

Pattern-Oriented Design for Multi-Agent System: A Process Framework

Radziah Mohamad, Safaai Deris
Faculty of Computer Science and
Information Systems
Universiti Teknologi Malaysia, 81310
Skudai, Johor, Malaysia

Hany H. Ammar
Computer Science and Electrical
Engineering Department, West Virginia
University
Morgantown, WV 26506, USA

Abstract - Agent design patterns represent solutions to the specific problem of developing multi-agent based applications that have evolved over time. They leverage the reuse level to the design phase by providing a common vocabulary of design, means of understanding designs and proven building blocks from which more complex applications are built. Much of the research work on agent design patterns has primarily focused on discovering and documenting patterns. To reap the benefits of deploying these proven design solutions, a systematic process to construct agent applications using patterns is needed. This paper discusses a process framework to design pattern-oriented multi-agent applications. This paper will show how the framework could be applied in the existing agent analysis and design methodology, MaSE methodology. The discussion is illustrated by a case study of building a framework for water treatment plant report management system.

Keywords: Agent design patterns, pattern-oriented design, design process framework

1.0 Introduction

The explosive growth of application areas such as electronic commerce, knowledge management, peer-

to-peer and mobile computing demand software that is robust, can operate within a wide range of environments, and can evolve over time to cope with changing requirements. Autonomous agents and multi-agent systems are being used to cope with ever increasing complexity in such system requirements. As the complexity of software system increases, developers look for approaches to facilitate the development of software applications. Patterns are reusable good-quality design practices that have proven useful in the design of software applications [3, 9]. Patterns can help in leveraging reuse to the design level because they provide a common vocabulary of designs and they are proven design units from which more complex applications can be built. Against this background, much work has focused on documenting agent design patterns [1, 7]. Other work is concerned with applying these reusable designs in constructing agent applications [4, 10, 11].

Yacoub and Ammar [17] classified design approach that utilize patterns into ad-hoc and systematic approach. Using an ad-hoc approach, a design pattern records a solution and forces and consequences of applying this solution. A systematic approach to design with patterns goes further beyond just applying a certain pattern. Systematic approaches can be classified as pattern languages and development processes. A pattern language provides a set of patterns that solve problems in a specific domain. Pattern languages not only document the patterns

themselves but also the relationships between these patterns. They imply the process to apply the language to completely solve a specific set of design problems. A systematic development process, on the other hand, defines a pattern oriented analysis and design steps, design models, and tools to automate the development steps. Such development process produces consistent designs each time the process steps are conducted. We are concerned here with systematic development processes because they are the way to repeatable software design practice. To improve the practice of systematically deploying agent design patterns, we need to define a process framework to design agent applications using patterns and apply this framework to the existed agent design methodology. The motivation is to make the existing agent design methodology like MaSE [5], Gaia [18] and Tropos [2] more mature.

In this paper, we discuss our approach for improving existing agent design methodologies and a process framework for developing pattern-oriented multi-agent designs. This paper is organized as follows. Section 2 will discuss our agent design methodology improvement process. In the section 3 we discuss our proposed design process framework. In the section 4, we discuss the application of the framework to one of the existing agent design methodology, MaSE methodology. The reasons of choosing the methodology are discussed in the section. Section 5 contains a review of related works. Finally, Section 6 presents our conclusion.

2.0 Agent Design Improvement Process

Although several agent design methodologies have been proposed recently, none of them is mature enough to develop commercial and industrial applications [18]. One step towards achieving mature methodologies is to enhance the current ones with the inclusion of software engineering best practices. One of those best practices is the use of patterns in key parts of the design processes. In order to improve the agent design methodology, we propose an improvement process based on survey on practices in methodology improvements by Self and DeLoach [13], DeLoach [6] and DiLeo et al. [8]. The improvement process is shown in Figure 1. By using this process, first we defined a case study named Water Treatment Plant Report Management System (WTPRMS) which could be implemented as a multi-agent system. Then, we have evaluated existing agent design methodology based on the resulted software

artifacts that produced in the methodology phases and the multi-agent Water Treatment Plant application which implemented based on these artifacts. During this evaluation, our focus was on the reusing previous design experience requirements.

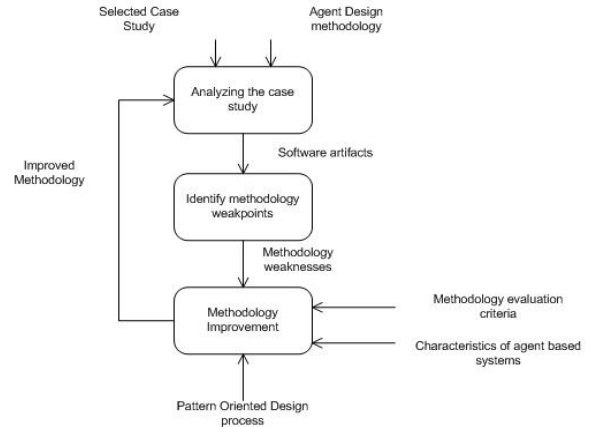


Figure 1 Agent Design Improvement Process

During the evaluation, we identified one general weakness in the existing agent design methodology i.e. they do not include a systematic process to design agents with patterns. Pattern-Oriented Analysis and Design methodology (POAD) proposed by Yacoub and Ammar [17] offers a systematic process to design object-oriented system using patterns as block of designs. However, the kind of decomposition that POAD offers is at odds with the kind of decomposition that agent oriented design encourages since agents are more coarse-grained than objects. Furthermore, it is hard to capture many aspects in agent system, for example, the agent coordination.

Inspired by a systematic process offered by POAD, this paper proposes a process framework to systematically design multi-agent systems with patterns. We then show how to apply the process framework to the existing agent design methodologies.

3.0 Pattern Oriented Design Process Framework

In this section, we present a systematic process to design agent application with patterns. Figure 2 illustrates the process framework. The following subsections will discuss the stages involved in detail.

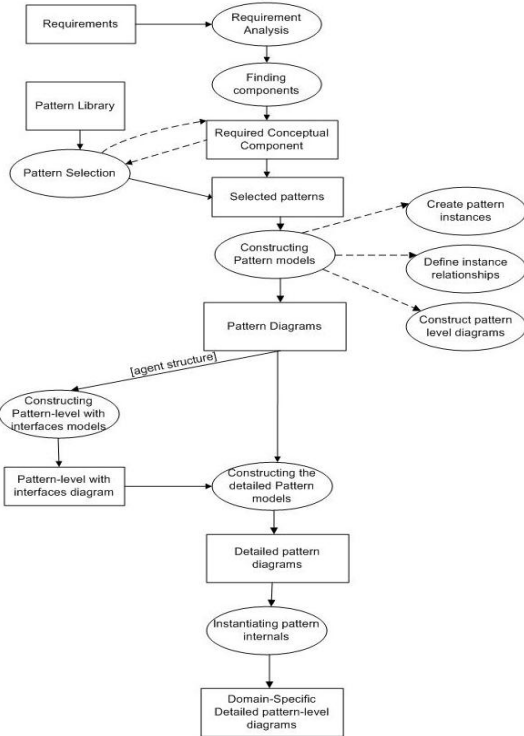


Figure 2 Agent Pattern Oriented Design Process Framework

A. Finding Components

The purpose of this step is to analyze the application requirements and identify the problems to be solved. We also determine the logical components and agent's roles used to address these problems. This step is applied at the agent analysis phase of one agent design methodology. Components identified in this step are used later to facilitate the process of assessing which patterns are suitable for the design of the agent application at hand.

Using the application requirement as an input, we start by analyzing the requirements by identifying the problems to be solved. We look for the functionalities that the application should provide, and then we articulate the problems to be addressed. At this step, it is useful to consider use cases and develop a use case diagram in which the context of the system is established. The use case diagram shows the main use cases that capture the required functionalities and their relationships and interactions with external actors. The use cases are documented using textual documents and realized using interaction diagrams in which logical components and objects are identified. Typically, a use case is realized by an interaction diagram showing the main flow of events and variations of this diagram to specify alternate and exceptional flow of events. Having identified a set of problems that we want to solve, we define logical

components that will be used to solve these problems and satisfy the application requirements.

The products of this step are a set of conceptual components, their use cases and their responsibilities, their sequence diagrams and agent roles.

B. Pattern Selection

The pattern selection process is more like a problem-solution matching technique. The problems are defined by the set of components and their responsibilities as defined from the requirement-analysis activity. The selection process is the process of matching solutions (design patterns) to the problems and responsibilities of the conceptual components.

Several tips on how to select a pattern from their catalog of 23 general-purpose patterns has been proposed by [10]. The proposed tips rely on the designer intuition and reasoning:

- Consider how the pattern solves the design problem by scanning the solution section and assessing whether the solution provided can be used in solving the problem at hand.
- Scan the intent section, which often describes what the problem is and hence could be matched to the problem at hand.
- Study how patterns interrelate. Look for similar patterns or patterns that are normally used in pairs or groups.

The outcome of this step is to select a set of patterns that fulfill the responsibilities of each conceptual component that was identified from the application requirements. Those selected patterns are then delivered to the pattern-level design phase. The outcome of this activity is the set of patterns that will be used in designing the application.

C. Constructing Pattern-Level Models

The purpose of this step is to create the pattern-level models for the application. In this step, the designer creates an instance for each pattern that is selected by the analyst. The instantiation process involves describing the patterns and their constituents in an application-specific context suitable for modeling the application at hand. The instantiation process includes two steps.

- The first is identifying the name and type of the pattern and is performed during the design phase.
- The second is related to giving application-specific names to the pattern internals and is performed during the pattern internal instantiation stage.

The product of this process is the pattern model for the application.

D. Constructing Pattern-Level with Interfaces Models

The purpose of this activity is to analyze the relationships between pattern instances and define the relationships between their interfaces. In this step, the dependency relationship between patterns in the pattern-level view is a conceptual high-level dependency that illustrates which pattern uses the others. To develop lower-level design elements that will be further used to glue the internal participants of a pattern with the internal participants of another pattern. This step is applied at the stage of modeling the agent structure (class) diagram.

E. Constructing Detailed Pattern-Level Models

The purpose of this activity is to reveal the internal design structure of the pattern instances used in the Pattern-Level models. As part of this stage, the designer studies the dynamics between the pattern participants and the roles played by each participant. This will help the designer at the next phase in instantiating the internal details of the pattern to solve the particular application domain problem.

F. Instantiating Pattern Internals

The purpose of this activity is to create an application-specific instance of the patterns used in the Detailed Pattern-Level diagrams. The process involves describing the patterns and their constituents in an application-specific context.

4.0 Applying the framework to the MaSE methodology

In this section, we discuss the application of the suggested framework to MaSE [5] methodology. The discussion is illustrated using the WTPRMS case study [12]. MaSE is one of the methodologies that have tried to be independent of particular multiagent system architecture, agent architecture, programming language or message-passing system [3]. It also provides a tool, called agentTool, which supports different phases of the methodology. Our study shows that, in comparison with some of the other methodologies, it can better model agent properties such as autonomy, proactiveness and goal-directed behavior. Also, it provides more expressive models for modeling communication and conversion in multiagent systems.

Next subsections will illustrate the application of the proposed framework to the MaSE methodology. Due to a space constraint, this paper will illustrate the application of the framework to design the agent structure diagram.

A. Analysis Phase

The analysis phase includes three steps: Capturing Goals, Applying Use Cases and Refining Roles. The first step, Capturing Goals, takes user requirements and turns them to top-level system goals. Using system-level use cases and defining sequence charts in the Applying Use Cases step, an initial set of system roles and communications paths are defined. In the Refining Roles step, using the system goals and roles identified in the use cases, an initial set of roles is refined and tasks to accomplish each goal are defined.

B. Finding components

Having identified a set of problems that we want to solve from the capturing goals and identification of roles [13] phase, we define logical components that will be used to solve these problems and satisfy the application requirements. In the following, we summarize the conceptual components that we identify in the system.

1. Scheduler
The scheduler component serves as the main driver for the report generation. It repeatedly processes the events and delegates actions to one or more components.
2. Report generator
The report generator component uses one of the distribution functions designated in the domain description to generate the content of the report based on its predefined template.
3. Notification
The notification component serves to notify the generated report to the plant staffs. The notification service is in terms of emailing the notification of the generated reports to the plant staffs.

C. Constructing Agent Structure Diagram

In this stage of MaSE methodology, the aim is to develop an agent class diagram which depicts the overall agent system organization. An agent class is a template for a type of agent in the system and is analogous to an object class in object-orientation [5]. The following subsections will discuss on the

application of framework to construct the agent structure diagram.

1. Pattern Selection

In this phase, we analyze the responsibilities and the functionalities of each component and identify candidate patterns that could provide a design solution for each component. In this step, we consider the design problem that we want to solve and match it to those documented in [1, 7, 9]. As an example, the notification component plays the role of notifying the staffs. In searching for domain-specific libraries of patterns for notification purposes, we select the Observer pattern [7] to design the structure of the Notifier agent.

2. Constructing Agent Pattern Level Diagram

The concern in this step is to represent the agent system as patterns and capture the relationship between various patterns. At this level, dependency is the relationship between patterns. It is a uses relationship. Pattern dependencies are further refined at later agent design phases to become associations between interface classes of two dependent patterns. The product of this process is the Agent Pattern-Level diagram of the framework. It describes the architecture of the WTPRMS using Agent patterns. The example of the WTPRMS Agent pattern diagram is as in Figure 3.

3. Constructing Agent Pattern Level with Interfaces Diagram

The purpose of this activity is to analyze the relationships between pattern instances and define the relationships between their interfaces. It is to explore the details of the relationship between the patterns used in the Pattern Level view. The dependency relationship between patterns in the pattern-level view is a conceptual high-level dependency that illustrates which patterns use the others. To develop lower-level design models for the application, these dependencies should be further traced to lower-level design elements that will be further used to glue the internal participants of a pattern with the internal participants of another pattern. It is the purpose of this activity to develop Pattern level with interface diagrams that will help the process of tracing these dependency relationships to lower-level class relationships. The Pattern level with interface diagram provides an intermediate step to bridge the gap between various design abstraction levels, where the higher abstraction level is the pattern instance dependency relationships and the lower abstraction level is the class association relationships between pattern participants. The uses relationship in the Pattern-Level view is refined in terms of relationships

between pattern interfaces. The relationship between pattern interfaces is the Service/Service relationship. This type of relationship models interactions, which usually reflect the designer perception of lower design details. Interactions are useful in developing agent behavioral description that could later be used to develop the conversation diagrams. Example of the WTPRMS Pattern level with interface model is as in Figure 4.

4. Constructing Detailed Agent Pattern Diagram

The purpose of this activity is to reveal the internal design structure of the structural-agent design pattern instances used in the structural-agent design pattern level diagrams. This internal structure is used in building the agent class diagram of the application.

5. Constructing Agent Class Diagram

To develop an agent class diagram from the Detailed Agent Pattern-Level diagrams, we have to trace all the relationships between pattern interfaces to class relationships between participants inside the pattern (after instantiating the participants). Figure 5 shows the agent class diagram for the WTPRMS.

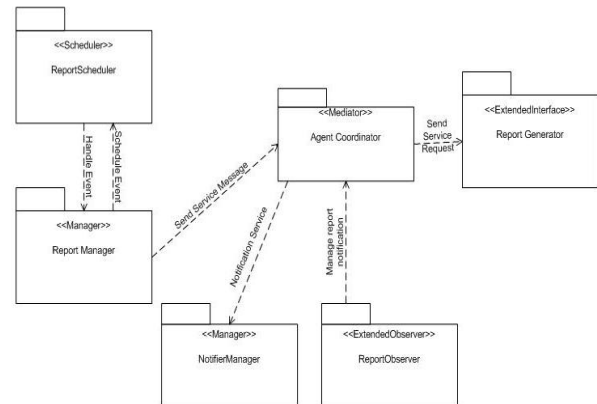


Figure 3 WTPRMS Agent Design Pattern Level Diagram

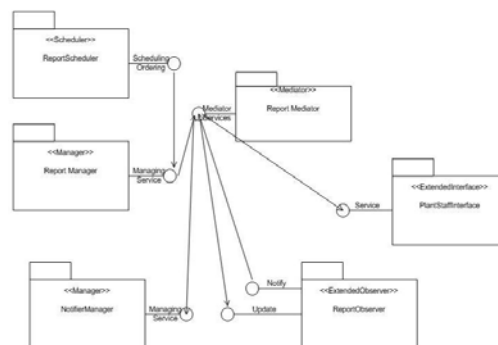


Figure 4 WTPRMS Agent Design Pattern with Interfaces Diagram

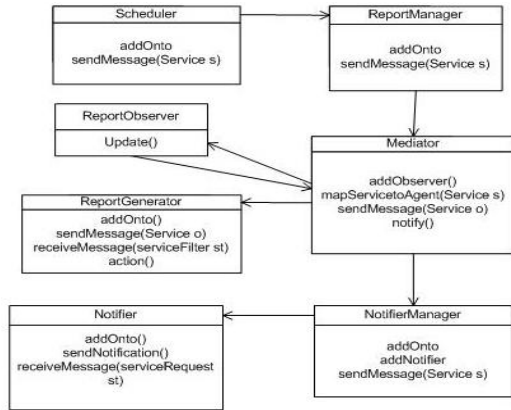


Figure 5 WTPRMS Agent Class Diagram

5.0 Related Works

Several successful experiences have reported on the advantages of using patterns in designing agent applications [4, 7]. Generally, we classify design approach that utilize patterns into adhoc and systematic approach. Using an adhoc approach, a design pattern records a solution and forces and consequences of applying this solution. Kolp et al. [10] presents a set of social patterns as part of the Tropos [2] methodology to describe the general architecture of a system under construction. Cossentino et al. [4] presents the design of a particular type of agent pattern immersed in the PASSI methodology. They defined a pattern as consisting of a model and an implementation code. However, this is not usually sufficient to systematically develop applications using patterns. This is simply because there is no process to guide the development and to integrate the pattern with other design artifacts.

A systematic approach to design with patterns goes further beyond just applying a certain pattern. Systematic approaches can be classified as pattern languages and development processes. A pattern language provides a set of patterns that solve problems in a specific domain. Pattern languages not only document the patterns themselves but also the relationships between these patterns. They imply the process to apply the language to completely solve a specific set of design problems. Weiss [16] proposed a generic agent pattern language. It documents the forces involved in agent-based design and key agent concepts. A domain-specific pattern language, respectively for agent-based manufacturing and electronic commerce, has been proposed by Shu and Norrie [14] and Weiss [15]. The relationships

between the patterns are made explicit in such a way that they guide a developer through the process of designing a system. A systematic development process, on the other hand, defines a pattern composition approach, analysis and design steps, design models, and tools to automate the development steps. Such development process produces consistent designs each time the process steps are conducted. This research is concerned with systematic development processes because they are the way to repeatable software design practice. To improve the practice of systematically deploying agent design patterns, this research suggests an approach to construct agent applications using patterns.

6.0 Conclusion and Future Works

The work in this paper stems from the need to develop systematic approaches using patterns in the development of agent applications and to develop pervasive pattern-level views that document a design as a composition of patterns. We are now working on developing a tool to support the proposed process. The aim is, the proposed methodology could be applied or embedded to the existing agent design methodology. The motivation is to make the existing agent design methodology more mature.

7.0 References

- [1] Aridor Y. and Lange D.: "Agent Design Patterns: Elements of Agent Application Design", in *Autonomous Agents (Agents '98)*, ACM Press, 1998.
- [2] Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J and Perini, A.: "TROPOS: An Agent-Oriented Software Development Methodology" in *Journal of Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers Volume 8, Issue 3, Pages 203 - 236, May 2004.
- [3] Buschmann F., Meunier R., Rohnert H., Sommerlad P. and Stal M.: *Pattern-Oriented Software Architecture - A System of Patterns*, (Addison-Wesley, 1996).
- [4] Cossentino, M., Sabatucci, L, Sorace, S. and Chella, A.: "Patterns reuse in the PASSI methodology", *Fourth International Workshop Engineering Societies in the Agents World*

(ESAW'03) - 29-31 October 2003, Imperial College London.

[5] DeLoach, S. A., Wood, M. F. and Sparkman, C. H.: "Multiagent Systems Engineering", The International Journal of Software Engineering and Knowledge Engineering, Volume 11 no. 3, pp. 231-258, June 2001.

[6] DeLoach S. A., "Modeling Organizational Rules in the Multiagent System Engineering Methodology", Proceedings of the 15th Canadian Conference on Artificial Intelligence, Calgary, Canada, 2002.

[7] Deugo, D., Weiss, M. and Kendall, E.: "Coordination of Autonomous Internet Agents: Models, Technologies and Applications", chapter Reusable Patterns for Agent Coordination, Springer, 2001.

[8] DiLeo J., Jacobs T. and DeLoach S., "Integrating Ontologies into Multiagent Systems Engineering", 4th. International Bi-conference Workshop on Agent-Oriented Information Systems (AOIS2002), Bologna, Italy (2002).

[9] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley, Longman, 1995.

[10] Kolp, M., Castro, J. and Mylopoulos, J.: "A social organization perspective on software architectures", in 1st. International Workshop from Software Requirements to Architectures, 2001.

[11] Lind, J.: "Patterns in Agent-Oriented Software Engineering". in Giunchiglia, F., Odell, J. and Weiss, G., editors, Agent-Oriented Software Engineering III, vol. 2585 of Lecture Notes in Computer Science, Springer, 2003.

[12] Mohamad R., Deris S. and Ammar H. H.: "Towards a Reusable and Maintainable Multiagent System Engineering (MaSE) Methodology", in the Proceedings of the International Conference on Artificial Intelligence (IC-AI'2005), Las Vegas, Nevada, USA. June 2005.

[13] Self A. and DeLoach S., "Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology," Proceedings of the Eighteenth Annual ACM Symposium on Applied Computing, Melbourne, Florida, USA (2003).

[14] Shu S. and Norrie D., Patterns for Adaptive Multi-Agent Systems in Intelligent Manufacturing, (International Workshop on Intelligent Manufacturing Systems, 1999).

[15] Weiss M., Patterns for e-Commerce Agent Architectures: Using Agents as Delegates, (Conference on Pattern Languages of Programming (PLoP), 2001).

[16] Weiss M.: "A Pattern Language for Motivating the Use of Agents", (P.Giogini et al. (Eds.): AOIS 2003, LNAI 3030, pp. 142-157, 2004).

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

[18] Zambonelli F., Jennings N. and Wooldridge M.: "Developing multiagent system: The Gaia Methodology", ACM Transactions on Software Engineering and Methodology, Volume 12, Issue 3, July 2003, pp. 317-370.