Malaysian Journal of Computer Science, Vol. 18 No. 1, June 2005, pp. 20-30

APPLICATION OF COMPONENT-BASED ANALYSIS PATTERNS FOR PATTERN-BASED REVERSE ENGINEERING OF MOBILE ROBOT SOFTWARE

Dayang Norhayati Abang Jawawi and Safaai Deris
Department of Software Engineering
Faculty of Computer Science & Information System
Universiti Teknologi Malaysia
81310 Skudai, Malaysia
email: dayang@fsksm.utm.my

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

ABSTRACT

Developing software for Autonomous Mobile Robot (AMR) is difficult and requires knowledge in embedded systems, real-time software issues, control theories and artificial intelligence aspects. To tackle the difficulty in developing software for AMR, many researchers have proposed the approach of reusable software component for mobile robot systems. Software pattern provides a way to reuse knowledge of expert across domain at all level of software development. In this paper component-based analysis patterns applicable to AMR software at high-level software development is proposed. Some important AMR component-based analysis patterns on AMR embedded software requirements are presented. How the analysis patterns can help in documenting two existing AMR software through pattern-based reverse engineering process is also illustrated.

Keywords: ***Analysis Pattern, Component-Based Software, Reverse Engineering, Mobile Robot Software***

1.0 INTRODUCTION

Modern software has becoming very large and complex due to increase demands on its requirement. This complexity leads to problems such as failure of software projects to meet their deadline, budget, and quality requirements and the increased in software maintenance cost. Recently, software reuse has becoming a popular approach to increase software productivity, improve software quality, consistency and reliability, and at the same time decrease the costs of software development.

Software pattern is an approach of software reuse; in which pattern provides a way to reuse expertise that can be used across domains at all level of development [1]. Software patterns are used to identify recurring problems and describe a generalised 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 categorised into three software development levels: conceptual patterns or analysis patterns for analysis level, design patterns for design level and programming patterns for implementation level [2]. Unlike design patterns, analysis patterns concern with system view of a software, which is important for requirements analysis and the usability of the final software.

Developing software for a heterogeneous system such as a mobile robot software, involves multi-disciplines of expert knowledge such as embedded systems, real-time software issues, control theories and artificial intelligence aspects. This requires analysis patterns which capture conceptual models in that domain in order to allow reuse across applications. In this paper we focus on analysis patterns as a means to facilitate mobile robot software knowledge reuse.

Two main areas where analysis patterns contribute to the software development process are: 1) to speed up the development of abstract analysis models that capture the main requirements of the concrete problem by providing reusable analysis models and 2) to facilitate the transformation of the analysis model into a design model by suggesting design patterns and reliable solutions for recurring constructs in the problem space [3].

The objectives of this paper are twofold: 1) to present results of our studies on some important mobile robot software analysis patterns; 2) to illustrate how existing AMR software can be documented using the analysis patterns through a pattern-based reverse engineering process. The aim of pattern-based document is for better understanding of the existing software and for identifying, evolving and applying reusable components [4].

This paper is organised as follows: Section 2 shows some analysis patterns theories detailing creation processes and documentation of AMR analysis patterns. Section 3 illustrates how the proposed AMR analysis patterns can be used to document two mobile robots software through a pattern-based reverse engineering process. Section 4 concludes the paper.

2.0 THE ANALYSIS PATTERNS FOR AUTONOMOUS MOBILE ROBOT

The software aspect of Autonomous Mobile Robot (AMR) has been recognised as the most difficult and challenging part [5], 6]. The reuse of AMR software components can reduce the need to start a new AMR project from scratch, and for robotic research group, this allows robotic researchers to focus on their particular field.

Software design patterns have previously been used in robotics software in a number of area and category. Graves and Czarnecki [7] use design patterns for behavior-based robotics systems focusing mainly on the area of man-machine interaction; Nelson [8] developed a design pattern for use in creating software control systems for autonomous or robotic vehicles; Rivero [9] proposed patterns-oriented framework for developing automatic and deliberative skills for mobile robots. However, till today there have been no analysis patterns for AMR documented anywhere.

The aim of the AMR analysis patterns help to recognise the reusable parts of AMR software and define explicitly the structure and the components of the AMR software system, and the relationships that exist within a component. Our patterns are referred to as component-based patterns because each analysis pattern proposed is solving a problem of a particular component in AMR software systems. Thus, each pattern is a unit of analysis. Partitioning a system into bounded units of analysis is necessary to enable a system to be independently extensible [10].

In this section some important analysis patterns are presented. These analysis patterns are documented in a catalog form for easy reference. Cataloging these analysis patterns involves two main processes: pattern mining process and documenting the analysis patterns based on a certain standard.

2.1 Analysis Patterns Mining Process of the AMR Software

Pattern mining process concerns with identification and documentation of patterns. The patterns mining process in this work is based on studies of AMR systems from books [1], [11], existing AMR software architectures [12], [13], [14] and experience from research works on AMR systems at the University of Teknology Malaysia (UTM). Some existing embedded and real-time domain-specific analysis patterns [15] and design patterns [16], [17], were also analysed in the mining process.

As a result of this patterns mining process, twelve software analysis patterns in typical AMR software were identified. The analysis patterns identified are: *input-output*, *actuator*, *sensor*, *signal processing*, *motor control*, *communication*, *Human-Robot Interface* (HRI), *poster*, *Behavior-Based Control* (BBC), *coordinator*, *planner* and *Real-Time Operating Systems* (RTOS). These analysis patterns were categorised according to hybrid deliberate layered architecture of robot software [12], [13], [14]. The relationships between the patterns in a layered architecture are shown in Fig. 1.

2.2 Documenting of the AMR Analysis Patterns

The essential information in AMR analysis patterns are cataloged based on guidelines of Meszaros [18], Gamma *et al.* [19] and Douglass [17]. The AMR software analysis patterns are documented using six essential elements: *name* - reference to the patterns; *context* - description of the context of the problem identified and the solution presented; *problem* - statement of problem solved by the patterns; *solution* - structural solution presented using class diagram, showing the elements and properties in the pattern, and interface to enable the pattern to communicate with other components; *related pattern* - 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 pattern. Fig. 2 shows a pattern catalogue for the *BBC* pattern documented using these six essential elements.

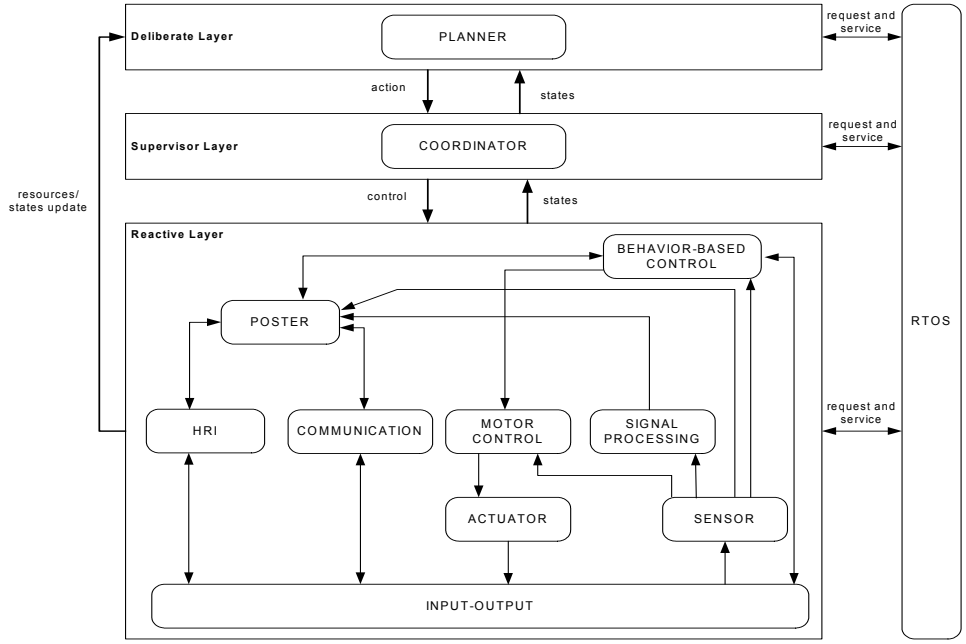


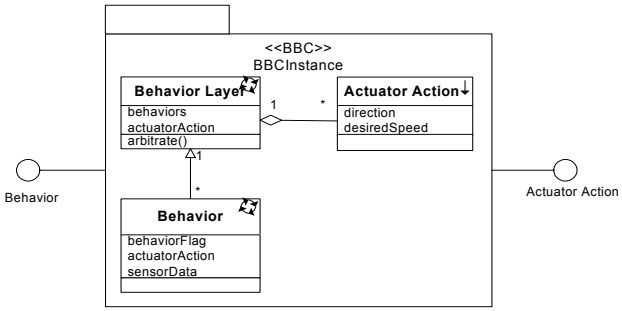
Fig. 1: AMR Software Pattern Relationships

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 intelligence 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: Pattern Catalogue for the *Behavior-Based Control* (BBC) Analysis Pattern

The Unified Modeling Language (UML) structural elements and diagrams have been adopted in describing the *solution* element of the pattern as UML provides a convenient and a lingua franca graphical representation in industry and academic software practice. Even though the *solution* is described using object-oriented technique, the implementation or realisation does not has to be in object-oriented approach.

The pattern *solution* is described using both *structural model* and *real-time behavior model*. The use of packages to represent constructional pattern and definition of pattern interface in describing structural model were adopted from Pattern-Oriented Analysis and Design (POAD) methodology [20]. The structural model describes classes that make up the pattern. The combination of classes in the structural model is arranged in a package to represent constructional pattern.

Interconnection between the components of AMR analysis patterns is supported by interfaces defined in the analysis pattern solution. POAD methodology definition of pattern interface is adopted to describe the AMR pattern interface. The result of wiring the pattern components using these interfaces is the functional structure of an AMR software system.

In describing real-time behavior of AMR software, the functional structure description using structural model is not sufficient. Hence, a real-time behavior model is required. We propose in this paper the enhancement of POAD to enable it to support real-time behavior of a component. The enhancements proposed are: 1) notations for identifying the types of real-time behavior of each class; and 2) introducing control interface between pattern components that can address the real-time behavior.

To specify the real-time behavior of a real-time component a set of stereotypes is introduced. The stereotypes are to describe different real-time behaviors using PECOS diagrammatical representation [21]. Based on this, three stereotypes are introduced in the AMR analysis patterns. 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 "↘".

As illustrated in Fig. 2 for the *BBC* pattern, the *Behavior Layer* and *Behavior classes* are active classes while the *Actuator Action* is a passive class. In some cases, a class can be specified with more than one type of real-time behavior depending on the application or implementation. For example, the class *Sensor Driver* from the *sensor* pattern can be specified as active class, event class or passive class. Fig. 3 illustrates this case for the class *sensor driver* of the *sensor* pattern, which the real-time behavior of the class is not posted in the solution because the form of the behavior will be decided during the implementation phase of the pattern.

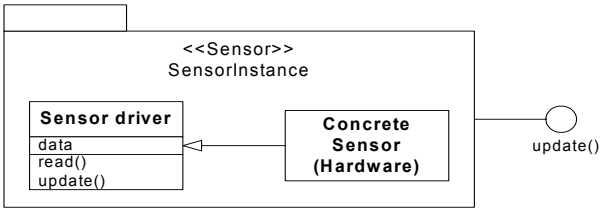


Fig. 3: *Sensor* Pattern Solution

Concurrency and multitasking capabilities of AMR software are supported by the real-time operating system (RTOS). Any component in AMR software that requires the RTOS services needs to be wired to the *RTOS* pattern using *control interface*. The control interface defines the attributes of real-time requirements of a pattern component, and this is only necessary in active or event classes. The control interface, however, is not explicitly showed in a pattern solution, because services required from *RTOS* pattern can only be specified during the wiring of pattern components.

In the documentation of the AMR analysis 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 examples, a commercial-of-the-shelf preemptive RTOS can be treated as a black box component in the *RTOS* pattern and Proportional-Integral-Derivative (PID) Controller patterns from Pont and Banner [16] can be used as white box component in AMR *motor control* pattern.

3.0 CASE STUDIES: REVERSE ENGINEERING OF EXISTING AMR SOFTWARE INTO PATTERN-BASED AMR DOCUMENTATION

Reverse engineering is the process of analysing 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 [22]. In this section a pattern-based reverse engineering process is illustrated in synthesizing higher abstraction of two existing mobile robot software and document their structural description using the proposed AMR analysis patterns.

The reverse engineering is performed on UTM intelligent AMR software and Fire Marshal Bill AMR software [22] to gain the graphical analysis representation of the software, based on the AMR analysis pattern. The first mobile robot 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 [22] which uses the Real Time Executive for Multiprocessor Systems (RTEMS) RTOS to support multitasking. The robot intelligent is implemented using a non-behavior-based architecture.

The two robot software are selected due to several reasons: 1) the C codes for both software are accessible to us; 2) both software are developed using conventional method without using any pattern; 3) both software are implemented using the traditional non-object-oriented approach; 4) the first software is based on behavior-based approach which represents common mobile robot software architecture, thus matched with the proposed analysis patterns; and 5) the second software is 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 are performed manually. The pattern-based reverse engineering processes are performed in three phases: 1) reverse to class; 2) reverse to pattern; and 3) reverse to pattern interface. Each phase is detailed out using the two case studies. These reverse engineering processes do not consider in detail the reuse components utilized in the codes such as the cooperative RTOS and some hardware interfaces code, as these components are already in the reuse forms.

3.1 Reverse to Class Phase

Reverse to class phase is to identify possible class structure that can be extracted from the code. The first step in this phase is to produce class diagrams from the functions in the program and the second step is to analyse the class diagrams and relationships between the diagrams. In this phase all the functions or modules names are maintained according to the original code names. Basically, this phase involves modeling of the functional structure of the software and identifying the classes or objects that required services from RTOS to support the cyclic and event classes.

Fig. 4 presents example of classes with their relationships, derived from the Fire Marshal Bill AMR software [22]. It illustrates the functional relationships between the classes, and also shows: 1) non-functional relation with a reused preemptive RTOS component to support multitasking behavior of the mobile robot; and 2) the classes with cyclic behavior which required services from RTOS class called *RTEMS*.

Mutual exclusion is very important when designing concurrent real-time systems such as the Fire Marshal Bill software, which used a preemptive RTOS. To ensure shared resources are accessed by one component at one time, the mutual exclusion services are required from RTOS. The *flame* and *robot* classes in Fig. 4 are examples of classes that required this service from *RTEMS* class.

3.2 Reverse to Pattern Phase

In this phase, three processes are performed, namely, matching process, grouping process and regrouping process. By comparing the software system class diagram with pattern solution documented in AMR analysis pattern catalogue, the matching process is performed. From this process, the component-based analysis patterns involved in an AMR software domain are identified. In the UTM intelligent AMR software, five AMR analysis patterns are identified, they are: *input-output*, *sensor*, *actuator*, *BBC* and *RTOS* patterns. For Fire Marshal Bill software, six AMR analysis patterns are identified, they are: *input-output*, *sensor*, *actuator*, *signal processing*, *motor control* and *RTOS* patterns. Up to this point of pattern-based reverse engineering process, we show how AMR analysis patterns can help to detect candidates for reusable software components from the two softwares.

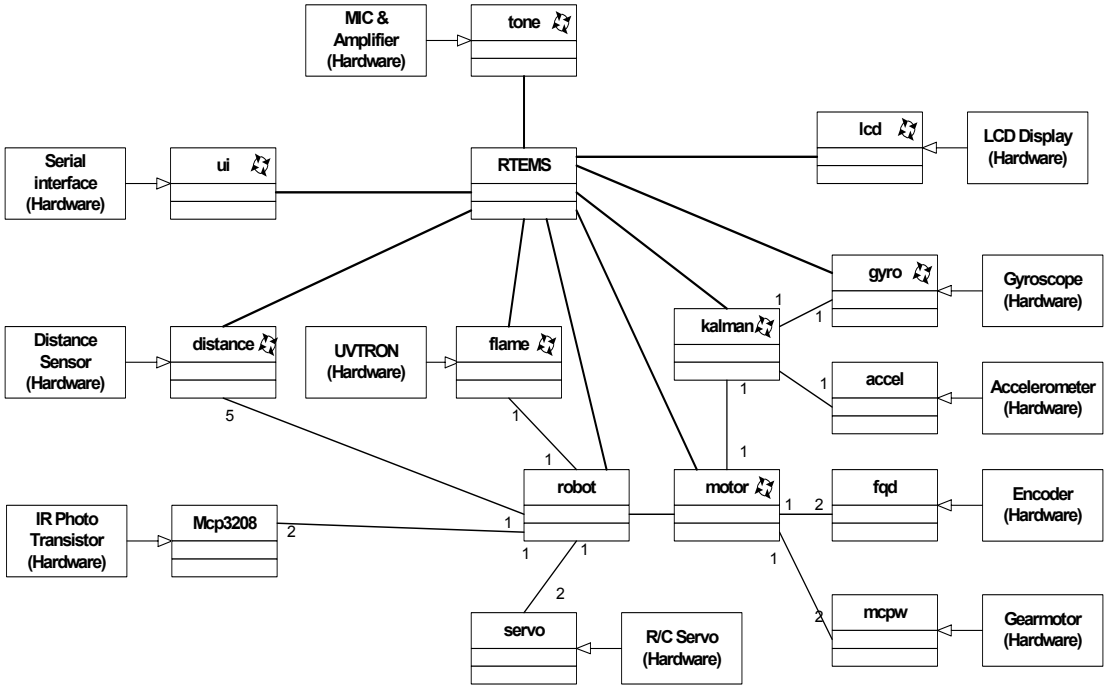


Fig. 4: Fire Marshal Bill class diagram

Using the identified patterns component, the classes of the mobile robot are grouped in the appropriate pattern packages or pattern instances. In this grouping process the matching of real-time control behavior are performed. Fig. 5 shows the example of two packages from the analysed UTM intelligent robot software, where *Arbitration* package is an instance of *BBC* pattern and *Motor* package is an instance of *actuator* pattern. In the *Arbitration* package eight classes shows the cyclic behavior, hence, require services from *Cooperative RTOS* package.

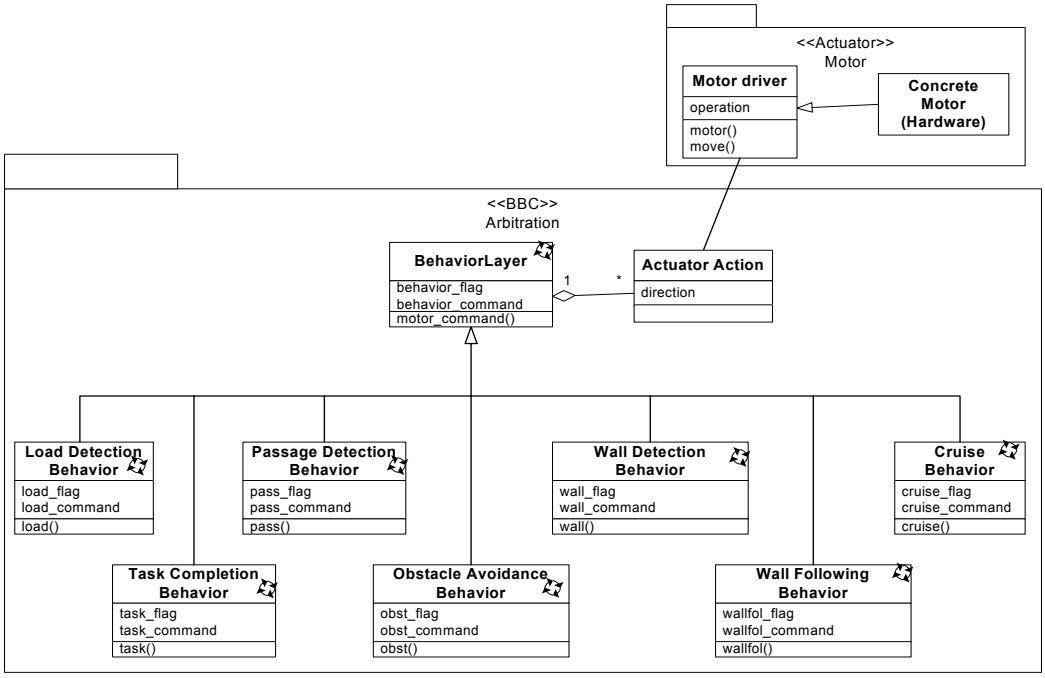


Fig. 5: The Arbitration and Motor Packages

Some classes or modules can be grouped and matched directly into appropriate pattern packages as illustrated in Fig. 5, however some classes or modules need to be regrouped or restructured in order to match with the proposed analysis pattern. For example, module *motor* in Fire Marshal Bill software consists of three PID control loops: 1) position PID to compute desired tilt; 2) balancing PID for velocity control of motor; and 3) heading PID for velocity control of motor. Only the balancing and heading PID matched with our *Motor Control* pattern therefore the two PID are grouped into the pattern.

The position PID to compute desired tilt should be included into a higher level of intelligence as compared to *motor control* pattern. Since the *robot* module also handles the robot high-level intelligence, the two modules can be grouped under different patterns. The structure of the two combined modules do not match with any pattern that we have proposed, therefore, the combination of position PID and *robot* module will form a user defined pattern that we called user High Level Robot Control (*user HLRC*) package. Fig. 6 shows the structure of Fire Marshal Bill level diagram after the matching, grouping and regrouping process.

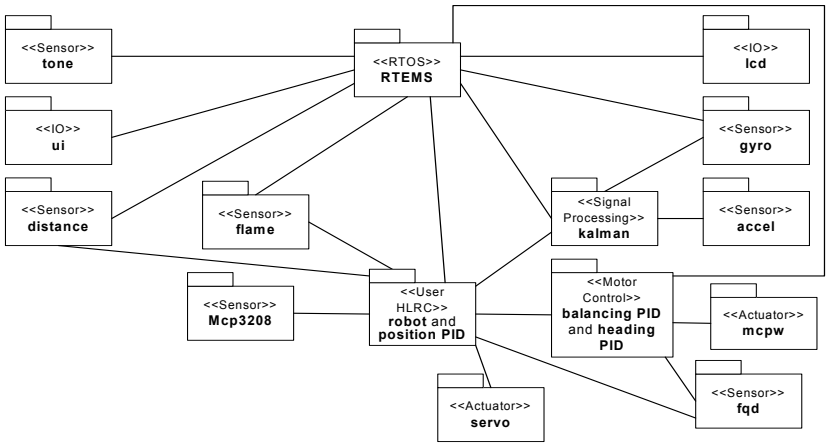


Fig. 6: Pattern Level Diagram for Fire Marshal Bill Mobile Robot

3.3 Reverse to Pattern Interface Phase

Interfaces are the means by which components connect in component technology [10]. The wiring of the components using AMR analysis patterns is supported by interfaces defined in the analysis pattern solution. The final stage of the reverse engineering process is to identify interfaces. Two types of interfaces need to be identified, i.e., functional interface and control interface.

Functional interface was specified based on the solution proposed in AMR analysis patterns. Example of functional interface selection for *Motor* package as shown in Fig. 5 is done as follows: from the analysis patterns, solution of *actuator* pattern suggested *scaler()* function to act as the functional interface for the pattern and from Fig. 5, for the *Motor* package, the *move()* operation is selected to be the interface for the package because the *move()* operation performed the scaling function as proposed in the functional interface of *actuator* pattern.

After all functional interfaces between packages are identified the functional composition of AMR software at analysis level is modelled. The composition is illustrated using Pattern-Level with Interface diagram as proposed in POAD methodology. The example of Pattern-Level with Functional Interface diagram is shown in Fig. 7. However, this composition does not synthesize the real-time behavior of the software. The real-time behavior within the intelligent AMR software has been analysed previously using PECOS temporal behavior as shown in Fig. 6.

The real-time behavior can be presented at pattern level analysis using control interface. The interface will define the interface to support the real-time temporal behavior and the synchronization services between packages. For example, the Fire Marshal Bill robot software requires two main services from RTEMS packages to handle the robot multitasking software: 1) task manager to manage nine cyclic tasks in the software; and 2) semaphore manager to provide mutual exclusion capability for operation in *run_maze()* to read flame. The control interface provides this additional information of components pattern as shown in Fig. 8.

The attributes for each control interface depend on the synchronization or types of real-time behavior in the pattern that are supported by RTOS. For example, in the UTM intelligent AMR software, the attribute for each control interface is the priority of each active component. However, for Fire Marshal Bill robot software, the control interface for active component needs to specify more detail attributes such as period.

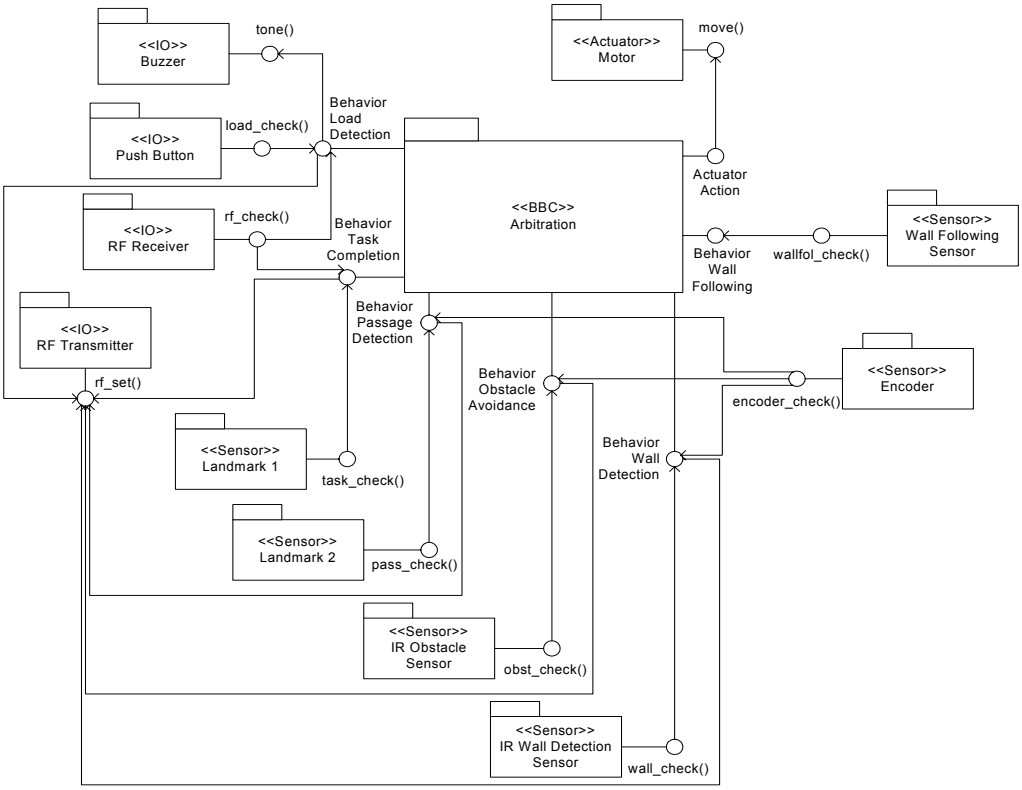


Fig. 7: Pattern-Level with Functional Interface Diagram for the Intelligent AMR

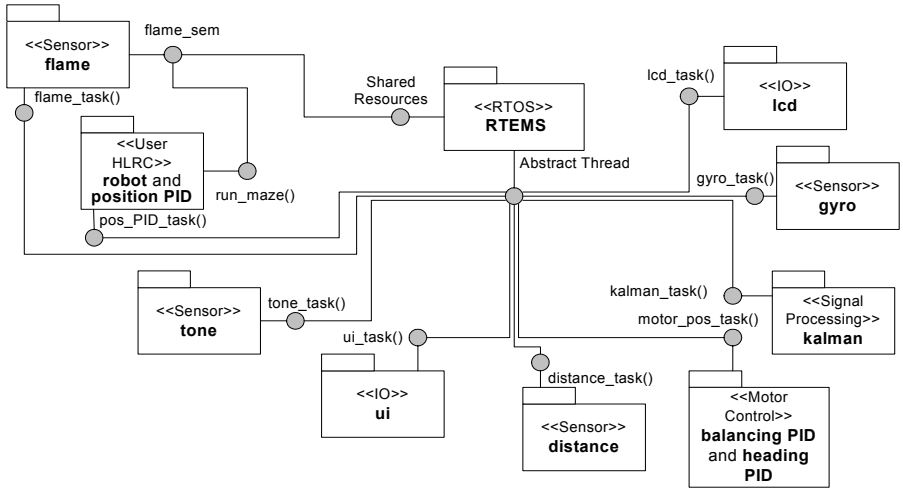


Fig. 8: Pattern-Level with Control Interface Diagram for the Intelligent Mobile Robot

Pattern-based reverse engineering is used to synthesize higher abstraction of the two case studies of AMR software. The software abstraction for each case study is presented using two separate pattern-level diagrams called Pattern-Level with Functional Interface and Pattern-Level with Control Interface diagram. Besides synthesizing higher abstraction of the software, the diagrams can also be used as graphical representation for documentation of the software. The main advantage gained from this higher software abstraction and graphical representation is software reusability. The AMR analysis pattern and the pattern-based reverse engineering can help to detect candidates for reusable software component from the existing software.

Based on reverse engineering experience of UTM intelligent AMR software and Fire Marshal Bill AMR software we found that both software designs can be structured according to our proposed analysis patterns. The UTM intelligent AMR software design structure is easier to be translated into our analysis pattern because software design was based on well-design architecture of behavior-based approach. For Fire Marshal Bill AMR software a few restructure works have to be done in order to document their design using our analysis patterns.

4.0 CONCLUSION

This paper has presented some important component-based analysis patterns as a result of our studies on AMR embedded software requirements. The representation of the patterns using structural elements of UML notation and POAD interface representation provides logical views to represent AMR functional composition. In concurrent and multitasking environment of AMR software this functional composition is not enough, the real-time form of each pattern need to be defined. Therefore, enhancement of POAD is presented in this paper to enable it to support real-time attributes of a component.

Based on the software analysis pattern, the pattern-based reverse engineering is performed on two existing mobile robot software. The results of the pattern-based reverse engineering process are the graphical documentation of software using pattern-level diagrams. Optimisation of the original code to improve the quality of the original subject software can also be proposed during the reverse engineering process based on the AMR analysis patterns. Another important conclusion derived from the reverse engineering is verification of the existence of AMR analysis pattern in existing mobile robot software, even though the software are originally not designed using any pattern approaches.

REFERENCES

[1] L. Rising, “Patterns: A Way to Reuse Expertise”. *IEEE Communications Magazine*, Vol. 37(4), 1999, pp. 34-36.

[2] D. Riehle and H. Zullighoven, “Understanding and Using Patterns in Software Development”. *Theory and Practice of Object Systems,* Vol. 2(1), 1996, pp. 33-13.

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

[4] G. Odenthal and K. Quibeldey-Cirkel, “Using Patterns for Design and Documentation”, in *Proceedings of 11th European Conference on Object-Oriented Programming (ECOOP '97)*, Vol. 1241(1997), Springer-Verlag Heidelberg Lecture Notes in Computer Science, Jyväskylä, Finnland, June 1997, pp. 511-529.

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

[6] D. W. Seward and A. Garman, “The Software Development Process for an Intelligent Robot”. *IEEE Computing and Control Engineering Journal,* Vol. 7(2), 1996, pp. 86-92.

[7] A. R. Graves and C. Czarnecki, “Design Patterns for Behaviour-Based Robotics”. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Human*, Vol. 30(1), 2000, pp. 36-41.

[8] M. L. Nelson, “A Design Pattern for Autonomous Vehicle Software Control Architectures”, in *Proceeding of 23rd International Conference on Computer Software and Applications*, 27-29 October 1999, pp. 172-177.

[9] D. M. Rivero, A. Khamis, F. Rodriguez and M. Salichs. "A Patterns-Oriented Framework for the Development of Sequential Movement Skills for Indoor Mobile Robots", in *Proceeding of 11th International Conference on Advanced Robotics,* Portugal, 30 June - 3 July 2003.

[10] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Second Edition, Addison-Wesley, Bostan, 2002.

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

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

[13] R. Alami, R. Chatila, S. Fleury, M. Ghallab and F. Ingrand, "Architecture for Autonomy". *Journal of Robotics Research*, Vol. 17(4), 199, pp. 8315-337.

[14] R. Volpe, I. A. D. Nesnas, T. Estlin, D. Mutz, R. Petras and H. Das, "CLARAty: Coupled Layer Architecture for Robotic Autonomy". *Jet Propulsion Laboratory Technical Report D-19975*, California Institute of Technology, 2000.

[15] S. Konrad and B. H. C. Cheng, "Requirements Patterns for Embedded Systems", *Proceedings. IEEE Joint International Conference on Requirements Engineering,* 2002, pp. 127-136.

[16] M. J. Pont, and M. P. Banner, "Designing Embedded Systems Using Patterns: A Case Study". *Journal of Systems and Software*, Vol. 71(3), 2004, pp. 201-213.

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

[18] G. Meszaros, and J. Doble, "A Pattern Language for Pattern Writing". *http://hillside.net/patterns/writing/patternwritingpaper.htm* available 21 June 2004.

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

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

[21] O. Nierstrasz, G. Arévalo, S. Ducasse, R. Wuyts, A. Black, P. Müller, C. Zeidler, T. Genssler, R. van den Born, "A Component Model for Field Devices", in *Proceedings of the First International IFIP/ACM Working Conference on Component Deployment*. Springer-Verlag Heidelberg, Vol. 2370, Berlin, Germany, 20-21 June 2002, pp. 200-209.

[22] E. J. Chikofsky and J. H. Cross II 1990, "Reverse Engineering and Design Recovery: A Taxonomy". *IEEE Software*, Vol. 7(1), pp. 13-17.

[23] Fire Marsal Bill at *www.dragonflyhollow.org/matt/robots/firemarshallbill/* available August 2004.

BIOGRAPHY

Dayang Norhayati Abang Jawawi received her B.Sc. Degree in Software Engineering from Sheffield Hallam University, UK and M.Sc. of Computer Science from University of Teknology Malaysia. Currently she is working towards her PhD. degree in software engineering. Her area of research is component-based software engineering for embedded real-time software.

Safaai Deris is a Professor at the Faculty of Computer Science and Information Systems, University of Teknology Malaysia. Prior to joining the university, he was a system analyst at the Ministry of Agriculture. He received his B.Sc. from Agricultural University of Malaysia, his M.Eng. and PhD. from University of Osaka, Japan. His fields of specialisations are artificial intelligence, software engineering, bioinformatics, database systems, object-oriented technology, planning and scheduling.

Rosbi Mamat is currently an Associate Professor and Head of Department of Mechatronics and Robotics Engineering at Faculty of Electrical Engineering of Universiti Teknologi Malaysia. He obtained his PhD. in Control Engineering from University of Sheffield, UK. His main areas of research interest cover intelligent control, robotics and mechatronics systems.