Software Reuse for Mobile Robot Applications Through Analysis Patterns

Dayang Jawawi[1], Safaai Deris[1], and Rosbi Mamat[2]
[1]Department of Software Engineering, Universiti Teknologi Malaysia, Malaysia
[2]Department of Mechatronics and Robotics Engineering, Universiti Teknologi Malaysia, Malaysia

Abstract: *Software analysis pattern is an approach of software reuse which provides a way to reuse expertise that can be used across domains at early level of development. Developing software for a mobile robot system involves multi-disciplines expert knowledge which includes embedded systems, real-time software issues, control theories and artificial intelligence aspects. This paper focuses on analysis patterns as a means to facilitate mobile robot software knowledge reuse by capturing conceptual models in those domains in order to allow reuse across applications. 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. Particularly, 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. To verify the existence of the pattern in AMR systems, pattern-based reverse engineering was performed on two existing AMR systems. The reuse potential of these patterns is evaluated by measuring the reusability of components in the analysis patterns.*

Keywords: *Analysis pattern, software reuse, component-based development, pattern-based reverse engineering.*

Received December 2, 2005; accepted March 3, 2006

1. Introduction

Autonomous Mobile Robot (AMR) 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 one of the challenging part [3, 22] for fully functional and successful AMR application. 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 through software reuse, in the form of components, architecture, framework, and 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 [25]. 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 [21].

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 [11].
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 AMR 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; to present the results of pattern-based reverse engineering process on two existing AMR software in order to verify the existence of the analysis pattern components in the software; and to measure the reusability of components from the analysis pattern.

This paper is organized as follows. In section 2, some other approaches of software reuse in robotics

software are reviewed. Section 3 describes the pattern mining process and catalogues some important AMR analysis patterns in a properly documented form. Section 4 illustrates the deployment of the AMR analysis patterns for analysis and early design of AMR software. The pattern-based reverse engineering process on two existing AMR applications is discussed in section 5. In section 6, the reusability of the components from the analysis pattern were measured using a metrics suite. Finally, the conclusion is presented in section 7.

2. Related Work

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.

Virtual Robot Framework (VRF) [23], NEXUS [7] and GenoM [15] are examples of frameworks proposed to support abstraction and components reuse in robotics. VRF provides an abstraction layer to limit the effects of diversity and lack of standardization of robot hardware, while NEXUS and GenoM frameworks were proposed for integrating software elements in robotics. These three frameworks assume that the software analysis phase or the conceptual level of the software is already defined before the framework can be used.

A number of software architectures was proposed to help in viewing a clear logical structure and components of the robotic software. CLARAty [17] architecture aims at developing flexible and reusable software components for robotic systems through architectural decomposition of a generic robotic system. Alami *et al.* [1] proposed three hierarchical levels architecture called LAAS, for mobile robot systems. OSCAR [2] is a component-based architecture for exploration of indoor environments with AMR. These three architectures are all targeted to be reused at architecture design level, in contrast, the analysis pattern discussed in this paper aims to be reused at analysis level.

Component reuse of AMR software has been used in industries. Sony's AIBO entertainment mobile robots were claimed to be developed using OPEN architecture (OPEN-R) component approach developed at Sony as a standard for hardware and software components interfaces in entertainment robotics by providing off-the-shelf components and basic robot systems [9]. However, since it is a Sony's commercial proprietary approach, the details of OPEN-R components were not available. Mobility [20] is another commercially available component-based system produced by Real World Interface (RWI) Company to support certain classes of mobile robot platforms which are produced by RWI.

Software patterns have also previously been used in robotics software. Graves and Czarnecki [12] used design patterns for behavior-based robotics systems focusing mainly on the area of man-machine interaction. Nelson [16] developed a design pattern for creating software control systems of autonomous or robotic vehicles.

Currently, the use of software patterns in robotics research communities is limited only to design patterns. 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, analysis pattern will serve this purpose appropriately. Based on two main tasks of analysis patterns proposed by [11], 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.
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.

3. Analysis Pattern for AMR Software

The analysis pattern proposed here aims to develop the conceptual level of the robot software. The pattern is to guide the software designer to develop the perception of AMR application domain and help the designer to understand the domain. The AMR analysis pattern consists of components and each pattern's component 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, then the analysis patterns and the associated pattern's components are documented using a standard catalogue template. These processes are elaborated in the following sections.

3.1. 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 [3, 13], existing AMR software architectures [1, 2, 17, 18], and experience from research works on AMR systems at the Universiti Teknologi Malaysia (UTM). Existing embedded and real-time design patterns [6] are 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.

3.2. Defining Analysis Pattern at Reactive Layer

At this stage we are focusing on defining patterns in reactive layer using behavior-based intelligent control approach, since, software at reactive layer is typically embedded onboard and constrained by limited resources and real-time requirements. The analysis pattern is documented based on template for documenting analysis pattern as proposed in [11]. An example of the documentation of the AMR analysis pattern is described in Appendix A.

3.3. Components of the AMR Analysis Pattern

The essential information in the components of AMR analysis pattern is catalogued based on guidelines of Gamma *et al.* [10] and Douglass [6]. The components of the analysis patterns are documented using five essential elements:

1. *Name:* Reference to the component patterns.
2. *Context*: Description of the context of the problem identified and the solution presented.
3. *Problem*: Statement of problem solve by the component patterns.
4. *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.
5. *Example of Reuse Component*: Name of components that can be reused in and with the component pattern.

Figure 1 shows the catalogue for the BBC component pattern documented using these five 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.

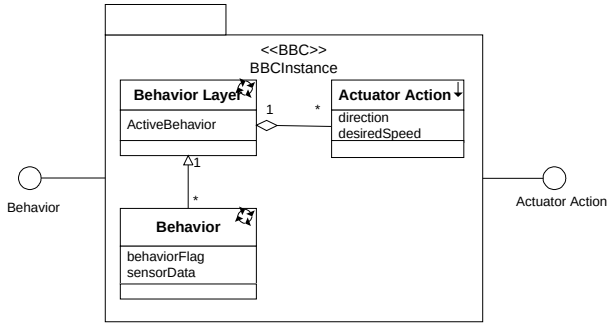
The component pattern solution is described using both structural model and real-time behavior model. The structural model describes classes that make up a particular component pattern. The combination of classes in the structural model is arranged in a package to represent constructional component pattern. Interconnection between the packages or components in the analysis pattern is supported by interfaces which are defined in the analysis pattern solution. The use of packages to represent constructional pattern and definition of pattern interface in describing structural model were adopted from the pattern-oriented analysis and design methodology [26].

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.

Example of Reuse Components: Subsumption, motor schema.

Figure 1. Behavior-based control pattern catalogue.

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 Figure 1 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 as suggested by domain experts in each component pattern are also proposed. This will facilitate the deployment of any

existing reusable black box or white box components in that particular pattern's component.

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

The deployment of the AMR software analysis patterns is illustrated using an AMR case study to show how the analysis pattern is deployed in a 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 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 which is based on AMD80C188ES microcontroller for computation of control signal to the DC motors every 50 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 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 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 [4]. 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.

4.1. The AMR Software Analysis Using POAD Methodology

The Pattern-Oriented Analysis and Design (POAD) [26] 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.
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 Figure 2. By matching the decomposed use-case of Figure 2 with the context and problem elements available in AMR analysis patterns, a mapping of the application requirements and the AMR analysis patterns are 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 Figure 3.

4.2. The AMR Software Early Design Using POAD Methodology

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 as shown in Figure 3 can be further detailed out to a lower-level design relationship using interfaces in pattern-level with interface diagram. For example, Figure 4 shows a section of the pattern-level with interface diagram relating the four packages involve in the AMR intelligence behavior: Switches, IR distance sensors, IR proximity sensors, and subsumption architecture.

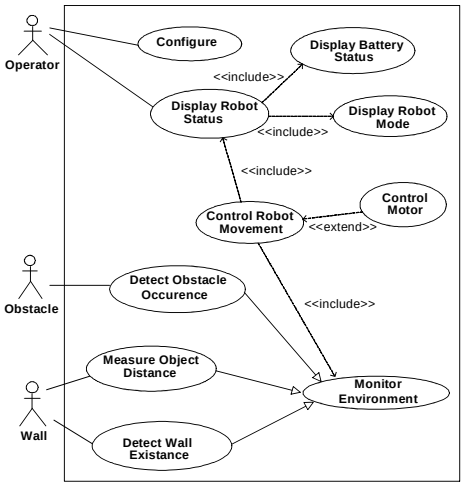


Figure 2. The AMR use-case diagram.

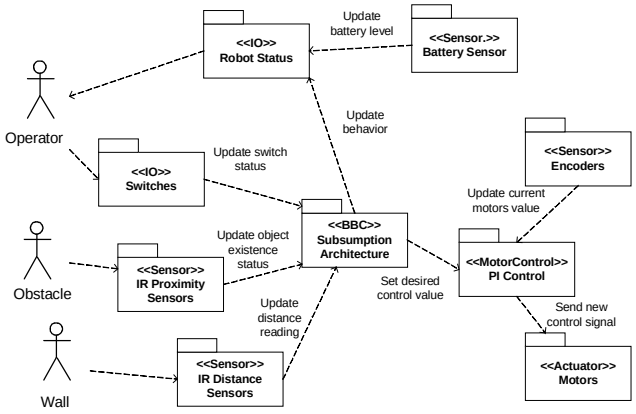


Figure 3. Pattern-level diagram for the AMR software.

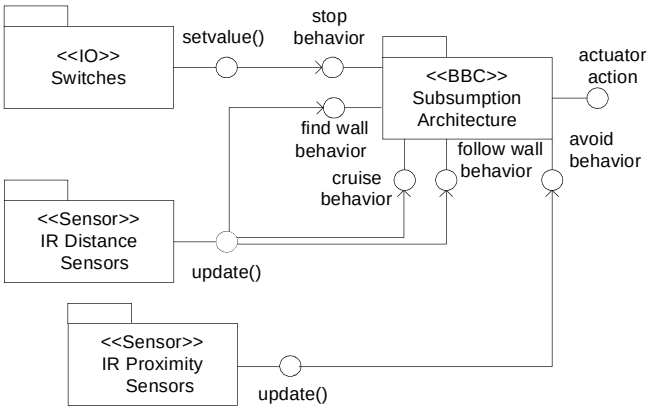


Figure 4. Pattern-level with interface diagram for AMR intelligence behavior.

Once the pattern-level with interface diagram similar to Figure 4 is 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. Figure 5 shows the results obtained from Figure 4 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.

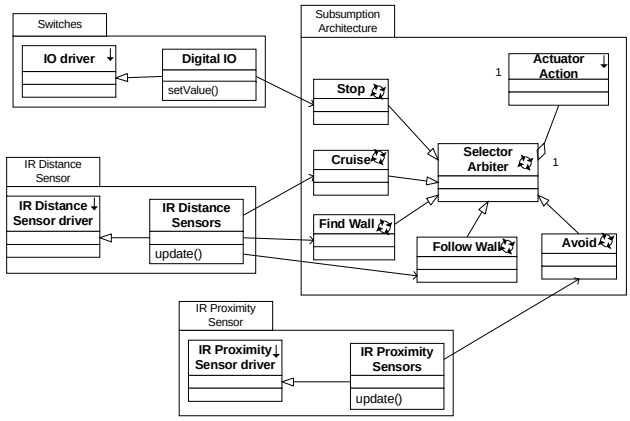


Figure 5. Detail internal classes representation of the AMR for intelligence behaviour.

Concurrency and multitasking capabilities of the AMR software are supported by the RTOS. The real-time behavior of AMR components specified in Figure 5, which require the RTOS services has to be wired to a concurrency or RTOS design pattern as proposed in [6]. Control interface is introduced for wiring real-time components to the RTOS design pattern as shown in Figure 6. The control interface defines the attributes of real-time requirements of a component pattern, 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 components pattern at design stage.

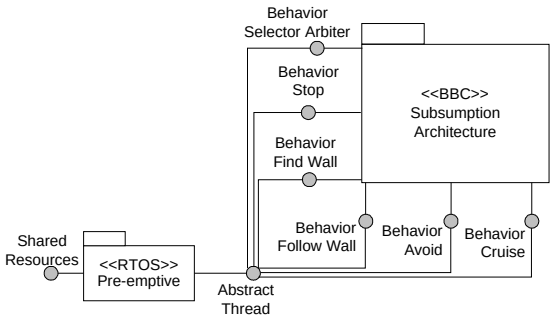


Figure 6. Pattern-level with control interface diagram for intelligence behavior.

The detail internal classes of Figure 5 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 components using structural elements of UML. Figure 6 enhances the static design model of Figure 5 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 [19] for generating ROMable code and µC/OS-II real-time kernel [14] for multitasking support.

5. Pattern-Based Reverse-Engineering on Two Existing AMR Software

Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships, and create representations of the system in another form or at a higher level of abstraction [5]. A pattern-based reverse engineering process was performed on two existing AMR software to obtain higher abstractions and document their structural description using the proposed AMR analysis patterns. This will verify the existence of the patterns in AMR systems.

The reverse engineering was performed on a Universiti Teknologi Malaysia (UTM) intelligent AMR software and a Fire Marshal Bill AMR software [8] to gain the graphical analysis representation of the software, based on the AMR analysis pattern. The first AMR software is the leader agent's code as a part of our own multi-agent mobile robots software. The real-time behaviors of the software are supported using a cooperative RTOS and the intelligent control is implemented using subsumption behavior-based

intelligent architecture. The second robot software is the Fire Marshal Bill balancing robot which uses the Real Time Executive for Multiprocessor Systems (RTEMS) RTOS to support multitasking. The robot intelligent was implemented using a non-behavior-based architecture.

The two robot software were selected due to several reasons:

1. The C codes for both software were accessible to us.
2. Both software were developed without using the proposed analysis pattern.
3. Both software were implemented using the traditional non-object-oriented approach.
4. The first software was based on behavior-based approach which represents common AMR software architecture, thus matched with the proposed analysis patterns.
5. The second software was not implemented using the behavior-based approach, thus posed a challenge in documenting it using the proposed analysis patterns.

To preserve the design history of the codes, the reverse engineering processes were performed manually. These reverse engineering processes did not consider in detail the reused components utilized in the codes such as the cooperative RTOS and some hardware interfaces code, as these components are already in the reuse forms. The results of the pattern-based reverse engineering are pattern level diagrams of the AMR systems. Figure 7 and Figure 8 show the pattern-level diagrams for the Fire Marshal Bill and UTM intelligent AMR, respectively.

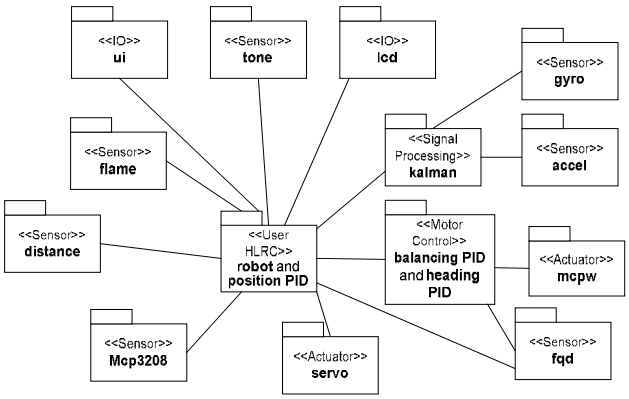


Figure 7. Pattern level diagram for the Fire Marshal Bill robot.

Table 1 summarized the number of pattern instances component from the AMR analysis pattern. The UTM intelligent AMR software which is based on behavior-based approach can be directly mapped into the AMR analysis pattern. All the components in the software are the component from the proposed patterns. For the Fire Marshal Bill AMR software, only a component called robot and position PID which handle the robot high-level intelligence does not match with the proposed patterns. This proved that this analysis pattern still can be used for analysis of non behavior-based approach with only a few of the components cannot be mapped directly with the AMR analysis patterns.

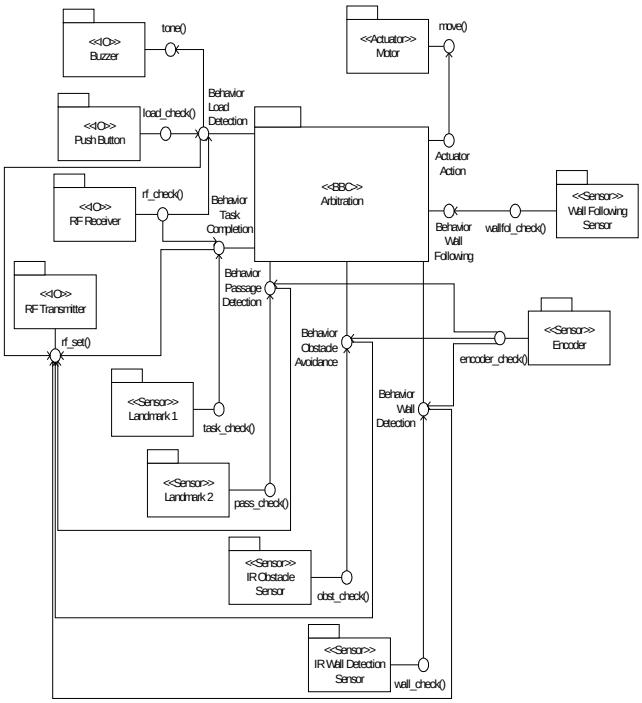


Figure 8. Pattern-level with functional interface diagram for the intelligent AMR.

Table 1. Number of components from the pattern used.

Analysis Component	Fire Marshal Bill	UTM Intelligent AMR
Input-Output	2	4
Sensor	7	6
Actuator	2	1
Motor Control	1	0
BBC	0	1
Signal Processing	1	0

6. Reusability of the Analysis Components

To quantify the benefit of using the analysis pattern, the reusability of the components from the analysis pattern is measured using a metrics suite proposed by Washizaki *et al.* [24]. 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 is utilized for assessment of pattern level reuse of the analysis pattern's component.

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 criteria measured by the metrics are: Observability (RCO), customizability (RCC) and external dependency (SCC), as defined in Table 2.

Table 2. Washizaki *et al.'s* metrics definition.

Metric Name	Definition
Rate of Component Observability (RCO)	A percentage of readable attributes in all attribute implemented within the interface class of a component.
Rate of Component Customizability (RCC)	A percentage of writable attributes in all attribute implemented within the interface class of a component.
Self-Completeness of Component's Return Value (SCCr)	A percentage of methods without any return value in all method implemented within a component.
Self-Completeness of Component's parameter (SCCp)	A percentage of methods without any parameter in all method implemented within a component.

The metrics were applied to five components of AMR analysis patterns. These five components were used in the analysis and design composition as shown in section 4. For each component, the values of adapted Washizaki *et al.'s* metrics were computed and tabulated in Table 3. Value of 1 indicates 'very high' and value of 0 indicates 'very low'.

Table 3. Washizaki *et al.'s* metrics applied on five 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 3, 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.
3. External dependency of all components is 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 as provided by the interface of the components. However, if the observability measurement is too high, it will lead to difficulty for users to find important readable properties from the interface, and if the customizability measurement is too high, it will lead to high possibility of misuse of components [24]. 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 sensors readings and to react to them appropriately through outputs and actuators.

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

7. Conclusion

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 early design of AMR software case study using the POAD methodology is illustrated. 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.

Based on the software analysis pattern, the pattern-based reverse engineering was performed on two existing AMR software. The result of the pattern-based reverse engineering process is the graphical documentation of software analysis using pattern-level diagrams. From the reverse engineering process, the existence of AMR analysis patterns in existing AMR software was verified.

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. These results suggest that further detail research on the benefits of patterns as a means to reuse domain knowledge is needed in domain such as AMR software.

Acknowledgements

The authors would like to thank Public Service Department of Malaysia for funding of this PhD work.

Appendix A

- *Name*: Behavior-based reactive layer.
- *Intent*: How to organize the components in the domain of reactive layer for AMR software?
- *Motivation*: Traditional robot control programs based on the sense-plan-act organization have been criticized due to the emphasis placed on construction of a world model and planning actions based on this model. The computation time required to construct a symbolic model has a significant impact on the performance of the robot. An alternative control organization is to use behavior-based approach which is a reactive system that do not use symbolic representation, and have been demonstrated capable of producing reasonably complex robot behavior.

Forces:

1. Reactive system reacts based on sensory data.
2. The concurrency resulted from different subsystem involved in the robot.
3. The dependency between the subsystem in the reactive layer are presented.

Solution: Build a reactive layer using behavior-based control paradigm. It models the sensory data and reaction toward the data in concurrent behaviors. A control mechanism will select the appropriate behavior for the robot in order to react toward the robot environment.

The relationships between the components in the analysis pattern in reactive layer are shown in Figure A1. All the components in the reactive layer are organized into a hierarchical organization based on their level of abstraction. Each subsystem will be treated as component-based analysis pattern to model the detail requirement of each domain.

Consequences: The benefits and liability of the pattern:

1. Difficult to manage with increasing system complexity. In this type of system, hybrid between reactive and deliberative planner is normally used.
2. Cannot perform high-level planning.
3. Avoiding symbolic representation of environment.

Design: Important point to be considered:

1. Identification of behavior from different types of sensory data and action in the reactive systems.
2. Interfaces to provide reuse components in the systems.

Known Uses: Mobile robot systems, autonomous vehicle systems

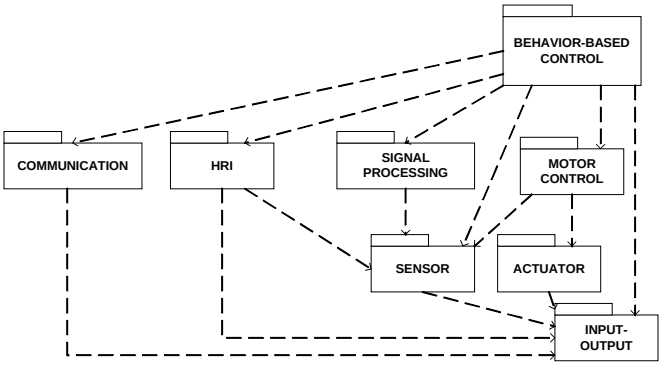


Figure A1. The structure of reactive layer.

References

[1] Alami R., Chatila R., Fleury S., Ghallab M., and Ingrand F., "Architecture for Autonomy," *Journal of Robotics Research*, vol. 17, no. 4, pp. 315-337, 1998.

[2] Blum S., "Towards a Component-Based System Architecture for Autonomous Mobile Robots," *in Proceedings of IASTED International Conference on Robotics and Applications (RA'01)*, pp. 220-225, 2001.

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

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

[5] Chikofsky E. J. and Cross II J. H., "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13-17, 1990.

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

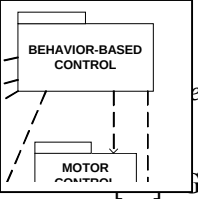
[7] Fernandez J. A., Gonzalez J., "NEXUS: A Flexible, Efficient and Robust Framework for Integrating Software Components of A Robotic System," *in Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, pp. 524-529, 1998.

[8] Fire Marshal Bill, available at: http://www.dragonflyhollow.org/matt/robots/firemarshalbill, August 2004.

[9] Fujita M. and Kageyama K., "An Open Architecture for Robot Entertainment," *in Proceedings of the 1st International Conference on Autonomous Agents*, pp. 435-442, 1997.

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

[11] Geyer-Schulz A. and Hahsler M., "Software Reuse with Analysis Patterns," *in Proceedings of the 8th Association for Information Systems (AMCIS)*, Dallas, TX, pp. 1156-1165, 2002.



Graves A. R. and Czarnecki C., "Design Patterns for Behaviour-Based Robotics," *Systems and Human*, vol. 30, no. 1, pp. 36-41, 2000.

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

[14] Labrosse J. J., *MicroC/OS-II The Real-Time Kernel*, R&D Books, USA, 1999.

[15] Mallet A., Fleury S., and Bruyninckx H., "A Specification of Generic Robotics Software Components: Future Evolutions of GenoM in the Orocos Context," *in Proceedings of the IEEE International Conference on Intelligent Robots and System*, vol. 3, pp. 2292-2297, 2002.

[16] Nelson M. L., "A Design Pattern for Autonomous Vehicle Software Control Architectures," *in Proceedings of 23rd International Conference on Computer Software and Applications*, pp. 172-177, October 1999.

[17] Nesnas I. A., Wright A., Bajracharya M., Simmons R., Estlin T., and Won S. K., "CLARAty: An Architecture for Reusable Robotic Software," *in Proceedings of SPIE Aerosense Conference, Unmanned Ground Vehicle Technology V*, vol. 5083, pp. 253-264, 2003.

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

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

[20] Real World Interface, "Mobility Robot Integration," available at: http://www.isr.com/rwi, December 2003.

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

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

[23] Smith G., Smith R., and Wardhani A., "Software Reuse Across Robotic Platforms: Limiting The Effects of Diversity," *in Proceedings of the Australian Software Engineering Conference*, pp. 252-261, 2005.

[24] Washizaki H., Yamamoto H., and Fukazawa Y., "A Metrics Suite for Measuring Reusability of Software Components," *in Proceedings of the 9th International Software Metrics Symposium*, pp. 211-223, 2003.

[25] Winn T. and Calder, "Is This a Pattern?," *IEEE Software*, vol. 19 , no. 1 , pp. 59-66, 2002.

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

Dayang Jawawi received her BSc degree in software engineering from Sheffield Hallam University, UK, and her MSc degree in computer science from Universiti Technologi Malaysia. Currently, she is working toward PhD degree in software engineering. Her area of research is component-based software engineering for embedded real-time software.

Safaai Deris received his Master degree in engineering and his PhD in computer and system sciences from Osaka Prefecture University, Japan. Currently, he is a professor in the Department of Software Engineering, Faculty of Computer Science and Information Systems, Universiti Teknologi Malaysia. His research interest include software engineering, artificial intelligence and bioinformatics. He has authored and co-authored more than 50 papers in international and local journals and conferences. Currently, he is the deputy dean of Graduate Studies, Universiti Teknologi Malaysia.

Rosbi Mamat is an associate professor and head of Department of Mechatronic and Robotics Engineering at the Faculty of Electrical Engineering, Universiti Teknologi Malaysia. He obtained his PhD in control engineering from University of Sheffield, UK. His research interests include intelligent control, robotics and mechatronic systems.