A Set of Agent Patterns for a More Expressive Approach

NUNO MEIRA IVO CONDE E SILVA ALBERTO SILVA

NAPFM@CAMOES.RNL.IST.UTL.PT IJCS@CAMOES.RNL.IST.UTL.PT ALBERTO.SILVA@ACM.ORG

INESC-ID & IST (TECHNICAL UNIVERSITY OF LISBON)
RUA ALVES REDOL, Nº9,1000 LISBOA PORTUGAL

Abstract. In this paper, we present a *set of agent patterns*, maintaining a tighter connection with the real world. In the real world there are many pattern behaviours, good solutions, that if transported to software agent paradigm will increase not only its capabilities and potential, as well as, possessing more expressiveness since they translate familiar concepts, make this emerging technology more easily accepted by the programming community.

1 Introduction

Several authors have proposed the use of *design patterns* as essential tools for intelligent software engineering [11,14]. We agree and would like to contribute to the enhancement of agent technology with our perspective about agent patterns.

An *agent* is a software entity with a well-defined identity, state and behaviour and represents, in some way, its owner in the accomplishment of tasks. The definition of agent here presented is by no means unique as there are many other attempts to define an agent. Being the issue of defining agent outsides this paper scope, we felt that the one we present is sufficiently clear and generic to serve the main purpose of this paper. A good compilation of agent definitions and explanations can be found, for instance in [2,10,15].

After some research on *agent patterns* [4,5,6,7], we found good patterns and solutions, which lacked some expressiveness that we feel, are essential. In the real world there are many good metaphors and analogies that can raise the expressiveness of those patterns, giving them a shape that is better defined and comprehensible. However, many times a pattern by itself takes a strange shape and it's hard to find a good metaphor or establish a good analogy, which means that, from our point of view, it is preferable to find an entity that contains a set of pattern behaviours (including the pattern in issue), that has a well defined form and from which may be possible to establish a direct relationship with its behaviour.

Some agent patterns that we are going to mention are based on patterns already defined by other authors and presented on papers to which we will make proper reference. However the work around these patterns of giving them the referred expressiveness recurring to metaphors, analogies and entities to which that type of behaviour can be delegated to. Other patterns will be new and constitute behaviours and solutions we got directly from the real world. These patterns catch solutions to problems like: how do agents communicate with other agents, located at the same place, without knowing which agents there are and how to reach them; how to contact or maintain contact with moving agents and others. Therefore, we think these patterns will capture your interest and bring you closer to this paradigm.

In the next section, we will briefly describe the pattern template that will be used for defining the agent patterns presented on Section 3 of this paper, namely the RECEPTIONIST pattern, the SECRETARY pattern, the SESSION pattern, the MOBILE SESSION pattern, the ANTENNA pattern, the PRIVATE SESSION pattern and the MEETING WITH MODERATOR pattern.

In the Section 4 ("Relating the Patterns"), we explain and justify the patterns existence by relating them with each other and better explain their differences.

In order to illustrate the potential of the defined patterns, we introduce on Section 5 ("Applying the Agent Pattern Language – An Example") a brief and simple example. Finally, on Section 6 ("Conclusions and Future Work "), we conclude this paper and introduce future work where the presented patterns have an important role.

2 Pattern Template

We present in this section the pattern template used in this paper. Our pattern template is an adaptation of the minimum format suggested by D.Deugo [5]. The template contains the headings that follow.

Name: Identification of the pattern using a representative name.

Introduction: Introductory description to the pattern recurring to metaphors and/or analogies.

Problem: Definition of the problem to be solved by the pattern.

Context: Description of one or more situations in which the pattern is applicable.

Forces: Description of the factors that may influence the decision of when should the pattern is applied.

Solution: Description of the solution generated by the pattern.

References: Indication of bibliographic references and/or patterns that inspired the defined pattern, in case they exist.

3 A Set of Agent Patterns

In this section we describe and discuss the set of patterns proposed in this paper. Namely, we describe, based on the pattern template referred on the Section 2, the following patterns: RECEPTIONIST, SECRETARY, SESSION, MOBILE SESSION, ANTENNA, PRIVATE SESSION and MEETING WITH MODERATOR.

As the following figures suggests the patterns discussed in this paper can be viewed as a specialized hierarchy of (1) agents themselves and (2) agent support communication mechanisms.

Related to the *specialized hierarchy of agents* (see Figure 1), we noticed that, in general, the majority of agent systems provide at least two basic notions: the *static agent* and the *mobile agent* [2]. Some systems, like AgentSpace or Aglets, provide a common and generic agent class (e.g., respectively `Agent` or `Aglet` class) independently of the mobility capability. Other systems, like D'Agents or Tacoma, provide two different top classes, and so, two different hierarchies.

Our work is defined on top of these concepts, so we introduce at least the following kind of software agents: at a high level, the “Generic Agent”; at a intermediate level, the “Mobile Agent”, and the “Static Agent”; and finally, at the lower level, the “Courier”, “Secretary”, “Receptionist” and “Specialized Service” agents. Other kind of agents can be defined for different contexts and application domains.

In this paper we don’t explain and consider the “Courier” and “Specialized Service” as relevant patterns because their understanding and application is easy and intuitive. So, we just consider the other kind of agents.

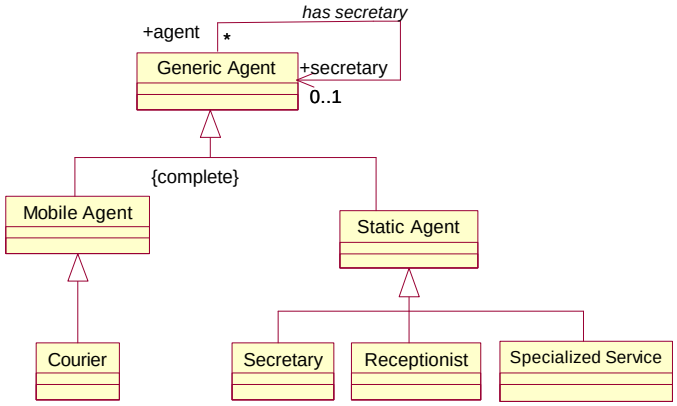

Figure 1: Agent hierarchy in typical agent-based applications.

Related to the *agent communication mechanisms* (see Figure 2), we identify and discuss the following concepts “Abstract Session”, “Session”, “Mobile Session”, “Private Session”, “Antenna” and “Meeting with Moderator”. Of course, other related mechanisms were already proposed and discussed elsewhere, such as “Meeting”, “Subscribe/Notify”, “Broadcast Session”, “Multicast Session”, “Two-Phase Commit Transaction”, “Contract-Net Protocol” [2,5,7,16].

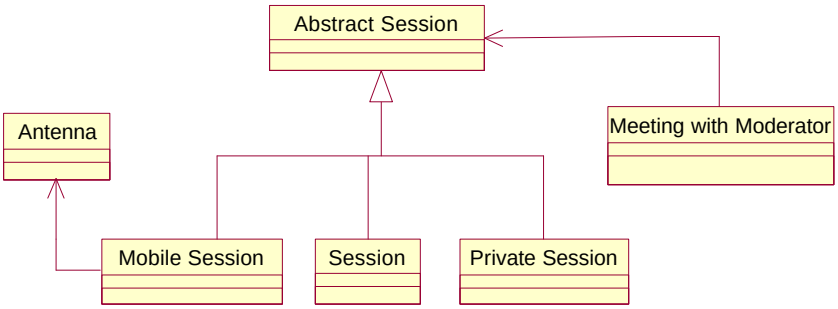

Figure 2: Agent support communication mechanisms.

It is important to clarify in this point that the “concepts” depicted above through UML class diagrams do not, necessarily, correspond to “classes” in the common sense of the object-oriented programming paradigm. On the contrary, these concepts correspond in general to sets of co-related classes described following the design patterns approach.

RECEPTIONIST Pattern

Introduction: Usually a person knows from start what is to be done in a given place although many times she has to obtain information when it gets there, like knowing where to go or whom to talk to in order to get information or obtain existing services. Sometimes you don't even know what kind of services there are so you have to ask. This kind of behaviour can be seen as a recurring situation on our daily life. It can and usually has a simple solution, which is to have a receptionist that provides to others this kind of information. It can also be used to one register oneself at the entrance, to ease certain kinds of interaction like warning someone that it is time to leave, or that there is someone who wants to talk with her. This solution can be easily transposed and adapted to the agent software paradigm.

Problem: How do agents communicate with other agents, located at the same place, without knowing which agents there are and how to reach them?

Context: A multi-agent system is being developed and collaboration between agents is necessary. In order to accomplish tasks, agents have to communicate with other agents physically located on the same machine.

Forces:

- When an agent reaches some place, he may not know what kinds of services are available at that place.
- An agent knows which services are available, but does not know hot to contact the agents that provide those services.

Solution: Use the RECEPTIONIST pattern, which can be implemented through a static agent, that contains: (1) a general description of the existing services; (2) the contact of every agent that provides those services as well as a description of what each agent does; and finally (3) a list of contacts of remaining agents present on that location. An agent, when reaching a place, registers itself at the RECEPTIONIST and asks what services are provided, if there is an agent that can provide a specific service and its respective contact. For this kind of interaction, the use of KQML [13] should be appropriate since this information exchange protocol is highly adequate to this kind of situation.

References: Inspired by the FACILITATOR pattern defined by Yariv Aridov and Danny B.Lange [7], but where the name "Receptionist" is associated with a well-known entity. An entity we can easily associate not only with the behaviour provided by the facilitator but also with other pattern behaviours, therefore not adding just one more strange entity for which there is no direct association with the real world.

SECRETARY Pattern

Introduction: In different kinds of work places there are secretaries that help their bosses accomplish their tasks like reminding them of their agenda, monitoring their tasks, present them intermediate results, control and contact people working for him and so on.

Problem: How to contact or maintain contact with moving agents?

Context: You are developing an application incorporating mobile agents. In order to accomplish tasks, agents have to reach others mobile agents that keep changing places, and therefore, their present locations are unknown.

Forces:

- Mobile agents change their locations frequently.
- An agent needs to contact another agent but does not know its present location.
- The owner of an agent wants to contact its agent to terminate a task or send intermediate results, but does not know its present location.

Solution:

By using the SECRETARY pattern that is itself an static agent where:

- Agents register themselves to the secretary on creation.
- An agent informs the secretary every time it changes its location or from time to time (in order to avoid message traffic overloading that can arise when agents change their locations too often).
- Every time an agent wants to send a message to another agent and doesn't know its present location, it sends the message to the secretary instead, and the secretary takes care of forwarding the message to the destination agent if possible, or can keep them so that the target agent can request them later (like a mail box).

References: Inspired by the PROXY AGENT defined by D.Deugo, F.Oppacher, J.Kerester, I.Von Otte [5], but once again we find that this pattern has no well-defined identity. That's not the case with the SECRETARY that can be easily associated to the kind of behaviour presented by the PROXY AGENT, but including as well other pattern behaviours, such as the control level by the user on the agent.

SESSION Pattern

(**Note:** This is the COMMUNICATION SESSION defined in [5].)

Introduction: Usually when we initiate a conversation with someone in a given place, we stay on that place talking until the conversation ends and only then we leave. If the contents of a given conversation are important it isn't usually good to jump from one place to another nor do other things at the same time, because this things can interfere negatively with the conversation taking place. This kind of behaviour can be seen as a simple pattern behaviour at the communication level of our daily lives, and can be seen as a session we are having with someone with a "beginning", a "middle" and an "end".

Problem: How do mobile agents or services can maintain a complex conversation between themselves in an easy and efficient way.

Context: A mobile agent system is being developed in which mobile agents interact with either mobile or static agents in complex conversations occurring over a period of time. The typical scenario is that two agents agree to communicate, interact by exchanging messages and finally stop their conversation.

Forces:

- Mobile agents change their position frequently.
- Messages of different conversations arrive interleaved in a random order. Complex interactions involve many messages.
- The design should be simple and easy to use. The design should restrict agents as little as possible.
- The communication should be as efficient as possible. You need to make it possible for agents to have simultaneously conversations and manage these conversations in a simple and efficient way.

Solution: Introduce the SESSION pattern. A session is an open communication link between two agents that is represented by a session object in each agent. Examples of this type of communication include message passing, RMI, sockets, or shared memory.

The use of sessions allows agents to manage several concerns. First, conversations are separate. Agents receive messages from a specific session object and can assign a handler to each. In this handler, the current state of the conversation can be encapsulated. Moreover, an open session indicates an ongoing interaction with a corresponding agent. Second, it is possible to separate session management from the mobility problem. We can force an agent not to change its location in a session or that such a change implicitly closes all open sessions. This allows a direct and simple communication between agents, which results in less overhead and higher performance.

References: This is the COMMUNICATION SESSION pattern defined by D.Deugo , F.Oppacher, J.Kerester, I.Von Otte [5]. We felt this should be included here since it introduces other patterns described later in this paper, namely the MOBILE SESSION and the PRIVATE SESSION.

MOBILE SESSION Pattern

Introduction: Many times, in order to talk to someone we don't need to stay on the same given place. For instance, when someone talks to another through the mobile phone she doesn't need to stop going where she is going.

This kind of behaviour can also be seen as pattern behaviour on our daily life, and therefore be seen as a session we are having with someone not only with a beginning, middle and end.

Problem: How do agents communicate with each other without having to worry about the mobility problem.

Context: A mobile agent system is being developed in which mobile agents interact with either mobile or static agents in complex conversations occurring over a period of time. The typical scenario is that two agents agree to communicate, interact by exchanging messages and finally stop their conversation, but during this process one or both agents may have to change their locations.

Forces:

- Mobile agents change their position frequently.
- Agents need to maintain a communication session and be able to change their locations at the same time.
- The design should be simple and easy to use.
- The design should restrict agents as little as possible.
- The communication should be as efficient as possible.
- The fact that agents change their locations should not increase the session's complexity.

Solution: Introduce the concept of MOBILE SESSION and ANTENNA (another pattern presented later in this article). The MOBILE SESSION is a communication link between two agents that is represented by a mobile session object in each agent and where messages should go through an antenna object, which is responsible for forwarding the involved messages.

The use of mobile sessions allows agents to solve several concerns related to the need they have to move. The conversations are separate. Agents receive messages from a specific mobile session object and can assign a handler to each, like in the SESSION pattern, and the details involved with their mobility can be encapsulated and solved by the ANTENNA pattern.

References: Inspired on the DIRECT COUPLING pattern defined by D.Deugo, F.Oppacher, J.Kerester, I.Von Otte [5] but adapted to a solution we feel is closer to the real world, without introducing any strange entities and providing a greater level of abstraction, since the agent never needs to know its communication partner location.

ANTENNA Pattern

Introduction: On the mobile session pattern we talked about mobile phones and, as we know, this phones use an antenna and that antenna is responsible for keeping the connection. When a phone starts getting out of range the connection enters the scope of another antenna so that the call doesn't break, and this happens without the intervenient knowledge.

Problem: How do agents communicate with each other without having to worry with the mobility problem.

Context: You want to support the mobile session concept already introduced in this article.

Forces:

- There is a need to keep the mobile session concept simple and separate from the way the message destination problem is handled, given the agents mobility.

Solution: Use an ANTENNA pattern. Each location (or at least each agent server) should have an agent of this kind that will be responsible for maintaining mobile sessions occurring between agents at a given place (by launching tasks for each link). Two agents communicating at the same location will be using the same antenna. When one of the agents moves, the antenna will be informed of that movement, using then the antenna that resides on the moving agent's destination. Note that even if an agent moves to more than one location he can always make sure that the number of antennas is reduced to a minimum, that is, two at the maximum (for only two agents). Also note that when an agent moves he only needs to tell the antenna and not all the agents with which the agent is having mobile sessions.

References: See the MOBILE SESSION pattern described previously in this paper.

There are other approaches to solve this problem. Agent servers are responsible for keeping the communication session between agents, providing a high-level of transparency (from the programmer's point of view). A possible approach to implementing this mechanism is based on the *proxy model* [17,18] (see the PROXY AGENT pattern [5]). Agents don't interact directly; they do it through the agent's proxy. When the agents moves to another node, it leaves a proxy to receive the messages addressed to it; the proxy then forwards the messages to the agent, as soon as it is established in the destination node. This gives full transparency to the programmer.

There are two ways to implement this mechanism; both of them also used in name management systems:

- *Centralized proxy*: There is only one proxy object, generally kept in the place in which the agent was created. In this case, any navigation operation implies a notification of the proxy, both before and after performing the operation.
- *Chain of proxies*: A chain of proxies is built. When the agent migrates, it leaves a proxy behind that is responsible for receiving messages from the previous proxy, and for forwarding them to the next proxy, and to the agent itself.

PRIVATE SESSION Pattern

Introduction: When you talk to someone and have a need for confidentiality you usually setup a safe place for that conversation to take place, and both intervenients move there and then start the conversation. This kind of behaviour can be seen as a pattern behaviour at the security level on our daily lives communications, and can also be seen as a private session we are having with someone that needs some restrictions for security reasons.

Problem: How do agents communicate with each other in a safe way.

Context: A mobile agent system is being developed in which mobile agents interact with either mobile or static agents in complex conversations occurring over a period of time. The typical scenario is that two agents agree to communicate, interact by exchanging messages and finally stop their conversation, but where some specified security restriction should be considered and the conversation must occur in a safe place.

Forces:

Mobile agents and other agents have to engage communication on a location that obeys security restrictions.

It is necessary to agree on the location where the communication will take place.

Solution: Introduce the PRIVATE SESSION pattern. A private session is a kind of session, represented by the private session object, where previous to the conversation itself, a negotiation phase takes place in order to define the location for that session. The private session object should have a list of safe locations for the session to take place, or any other way to determine it.

References: See also the SESSION pattern and the MOBILE SESSION pattern referred earlier in this paper.

MEETING WITH MODERATOR Pattern

Introduction: Many times when we have to talk over a given subject with more than one person we usually setup a meeting, hope everyone appears and then start the conversation without having the need to repeat the same thing to each of the persons involved. On such a meeting there is normally a person whose job is to coordinate the meeting which we call a moderator. A clear example of this could be the Parliament. This is also a recurring situation on our daily lives.

Problem: How can an agent communicates over a given subject with more that an agent in a simple way.

Context: A multi-agent system is being developed and collaboration between agents is necessary. In order to accomplish tasks, agents must communicate dynamically with other agents.

Forces:

Agents need to maintain communication with other agents over a common subject

An agent doesn't need to constantly worry about sending messages to multiple agents.

It is useful to abstract the synchronization in time and place needed for the interaction to take place.

Solution: Using a MEETING WITH MODERATOR pattern. Every time there is a need for a conversation between several agents, a "Meeting with Moderator" agent should be invoked. This agent is responsible for initiating a session (that can be either a session, a mobile session and/or even a private session) well suited for the situation, synchronizing messages between agents and terminating communication sessions.

References: Inspired by the MEETING pattern defined by Yariv Aridov, Danny B.Lange [7], but without any restriction to local interactions.

4 Relating the Patterns

Relating the patterns is probably the best way to clarify their existence, through their conceptual distinctions and also justifying the needs for such distinctions.

Starting with the RECEPTIONIST and SECRETARY, we can ask ourselves why shouldn't they both be the same agent. The answer in this case is a simple one. The RECEPTIONIST role is just an informative one, that is, it should not have and does not have any kind of control over the agents involved on.

On the other hand, the SECRETARY's role has to do a lot with agent control, that is, doing tasks in order to supervise, facilitate or coordinate the agent's jobs. As such, it also makes sense to exist a secretary for a given group of agents (or even several), while there is no need to have more that one receptionist per place. We can see the SECRETARY as the skeleton of a kind of service agent that can take a distinct shape, accordingly with the kind of work the agent that works for the SECRETARY, its capacity to manipulate that work's results and the way it can be presented to his creator (a user or any other agent owner). Furthermore, nothing forbids an agent that works for a SECRETARY to be also a SECRETARY. We think the reasons just presented are more than enough to justify the distinction, and certainly would not be difficult to find even more.

A similar question could be posed to the ANTENNA pattern: why shouldn't the RECEPTIONIST or the SECRETARY do what the ANTENNA is doing? The answer to this question may not be as straightforward as the previous one, but still exists and is conceptually important. As the SECRETARY is concerned there are two important issues to consider. First, an agent does not necessarily need to have a SECRETARY, which would mean in this case that without the ANTENNA functionality the agent would be impeached to have a MOBILE SESSION. Second, if the SECRETARY acts like an ANTENNA, and knowing the fact that it resides on the user's machine, then we could not take advantage of what distributed computing has to offer to the agent technology.

In what concerns the RECEPTIONIST, it doesn't make sense in conceptual terms to have it being responsible for the communication process between agents (or even being capable of). That's the ANTENNA responsibility. The ANTENNA represents a structure, like a building or a car, and as such it makes sense that it belongs to the system structure, i.e., as a component of the agent support system. Furthermore, only this way can the system administrator have some control over the extra message traffic originated by this technology, a problem for which in the future should be necessary to provide easy solutions.

In what concerns the remaining patterns (SESSION, MOBILE SESSION, Private Session, MEETING WITH MODERATOR) we feel there is no need to face them against each other, as this patterns only represent four distinct ways of well known interactions and therefore we think no special issues are raised here.

5 Applying the Agent Pattern Language – A Simple Example

To conclude, we present a simple example where the potentials of the presented patterns are illustrated, and how these can be related to each other and are used on a near future where the agent technology would be more spread and accepted throughout the Internet. Continuing with our example, we shall consider the following context:

- There is a courier (mobile agent), called "Agent-A", which goal is to get information about Australia (this is the reason it is illustrated as a "kangaroo"!).
- The Agent-A has a secretary.
- There is a service agent (called "yellow pages"), which can be seen as a search engine. This agent has information about what kind of services agents are able to get in a certain place, but does not have the contacts of these agents that can provide that information.

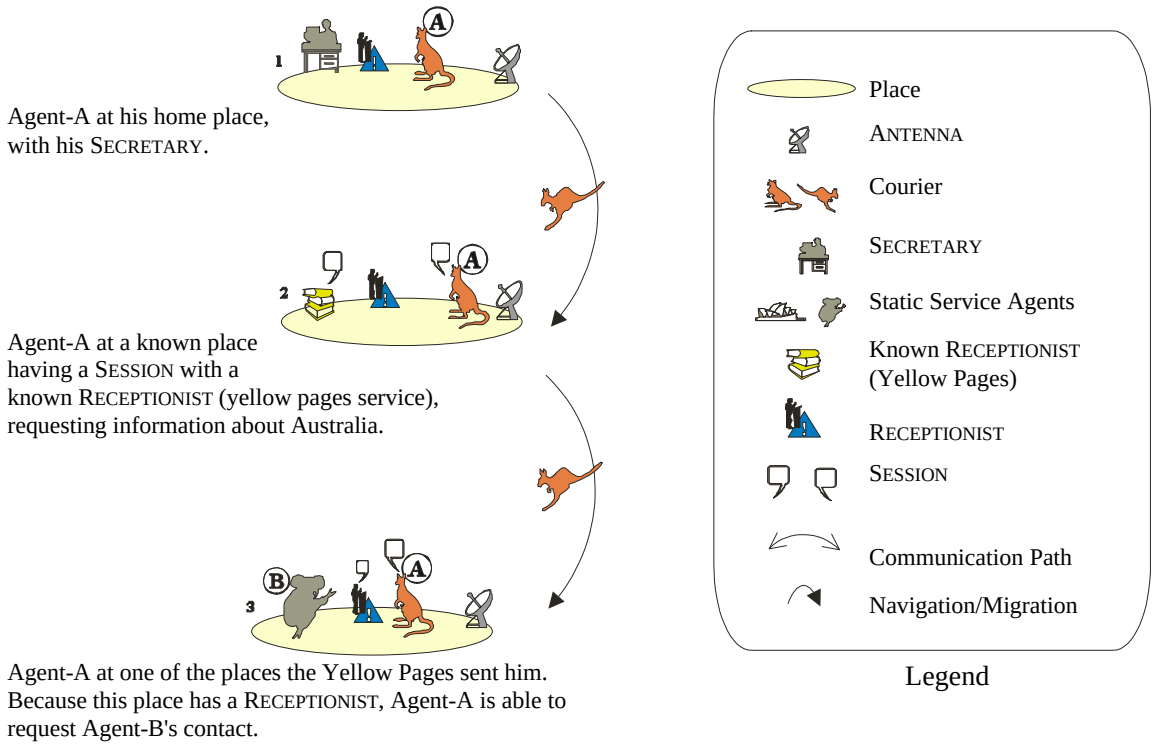

Figure 3: How does Agent-A discover the Agent-B's contact?

Because the yellow pages service does not have the direct contacts of the agents that can provide information about Australia, Agent-A tries to discover them (such as the "Agent-B") through the process illustrates in the Figure 3, based on the RECEPTIONIST pattern.

After Agent-A knows Agent-B's contact, if they have the option to establish a SESSION or MOBILE SESSION, they can communicate at least through four different ways as depictured in the Figures 4, 5, 6 and 7.

Communication based on the SESSION pattern (see Figure 4): Agent-A can start a SESSION with Agent-B, wait and get his results (4), and then go to another place (5).

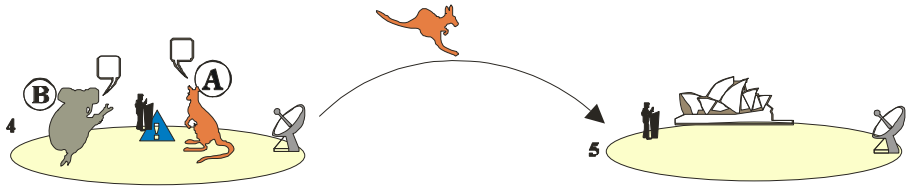

Figure 4: Case 1 – Communication based on the SESSION pattern.

Communication based on the MOBILE SESSION pattern (see Figure 5): Agent-A can start a MOBILE SESSION with Agent-B (4) and consequently, doesn't have to wait or terminate his session, if he wants to go to another place. Due to the establishment of the MOBILE SESSION, both agents would keep their interaction even though they are in different places (5).

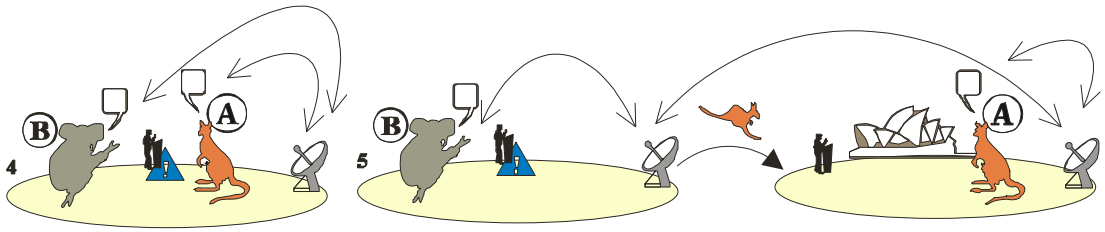

Figure 5: Case 2 – Communication based on the MOBILE SESSION pattern.

Communication based on the SECRETARY pattern (see Figure 6): Because Agent-A has a SECRETARY, he can ask Agent-B to send Agent-A's SECRETARY the results (4); and go to the next place to request more services (5). Then, Agent-B sends them to the SECRETARY, where she gets and holds the results (6) until Agent-A gets home or requests them in somehow.

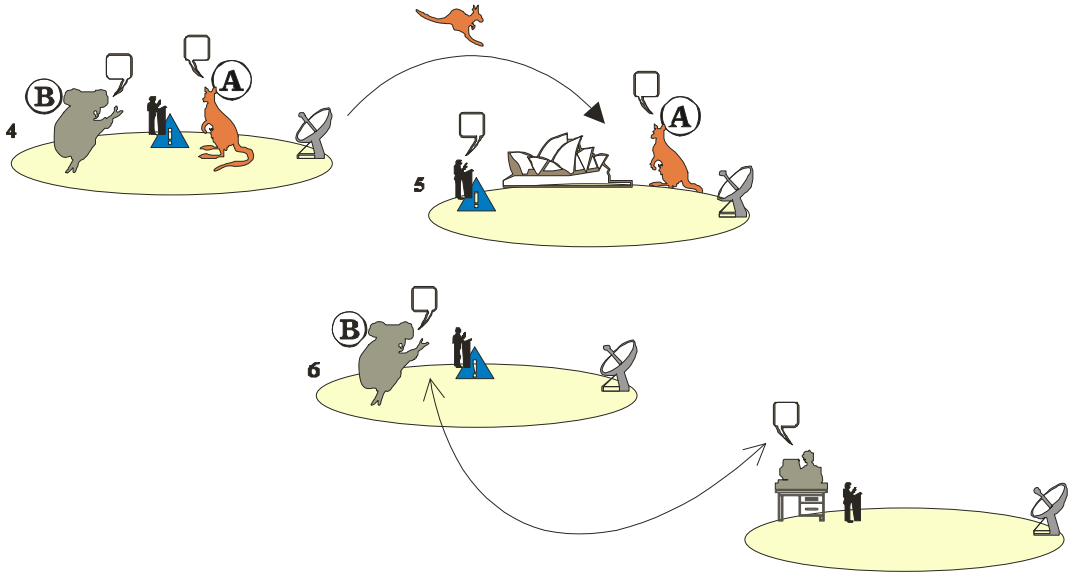

Figure 6: Case 3 – Communication based on the SECRETARY pattern.

Communication based on the SESSION and SECRETARY patterns (see Figure 7): Agent-A can also ask Agent-B to notify Agent-A's SECRETARY when the service is complete (4) and go immediately to the next place to request more services (5). Then Agent-B, when his task is completed (6), notifies the SECRETARY (i); she gets in touch with Agent-A (ii); and, finally, Agent-A can establish again a SESSION with Agent B (iii).

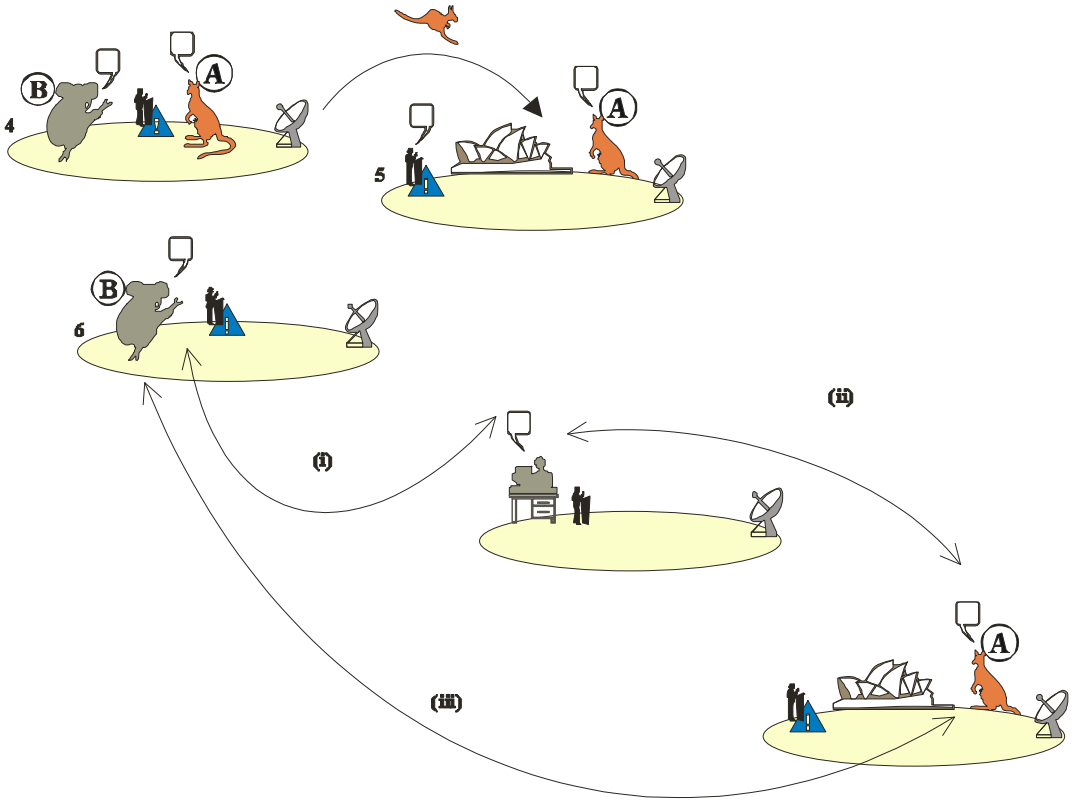

Figure 7: Case 4 – Communication based on the SECRETARY and SECRETARY patterns.

There can also be agents that move from place to place, consulting local RECEPTIONISTS for services provided and therefore updating the yellow pages in an easy and efficient way. The reason why the yellow pages don't also keep the agents contacts, is because, the agents that provide services in a certain place can change as well as their contacts. (For instance, if you want to buy a computer, you only need to know where you can buy it, not the person who is going to sell it.)

As this example, there are many others that can be easily imagined and depictured. However, this simple example allows us to show the application and use of the proposed set of agent patterns in an attractable and understandable way.

6 Conclusions and Future Work

Associating agent systems to a perspective based on the common sense of our society, will bring, from our point of view, not only a larger acceptance from the programmers community but also from future agent users. Our society is full of pattern behaviours that are familiar to everyone and that reflect many years of experience and therefore these patterns can generate good solutions to build agent societies.

Although it is not part of this paper main intent, we think a note should be added about the notation used on the previous section figures. The “cartoonish” style of the example on Section 5 is a symptom of the lack of a standard notation to represent concepts like places, static and mobile agents, agents in general, communication and interaction, messaging, mobility, and so on. Our first suggestion for an approach to this problem would be the use of a modeling language such as the Unified Modeling Language (UML) [9,12] and its emergent stereotypes [19]. Whatever the solution will be, future work on this subject should be important to avoid the proliferation of multiple notations, providing the community with a common language on graphical representations of agent systems, places and others.

We would also like to refer the fact that these patterns won’t end with this paper, since they will be developed and implemented on a developing platform on top of the AgentSpace framework [1]. In short, this work will concentrate on two main areas.

First, development of an API based on the implementation of agent patterns (where the patterns presented on this paper are obviously included).

Second, providing a high level development environment through the use of a modeling language like the UML, so that modelers, software engineers and programmers in general can develop their agent based applications in an easy and powerful way.

On a more theoretical basis, aspects of agent theory such as mobility, communication, interaction, tasks and others, would be explored in the future work

References

[1] “AgentSpace: An Implementation of a Next-Generation Mobile Agent System”, Alberto Silva, Miguel Mira da Silva, José Delgado, (Mobile Agents’98) *Lecture Notes in Computer Science, 1477*, Springer Verlag, 1998.

[2] “Towards a Reference Model for Surveying Mobile Agent Systems”, Alberto Rodrigues da Silva, Artur Romão, Dwight Deugo and Miguel Mira da Silva, *Autonomous Agents and Multi-Agent Systems Journal*. Kluwer Publisher (to be Published).

[3] "A case for Mobile Agent Patterns", Dwight Deugo, Michael Weiss, Carleton University, 1999.

[4] "Communication as a Means to Differentiate Objects, Components and Agents", Dwight Deugo, Franz Oppacher, Michael Weiss", Carleton University, 1999.

[5] "Patterns as a Means for Intelligent Software Engineering", D.Deugo, F.Oppacher, J.Kerester, I.Von Otte - IC-AI99, 502SA, 1999.

[6] "The Agent Pattern for Mobile Agent Systems", Alberto Silva, José Delgado, 1998, In European Conference on Pattern Languages of Programming and Computing, EuroPlop’98.

[7] "Agent Design Patterns - Elements of Agent Application Design", Yariv Aridov, Danny B.Lange, Proceedings of the Mobile Agents'98, ACM Press, 1998.

[8] “The Timeless Way of Building”, C.Alexander, Oxford University Press, 1977.

[9] “The Unified Modeling Language User Guide”, G.Booch, J.Rumbaugh, I. Jacobson, Addison Wesley 1999.

[10] “Is it an agent or just a program? A Taxonomy for Autonomous Agent”, S. Franklin, A. Graesser, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, 1996.

[11] “Design Patterns: Elements of Reusable Object-Oriented Software”, E. Gamma, R. Helm, R. Johnson, J. Vlissides, Addison-Wesley, Reading, MA, 1995.

[12] Object Managament Group, UML Revision Task Force. *OMG Unified Modeling Language Specification*. Version 1.3, 1999.

[13] "KQML - A language and Information Exchange", Tim Finnin, Rich Fritzon, University of Maryland, Technical Report CS-94-02,1994.

[14] Pattern Languages of Program Design 1-4 (Software Patterns Series), Addison-Wesley, 1995-1999.

[15] N. Jennings, K. Sycara, M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1), Kluwer Academic Press, 1998.

[16] “FIPA Contract Net Protocol Specification”, Foundation for Intelligent Physical Agents, 2000

[17] “Designing a Process Migration Facility: The Charlotte Experience”, Y. Artsy, R. Finkel. *IEEE Computer*, September 1989.

[18] “Concurrency, a Case Study in Remote Tasking and Distributed IPC”, D. Milojicic et al., *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, January 1996.

[19] “Extending UML for Agents”, James Odell, H. Parunak, Bernhard Bauer, AOIS Worshop at AAAI 2000.