Journal of Computing and Information Technology - CIT 17, 2009, 2, 195–202
doi:10.2498/cit.1001212

A Software Architecture Framework for Home Service Robots

Mohamed T. Kimour[1], Ammar Bessam[2] and Rachid Boudour[1]

[1] Department of Computer Science, University of Badji Mokhtar-Annaba, Algeria
[2] Department of Computer Science, University of Jijel, Algeria

Over the last years, home service robots have a wide range of potential applications, such as home security, patient caring, cleaning, etc. When developing robot software, one of the main challenges is to build the software architectural model. Software architecture is used throughout the software life-cycle for supporting analysis, guiding development, and acting as a roadmap for designers and implementers. Though many software architectures for robotic systems have been defined, none of them have reached all its objectives due to the great variability among systems behaviors, and still lack of systematic techniques to derive the robot software architecture from its requirements model. In this paper, we present a generic architectural model for home service robots, allowing for software architecture design, and preserving a continuous architectural view all along the development cycle. While avoiding the predominant decomposition problems, our approach allows for integration of the architectural components in a systematic and comprehensive way for efficient maintainability and reusability.

Keywords: home service robots, software architecture, embedded systems, separation of concerns

1. Introduction

Nowadays, robots are used extensively in industrial environments to perform repetitive, dangerous, unpleasant, and hard works, and are more and more entering everyday life as toys and service robots [1].

Robots are complex systems whose complexity is continuously increasing as more and more intelligence (decisional and operational autonomies, human-machine interaction, robots cooperation, etc.) is embedded into their controllers [2].

In the last decade, both academic and industrial teams have been conducting intensive research on the emerging field of home service robots [3]. Home Service Robots (HSRs) are individually designed to perform tasks in a specific environment for working with, or assisting humans and must be able to perform services semi- or fully automatically [4]. A number of commercialized service robots have recently been introduced, such as vacuum cleaning robots, home security robots, entertainment robots, guide robots, etc. [5,6,7].

Due to the increasing complexity of these robots, the design of their embedded software is a difficult task. The complexity mainly depends, on the mechanical portion of the robot that the controller has to deal with. These two aspects of a robot, its mechanical aspect (including its sensors and actuators) and its control logic are intrinsically interdependent.

Robots are embedded systems, complex and reactive, where the new way of building their control software is centered on the design of the software architecture as a combination of software entities, allowing for reuse and reasoning about system properties before its implementation [6,8]. Software architecture is represented by an architecture description language, highlighting various system properties allowing for analyzing and evaluating trade-offs among alternative solutions [9].

For HSRs, architectures form the backbone of complete robotic systems. A good choice of architecture allows for facilitating the specification, implementation and validation of robotic systems.

Though many software architectures have been defined for developing control software for robots as embedded systems, and reuse common

components, none of them have reached all its objectives, especially for HSRs, due to the great variability among systems behaviors [10].

It is still a challenging problem to develop the robot software architecture by carefully taking into account user needs and requirements, implement robot application components based on the architecture, and integrate these components in a systematic and comprehensive way [3].

Many robotics controllers software inspired by Component-based Software engineering, have been developed [4,3,11,12,13,14,15]. Some approaches are devoted to specific robotic systems; others propose common frameworks which cover a broad range of mechanisms and missions, each of them with their specific features and requirements. However, none of them have addressed the problem of the predominant decomposition and important aspects inherent to technology evolution.

Designing the software architecture is one of the most challenging issues for the development of the robots [3]. Specifically, HSRs need to add or update services frequently, according to the changing needs of users. Among others, this reason leads to integrating technology-oriented components to benefit from their advantages in reusability, upgradeability and cost.

Thus, it becomes more important to develop reusable software control for the HSRs. Such a software control has to not only exhibit modularity and scalability, but to support generalized software architecture allowing rapid developments, decreased cost and increased usability.

In this paper, we present a generic architecture model for home service robots facilitating the software architecture design, which would rely on a continuous refinement of application models, with system elements preserving a continuous architectural view all along the design cycle.

The main idea of our approach is based on an appropriate component categorization leading to efficient separation of concerns. Separation of concerns is the primary means of splitting an artifact into parts in order to make it more manageable and comprehensible [16,17].

The proposed generic software architecture is composed of three layers, each of which is divided into three sub-layers. Also, shared data and crosscutting concerns are orthogonal to all the layers and sub-layers. This orthogonality constitutes the main difference between the most notable software architectures for robotic systems, especially those proposed for industrial robotic arms and our architectural framework. Furthermore, our approach is distinguished by at least two fundamental characteristics:

Firstly, while separating between data flow and control flow, it advocates modeling the robot arm function at the mechanism and I/O layers. At the mechanism layer, a software component communicates not only with components at the I/O layer, but also with the higher level ones for e.g. coordinated actions, thus improving the efficiency of the global architecture and following recent trends in the embedded systems design community. It would be useful to correctly define specific requirements, especially the timing constraints and safety concerns. From the users and programmers point of view, this facilitates accessibility to them with different domain(s) of expertise.

Secondly, our approach supports the modeling of crosscutting concerns, called Non-functional Requirements (NFRs). NFRs are concerns on qualities and constraints of the system (such as security, usability, safety, etc.) that cannot be effectively described and modularized using the concept of composition of high cohesion functional modules.

The rest of the paper is organized as follows: Section 2 summarizes the home service robot domain and characterizes it. Based on a separation of concerns principles, Section 3 describes the proposed component categorization for HSRs. This component categorization is used in Section 4 to present our software architecture framework for HSRs. Finally, in Section 5, conclusion and the future directions of our research are outlined.

2. Home Service Robots

HSRs are robotic systems that can be used for real useful work around the home to make our lives more convenient and entertaining [3]. They aim to increase the quality of human life in a wide range of application areas. They are designed for providing various services to human user such as: home security, patient caring,

cleaning, etc. In addition to providing assistance to the elderly, it can further be envisaged that such robotic appliances will be of general utility to humans in their homes [18]. The role of robots in aiding the disabled and elderly is becoming very important with growing health care cost and the increase of the elderly population.

HSR applications integrate robots and humans both in a common workspace and in the execution of the same work tasks. Such tasks are involved when a person needs help in the same living environment and for the same application, such as eating, drinking, washing, shaving, etc.

Examples of HSR tasks include a direct interaction with humans as well as control of different equipment through push buttons, such as computers, copy machines, turn on/turn off lights, functions in the kitchen, etc. In addition to fetch-and-carry duties, this includes services such as setting the table or performing basic cleaning, leading to the capability of HSRs to move through various rooms of the home without colliding with furniture or people.

Among other services, a HSR can recognize voice commands from a user via microphones the robot is equipped with, and can synthesize voices for services. A user can call a robot's predefined name, to let the robot recognize the call while the robot knows the direction to move to the user. This service analyzes audio data and is based on speech recognition to recognize the command from the user (see Figure 1).

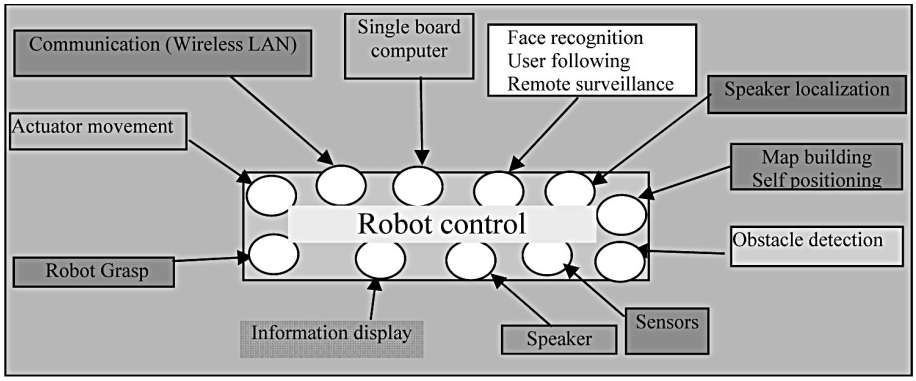

Figure 1. Example of home service robot components.

HSRs work directly with people, thus placing a central importance on making interactions between people and machines as natural as possible. The scenario of house keeping and home care robot assistants focuses on the employment of HSRs in everyday domestic settings.

HSRs should have a high robustness and be able to be operated by a disabled user which means a semi-automatic operation using different types of user interface devices for operation in unstructured environment.

The mechanical design of HSRs should consider different specifications compared to those used in industrial applications. Examples of differences are: payload in the lower range, low total weight of the robot and high payload/weight ratio, low life duty cycle and low acceleration and velocity performance.

On the other hand, control Software for HSRs, differs from other software applications in many ways. Foremost are the needs to achieve high-level, complex goals, the need to interact with a complex, often unstructured environment, while ensuring the system's own dynamics, the need to handle noise and uncertainty, and the need to be reactive to unexpected changes. These needs influence the ways how HSRs are designed, how they operate, and how they are validated.

Furthermore, there is a need to have capabilities and functions made specific to the service tasks. This leads to more complex applications that are required to achieve multiple tasks in reactively and concurrently manner, while satisfying stringent safety requirements.

A control software architecture for such systems, should provide HSR software designers with means to analyze a given software architecture with respect to system properties, early in the design process.

It becomes more important to follow architecture-based development to build HSRs in reasonable project time, while evolving in accordance with quickly changing market demands, and satisfying required quality attributes such as safety, human-robot interaction, power consumption, chemical compatibility, price, life time, and ease of use.

Modularity and flexibility allow among others, for decreasing cost, increasing usability, and integrating not only technical aspects, but also user interactions. Scalability and upgradeability permits to cope with technological advances that continually offer new devices for communication, localization, computing, etc.

3. Component Categorization

Overall HSR complexity can be reduced by decomposing it into smaller components with well-defined abstraction levels and interfaces between them.

Usually, robot architectures are determined through hierarchical decomposition, where modular components are themselves built with other modules. This type of decomposition can lead to more modular systems, which has a positive impact on the performance, maintainability and evolution. However, such decomposition is a difficult task and requires systematic technique to do it.

We believe that a characterization of the application domain can go a long way in making the hierarchical decomposition easier and seamless. It is the starting point to define the functional and quality requirements that guide the architecture design.

To this end, we define an HSR categorization by applying separation of concerns principles. Separation of concerns is a viable principle for reducing complexity and increasing reusability and upgradeability.

Indeed, physically, a HSR consists of mechanical, electrical, and sometimes hydraulic components (e.g., links, drive trains, induction motors, hydraulic actuators, encoders), often controlled by embedded software running on a central processing unit, or a dedicated digital signal processor [19].

A crucial part in the robot control is the model of its mechanical structure, providing the kinetic and dynamical states of the robot, which is vital to control functionality such as trajectory generation and loop control [1]. Moreover, behavior of HSR is context-dependent in the sense that it reacts to the environment change and new system status by supervising the outside world and controlling its parameter values.

We thus categorize HSR components according to a combination of layered architecture, where each layer exhibits specific behavior characteristics and constraints, on one hand, and four separations of concerns principles on the other.

3.1. Separating Control from Data Components

Separating the control aspect, which consists of control components, from the data aspect, we could separate data flows from control flows, thus making it possible to visualize and analyze behaviors of the system. As a consequence, addition/removal of components becomes easier because responsibilities of each component become clear.

A control component is defined to encapsulate state-dependent computation that generates commands to be addressed to actuators. A control component may use information from data and/or algorithmic components.

A data component is a persistent container that encapsulates a state (knowledge) and a behavior. Data component is used to describe a robot's knowledge of its environment, its mission and its concrete entities, such as those relating to the robot's mechanical elements (vehicle, manipulator, etc.) or elements of its environment.

In the architecture description language view, a data component consists of a set of provided ports that allow other components to obtain some of its static physical properties (wheel diameter, frame width, etc.) and set or obtain its dynamical properties (velocity and orientation of wheels, etc.).

3.2. Algorithmic Components

Home service robots consist of various components (e.g. vision recognizer, speech recognizer, and actuator) to provide services to a user.

A component of this type encapsulates specific computation that describes how to compute a set of outputs based on a given set of inputs. Usually, it prepares some specific computations to a control component. The domain is characterized by various types of control algorithms, from very simple reactive actions to extremely complex algorithms and navigation strategies, depending on the applications. Moreover, as a decomposition principle, we isolate the computation related to a technology that is characterized by a relatively fast evolution of the more stable one.

3.3. Interface Components

To do its functionalities, HSR has various sensors and actuators, as output peripherals such as information display devices, speaker for speech generation, etc. The actuators model the simplest active elements, for example a motor for left and right wheels. A remote user can control the robot using a PDA.

The sensor interface components are components that provide the information required for controlling a given active element, for example, the encoder and switch limits associated to a given joint. Sensors detect events, which are notified to higher-level control or coordination components. An actuator interface component receives commands generated by a control component, and concerns actuators that belong to the set of physical elements controlled by such control component. An interface component is defined to encapsulate external interaction patterns of the robot by means of its sensors and actuators or other I/O drivers.

3.4. Coordination Components

The coordination responsibility, among different services in a group of related control components, is assigned to a special component called coordination component which controls global behavior of such a component's group. This component is responsible for any activity dispatching or component selection to perform some tasks.

4. Layered Architecture

As shown above, the domain is characterized by high degree of specialization and, therefore, a great variability of functionality and physical characteristics. We note that the presence of hard real-time requirements and safety is usually a main concern. However, requirements nature and characteristics differ from sensors and actuators to vehicle control or manipulators. This is why a layering approach would be useful to correctly define these requirements, especially the timing constraints and safety concerns. From the users and programmers point

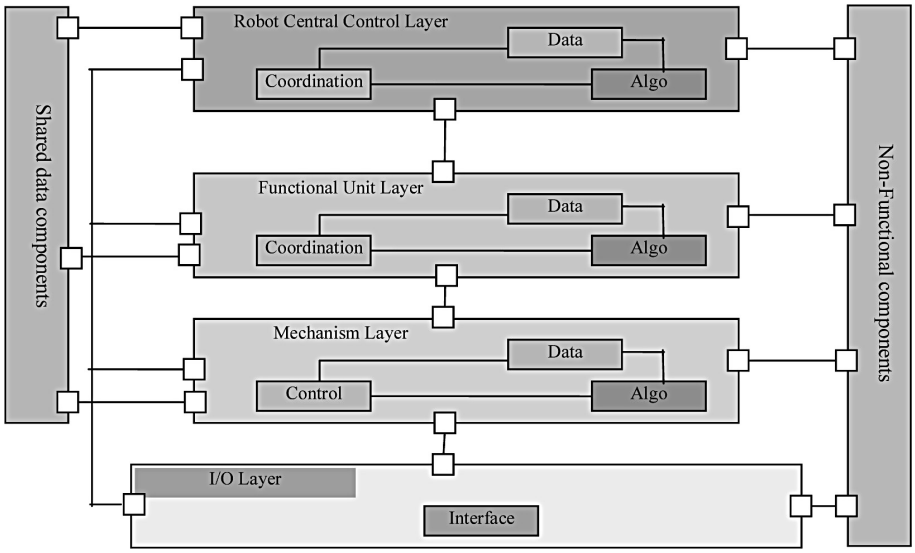

Figure 2. Layered HSR architecture.

of view, the specification of a robotic application must be modular, structured and accessible to users with different domain of expertise.

Therefore, while avoiding the predominant decomposition problem, we introduce architectural layers and organize the above mentioned components into these layers (see Figure 2).

Indeed, in an HSR, at the lowest level, we can find the actuators and sensors. At a higher level we find the controllers for simple actuators (for example a motor controller). At the next higher level, we find the controllers for groups of actuators (for example a motion card capable of controlling the joints of a mechanism) and so on. Many of these components can be found in the market, either as hardware devices and control cards or software packages for a given platform. We identify four layers of granularity at which the components can be defined:

- *I/O layer*: A component at this layer is of interface type. It abstracts the characteristics of I/O drivers that supervise and control the outside world, such as sensors and actuators. The sensors are components that provide the information required for controlling a given active element, for example, the encoder and switch limits associated to a given joint. Actuators model the simplest active elements, for example a motor.
- *Mechanism layer*: A component at this layer models the control over the actuators and the collection of data from sensors. For example, there will be mechanism component defined for controlling the joints of a given mechanism. This component is a composite one that may contain *control*, *data*, and *algorithmic* sub-components. Usually, a mechanism component needs to accomplish hard real-time requirements and, therefore, it is generally implemented in hardware. When it is implemented in software, it serves to impose severe real-time requirements.
- *Functional unit layer*: a component at this layer abstracts a homogenous functionality of entities that are related together.

To determine the functional unit components, the decomposition of the application is guided by the structure inherent in the application, in accordance with the proven architecture principle of "form follows function". A *control component* at this layer, defines control over a whole functional area (vehicle, manipulator, etc), which is defined by special constraints or homogeneous characteristics.

Besides control components, other components of type *data* and/or *algorithmic*, may be defined. For instance, the speeds of the motors are sensed using shaft encoders (interface component) and fed back to the embedded controller for computation of control signal (algorithmic or control component) to the DC motor (interface component) every 10 to 50 milliseconds using the proportional-integral control algorithm. The embedded controller component also monitors the robot environment using four infrared proximity sensors and switches (interface components).

Moreover, we may define at this layer, a *coordinator component* for managing an aggregation of mechanism components and coordinating their actions according to the command and information that it receives, as well as the coordination strategy that it comprises.

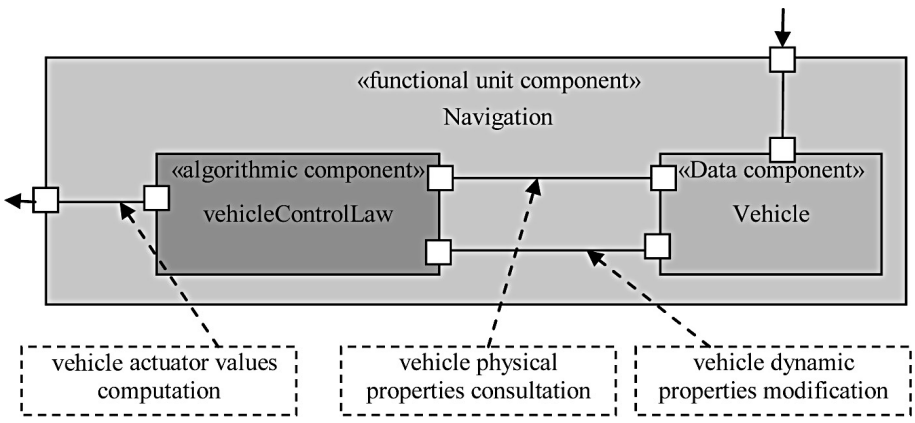

Figure 3. Simple example of a functional unit component.

This coordination strategy is an interchangeable part of that component. For example, the *CoordinationStrategy* of a given manipulator could be a given solution for its inverse kinematics, the coordinator strategy for a given vehicle could be a given navigation strategy, etc.

Figure 3 depicts an example of functional unit component, which contains two sub-components: a *robotVehicle* component having "data" as type and a *vehicleControlLaw* component having "algorithmic" as type.

- *Robot central control layer*: A component at this layer, models the control over a whole robot. For example, a robot composed of a vehicle with an arm and several interchangeable tools. This component is an aggregation of functional unit components and a global coordinator that generates the commands for these functional unit components and coordinates their actions, according to the order and the information that it receives and the coordination strategy that it comprises. Such strategy is an interchangeable part of the robot component. Also, the *robot central component* may be a container for three types of components: *data*, *coordination*, and *algorithmic* ones.

In addition, because usually there may be behavior and states that crosscut several entities, we model them as components of a separate layer, with possible interaction with components of the other layers. To this end, we define two other layers that are orthogonal to the aforementioned ones, as follows:

- *Non-functional requirements layer:* Our main goal is to preserve the reusability of entities and improve the comprehensibility, maintainability, evolution, etc. of the system. To do this, we keep "clean" the functionality of the software components composing a system, by separating the properties induced by the environment in which software entities are being used, such as the response time or security aspects. Those properties are called non-functional requirements (NFRs) that are related to crosscutting concerns [17,20].
- *Shared data components:* There may be common data to be used by components at different layers. To this end, we define a shared data component as a wrapper of data structure classes that provide acquisition, transformation, and storage of data.

A shared data component encapsulates states and behavior, which represent an abstract view of a physical entity. An internal state machine may be attached to a component at this layer to represent the behavior aspect.

Access to the states should be done through public interfaces. Components from other layers can query any state of such component type at any time. This allows for local state estimation based on information available within the shared data component, or provides resource usage prediction in response to queries from the components of other layers.

5. Conclusion

We have presented an approach that focuses on the definition of a generic component framework for the building of architectural components that can be reused in different home service robotic systems, and identifying software components based on the separation of concerns principle and interchangeable aspect inherent to the degree of technology evolution.

Developed for the domain of home service robots, the proposed generic architecture is defined according to various dimensions as layers representing different levels of computation. It avoids the predominant decomposition problems by flexibly combining a layered decomposition of the system with views and aspects. In particular, we have demonstrated how functional decomposition and non-functional ones can be combined systematically. Future work will concern the development of a software engineering environment based on our approach and the investigation of other categories of robotic systems.

References

[1] R. HÖPLER, P. J. MOSTERMAN, Model-integrated Computing in Robot Control Synthesize Real-time Embedded Code. In *proc. of IEEE Intl Conf. on Control Applications, CCA'01*, 2001.

[2] D. BRUGALI, P. SALVANESCHI, Stable Aspects in Robot Software Development. *International Journal of Advanced Robotic Systems*, Vol. 3, No. 1, pp. 017–022, 2006.

[3] M. KIM, S. KIM, M.-T. CHOI, M. KIM, H. GOMAA, UML-based Service Robot Software Development: A Case Study. ICSE, pp. 534–543. 2006.

[4] T. VALLIUS, J. RÖNING, A Telepresence Robot System Realized by Embedded Object Concept. *Proc. Intelligent Robots and Computer Vision XXIV: Algorithms, Techniques, and Active Vision*, October 3-4, Boston, Massachusetts, 2006.

[5] R. D. SCHRAFT, Mechatronics and Robotics for Service Applications. In *IEEE Robotics and Automation Magazine*, No. 4, pp. 31–35, December 1994.

[6] T. ROFER, A. LANKENAU, R. MORATZ, Service Robotics-applications and Safety Issues in an Emerging Market, Workshop W20, *proc. ECAI2000*, Berlin, 2000.

[7] B. YOU ET AL., Development of a Home Service Robot 'ISSAC'. *Proc. of the 1994 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, Nevada, pp. 2630–2635, 2003.

[8] D. GARLAN, Formal Modeling and Analysis of Software Architectures. Formal Methods for Software Architectures. M. Bernado, P. Inverardi (Eds.). Lecture Notes in Computer Science 2804. *Springer*. Berlin, Germany, pp. 1–24, 2003.

[9] N. MEDVIDOVIC, R. N. TAYLOR, A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1), pp. 70–93, January 2000.

[10] A. BROOKS, T. KAUPP, A. MAKARENKO, S. WILLIAMS, A. OREBÄCK, Towards Component-based Robotics. *Proceedings IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Canada, 2005.

[11] M. KIM, J. LEE, K. CH. KANG, Y. HONG, S. BANG, Reengineering Software Architecture of Home Service Robots: A Case Study. *International Journal of Advanced Robotic Systems*, Vol. 3, No. 1 ISSN 1729-8806, pp. 017-022, 2006.

[12] R. PASSAMA, D. ANDREU, COSARC: Component-based Software Architecture of Robot Controllers. In *1st National Workshop on Control Architecture of Robots: software approaches and issues*, Montpellier, 2006.

[13] M. J. BAKARI, K. M. ZIED, D. W. SEWARD, Development of a Multi-arm Mobile Robot for Nuclear Decommissioning Tasks. *International Journal of Advanced Robotic Systems*, Vol. 4, No. 4, 2007.

[14] F. KANEHIRO, H. HIRUKAWA, S. KAJITA, OpenHRP: Open Architecture Humanoid Robotics Platform. *International Journal of Robotics Research*, 23(2):155–165, 2004.

[15] B. ÁLVAREZ, A. IBORRA, J. A. PASTOR, C. FERNÁNDEZ, Software Architecture for Development of Mechatronic Systems: Service Robots. *Dedicated Systems Magazine*, 04, 2001.

[16] D. L. PARNAS, On the Criteria to be Used in Composing Systems into Modules, *Communications of the ACM 15*, 12, 1053–1058, 1972.

[17] P. OSSHER, H. HARRISON, W. SUTTON, N Degrees of Separation: Multi-dimensional Separation of Concerns. *Proc. of the 21st Intl Conf. on Software Engineering*, Los Angeles, 107–119, 1999.

[18] M. ZOBEL, J. DENZLER, B. HEIGL, E. NÖTH, D. PAULUS, J. SCHMIDT, G. STEMMER, MOBSY: Integration of Vision and Dialogue in Service Robots. *Machine Vision and Applications*, 1(14), pp. 26–34, 2003.

[19] M.-P. ROYER, A Qualitative Study of In-home Robotic Telepresence for Home Care of Community-living Elderly Subjects. *Journal of Telemedecine and Telecare*, 2006.

[20] A. NAVASA, M. A. PÉREZ, J. M. MURILLO, J. HERNÁNDEZ, Aspect-oriented Software Architecture: a Structural Perspective. Workshop on Early Aspects: Aspect-oriented Requirements Engineering and Architecture Design. *Aspect-oriented Software Development Conference*, April 2002.

Received: February, 2008
Revised: December, 2008
Accepted: February, 2009

Contact addresses:
Mohamed T. Kimour
University of Annaba
BP 12, 23000, Annaba, Algeria
e-mail: mtkimour@hotmail.fr

Ammar Bessam
University of JiJel
JiJel, Algeria
e-mail: bessamamar@yahoo.fr

Rachid Boudour
University of Annaba
BP 12, 23000, Annaba, Algeria
e-mail: racboudour@yahoo.fr

MOHAMED TAHAR KIMOUR received his Ph.D. in computer science from the University of Annaba (Algeria) in 2005. His research interests include embedded system design, software architecture, UML/SysML, and wireless sensor/actor networks.

AMMAR BESSAM received the M.S. degree in computer science from the University of Constantine (Algeria). He is interested in software design, software architecture description languages, and embedded systems.

RACHID BOUDOUR received his Ph.D. in computer science from the University of Annaba in 2006. He is the head of the research group on embedded systems co-design and co-verification. His research interests include embedded systems co-design, high-level synthesis and verification, and formal tools.