kalman Filter. The configuration estimation produced by this component



(a) BIBA robot.



(b) RoboX.



(c) A component-based software system. Software components that are executed as real-time tasks are mentioned with RT. Arrows represent data flow between components.

Fig. 3. An autonomous indoor navigation system for the BIBA robot and RoboX.

is used by the *obstacle avoidance* component in the indoor navigation scenario. Note that if this component is disabled or is not able to produce an acceptable estimation, the indoor navigation system can rely on the *odometry* component for a rough configuration estimate. As this component is not critical for security, it is executed as a non real-time task.

- Input: Map of the environment, cartesian scan from SICK, configuration from *odometry*.
- Output: Robot configuration (x, y, θ, C) .

This indoor navigation system is successfully used on both RoboX and BIBA robot. The real-time capabilities of the system ensure that the navigation is safe. Moreover, the component-base approach selected for the framework provides for the possibility to replace any component by another implementation providing the same services. Also, each component of the system is developed by a specialized roboticist in the context of the proposed framework, resulting in a high quality system.

IV. DRIVER ASSISTANCE FOR SMART CARS

In this section, we present another project that benefits from the software framework introduced above: a driver assistance for smart cars. A Smart ForTwo passenger car

is equipped with several exteroceptive and proprioceptive sensors in order to develop and to test algorithms for driver assistance and autonomous driving tasks. This involves domain specific tasks like lane keeping as well as classic robotic tasks like obstacle detection and avoidance.

A. Hardware Platform

The computing power is provided by a standard laptop running Linux. The laptop is connected to the various perception devices as well as to the vehicle internal bus system for accessing internal sensors for velocity and steering angle. The sensing devices added to the vehicle are a SICK laser range finder mounted in front of the car, a firewire camera and an inertial measurement unit (IMU). The vehicle can be seen in figure 4(a).

B. Software Components

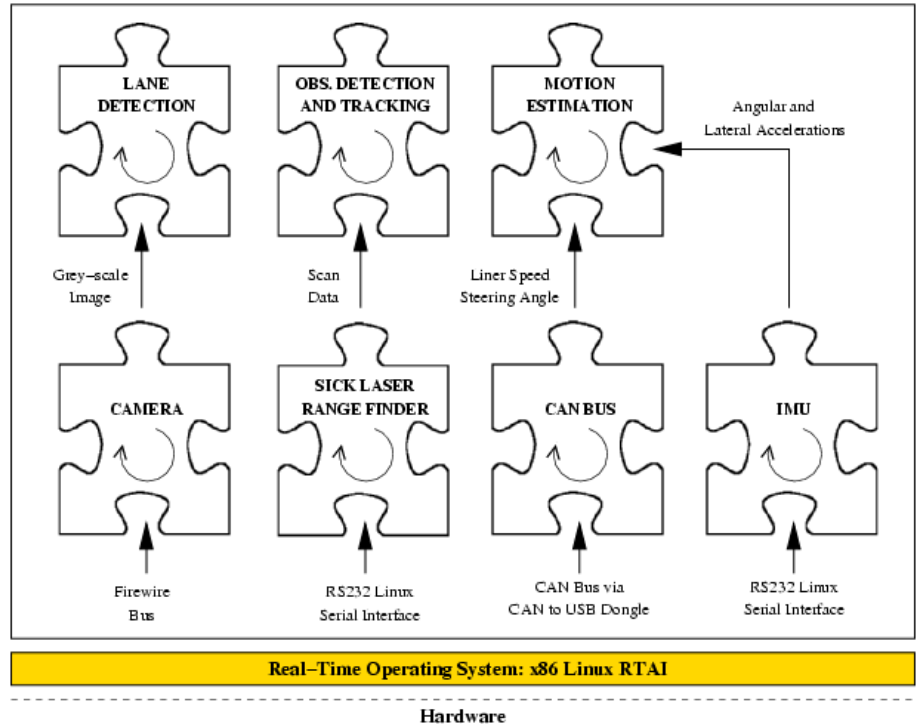
In the driver assistance software system, each perception task and each processing task is implemented in a software component. The complete system with data flow between components is shown in figure 4(c).



(a) Smart ForTwo passenger car equipped with a SICK laser range finder.



(b) Lane detection.



(c) A component-based software system. Arrows represent data flow between components.

Fig. 4. Driver assistance system for smart cars.

1) *SICK Laser Range Finder*: As in the indoor navigation application presented above, the *SICK* component is responsible for managing the *SICK* laser range finder mounted in front of the car. The *SICK* sensor is connected to the laptop through a RS232 serial interface but a high speed RS422 serial connection is planned. Data is published by this component with a refresh rate of $20Hz$ and it is used by the *obstacle detection and tracking* component.

- Input: Data from Linux standard serial interface.
- Output: Scans in polar and cartesian coordinates.

2) *Obstacle Detection and Tracking*: The *obstacle detection and tracking* component uses the scan data generated by the *SICK* component to perform a segmentation, objects extraction by means of line segments and tracking of these objects to obtain their relative velocity.

- Input: Scan from *SICK* component.
- Output: List of obstacles with relative position, relative speed and size.

3) *Camera*: The *camera* component reads data from the firewire bus and produces several images, namely a RGB, a YUV and a grey-scale image. The *lane detection* component relies on the grey-scale image produced by this component.

- Input: Data from the firewire bus.
- Output: RGB, YUV and grey-scale images.

4) *Lane Detection*: The *lane detection* component performs a combination of a Canny filter to obtain directed

edges in the picture. Then, a modified Hough transform is applied to classify these edges by the angle in which they occur in the image and a particle filter is used to choose the most likely candidates forming the lane boundaries. The result of the lane detection can be seen in figure 4(b).

- Input: Grey-scale image from the *camera* component.
- Output: Lane width, offset to the vehicle and yaw, modelling the lane by means of a straight lane.

5) *CAN Bus*: The *can bus* component monitors the vehicle bus using a CAN to USB dongle attached to the laptop. The output of this component is the current vehicle speed and the current steering angle, calculated using geometric constraints from the steering wheel angle delivered on the CAN bus.

- Input: Data from the car CAN bus.
- Output: Current vehicle speed and steering angle.

6) *IMU*: The *IMU* component communicates with the IMU (Inertial Measurement Unit) device via a serial RS232 connection and publishes the angular and lateral accelerations. The IMU was introduced in the system to overcome large and unsystematic errors on the driving angle obtained when relying only on the data available through the *CAN* component.

- Input: Data from the IMU device.
- Output: Angular and lateral accelerations.

7) *Motion Estimation*: As we use moving coordinate systems in our application, the *motion estimation* component

forms a kind of anchor to the real world and helps estimating the motion of other traffic participants, as well as anticipating the car movement in respect to the driving lane. Thereafter a good estimate of the motion of our own vehicle forms a basis for further processing. This component fuses data from the CAN and the IMU components using an Extended Kalman Filter.

- Input: Data from CAN and IMU components.
- Output: Corrected vehicle speed and yaw rate.

This early-stage driver assistance system for smart cars is currently built using non-real time software components. The use of the presented real-time framework ensures that timing constraints can later be introduced without any modifications to the value-adding algorithmic parts that are currently being tested.

V. CONCLUSIONS AND OUTLOOK

We presented an initial implementation of a real-time software framework for complex mechatronic systems that helps managing system complexity, promotes software components reuse and increases software portability across hardware platforms. The framework also empowers specialists to contribute software components that can be integrated into embedded software systems which have the ability to meet strict timing constraints, which is mandatory for real-world mobile mechatronic systems. We also presented two applications taking advantage of the proposed framework. The first project is a autonomous indoor navigation system for the RoboX and BIBA robots. The system is built with software components developed by specialized roboticists and the integration into a real-time system that is reliable and safe was made possible by the proposed framework. The second project is a driver assistance system for smart cars that relies on proprioceptive and exteroceptive sensors mounted on a Smart ForTwo passenger car. The use of the proposed software framework provides for component reuse across applications. It also ensures that software components can be developed and tested without timing constraints during the first part of the project and that they can later be switched to real-time tasks when necessary.

The current implementation focuses on component generation and integration into a complete system. Future work includes the extension of the software framework with tools and mechanisms for precise performance monitoring and timing constraints verification in order to ensure safety and compliance to requirements. As the framework is meant to be used by specialists that are not software engineers, some improvements are also planned to shorten the learning curve in order to promote the use of the framework.

VI. ACKNOWLEDGMENTS

The work reported in this paper has been supported in part by the IST projects RECSYS (IST-32515) and SPARC (IST-507859). Moreover, the authors would like to thank Agostino Martinelli, Viet Nguyen and Anthony Mallet for their precious contributions to the real-time software framework and to the indoor navigation system.

REFERENCES

- [1] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, March 1986.
- [2] R. C. Arkin, *Behaviour Based Robotics. Intelligent Robots and Autonomous Agents*. MIT Press, 1998.
- [3] E. Gat, "On three-layer architectures," *Artificial Intelligence and Mobile Robots. MIT/AAAI Press*, 1997.
- [4] K. Konolige and K. Myers, "The saphira architecture for autonomous mmobile robots," Artificial Intelligence Center, SRI International, Tech. Rep., 1996.
- [5] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "The CLARAty architecture for robotic autonomy," in *IEEE Aerospace Conference Proceedings*, Big Sky, Montana, March 2001, pp. 121–132.
- [6] B. Gerkey, R. Vaughan, K. Sty, A. Howard, G. Sukhatme, and M. Mataric, "Most valuable player: A robot device server for distributed control," in *Proceedings of the International Conference on Intelligent Robots and Systems*, Wailea, Hawaii, October 2001, pp. 1226–1231.
- [7] R. T. Vaughan, B. P. Gerkey, and A. Howard, "On device abstractions for portable, reusable robot code," in *Proceedings of the International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, October 2003, pp. 2121–2427.
- [8] "ORCA-Robotics," <http://orca-robotics.sourceforge.net/>.
- [9] W. Li, H. I. Christensen, A. Oreback, and D. Chen, "An architecture for indoor navigation," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, April 2004.
- [10] A. Oreback, "A component framework for autonomous mobile robots," Ph.D. dissertation, KTH Stockholm, 2004.
- [11] G. T. Heineman and W. T. Councill, *Component-Based Software Engineering: Putting the Pieces Together*. Addison Wesley Professional, 2001.
- [12] "Orocos," <http://www.orocos.org/>.
- [13] C. Cote, D. Letourneau, F. Michaud, J.-M. Valin, Y. Brosseau, C. Raievisky, M. Lemay, and V. Tran, "Code reusability tools for programming mobile robots," in *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2004.
- [14] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit," in *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.
- [15] D.-J. Chen and M. Torngren, "Towards a framework for architecting mechatronics software systems," in *Proceedings of the Seventh IEEE International Conference on Engineering of Complex Computer Systems*, 2001, pp. 170–179.
- [16] A. Pasetti, "Software Frameworks and embedded Control Systems", ser. Lecture Notes in Computer Science. Springer-Verlag, 2002, vol. 2231.
- [17] A. Blum, V. Cechtický, A. Pasetti, and W. Schaufelberger, "A java-based framework for real-time control systems," in *Proceedings of ETFA*, vol. 2, 2003, pp. 447–453.
- [18] "RTAI: Real-time application interface," <http://www.rtai.org>.
- [19] "RoboX," <http://robotics.epfl.ch>.
- [20] F. Pont and R. Siegwart, "Towards improving robotic software reusability without losing real-time capabilities," in *ICINCO (2)*, 2004, pp. 291–294.
- [21] S. Fleury, M. Herrb, and R. Chatila, "GenoM: a tool for the specification and the implementation of operating modules in a distributed robot architecture," in *Proceedings of the International Conference on Intelligent Robots and Systems*, Genoble, France, September 1997, pp. 842–848.
- [22] A. Mallet, S. Fleury, and H. Bruyninckx, "A specification of generic robotics software components: future evolutions of genom in the orocos context," in *International Conference on Intelligent Robotics and Systems*. Lausanne (Switzerland): IEEE, Oct. 2002.
- [23] "Laas open software for autonomous systems," <http://softs.laas.fr/openrobots/>.
- [24] J. Minguez and L. Montano, "Nearness diagram navigation (nd): A new real time collision avoidance approach for holonomic and no holonomic mobile robots," 2000.
- [25] T. Pavlidis and S. Horowitz, "Segmentation of planar curves," *IEEE Transactions on Computers*, vol. C-23, no. 8, pp. 860–870, 1974.