An Agent Design Pattern Classification Scheme:
Capturing the Notions of Agency in Agent Design Patterns

Ayodele Oluyomi  Shanika Karunasekera  Leon Sterling
Department of Computer Science and Software Engineering
The University of Melbourne
111 Barry Street, Carlton, Victoria 3053, Australia
{aoo, shanika, leon}@cs.mu.oz.au

Abstract

Agent technology is increasingly being used to develop software systems in different domains. Hence particular design problems are recurring in different multi-agent systems development projects. In order to enhance the reuse of proven solutions to recurring design problems, agent design patterns are being identified and documented. Understandably, most of the available work on agent design patterns reflect Object Oriented concepts (classes, objects, inheritance, etc) and exhibit an implementation bias. It is preferable for these patterns to reflect the notions of agency and be described at the right level of abstraction. In this paper, we present a new agent design pattern classification scheme that is structured to better reflect the notions of agency and allow varying levels of abstraction in describing agent design patterns. We also show how four different agent design patterns fit into our classification scheme.

1. Introduction

As the application of agent technology in building software systems is increasing, the awareness of agent system design problems is also increasing. Agent system developers are identifying recurring design problems. The concept of software design patterns has been successfully applied to the object oriented (OO) paradigm to address recurring design problems. Although different definitions exist for design patterns, they all convey a common theme of a pattern being a general (reusable) solution to a specific problem in context [15]. Inherent in the definition of a pattern are the properties that determine if the description of a design solution is a pattern or not. As a result, a design solution needs to be carefully described for it to be a valid design pattern.

Software agent developers and researchers are also investing efforts in identifying and describing design patterns for recurring design problems in agent system development [1, 2, 4]. However, most of the available work on agent design patterns reflect Object Oriented concepts (classes, objects, inheritance, etc) [1, 3] and are usually implementation biased. These indicate that such patterns describe more of the specifics of a solution rather than the core of the solution. These patterns are not described at the right level of abstraction to facilitate appropriate reasoning about the qualities of the various design options (patterns) for agent systems in the context of the problem at hand. Also, this does not allow for appropriate adaptation of existing patterns to suit the peculiar design needs of individual projects and it tends to lock developers into a particular implementation. In this paper, we present a two way classification scheme for describing patterns at the appropriate level of abstraction while reflecting notions of agency.

The paper is organized as follows. Section 2 presents the need for a different way of perceiving and describing design patterns for agent systems. The challenges of analyzing and designing agent oriented systems are analyzed in section 3. In section 4, we describe our scheme for organizing design patterns for agent systems, and some examples are presented in section 5. Section 6 is a discussion of this work while section 7 concludes the paper and highlights future work.

2. Motivation

Developers of software systems hold different views on the relationship between agents and objects. While some consider agents to be no more than objects with additional capabilities, others are of the opinion that agents are different from objects despite some similarities [5]. A crucial point to note is that both agents and objects have their unique applications in the development of software systems. The decision on whether objects or agents are most suited for a particular software system is largely informed by the nature and peculiarities of the problem.
domain. Perhaps, a combination of agents and objects is the most effective choice.

A fundamental difference is that an agent has a goal that it is designed to achieve. As such an agent needs to have strong control over its internal state and behaviour i.e. has a high degree of autonomy. Agent interaction is another major difference between agents and objects. While objects interact via method invocation, agents interact through a structured sequence of message exchanges i.e. agents are capable of social behaviour presenting a strong notion of collaborative behaviour in a multiagent system. Agents are reactive, they are capable of timely response to changes in their environment. Also, agents are proactive, they take goal motivated actions by themselves. Other differences include thread of control, data persistence and so on. [5] and [6] give detailed discussion of these differences.

It is necessary to identify and understand these differences between agents and objects because they impact significantly on the way agent based systems are engineered. This is evidenced by the efforts that are put into developing new Agent Oriented Software Engineering (AOSE) methodologies [9, 10]. The first impact of these differences on the engineering of agent systems is that agents fundamentally change the way we reason about systems that are being developed. Instead of having an encapsulated software system which external actors interact with, agents allow us to reason about the system as a virtual organization in which the players are defined as roles [7, 8]. This concept introduces a higher level of abstraction in reasoning about software systems thereby reducing the gap between human organizational behaviour and software systems that implement such behaviours. Secondly, different architectures have been defined for the internal structure of an agent such that the notions of agency that describe each agent is appropriately represented and implemented. These architectures include belief-desire-intention (BDI), layered architectures, logic based architectures and so on [6]. Hence the internal structure of an agent is more complex than that of an object. An agent (or agent systems) could be implemented using different technologies including object oriented programming, logic programming, and so on [11].

These discussions emphasize the different levels of complexity and abstraction that are introduced in agent systems and the differences in how agent systems are realized – implementation. Hence, reasoning about design and design patterns for agent based systems in terms of object oriented constructs significantly obscures the different levels of decomposition that is required and tends to restrict implementation of agent based systems.

3. Analysis and Design in Agent Oriented Development

Analysis and design in agent oriented development also draws from the lifecycle phases of software engineering. However, it is necessary to understand the different levels of abstraction in agent oriented analysis and design in order to achieve a systematic top down approach to developing agent based systems. The following stratification of the analysis and design of agent oriented development is based on the ROADMAP methodology [8].

In the recent revisions to the ROADMAP methodology, the highest level is the Goal model. The Goal model represents the overall view of the system describing its main objectives. Next to the Goal model is the analysis level, requirements specification is analyzed into roles and also the relationship amongst the roles in the system are defined. Next to this is the first level in design, at this level, agents are defined through either decomposition or aggregation of the different roles identified in the analysis stage. The agents’ definitions at this stage include their functionality and domain knowledge requirements. Also, at this level, the relationship and interaction between the agents are defined. The next level is the system design level (software architecture). This level describes the logical organization of the agents in the system into subsystems and the strategies for building the system. The next level of the agent oriented design is the agent architectural design level where the architectures of individual agents are designed. The internal architecture which could be BDI, layered, reactive etc represent the structural model of the components of the agent and how these components are interconnected [4]. The degree of the notions of agency that describe each agent is represented at this level. The next level in design is the agent implementation design level. At this level, the components of the individual agent’s architecture, the agent relationships and their interaction protocols are designed using the constructs of the technology (object oriented programming for instance) chosen to implement the agents. See figure 1 for a layout of these levels of abstraction in agent oriented analysis and design.
It is essential to have this stratification of agent oriented development analysis and design because it ensures that agent design patterns are identified and described at the appropriate level of abstraction, the notions of agency are captured in agent design pattern descriptions and appropriate categories are specified for classifying agent design patterns [4].

4. Two way classification scheme

Different design pattern categorization schemes have been presented both in the OO world and for the attempts at describing agent design patterns. The Gang of Four [12] specified three categories for OO design patterns namely creational, structural and behavioural. Design patterns are categorized as architectural patterns, design patterns and programming idioms in [13]. We agree with [14] that these are not quite applicable to agent systems based on reasons discussed in the preceding section. [1] categorizes agent design patterns into traveling, task and interaction strictly in line with the patterns they have so far identified. A view oriented approach for classifying agent design patterns is presented in [4]. It is however not clear where each of these views fall in the agent design levels and what
aspects of the design they relate to. [14] presents a set of agent design patterns described as agent architectural patterns and proposes other categorizations for agent patterns which are behavioural, cognitive, social and mobility.

Based on our reviews of the different categorizations and the understanding of design of agent systems, we conclude that a one way classification is not adequate for describing agent design patterns. We therefore present a two way classification scheme for agent design patterns. This scheme describes an agent design pattern in terms of the aspect and the level of design that it relates to. Horizontal classification describes the patterns that address the different aspects of design while the vertical classification describes the different levels of design as we analyzed in the preceding section. The horizontal classifications include:

**Structural patterns** Structural patterns describe the components that make up an entity and the connections amongst these components. An entity could either be the different subsystems at the system design or the internal architecture of an agent.

**Behavioural patterns** Even though this same word is used in the OO pattern classification according to the GoF to represent the interaction between classes and objects, our usage of the word implies a different meaning which is equally slightly different from its usage in [14]. Behavioural patterns here describe the characteristics of individual agents in terms of the degrees of their notions of agency, i.e. autonomy, pro-activeness, etc.

**Social patterns** These patterns describe relationships (e.g. team structures) as well as the interactions among the different entities in the system.

**Mobility patterns** These are patterns that address problems related with the migration of mobile agents around networks.

Vertical classification represents the different levels of abstraction in agent system design as described in the preceding section. These are role analysis patterns, agent design patterns, system design patterns, agent architectural patterns and agent implementation patterns.

Based on this classification scheme, an agent design pattern will for instance, be described as a *structural-internal agent design pattern*. Table 1 presents this classification scheme matrix.

This is a more comprehensive approach to categorize agent design patterns since the level of abstraction in agent design which the pattern applies to can be identified as the aspect of design it addresses is identified as well. This classification scheme makes for more effective reuse of agent design patterns as developers can adapt particular agent design patterns to the right stage and aspect of their development projects.

Table 1: Two way agent design pattern classification scheme

<table>
<thead>
<tr>
<th>Vertical</th>
<th>Horizontal</th>
<th>Structural</th>
<th>Behavioural</th>
<th>Social</th>
<th>Mobility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role analysis (Level 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent design (Level 2)</td>
<td></td>
<td><strong>x</strong></td>
<td><strong>x</strong></td>
<td></td>
<td><strong>x</strong></td>
</tr>
<tr>
<td>System design (Level 3)</td>
<td></td>
<td><strong>x</strong></td>
<td></td>
<td></td>
<td><strong>x</strong></td>
</tr>
<tr>
<td>Internal Agent design (Level 4)</td>
<td></td>
<td><strong>x</strong></td>
<td><strong>x</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent implementation (Level 5)</td>
<td></td>
<td></td>
<td><strong>x</strong></td>
<td><strong>x</strong></td>
<td><strong>x</strong></td>
</tr>
</tbody>
</table>

The matrix in table 1 shows how the combination of the horizontal and vertical classifications is used in describing a design pattern. Each unchecked box in the table indicates the high unlikelihood of having a pattern.
that will be described by such combination of categories. For instance, it is unlikely to have a "structural-role analysis design" pattern. However, it should be noted that the checked boxes on a particular level are not mutually exclusive as shown in the examples in section 5. In fact, this throws up other issues about the make up of some of the existing agent patterns. For instance, a pattern that structural and behavioural at the agent implementation level could suggest that it is really an implicit combination of two patterns. This and other such considerations are however, beyond the scope of this paper.

The following pattern template, based largely on the work of [13], was adopted by [14] after review and comparison of the major existing templates for describing design patterns, we augment this template to reflect our classification scheme:

**Classification and Name** – provides a memorable description of the pattern.

**Example** – provides a description of an actual problem that demonstrates the need for the pattern.

**Context** – is a statement of the conditions under which the developer might consider using this pattern.

**Problem** – describes the issues, problems, and challenges that the pattern is designed to address.

**Solution** – the basic principle that defines the pattern that solves the problem.

**Structure** – presented graphically to assist understanding of the functioning of the pattern.

**Dynamics** – the interactions between the participants of the pattern are described here.

**Implementation** – captures any known features and guidelines that will facilitate the use of the pattern.

**Variants** – any notable variants of the basic pattern described.

**Known uses** – description of any examples of the usage of the pattern.

**Consequences** – documents the particular benefits and liabilities resulting from the use of the pattern.

**See Also** – describes relationships to other patterns either as alternatives or dependencies.

**Aspect of Design** – describes the horizontal classification of the pattern according to the two way classification.

**Level of Design** – describes the vertical classification of the pattern according to the two way classification.

5. Examples

Here we present examples of how different agent design patterns that have been identified by different organizations fit into our two way classification scheme.

For the purpose of this illustration, we limit our description of each of these example patterns to a brief overview of the problem it solves and the structure of the solution. From our review of these, we are able to identify the aspect of design (horizontal classification) and the level of design (vertical classification) that each pattern addresses.

**Name:** Contract Net [16]

**Problem:** An agent (initiator) must perform a task using one or more other agents (participants) and it must ensure the optimization of an attribute of the task for instance time to completion, cost, etc.

**Structure of Solution:** The initiator requests bids of potential participants. The participants respond by either sending a bid or refusing to send a bid within a stipulated time frame. At the expiry of the stipulated time, the initiator selects one or more agents to perform the task based on the evaluation of their bids while rejecting the bids of others. The agents selected to execute the task then send the results to the initiator at the completion of the task. See figure 2 for a representation of the Contract Net pattern.

This overview of the pattern depicts that it addresses interaction amongst the agents in a multiagent system. According to the descriptions of the classes of patterns as presented in the preceding section, the horizontal classification of this pattern is a social pattern. The design of interaction amongst agents is carried out at level 2 (see figure 1) therefore the vertical classification of this pattern is Agent Design Pattern.

![Figure 2. The FIPA Contract Net Protocol](image-url)
**Name:** InteRRaP [17, 6]  
**Problem:** The agent architecture [requires distinction of the components that handle reactive, pro-active and social interaction functionalities of the agent] [requires distinct representation of the reactive, pro-active and social interaction related functionalities of the agent]. There should not be conflicts in the discharge of the duties of the separate functional components. An efficient representation of the world relative to each functional component is required.

**Structure of Solution:** The components are represented in a two pass layered structure with a control pass and a knowledge representation pass. The control layers from lowest to highest deal with reactive, pro-active/everyday planning and social interactions. There are three knowledge representation layers as well mapping directly to each of the control layers. The layers interact with each other and use a common world interface for perception and action in order to avoid the overhead of a supervisory control component. See figure 3.

From this brief description of the InteRRaP pattern, it is identified that it relates both to the design of the degree of notions of agency and also the internal architecture of the agent. This results in a combined horizontal classification of structural/behavioural and a vertical classification is internal agent design.

**Figure 3. InteRRaP – vertical layered**

**Name:** Ecological Recogniser [14]  
**Problem:** An intelligent agent (recognizing agent) must recognize the intention of other agents (intending agents) but has access only to the state data. Furthermore, the state data is complex and the intentions to be recognized are difficult to describe by sets of rules.

**Structure of Solution:** In this domain, the basic architecture of an agent is made up of three components which are perception, reasoning and action. To address this problem, include a recognition module called Pattern Matcher in the architecture of the recognizing agent. This pattern matcher is however a part of the perception component of the agent and it maps the pattern of incoming data stream onto learned, stored or otherwise supplied examples of the recognized intention. See figure 4.

From this brief description of the ecological recogniser pattern, it is identified that it relates to design of the internal architecture of the agent. Hence, the horizontal classification is social while the vertical classification is internal agent design.

**Figure 4. Ecological Recogniser Pattern**

**Name:** Meeting [1]  
**Problem:** A collection of mobile agents should carry out a task involving exchange of their resources e.g. mobile commerce agents to buy and sell goods on behalf of their clients. They should be able to continue interacting to carry out their task even when their origins (client’s machines) are disconnected from the network or located inside firewalls. The agents need to be able to identify each other when they arrive at the common destination for the transaction.

**Structure of Solution:** This pattern uses a Meeting class that encapsulates a destination and a unique identifier for the meeting. Participating agents are equipped with a meeting object and will locate a meeting manager object using the unique identifier to register itself. The meeting manager object notifies the registered...
participants about the newly arrived agent and vice versa. Each agent should unregister itself before leaving the meeting place. See figure 5.

This brief description of the meeting pattern shows that it addresses implementation issues (using the object orientation implementation technology) regarding additional entities that will realize the problem and interactions amongst participating agents/entities. Hence, the horizontal classification is structural, social and mobile while the vertical classification is agent implementation design.

A summary of the classification of the patterns presented in this section is presented in Table 2 below.

![Figure 5. Meeting Pattern](image)

### Table 2: Positioning existing patterns in the two way classification scheme

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Reference</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract Net</td>
<td>[16]</td>
<td>Social</td>
</tr>
<tr>
<td>InteRRaP</td>
<td>[17], [6]</td>
<td>Structural/Behavioural</td>
</tr>
<tr>
<td>Ecological Recogniser</td>
<td>[14]</td>
<td>Structural</td>
</tr>
<tr>
<td>Meeting</td>
<td>[1]</td>
<td>Structural/Social/Mobile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Agent Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal Agent Design</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internal Agent Design</td>
</tr>
</tbody>
</table>

6. Discussion

It is essential to clearly specify the problem to which the pattern applies in its particular context. The constraints (in terms of non functional requirements and system limitations) that give definition to the problem must be explicitly described as well. Otherwise developers may be applying a right solution to a wrong problem.

The description of a design pattern should emphasize the core of the solution to the problem in context [15]. That is, it should be general enough to allow adaptations of the solution to the specifics of the project to which it is being applied. The emphasis on the generality of the solution is to achieve reusability. The whole essence is for different software developers to be able to clearly identify similar problems in their projects and appropriately apply the pattern by adapting the solution it proffers.

Broadly speaking, two concepts stand out in describing design patterns that reflect the paradigm of their application. One is classification or categorization of design patterns, the other is the pattern template. The appropriate classification or categorization of patterns helps to put the problem in perspective and as such, this guides the identification of the right pattern that applies to a design problem by potential pattern users. The components or elements of a pattern template determine how well the problem, its context, and constraints are described. Also, they determine the quality of the solution that the pattern prescribes. Hence the importance of a comprehensive classification scheme as we have presented in this paper. We extend the template presented in section 4, to include two fields to identify the location of a pattern on the classification scheme.

7. Conclusion and future work

In this paper, we have described our two way classification scheme for categorizing agent design patterns. This scheme is a first step in capturing the notions of agency in describing design patterns for agent based systems. As such it follows a comprehensive stratification of the design of agent systems. The scheme is the combination of horizontal and vertical classifications such that each pattern can be described in terms of the aspect (horizontal) and level (vertical) of agent system design to which it is applicable. It is
expected that this will aid the work of pattern writers and improve the quality of patterns written for agent systems. Also, agent system developers will be better guided in choosing design patterns.

Work continues on these classifications and revisions will be made as more experience is gained in understanding the agent design space. The other major area of future work is the identification of what constitutes required, comprehensive but efficient elements of the patterns at the various levels of the classification scheme.

8. Acknowledgement

Supported by Melbourne Research Development Scheme

References

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
[16] Design Patterns Explained:A New Perspective on Object-Oriented Design by Alan Shalloway and James R. Trott - Addison-Wesley Pub Co; 1st edition (July 9, 2001)