ABSTRACT

A multi-robot system is a system in which several robots function at the same time to achieve a goal. Multi-Robot design does however bring with it the complications of increased complexity of communications and the possibility of physical collisions between robots. As the number of robots increases, functioning as a team can be daunting. Design framework for agent-oriented multi-robot system is a framework documented at the high design level and is reusable as an initial phase in designing multi-robot applications. The framework is developed using the AgentPOD approach; the approach that fully utilize design pattern in its development. An Agent-Structure Design Pattern-Level and an Agent-Architecture Design Pattern-Level are described as higher design level of class diagram, which represents the framework in terms of patterns, subsystems, and associations. An Agent-Conversation Design Pattern-Level is described as higher design level of state diagram, which represents the framework in terms of patterns, states, and transactions. This framework can be reused in multi-robot application that shared an environment and sense each other and may interact, but do not depend on one another (members of the group can be removed without significant effect) such as load transport robot.

Keywords: Multi-Robot, Design, Framework, Pattern-Oriented, Agent-Oriented.

1 INTRODUCTION

Reusing software components in practical applications is a very difficult task, yet, it is required to reduce the development effort and assure the high quality of the developed software. Code reusability provides a low percentage of the effort savings but it is the most popular and common one. The level of reusability is determined by the design decisions taken by the framework designer on behalf of the framework users. A simple design decision, like the reuse of a library class or API, is the most common and easiest to use. The next level is the reuse of design patterns and the most difficult and sophisticated is the reuse of frameworks in which a wide range of design decisions is already taken.

A multi-robot system is a system in which several robots function at the same time to achieve a goal. Multi-Robot design does however bring with it the complications of increased complexity of communications and the possibility of physical collisions between robots. As the number of robots increases, functioning as a team can be daunting. One way to facilitate the programming of multi-robot systems is to use an agent oriented approach. Thus, a generic agent design for the multi-robot system will serve many robot software designers. Recently, design patterns have been evolving to describe a problem that frequently occurs in the software applications, and then describe the solution to that problem in such a way that it can be reused [1]. A framework is a set of classes and objects that collaborate to carry out a number of responsibilities. In developing our framework we used agent design patterns such as Strategy, Extended Observer, Blackboard patterns [2, 4, 7], FIPA-request and FIPA-Inform conversation patterns and BDI architecture pattern. Description of the framework in terms of design patterns adds flexibility and reusability to the framework. In fact, the research in [6] describes the benefits of using a design pattern level as an intermediate level of a system description between the analysis and class levels to reduce the descriptive complexity of the design.

Design patterns have recently been deployed in the framework design of many domain object-oriented specific applications. For manufacturing frameworks, an architecture design proposed in [10] started from an existing object-oriented analysis and targeted to more generic and flexible architecture using design patterns. The design was made general for automated manufacturing systems like assembly line. In this paper, we propose a framework for
258

The 5th International Conference on Information & Communication Technology and Systems

multi-robot system. Instantiation of an example from the framework is also presented. The framework is presented in terms of agent design patterns. In section 2, we give a brief introduction on multi-robot systems. Section 3 mentions the design patterns used in designing the framework. In section 4 we describe the proposed framework structure. Section 5 shows an example of instantiating the framework and finally, we conclude the paper in section 6.

2 MULTI ROBOT SYSTEM

Many issues must be considered when designing a multi-robot system such as autonomy, communication structure and coordination. Collective autonomy refers to the ability of the robots to work individually and without human intervention. Coordination addresses the interdependency management among the cooperative robots to achieve individual or collective goal(s). Referring to multi-robot architecture literature, the generic block diagram of multi-robot system proposed by [5] represents initial architecture documentation to start with. Figure 1 illustrates the initial architecture. This research also follows the Leader-Follower strategy to model the coordination behaviour among the robots. With this strategy, one mobile robot is selected to be a leader and the rest of the robots are the followers. By following this strategy, we have eliminated the conflict that may emerge due to the decentralized control posses by each robot.

The Action Layer is where the physical actions and sensory parts of the robot are located. The Action Layer consists of three key elements: the Executor, the Repository and the State Monitor as shown in Figure 1. The Executor is responsible for controlling and performing all of the physical actions of the robot. It manages the actuators and receives feedback from the perception modules. The State Monitor communicates with the Executor to choose the tasks to be executed. The Repository captures how and action should be performed. The Repository is a collection of known tasks that can be run by the Executor. It communicates with the State Monitor to determine which tasks are required in the Executor and to respond to queries about the available tasks. The State Monitor is the Action Layer’s communication channel through which is the only method that it can communicate with the Cognitive Layer. The Cognitive Layer performs all high level decision making. It receives status updates from the Action Layer’s State Monitor and uses these updates to determine which course of action to pursue. The Cognitive Layer consists of two main elements are always required in the Cognitive Layer: the Decision Maker and the Coordinator (see Figure 8.1). The Decision Maker represents the problem solver of the robot. Without this element, the robot would not be autonomous, would not be able to adapt to new situations and would not be able to form consensus with other robots. From the Decision Maker, commands are sent to the Action Layer via the Coordinator and inter-robot communications are facilitated. The Coordinator is the element that is in charge of communications with the Action Layer. It should only receive communications from the Action Layer and the Decision Maker.

3 AGENT DESIGN PATTERN

Definition of a framework in terms of design patterns provides a high level of generality to a framework, which in essence increases its reusability. In this section we briefly mention the patterns used in the framework. A detailed description of each pattern is found in [2, 4, 7].

3.1 The Strategy Pattern

The intent of a strategy pattern is to define a flexible design that solves the problem of choosing between alternative strategies and algorithms. The strategy pattern lets the algorithm vary independently from other system objects that use it [4].
3.2 The Extended Observer Pattern

The intent of the observer pattern is to provide a way to define dependencies between objects. When an object's state (a subject) is changed, it is sometime required to automatically notify and update other objects (observers) [7].

3.3 Blackboard Pattern

The intent of the blackboard pattern is to reduce the amount of information that needs to be studied and processed simultaneously, by partitioning the data objects into similar data. We will use a simplified version of the pattern of [4].

4 THE PROPOSED FRAMEWORK

The framework consists of a suite of concrete and abstract classes. A well-designed framework is a one that takes the maximum advantage of reusable elements. The proposed feedback control framework has a predetermined structure in which the collaboration between the objects and the type of classes are predefined. Domain specific designers can also modify and implement different parts of the framework according to the domain requirements. The proposed framework is developed using the AgentPOD approach [8]; the approach that fully utilize design pattern in its development. We describe the structure of the multi-robot system framework using the strategy, observer, and blackboard patterns as its building blocks. First, the framework is presented in terms of patterns and subsystems, then we show the final class diagram of the framework. The Unified Modeling Language notation is used to represent the class and object diagrams.

4.1 Agent Class Diagram

The multi-robot system can be divided into subsystems. Each subsystem provides a group of related functions in the framework. The requirements of each subsystem are studied and accordingly we choose a set of design patterns to be used.

A multi-robot system framework is divided into the following subsystems:

- The feedback component receives environment state and applies a feedback control strategy. It feeds the result to the local control component of the robot. The environment interaction unit observes and measures data from the sensors and feeds it to the feedback branch. Thus, environment observations can be communicated to the feedback controller using the Extended Observer pattern [7]. The environment data is fed to the feedback control strategy that can be implemented using the Strategy pattern [4].
- The local control component implements a control strategy. The change in the strategy should be flexible and hidden from any calls and invocations form any other component. Thus, a Strategy pattern [4] is a good candidate for the task. The feedback controller notifies local control component with the feedback data. The feedback controller can be viewed as the subject that notifies the state observer with changes in the feedback data. An Extended Observer pattern [7] can implement this behaviour.
- The coordination component implements a coordination control strategy. The change in the strategy should be flexible and hidden from any calls and invocations form other robots. Thus, a Strategy pattern [4] is a good candidate for the task.
- Data of different types need to be exchanged between the framework components. We can use a Blackboard pattern [2] for managing the robot behaviour repository.

The next level in the design of the framework is to expand the patterns in each subsystem in terms of their constituting classes and rename the classes of the patterns to be meaningful in the framework. Figure 2 shows the initial agent class diagram of the framework.

- The local control subsystem: The local control subsystem is a simple subsystem containing only one design pattern; the strategy pattern which is composed of the class LCtrlAgent.
- The Coordination Subsystem: The coordination system consists of the strategy pattern which is composed of LCoordinationCtrlAgent and FCoordinationCtrlAgent.
- The Feedback Subsystem: The feedback subsystem consists of the extended observer pattern, the strategy pattern and the blackboard pattern. The FeedbackObserverAgent and HWSubjectAgent represent the extended observer pattern. The Blackboard, SensorData, and FeedbackData represent the Blackboard pattern. The FeedbackAgent represents the strategy pattern. The SensorData is updated by the HWSubject and is used by the FeedbackObserverAgent class. The FeedbackObserverAgent invokes the control routine of the FeedbackAgent class that applies the feedback control strategy required from the subsystem. The FeedbackAgent class interacts with the FeedbackSubjectAgent of the
ExtendedObserver Pattern in the local control subsystem and invokes its notify plan procedure. This establishes the link between the feedback subsystem and the local control subsystem.

We then use grouping activity to optimize the initial agent class design diagrams. In this step, more optimization in class usage is projected by merging concrete classes together depending on their interaction and responsibilities. This step brings the framework’s initial class diagram to a more reduced form, however it mainly depends on the framework designer skills. We mark out that the classes FeedbackObserverAgent, FeedbackSubjectAgent and FeedbackAgent perform highly related tasks which are summarized as receiving environment data notification, applying control strategy, and notifying the state observer that the feedback data is ready. Instead of implementing a primitive task in each class, the three agent classes are merged into one class FeedbackSubjectObserverAgent. This step eases the use of the framework and brings the framework to a more reduced form. Figure 2 shows the final class diagram.

![Figure 2 The final agent class diagram for the framework](image-url)
4.2 Agent Conversation Diagram

In constructing conversation, AgentPOD[8] represents one conversation with two Communication Class Diagrams: Initiator and Responder. From the agent class diagram, two conversation patterns are identified suitable to be applied in designing the conversation between the agents. The conversation patterns chosen are the FIPA-Request and FIPA-Inform conversation patterns. The FIPA-Request conversation pattern can be used to model the RequestFeedback conversation between the FeedbackSubjectObserver agent with the HWSObject agent and the RequestEnvironmentState conversation between the StateObserver agent with the FeedbackSubjectObserver agent. The FIPA-Inform conversation pattern can be used to model the InformState conversation between the StateObserver agent and LCtrl agent, the InformMotorControl conversation between LCtrl agent the HWSObject agent, the InformState conversation between the StateObserver agent and LCoordinationCtrl agent, the InformFeedbackData conversation between the HWSObject agent and the FeedbackSubjectObserver agent and the InformLeaderState conversation between the LCoordinationCtrl agent and FCoordinationCtrl agent. This is illustrated in Figure 3.

4.3 Agent Architecture Diagram

In constructing the agent architecture class diagram, select patterns are selected from the literature based on the components identified in the analysis phase. The agent architecture pattern chosen is the BDI architecture pattern [9]. This is illustrated in Figure 4.
5  EXAMPLE: LOAD TRANSPORTATION ROBOTS

In this subsection, we illustrate a simple example for instantiating the framework of UTM intelligent Autonomous Mobile Robot (AMR) software. The system consists of two mobile robots where each robot is a differential drive wheeled mobile robot, capable of traversing in an environment, which is surrounded by four walls. The task of the robots is to find a passage and exiting through the passage while holding one large object on top of them. Therefore, the main goal of these mobile robots is to transfer one large object through an unknown passage. Leader-followers strategy has been used where one mobile robot will be a leader while the other will be a follower.

The software has to run on embedded processor. The main processor in the controller is based on Motorola68HC11 microcontroller with 32K RAM, Parallel I/O, Serial I/O, ADC/DAC, Switches on/stop and Status Liquid Crystal Display (LCD). The embedded controller can be represented in a block diagram form as in Figure 5. The main function of the controller is to move the wheels of the robot until the robot achieve its goal. In order to move the robot and achieve the goal, the software needs to control the motor at each robot drive wheels, monitor the environments, navigate the robot and cooperate with other robot. The controller monitors its environment using four IR proximity sensors and two IR distance sensors. The load detection is done via one push button. The controller also communicates with other robot via a Radio Frequency (RF) transceiver. The controller also communicates with human through LCD and a switch.

5.1  Instantiation of the framework

We instantiate the framework by starting with the framework class diagram and define the functionality and specification of each class accordingly:

- The HWSubjectAgent. The function of this agent class is to monitor for any obstacles and walls and monitor the speed of the motors. The scanObstacle(), scanWall(), scanLoad() and getMotorSpeed() plans will enable the sensors to monitor for the obstacle and wall, to detect the load and get the current speed of the motors. The moveMotor() plan will be invoked by the LCtrlAgent to move the motor as according to the result of the invocation of the MotorControlAlgorithm().
- The FeedbackSubjectObserverAgent. The necessary feedback algorithm is implemented this agent class. The invokeFBAldgo() plan will implement a feedback algorithm to get a feedback on the environment data of the robot. The result of the algorithm will be stored in a FeedbackData object.
- The LCoordinationCtrlAgent. The necessary coordination control algorithm is implemented in this class. The invokeCoordCtrlAlgo() plan will implement a coordination control algorithm to coordinate the leader and follower robots.
- The FCoordinationCtrlAgent. The necessary coordination control algorithm is implemented in this class. The invokeCoordCtrlAlgo() plan will implement a coordination control algorithm to coordinate the follower with its leader.
- The StateObserverAgent. After being notified by the FeedbackSubjectObserverAgent, the agent will get the current state of the robot.

The LCtrlAgent. The local controller of the robot invokes the motor control algorithm via the invokeCtrlAlgo() plan to control the robot’s motor according to the current robot state obtained from the StateObserverAgent.

![Figure 5 Block Diagram of the UTM intelligent AMR controller](image-url)
Figure 6 The class diagram of the load transportation robot: Leader Robot

Figure 7 The RequestEnvironmentFeedback conversation diagram: FeedbackSubjectObserverAgent (Initiator)
Next, we instantiate the framework agent conversation diagram and define the transactions and action for each state accordingly. Figure 7 illustrates the instantiation of the RequestFeedback conversation for the load transportation robot.

Lastly, we instantiate the framework agent architecture diagram and define the functionality and specification of each class accordingly. Figure 8 illustrates the instantiation of the LCtrl agent framework architecture diagram for the load transportation robot.

## 5.2 Remark

From the previous example, we deduce the following:

- The instantiation process is direct and simple and removes a lot of burdens from the framework user. The design is self-explanatory, it is clear to the user where to insert application specific functions and attributes and where to modify them.

- Decreasing the dependencies between several parts of the framework adds simplicity to the design and increases the flexibility of using and instantiating the framework.

## 6 CONCLUSION

In this paper we presented a multi-robot system framework using design patterns as its building blocks. The framework consists of the Strategy, Extended Observer and Blackboard patterns. We explained the framework design in terms of patterns, an agent class diagram, an agent conversation diagram and an agent architecture diagram. The framework shows a design using patterns and then exploding the design in terms of classes, states and transitions. Ordinary techniques represent the framework in terms of classes and their interaction or even in block diagrams [2]. The approach of using design patterns in representing a framework makes it more reusable, flexible and well documented [6].

There are two approaches to instantiate the proposed framework in an application. The first is to start with the diagram of patterns and proceed with the instantiation and implementation of each pattern. The second approach is to use the "class diagram" of the framework and implement the classes' attributes, operations, and associations. We showed an example in which we used the final class diagram and instantiated the application for a load-transportation robot system. The instantiation example shows the simplicity, flexibility and ease of use of the generic design.

## REFERENCES


[5] Gruneir B., Tehrani A., Miners B., Khamis M., Li H., El-Abd M., Song I., Kamel M., Basir O., and


[This page is intentionally left blank]