A Case for Mobile Agent Patterns

Dwight Deugo
School of Computer Science,
Carleton University
1125 Colonel By Drive,
Ottawa, Ontario, Canada, K1S 5B6
deugo@scs.carleton.ca
http://www.scs.carleon.ca/~deugo

Michael Weiss
Business Communications Systems
Mitel Corporation
350 Legget Drive
Ottawa, Ontario, Canada, K1K 1X3
michael_weiss@Mitel.COM

Abstract: In this position paper, we make a case for the development of mobile agent patterns. Patterns have proven extremely useful to the object-oriented programming community. However, of the large amount of pattern research, little effort has been devoted to developing mobile agent patterns. We wish to correct this situation. We believe that the ongoing success of mobile agent systems depends on the development of software engineering principles for them. Patterns are a recognized means to this end, and one that we wish to promote.

Introduction

In most areas of research there comes a time when the researchers begin to understand the principles, facts, fundamental concepts, techniques, architectures, and other research elements in their fields of study. Research into agents, specifically mobile agents, is reaching that time. As evidence of this fact, one need not look any further than the calls for papers to upcoming conferences and workshops on mobile agents. Take for example, the call for papers for the workshop on Mobile Agents in the Context of Competition and Cooperation [MAC3 99]. We find comments such as, "… gaining more widespread acceptance and recognition as a useful abstraction and technology" and, "we are uninterested in papers that describe yet another mobile agent system." The question to answer now is what is important for us to do next as a research community.

To answer that question we must first consider why we do not want to hear about another mobile agent system. We will not answer for the various workshop and program committees, but we can propose one that we hear often from others. Since there are many mobile agent systems and frameworks in existence doing many of the same things, there is no use discussing another one because it too will do the same things and in similar ways. In other words, a new system does not often provide any new insights that are useful to the research community. However, we claim that these 'new' systems do validate, refine and show the reuse of many of the previously proposed and discussed research elements. Moreover, they bring with them additional thoughts, understanding and clarifications of the research elements. The problem is that when reporting on these 'new' systems, these insights get lost in the discussion of the system as a whole, or they are just not reported at all. For example, we have lost track of how many times we have read a paper that indicated it used KQML for the communication between agents and not been able to understand why it was used? There may have been an obvious advantage, or maybe it just did not matter. What we wanted to understand was which forces and context lead to this decision, because if we need to make a similar decision in the future we need this information.

We propose that, since as a research community we have reached a stage where some research elements in mobile agents are well understood and that there are several examples of each, it is time to begin the effort of documenting these elements as software patterns. This is not a matter of documenting the solution and problem surrounding each research element; this material is already evident in most agent papers. We need to go further and deeper in order to understand the forces and context of the problems that give rise to the proposed solutions. These are the undocumented and often misunderstood features of the research elements, which need to be elaborated before agent systems can enter the mainstream of software engineering and business applications.

Since many are not familiar with software patterns, and those that are think of them as only problem and solution pairs, we introduce patterns and pattern languages in order to help with their understanding. Next, we enhance our argument as to why agent patterns are important for agent research. We then compare and contrast agent patterns with their object-oriented counterparts. Finally, we present the current layout of the agent pattern landscape, identifying what some have started to do and what else we feel needs to be done.

What are Patterns and Pattern Languages?

Software patterns have their roots in Christopher Alexander's work [Alexander 79] in the field of building architecture. After reading his work, it was clear to software engineers that, like building designs, there are many

recurring problems and solutions used in the design of software systems. Unfortunately, they noted that many of these combinations were hard to find except for in the minds of the most experienced developers, for if they were, projects would have been built on time, within budget and without bugs! Moreover, knowing the problem and solution were not enough. Software engineers needed to know when the solutions were appropriate for the given problems.

Alexander's proposed the following definition for a pattern:

- A three-part rule which expresses a relation between a certain context, a problem, and a solution.
- As an element in the world, each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves.
- As an element of language, a pattern is an instruction, which shows how the spatial configuration can be used, repeatedly, to resolve the given system of forces, wherever the context makes it relevant.
- The pattern is, in short, at the same time a thing, which happens in the world, and the rule which tells us how to create that thing, and when we must created it. It is both a process and a thing; both a description of a thing which is alive, and a description of the process which will generate the thing

Although there are different pattern formats, the minimal format contains the following headings or ones dealing with similar subject matters.

- **Name:** As the saying goes in the object-oriented community, a good variable name is worth a thousand words and a good pattern name, although just a short phrase, should contain more information than just the number of words would initially suggest. Would the word agent be a good pattern name? The answer is no. Although it means a lot more than the single word suggests, it has too many meanings! One should strive for a short phrase that still says it all.
- **Problem:** A precise statement of the problem to be solved. Think of the perspective of a software engineer asking himself, How do I ….? A good problem for a pattern is one that software engineers will ask themselves often.
- **Context:** A description of a situation when the pattern might apply. However, in itself the context does not provide the only determining factor as to situations in which the pattern should be applied. Every pattern will have a number of forces that need to be balanced before applying the pattern. The context helps one determine the impact of the forces.
- **Forces:** A description of an item that influences the decision as to when to apply the pattern in a context. Forces can be thought of as items that push or pull the problem towards different solutions or that indicate trade-offs that might be made [Coplien 96].
- **Solution:** A description of the solution in the context that balances the forces.

Other sections such as resulting context, rationale, known uses, related patterns and implementation with sample code are usually included to help with the pattern's description.

A good pattern provides more than just the details of these sections; it should also be generative. Patterns are not solutions; rather patterns generate solutions. You take your 'design problem' and look for a pattern to apply in order to create the solution. The greater the potential for application of a pattern, the more generative it is. Although very specific patterns are of use, a truly great pattern has many applications. For this to happen, pattern writers spend considerable time and effort attempting to understand all aspects of their patterns and the relationships between those aspects. This generative quality is so difficult to describe that Alexander calls it "the quality without a name", but you will know a pattern that has it once you read it. It is often a matter of simplicity in the face of complexity.

Although useful at solving specific design problems, we can enhance the generative quality of patterns by assembling related ones, positioning them among one another, to form a pattern language, enabling us to use them to build systems. For example, individual patterns might help you design a specific aspect of your mobile agent, such as how it models beliefs, but a pattern language might be able to help you build all types of mobile reactive agents.

Agent pattern languages are very important for agent patterns to be successful. Forcing each pattern to identify its position within the space of existing patterns is not only good practice, it is also good research. In other words, all agent patterns should be part of an agent pattern language. It is not only helpful to you, but to all those other software engineers who will use the patterns to develop their systems in the future.

Why is Research into Patterns Important for Agent Research?

For any software system to be successful and run safely, it must be constructed using sound software engineering principles, and not constructed in an add-hoc fashion. Unfortunately, much of agent development to date has been done ad hoc [Bradshaw, et al., 1997], creating many problems – the first three noted by [Kendall et, al, 1998]:

1. Lack of agreed definition of an agent
2. Duplicated effort
3. Inability to satisfy industrial strength requirements
4. Difficulty identifying and specifying common abstractions above the level of single agents
5. Lack of common vocabulary
6. Complexity
7. Only goals and solutions presented

These problems limit the extent to which "industrial applications" can be built using agent technology, as the building blocks have yet to be exposed. Objects and their associated patterns have provided an important shift in the way we develop applications today, since the level of abstraction that we develop at is greater than doing functional and imperative programming. Since we believe that agents are the next major software abstraction, we find it essential to begin an effort to document that abstraction so that others can share in the vision. Patterns provide a good means of documenting these building blocks in a format already accepted by the software engineering community. Patterns also have the added benefit that no unusual skills, language features, or other tricks are needed to benefit from them [Lange and Oshima, 1998].

Are Agent Patterns different from their Object-Oriented Design Patterns?

Since many mobile agent frameworks, such as Grasshopper, AgentSpaces, Voyager, and Aglets, are implemented using Java, an object-oriented language, we argue that there must have been a few object-oriented design patterns used by the developers constructing them. Some patterns such as Mediator, Adapter, Broker, Strategy, and Composite, have been explicitly identified as being useful for the development of agent systems. [Kendall et al., 97]. However, since agents and agent systems are more dynamic and able to adapt to their environments than object-oriented systems, there must obviously be other patterns that are applicable only to them. We have already seen some of these patterns documented [Lange and Oshima, 98, Silva and Delgado 99], but we are positive there are a great many more.

In general, we find the format and structure of Agent Patterns similar to Object-Oriented Design Patterns. We also find that since many mobile agent frameworks are implemented using object-oriented technology, most object-oriented design patterns are applicable to them. However, differences in the way agents communicate, their level of autonomy and intelligence, and the fact that they are often highly mobile, has created a void not filled by existing patterns: agent, object, or otherwise.

One criticism of the previous breed of pattern writers is that they have not done a good job of relating their patterns to others – a pattern language. This is starting to change now as pattern languages become more frequent, but historically patterns have typically only linked themselves to others without identifying exactly what the relationship is. This is a problem that agent patterns writers must consider immediately. Describe how your patterns relate to others and how they can be used together or separately. Create a pattern language that can be used to build things: mobile agents and mobile agent systems. Do not create patterns that will only frustrate their readers, forcing them to discover how to apply the patterns together.

The Agent Pattern Landscape

The application of design patterns has shown us that it is very important and useful to start formalizing the experiences of developers and relating these formalizations with one another. There are a small number of agent patterns compared to the number object-oriented design patterns. As a consequence little work has been devoted to classifying them and, as mentioned in the last section, to defining pattern languages for them. The few classifications already proposed include the following:

- **Agent Patterns:** deal with the architectures of agents and agent-based applications [Kendall et al. 98, 97; Silva and Delgado 98; Aridor and Lange, 98].
- **Communication Patterns:** deal with the way agents communicate with one another [Deugo and Weiss, 99].

- **Travelling Patterns:** deal with various aspects of managing the movement of agents such as routing and quality of service Lange and Oshima 98].
- **Task Patterns:** deal with the breakdown of tasks and how these tasks are delegated to one or more agents Lange and Oshima 98].
- **Interaction Patterns:** deal with the way agents locate one another and facilitate their interactions [Lange and Oshima 98].
- **Coordination Patterns:** deal with managing dependencies between agent activities. [Tolksdorf 98]

As a general comment, patterns cover many different levels of abstraction. For example, some of them are used to describe the structure of an mobile agent system. Other patterns support the structure of agents and their relationships with different agents. While other patterns can be used to specify the design aspects of individual agents. The important feature here is not in developing the definitive classification. Rather it is more important for the mobile agent community to identify, specify and agree on the abstractions so that we can provide a common vocabulary for discussing and enhancing them, and, more importantly, building industrial strength mobile agent applications based on well-grounded software engineering principles.

Conclusion

What is in the future of mobile agent pattern research? Our prioritized list is as follows:

- **Identify an initial set of agent pattern classifications:** These classifications are to help focus pattern writers on targets that are of the greatest importance to those developing 'real' mobile agent systems.
- **Identify pattern languages within each classification:** These pattern languages are for pattern writers to develop and extend and will permit writers to position their new patterns within a know space of existing patterns.
- **Write the patterns.**

It is at MAC3 that we wish to debate and discuss these foundations with the participants in order to fill in the details of how to proceed and why bother at all. We feel it necessary to remind those involved with mobile agent research to not only write about solutions. Think, discuss, and write about the problems their solutions are intended to address and what context and forces led them to that particular solution. In short, we believe that following this approach, we will not have to read about "yet another mobile agent framework" anymore. Rather, we will be able to read and understand what problems a mobile agent system solves, and when we **should** consider using the approach!

References

Aridor, Y., Lange, D., "Agent Design Patterns: Elements of Agent Application Design", Proceedings of the Second International Conference on Autonomous Agents (Agents 98), ACM Press, 1998, 10-115.

Bradshaw, J.M., S. Dutfield, P. Benoit, J.D. Woolley, "KaoS: Towards and Industrial-Strength Open Distributed Agent Architecture", J.M. Bradshaw (Ed.), Software Agents, AAAI/MIT Press, 1997.

Coplien, J.O., "Software Patterns", SIGS Management Briefings Series, SIGS Books & Multimedia, 1996.

Deugo, D.L, "Communication as a Means to Differentiate Objects, Components and Agents", submitted to TOOLS USA 99, 1999.

Kendall E. A., P.V. Murali Krishna, Chirag V. Pathak, C.B. Suresh, "Patterns of Intelligent and Mobile Agents", Autonomous Agents '98 (Agents '98), 1998

Kendall E. A., M.T. Malkoun and C.H. Jiang , "Multiagent System Design Based on Object Oriented Patterns", The Report on Object Oriented Analysis and Design in conjunction with The Journal of Object Oriented Programming, June 1997

Lange, D.B., M. Oshima, "Programming and Deploying Java Mobile Agents with Aglets", Addison Wesley, 1998.

MAC3, "Mobile Agents in the Context of Competition and Cooperation", Autonomous Agents '99, http://mobility.lboro.ac.uk/MAC3, 1999.

Silva A. and J Delgado, "The Agent Patterns: A Perspective from the Mobile Agent System Point of View", EuroPLoP '98, 1998.

Tolksdorf, R., "Coordination Patterns of Mobile Information Agents", Proceedings of Cooperative Information Agents II, Second International Workshop, CIA'98, Springer, 1998, 246-261.