An Approach to Reusable Software for Mobile Robot Applications through Analysis Patterns

Dayang N. A. Jawawi, Safaai Deris
Department of Software Engineering
Faculty of Computer Science and Information System
Universiti Teknologi Malaysia
81310 UTM, Skudai, Malaysia
dayang@fsksm.utm.my

Rosbi Mamat
Department of Mechatronics and Robotics Engineering
Faculty of Electrical Engineering
Universiti Teknologi Malaysia
81310 UTM, Skudai, Malaysia

Abstract

The use of software analysis patterns as a means to facilitate Autonomous mobile robots (AMR) software knowledge reuse through component-based software engineering is proposed. The software analysis patterns for AMR were obtained through a pattern mining process, and documented using a standard catalogue template. These analysis patterns are categorized according to hybrid deliberate layered architecture of robot software: reactive layer, supervisor layer and deliberative layer. In this paper, the analysis patterns in the reactive layer are highlighted and presented. The deployment of the analysis patterns are illustrated and discussed using an AMR software case study. The reuse potential of these patterns is evaluated by measuring the reusability of components in the analysis patterns.

1. Introduction

Autonomous mobile robots (AMR) have found their applications in industries, education and research. It represents a mechatronics system, which involves expertise from multi-disciplines in the domains of artificial intelligence, mechanical, electronics, computer and software engineering to develop it. The software aspect of AMR has been recognized as the most difficult and challenging part [1,2] for fully functional and successful AMR. Developing software for AMR requires knowledge in embedded systems, real-time software issues, control theories and artificial intelligence aspects. Thus, reusing existing knowledge from previous projects can significantly reduce the efforts and speeding up the AMR software development process. A widely accepted solution to this is software reuse, in the form of components, framework or software patterns.

Software patterns are used to identify recurring problems and describe a generalized solution to the problems and help software developers to understand how to create an appropriate solution, giving certain domain-specific problem. Software patterns can be categorized according to three software development levels: analysis patterns or conceptual patterns for analysis level, design patterns for design level and programming patterns for implementation level [3].

Previously, robotics research communities had recognized and practiced software reuse in general robotics software. The reuse approaches include reuse architecture, framework, design patterns, code or library components. However, the use of software patterns in robotics research communities is limited to design patterns only. Software analysis pattern has not yet received much attention. Since the focus of this work is on reuse of domain specific knowledge for analysis of AMR software, it was decided that analysis pattern is more appropriate here.

The focus of this paper is on analysis patterns as a means to facilitate AMR software knowledge reuse. The main reasons for concentrating on analysis patterns are: 1) analysis patterns speed up the development of abstract analysis models that capture the main requirements of the concrete problem by providing reusable analysis models [4]; 2) due to multi-disciplines nature of AMR software, conceptual models of experts knowledge in a particular domain can be captured independently using analysis patterns; 3) analysis patterns can served as basis for development of components and framework.

The main objectives of this paper are: to present AMR analysis pattern and some important components in the software analysis pattern as a result of our works and experience in AMR software requirements; to illustrate how the AMR software analysis patterns can be used for analysis and early design of AMR software; and to measure the reusability of components from the analysis pattern.

This paper is organized as follows. Section 2 describes the pattern mining process and catalogues some important AMR analysis patterns in a properly documented form. Section 3 illustrates the deployment of the AMR analysis patterns for analysis and design of AMR software. In Section 4 the reusability of the components from the analysis pattern were measured using a metrics suite [5]. Finally, the conclusion is presented in Section 5.

2. Analysis Pattern for AMR Software

In this paper, the use of analysis pattern for system level view of AMR software is proposed. The AMR analysis pattern consists of components and each component's pattern acts as a unit of analysis and the pattern will facilitate the transformation of the analysis model into design model.

The software analysis patterns for AMR were obtained through a pattern mining process, and

documented using a standard catalogue template. Cataloging these analysis patterns involves two main processes: pattern mining process and documenting the analysis pattern and the pattern's components. These processes are described as follow.

2.1 Analysis patterns mining process Pattern mining process concerns with identification and documentation of patterns. The patterns mining process in this work is based on studies of numerous AMR systems from books such as [1, 6], existing AMR software architectures [7, 8], and experience from research works on AMR systems at the Universiti Teknologi Malaysia (UTM). Existing embedded and real-time design patterns [9] is also analyzed in this process.

As a result of this pattern mining process, currently ten software components were identified in the analysis pattern for typical AMR software. The components identified are: input-output, actuator, sensor, signal processing, motor control, communication, Human-Robot Interface (HRI), Behavior-Based Control (BBC), coordinator and planner. These components are categorized according to hybrid deliberate layered architecture of robot software.

The relationships between the components of the analysis pattern in reactive layer are shown in Fig 1. All the components in the reactive layer are organizes into a hierarchical organization based on their level of abstraction. This subsystem organization of AMR domain represents the architecture pattern of the systems. Each subsystem will be treated as component-based analysis pattern to model the detail requirement of each domain.

2.2 Defining Analysis pattern at reactive layer At this stage we are focusing on defining patterns in reactive layer using behavior-based control intelligent.

The analysis pattern is documented based on template for documenting analysis pattern as proposed in [4]. The documentation of the AMR analysis pattern includes the *name* of the pattern, the *intent* of the pattern, the *motivation* for the pattern, the *forces* resolved by the pattern, the *solution* brought by the pattern, the *consequences* of the pattern, the important *design* points to be considered, and the *known uses* of the pattern.

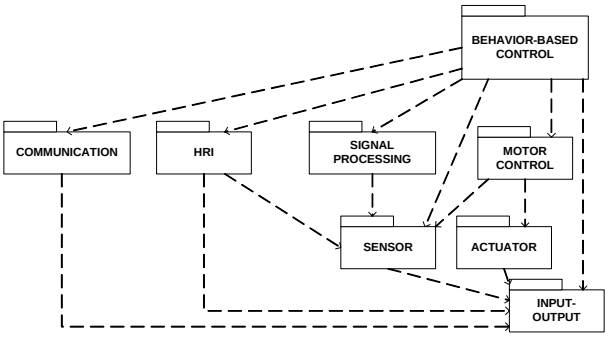


Fig. 1: The structure of reactive layer

2.3 Components of the AMR analysis pattern The essential information in components of AMR analysis pattern are cataloged base on guidelines of Gamma *et al.* [10] and Douglass [9]. The components of the analysis patterns are documented using six essential elements: *name* – reference to the component patterns; *context* – description of the context of the problem identified and the solution presented; *problem* – statement of problem solve by the component patterns; *solution* – structural solution presented using class diagram, showing the elements and properties in the component pattern, and interface to enable the component pattern to communicate with other components; *related pattern* – component patterns that may relate or will have interaction to this pattern during composition process; *example of reuse component* – name of components that can be reused in and with the component pattern. Fig. 2 shows the catalogue for the BBC component pattern documented using these six essential elements.

The Unified Modeling Language (UML) structural elements and diagrams were adopted in describing the *solution* element of the patterns as UML provide a convenient and a lingua franca graphical representation in industry and academic software practice. Even though the *solution* is described using object-oriented technique, its implementation or realization need not be in object-oriented approach.

BEHAVIOR-BASED CONTROL

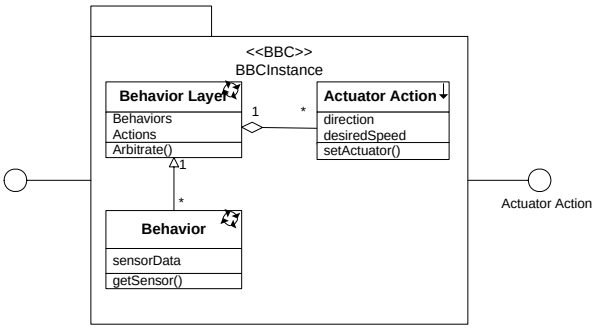
Context

A mobile robot will have multiple behaviors, to react to the robot's environment while trying to achieve its' goal. Behavior-based control will provide intelligent to select the parallel behaviors.

Problem

How to provide intelligence to decide on which behavior to select depending on the environment and the robot's goal.

Solution



- *Behavior Layer* – Responsible to select action based on the fixed behavior-based architecture layers of the robot behaviors.
- *Behavior* – A definition of a behavior in the robot system.
- *Action* –Action that will affect the robot's actuator.

Related Patterns

Motor control, sensor, poster, coordinator, input-output.

Example of Reuse Components

Subsumption, motor schema.

Fig. 2: Behavior-based control pattern catalogue

The component pattern *solution* is described using both *structural model* and *real-time behavior model*. The structural model describes classes that make up a particular component pattern. The combination of classes in the structural model is arranged in a package to represent constructional component pattern. Interconnection between the packages or components in the analysis pattern is supported by interfaces which are defined in the analysis pattern solution. The result of wiring the pattern components using these interfaces is the functional structure or *structural model* of the AMR software system. The idea of packages to represent constructional pattern and definition of pattern interface in describing structural model were both available in the Pattern-Oriented Analysis and Design methodology [11].

For a real-time system such as the AMR software, the functional structure description using structural model alone is not enough. A *real-time behavior model* is required to specify the real-time behavior of real-time components in the software. The real-time behaviors for classes are categorized in four: passive class, active class, event class, and implementation dependence class. A passive class is a class that does not has its own thread of control, and it is marked with a stereotype "↓". An active class is a class with its own thread of control, and it is marked with a stereotype "". An event class is an active class whose behavior in triggered by event, and it is marked with a stereotype "". An implementation dependence class is a class whose real-time behavior can only be specified or decided during the implementation phase of the pattern depending on the application. This class is not marked with any stereotype. As illustrated in Fig. 2 for the BBC component pattern, the Behavior Layer and Behavior classes are active classes while the Actuator Action is a passive class.

In the documentation of the AMR component patterns, the typical reusable components in each pattern are also suggested. This will facilitate the deployment of any existing reusable black box or white box components in that particular pattern. For example Proportional-Integral-Derivative (PID) Controller design patterns from [12] can be used as white box component solution in AMR Motor Control component pattern.

3. Deployment of the Analysis Patterns in Component-Based Development

Based on two main tasks of analysis patterns proposed by [4], the AMR analysis patterns tasks are: 1) to speed up the analysis of structural model and identify the real-time behavior of each object in the structural model at analysis level, and 2) to facilitate the transformation of structural analysis model into design model by suggesting reuse component that can be used to solve the identified problems in the analysis model. The deployment of the AMR software analysis patterns is illustrated using an AMR case study to show how the analysis pattern performs its tasks in Component-Based Development (CBD).

The AMR considered in this case study is a wheeled AMR, capable of traversing in a structured environment, which is surrounded by walls. The goal of the robot software is to navigate the robot in finding a passage and exiting through the passage while avoiding obstacles during its motion. The AMR consists of a body and a pair of wheels. Each drive wheel is move by a Direct-Current (DC) motor. The speeds of the motors are sensed using shaft encoders and fed back to the on-board embedded controller for computation of control signal to the DC motors every 100 milliseconds using the Proportional-Integral (PI) control algorithm. The embedded controller also monitors the robot environment using four Infra Red (IR) proximity sensors and a distance sensor.

The embedded software must support the intelligence aspect of the robot in order to response to the conditions in the environment in achieving the goal. The intelligence of AMR is supported by a behavior-based control using subsumption architecture [13]. To support concurrent behavior in subsumption architecture and to satisfy the multi-tasking requirements for these major tasks, a pre-emptive Real-Time Operating System (RTOS) is used in the robot software. The embedded controller also communicates with human through Liquid Crystal Display (LCD) and switches.

3.1 The AMR software analysis and early design using POAD methodology The POAD methodology is used in the analysis and early design of the AMR software using the software analysis patterns. The choice of POAD methodology is due to several reasons: 1) POAD takes structural composition approach to glue patterns at high-level; 2) POAD provides logical views to represent AMR application analysis and design as a composition of the patterns; and 3) POAD provides the necessary means to trace participants of those patterns into the application's final class diagram.

In this analysis phase, suitable candidates from the AMR analysis patterns that could capture the main requirements of the problem are identified. The AMR software requirements were modeled using the UML use-case diagram as shown in Fig. 3.

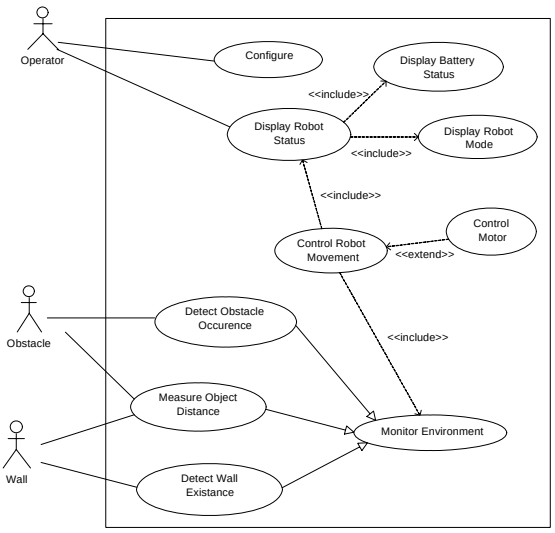


Fig. 3. The AMR use case diagram

By matching the decomposed use-case of Fig. 3 with the context and problem elements available in AMR analysis patterns, a mapping of the application requirements and the AMR analysis patterns is obtained. From this, a POAD Pattern-Level Diagram which specifies the AMR pattern instances and their relationships for this case study is developed as shown in Fig 4.

In CBD, interfaces are the means by which components connect. The composition of the components using AMR analysis patterns is supported by interfaces defined in the analysis pattern solution.

The relationship between patterns instances in Fig. 4 can be further detailed out to lower-level design relationships using interfaces in Pattern-Level with Interface diagram. For example, Fig. 5 shows a section of the Pattern-Level with Interface diagram relating the four packages involve in the AMR intelligence behavior: Switches, IR Distance Sensor, IR Proximity Sensor, and Subsumption Architecture.

Once the Pattern-Level with Interface diagrams similar to Fig. 5 were obtained, each of the generic analysis patterns in the diagram needs to be renamed, classes in the pattern need to be detailed out according to the specific AMR system, and the tracing of pattern interfaces to internal classes need to be defined. Fig. 6 shows the results obtained from Fig. 5 following those processes. All function and classes defined as the pattern interfaces are connected directly to show the relationship between the internal classes in each pattern.

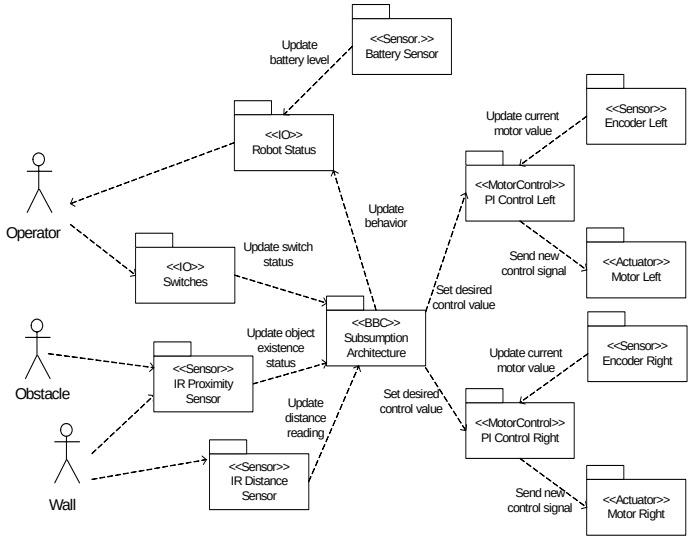


Fig. 4. Pattern-level diagram for the AMR software

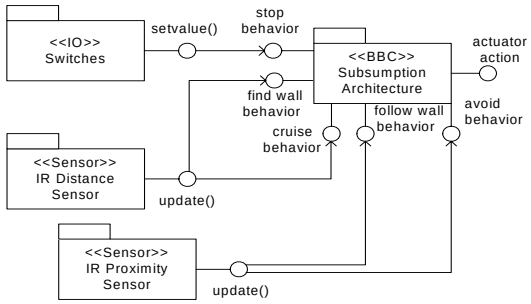


Fig. 5. Pattern-level with interface diagram for AMR intelligence behavior

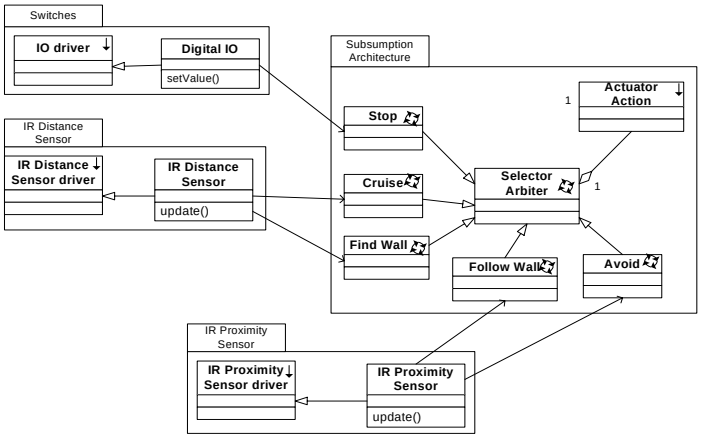


Fig. 6. Detail internal classes representation of the AMR for intelligence behavior

Concurrency and multitasking capabilities of the AMR software are supported by the RTOS. The real-time behavior of AMR components specified in Figure 6, which require the RTOS services has to be wired to a concurrency or RTOS design pattern as proposed in [9]. Due to lack of support for real-time in POAD, we introduced the *control interface* for wiring real-time components to the RTOS design pattern as shown in Fig. 7. The control interface defines the attributes of real-time requirements of a pattern component, and this is only necessary in active and event classes. The control interface, however, is not explicitly showed in a pattern solution, since services required from RTOS design pattern can only be specified during the wiring of pattern components.

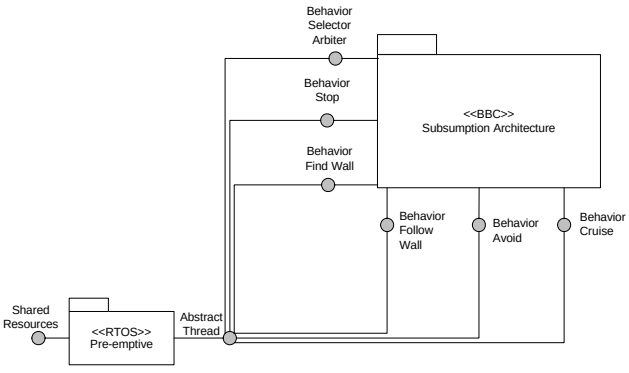


Fig. 7. Pattern-level with control interface diagram for intelligence behavior

The detail internal classes of Fig. 6 acts as the initial class diagram for static design model of the AMR software. Up to this point, POAD methodology provides logical views to represent AMR application analysis and design as a composition of the patterns using structural elements of UML. Fig. 7 enhances the static design model of Fig. 6 by detailing the real-time concurrency support required in the design model.

Once the detail software behaviors of the AMR are defined, the software implementers can choose any appropriate ways and suitable programming languages for implementing the AMR software. For this case study the software implementation process includes writing functions, modules and tasks in C programming

language, program translation into executable code, testing and debugging. The software tools used for the software implementation are Paradigm C/C++ compiler [14] for generating ROMable code and µC/OS-II real-time kernel [15] for multitasking support.

4. Reusability of the Analysis Components

To quantify the benefit of the used of the analysis pattern, the reusability of the components from the analysis pattern is measured using a metrics suite proposed by Washizaki et al. [5]. This metrics suite is suitable for measuring the usability of black-box components in analysis patterns without the availability of source codes.

The metrics suite consists of five metrics originally for JavaBeans component reusability assessment. The metrics suite however, was modified for assessing the components of the AMR analysis pattern. The main modification is made in trying to match the Facade class features in the original metrics with the interfaces proposed in each components of the AMR analysis pattern.

Three relevant metrics were considered in the measurement process. These metrics are: observability, customizability and external dependency. The description of these metrics as adapted for components of the AMR analysis pattern are as follow.

(1) Observability

Objective: To measure how easy a component in terms of its operational behavior, input parameters and output parameters.

Definition: Rate of Component Observability (RCO), *RCO(c)* is a percentage of readable attributes in all attribute implemented within the interface class of a component.

Formula:

$$RCO(c) = \begin{cases} \frac{\Pr(c)}{A(c)} & (A(c) > 0) \\ 0 & (otherwise) \end{cases}$$

where,

Pr(c) : number of readable attributes in c

A(c) : number of attributes in c's interface class

(2) Customizability

Objective: Indicates the built-in capability for supporting the customization and configuration of a component's internal functional features.

Definition: Rate of Component Customizability (RCC), *RCC(c)* is a percentage of writable attributes in all attribute implemented within the interface class of a component.

Formula:

$$RCC(c) = \begin{cases} \frac{Pw(c)}{A(c)} & (A(c) > 0) \\ 0 & (otherwise) \end{cases}$$

where,

Pw(c) : number of readable attributes in c

A(c) : number of attributes in c's interface class

(3) External dependency

Objective: Indicates the component's degree of independence from the rest of the software which originally used the component.

Definition 1: Self-Completeness of Component's Return Value (SCCr), *SCCr(c)* is a percentage of methods without any return value in all method implemented within a component c.

Formula:

$$SCCr(c) = \begin{cases} \frac{Bv(c)}{B(c)} & (B(c) > 0) \\ 1 & (otherwise) \end{cases}$$

where,

Bv(c) : number of methods without return value in c

B(c) : number of methods in c

Definition 2: Self-Completeness of Component's parameter (SCCp), *SCCp(c)* is a percentage of methods without any parameter in all method implemented within a component c.

Formula:

$$SCCp(c) = \begin{cases} \frac{Bp(c)}{B(c)} & (B(c) > 0) \\ 1 & (otherwise) \end{cases}$$

where,

Bp(c) : number of methods without parameter in c

B(c) : number of methods in c

The proposed metrics were applied to six components of AMR analysis patterns. These six components were used in the analysis and design composition as shown in previous Section 3.1. For each component the value of adapted Washizaki et al.'s metrics can be computed as tabulated in Table 1.

Table 1. Washizaki et al.'s metrics applied on six components of the AMR analysis pattern

Components	RCO	RCC	SCCr	SCCp
Input	1	0	1	1
Output	0	1	1	1
Sensor	1	0	1	1
Actuator	0	1	1	1
Motor Control	0.25	0.75	1	1
BBC	0.66	0.33	1	1

From Table 1, it can be concluded that: (1) the observability of component Input and Sensor is very high; (2) the customizability of component Output and Actuator is very high; and (3) External dependency of all components are very high.

High observability for Input and Sensor components, and high customizability for Output and Actuator are due to the readable and writable attributes in the component are provided by the interface of the components. However, if the observability measurement is too high, its will lead to difficulty for users to find important readable properties from the interface, and if the customizability measurement is too high, its will leads to high possibility of misuse of components [5].

In the AMR domain it is important for the Input and Sensor components to have very high obsevability, and Output and Actuator components to have very high customizability, since, the main objective of the components are to observe its environment from hardware reading and to configure the hardware in order to react in environment.

The high external dependency of five components of AMR analysis patterns are due to the implementation of the operation in the interface class are without the use of parameters or return values, this lead to self-completed within component.

5. Conclusions

The use of software analysis patterns as a means to facilitate AMR software knowledge reuse through component-based software engineering is proposed. The software analysis patterns for AMR were obtained through a pattern mining process, and documented using a standard catalogue template.

Based on this AMR software analysis pattern, the pattern level analysis and design of AMR software case study using the Pattern-Oriented Analysis and Design (POAD) methodology is illustrated and discussed.

The results of this pattern level analysis and design is the initial class diagram for static design model of the AMR software system. Once the detail internal classes' representation of AMR software is obtained, it will serve as best starting point for two groups of software implementer: 1) application software engineer who can easily implementing the AMR software in any way, not necessarily based on CBD; 2) software engineer who develops component can develop black box or white box version of components, which can later be used by application software engineer to compose the AMR software based on the black box or white box components.

The reuse potential of the analysis pattern is evaluated by measuring the reusability of components in the analysis patterns using a metrics suite. From the measurement the reusability of the component in the patterns are found to be high. From this result it is believe that further detail research on the benefits of patterns as a means to reuse domain knowledge is needed in domain such as AMR software.

References

[1] Braunl. T., Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems, Springer-Verlag, New York, 2003.

[2] Seward, D.W. and Garman, A., "The Software Development Process for an Intelligent Robot", IEEE Computing and Control Engineering Journal, vol. 7, no.2, (1996), pp. 86 –92.

[3] Riehle, D., and Zullighoven, H., "Understanding and Using Patterns in Software Development", Theory and Practice of Object Systems, vol. 2, no. 1, (1996), pp. 33-13.

[4] A. Geyer-Schulz and M. Hahsler, "Software reuse with analysis patterns", In Proceedings of the 8th Association for Information Systems (AMCIS), Dallas, TX, (2002), pp. 1156-1165.

[5] Washizaki, H., Yamamoto, H., Fukazawa, Y., "A Metrics Suite for Measuring Reusability of Software Components", Proceedings of the Ninth International Software Metrics Symposium 2003, (2003), pp. 211 – 223.

[6] Jones, L.J., Seiger, B.A., and Flynn, A.M., Mobile Robots Inspiration to Implementation, Second edition. A K Peters, Natick, 1999.

[7] A. Oreback, and H. I. Christensen, "Evaluation of Architecture for Mobile Robotics", Autonomous Robots, vol. 14, pp. 33-49, 2003.

[8] Alami, R., Chatila, R., Fleury, S., Ghallab, M. and Ingrand, F., "Architecture for Autonomy", Journal of Robotics Research, vol. 17, no. 4, (1998), pp. 8315-337.

[9] Douglass, B. P., Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems, Addison Wesley, Boston, 2002.

[10] Gamma, J., Helm, R., Johnson, R. and Vlissides, J., Design Patterns: Elements of Reuse Object-Oriented Software, Addison-Wesley, Reading, 1995.

[11] Yacoub, S. M. and Ammar, H. H., Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems, Addison-Wesley, Boston, 2004.

[12] Pont M.J., and Banner, M.P., Designing embedded systems using patterns: A case study**,** Journal of Systems and Software 71(3), (2004), pp. 201-213.

[13] Brooks, R.A., A Robust Layered Control System for a Mobile Robot, IEEE Journal of Robotics and Automation, vol. RA-2, no. 1, (1986.), pp. 14-23.

[14] Paradigm C++ Reference Manual Version 5.0, Paradigm Systems, Endwell, 2000

[15] Labrosse, J. J.. MicroC/OS-II The Real-Time Kernel, 2nd edition, R&D Books, USA , 1999.