

# Pattern Repository and Reuse in the PASSI Methodology

Luca Sabatucci<sup>(1)</sup>, Massimo Cossentino<sup>(2,3)</sup>, Salvatore Gaglio<sup>(1,2)</sup>

(1) DINFO - Dipartimento di Ingegneria Informatica, Università degli Studi di Palermo - Viale delle Scienze, 90128 Palermo, Italy

(2) Istituto di Calcolo delle Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche;

(3) SET - Université de Technologie Belfort-Montbéliard - 90010 Belfort cedex, France

sabatucci@csai.unipa.it; cossentino@pa.icar.cnr.it; gaglio@unipa.it

## I. INTRODUCTION

In the last years, multi-agent systems (MAS) have achieved a remarkable success and diffusion in employment for distributed and complex applications; experiences of industrial applications have been done, for instance, in e-commerce/e-market contexts where usage scenarios require high quality of design as well as secure, affordable and well-performing implementation architectures. In our research we deal with design process of agent societies; this activity involves a set of implications such as capturing the ontology of the domain, representing agent interactions (social aspects), and modelling the ability of performing intelligent behaviours. We consider that a fundamental contribution to the agent oriented software engineering (AOSE) could come by the adoption of proper reuse techniques and tools providing a strong support during the design phase. In pursuing these objectives we defined a reuse technique based on design patterns; this approach is integrated with the PASSI methodology [6], a step-by-step requirements-to-code methodology for developing multi-agent software.

## II. BACKGROUND

A typical approach to patterns classification can be found in the Gang-of-Four (GoF) pattern catalogue [1], where design patterns are classified according to two criteria: purpose and scope. With “purpose” authors refer to what a pattern does: they enumerate *creational* patterns (dealing with the process of object creation), *structural* patterns (dealing with the composition of classes and objects) and *behavioral* patterns (describing the interactions of classes/objects). The “scope” is directed to clarify patterns according to the different kind of elements they can be applied to (i.e. classes or objects).

A different, agent-oriented, classification is proposed by Lind in [3] where patterns are classified in accordance to the views defined in the MASSIVE methodology [2] that are: i) *Interaction* view, ii) *Role* view, iii) *Architecture* view, iii) *Society* view, iv) *System* view, v) *Task* view, and vi) *Environment* view.

Another interesting example of methodology dealing with patterns is Tropos [11], that uses a classification composed by five categories (or dimensions): i) the *social* dimension specifies agents and their interactions using a second order logic language; ii) the *intentional* dimension is focused on

services; iii) the *structural* dimension explores the internal composition of agents in terms of beliefs, events and plan; iv) the *communication* dimension focuses on agent interaction using specific protocols for describing communications; finally v) the *dynamic* dimension describes intentional and social actions.

A significant different way for organizing patterns in a catalogue uses a functional oriented classification; in [4] patterns are clustered in three categories: i) *traveling* (dealing with agent mobility issues), ii) *task* (regarding the breakdown of agent’s tasks and the delegation of them from one agent to another) and iii) *interaction* (dealing with agent communications).

## III. A REPOSITORY TEMPLATE FOR AGENT PATTERNS

In our approach [12], we summarize these classifications using a bi-dimensional space for the classification; this is summarized in Table 1: the first dimension concerns the *application context* that is a specific view (or design perspective) of the methodology to which apply the pattern. We enumerate four kinds of perspectives concerning with the PASSI methodology:

- *Multi-Agent patterns*. Concerning collaborations among two or more agents; they can be thought as an aggregation of roles (played by several agents) and rules to observe during the interaction.
- *Single-Agent patterns*. They are entire-agent patterns; these patterns propose a solution for the internal structure of an agent together with its plans for realizing specific services.
- *Behaviour patterns*. They propose solutions addressing specific agent capabilities, introducing features to agent behaviors; we can look at each of them as a collection of actions.
- *Action Specification patterns*. They address an atomic functionality of an agent; their granularity can be resembled to a method of a class.

The second dimension is the *functionality*; actually we consider four items in it:

- *Resource management*. It contains patterns dealing with information retrieval, manipulation of data sources, access to external resources.

Table 1 - The Pattern Classification in our Repository

		Application Context			
		Multi-Agent	Single-Agent	Behaviour	Action Specification
Functionality	Resource Management		Resource Sharing, Parallel Resource Sharing, Sequential Resource Sharing, Publish-Subscribe, Resource Caching		53*
	Communication	Request, Query, Inform, Contract-Net, Secure-Request, Secure-Query		Request (I/P), Query (I/P), Inform (I/P), ContractNet (I/P)	66*
	Internal Architecture		GenericAgent, LogAgent, Planner, Memento, Persistent-Agent	AStarPlanner	44*
	Deployment and Mobility	Explorer			7*

\* Names of these patterns have been omitted because the list would be too long and not really significant to the purpose of this paper

- **Communication.** It contains patterns representing solutions to the problem of interaction among agents using an interaction protocol.
- **Internal Architecture.** It contains patterns dealing with deliberation, plan management, message dispatching, knowledge management and other internal agent's basic functionality.
- **Deployment and Mobility.** It contains patterns that describe the physical environment and solutions involving the ability of moving from one platform to another

The documentation template we used in our repository is derived from [1], however the meaning of the keys is specialized for the agent-oriented paradigm.

- **Name.** The name of the pattern (preferably a single word or short phrase).
- **Classification.** The classification of the pattern addressing our bi-dimensional space.
- **Intent.** A short description of the pattern solution, its rationale and intent.
- **Motivation.** A description of pattern relevant *forces*, how they interact/conflict with one another, and goals they achieve. A concrete scenario which serves as the motivation for the pattern is frequently employed. Forces reveal the intricacies of a problem and define the kinds of *trade-offs* that must be considered in the presence of the tension or dissonance they create.
- **Preconditions.** The *preconditions* under which the problem and its solution seem to recur, and for which the solution is desirable. This shows us the pattern applicability context. It can also be thought of as the initial configuration of the system before the pattern is applied to it.
- **Postconditions.** It describes the state or configuration of the system after the pattern has been applied, including the consequences of applying the pattern, and other problems and patterns that may arise from the new context.
- **Solution (Structure, Participants and Collaboration).** Static relationships and dynamic rules describing how to realize the desired outcome. This is often equivalent to

giving instructions which describe how to construct the required work products. The description of this solution may indicate guidelines to keep in mind (as well as pitfalls to avoid) when attempting a concrete implementation.

- **Implementation availability.** Availability and compatibility of the implementation code of the solution with a selection of agent platforms. In our examples we consider Jade [8] and FIPA-OS [9] as implementation platforms.
- **Implementation description.** Comments on the most significant code fragments for illustrating the pattern implementation in the specific agent platforms
- **Related Patterns.** A guideline for browsing the pattern repository; this section expresses some functional relationships (if any) between the pattern with other ones. Related patterns often have an initial or resulting context in common, but a different way to approach the problem. Such patterns might be *predecessor* patterns whose application leads to another one; *successor* patterns whose application follows from the current one; *alternative* patterns that describe a different solution led by different forces and constraints; finally *codependent* patterns that may (or must) be applied simultaneously to the same context.

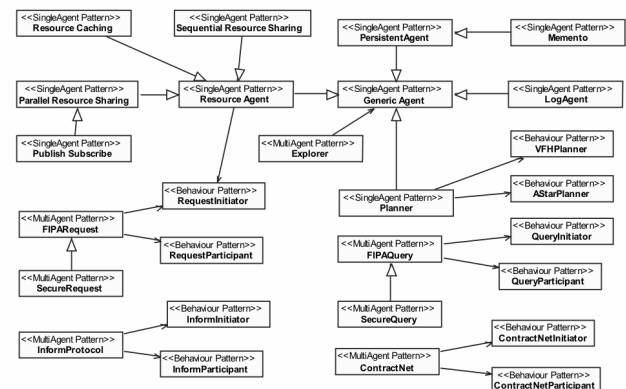


Figure 1 - Relationships among patterns in the repository

Our repository actually contains 27 patterns (among multi-agents, single-agent and behaviour patterns), and about 170 action-specification patterns. In Table 1 we can find a summary of our patterns according the classification described above. A specific interest has been focused on analyzing the relationships occurring among the patterns; *predecessor*, *successor* and *alternative* relationships are very useful for browsing the repository (user's point of view), while the *codependent* relationship is greatly useful when developing a tool for supporting the pattern reuse phase. In order to build our tool for the pattern support in the PASSI methodology, we have further distinguished the codependent relationship in *generalization* and *inclusion*.

The *generalization* (shown in Figure 1 using the UML inheritance association) is used to extend the responsibility of a pattern, adding new properties and features. It is the case of the *ParallelShareResource* that is a specialization of the *ResourceAgent*. This latter pattern specifies the structure of an agent who offers a service depending by a resource. The *ParallelShareResource* add a specific mechanism for access to the resource and updating its status.

The second type of relationship the *inclusion* (shown in Figure 1 using a standard UML association) occurs when a pattern includes another one in its solution for solving a sub-problem; for instance, the *ParallelShareResource* uses the *FIPRequestInitiator* pattern for handling incoming request communications.

#### IV. AN EXAMPLE OF AGENT PATTERN

As an instance of pattern documentation here we report the *ParallelShareResource* pattern.

**Name:** ParallelShareResource

**Classification:** resource management/single-agent

**Intent:** this pattern solves a problem of coordination: while continuously reading the status of a resource the agent has to provide a service based on this resource.

**Motivation:** when an agent provides a service depending from a resource (resource sharing) which status changes in time, there is a problem of coordinating the update activity with the resource sharing. The pattern solves this problem executing these two activities in parallel: a listener task is always ready for offering the service, and in the same time a cyclic behaviour continuously reads the status of the resource. In this

way when a request incomes, a reply is sent without any delay (thus obtaining a good response time). Two side effects are: i) the cyclic behaviour requires a lot of computational resources (it is always running), so other agents deployed in the same node may suffer of a significant slow down; ii) incorrect or older information may be given as a response because of the independency among service providing and updating cycle.

**Pre-conditions:** none

**Post-conditions:** the pattern uses the *GenericAgent* pattern for registering the service to the yellow pages and the *FIPRequest Participant* pattern for the incoming communications. The pattern specifically introduces a resource handling ability (to read/change the resource status) a synchronization mechanism to access the resource and a cyclic task for updating the resource status readings.

**Solution:** the solution proposed by this pattern is composed by only one participant: the *ParallelShare* agent. Figure 1-a shows the structure of this agent: it inherits the ability of registering itself to the platform (from the *GenericAgent* pattern) and the ability of manipulating a resource (regarded as an abstract element of the ontology domain). The agent also receives the ability of participating to a conversation with other agents (using the *FIPRequest* pattern) and uses a cyclic task for updating the *ResourceStatus* value (see Figure 1-b); this task has an abstract method *updateStatus()* where the programmer has to put the code for reading the resource.

**Related Patterns:** the *SequentialShareResource* and the *Publish-Subscribe* solve the same problem using different approaches. A solution could be to read the status of the resource only when required thus avoiding the cost in computational time (*SequentialShareResource*). When several agents are interested on the same resource the *Publish-Subscribe* pattern may be useful: the agent responsible of the resource manages a list of subscribed agents to notify every time the resource status changes.

#### REFERENCES

- [1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley
- [2] Lind J. 2001. *Iterative Software Engineering for Multiagent Systems - The MASSIVE Method*, LNCS 1994, Springer-Verlag
- [3] Lind J. 2002. Patterns in agent-oriented software engineering. online paper. "http://www.agentlab.de/agent patterns.html".
- [4] Malyankar R. 1999. A pattern template for intelligent agent systems. In *Workshop in Agent-Based Decision Support for Managing the Internet-*

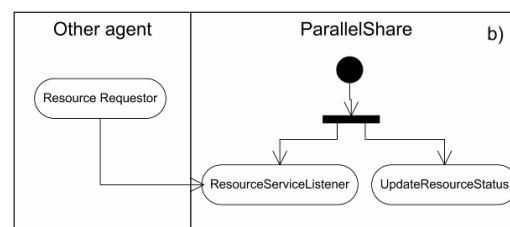
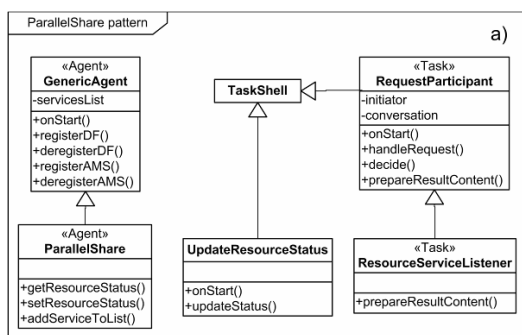


Figure 2 - Solution of the ParallelShareResource pattern: a) structure and b) behaviour

*Enabled Supply Chain*, Seattle, WA

- [5] Meira N., e Silva I. C., and da Silva A. R. 2000. An agent pattern language for a more expressive approach. In Proceedings of EuroPLOP
- [6] Cossentino M. 2005. From Requirements to Code with the PASSI Methodology, In *Agent-Oriented Methodologies*, edited by B. Henderson-Sellers and P. Giorgini, Idea Group Inc., Hershey, PA, USA
- [7] Aridor Y. and Lange D. B. 1998. Agent design patterns: Elements of agent application design. In Proceedings of the second international conference on Autonomous agents (Agents'98) Minneapolis, Minnesota, United States, pp. 108 – 115
- [8] Bellifemine F., Poggi A. and Rimassa G. 2001. Developing Multi-agent Systems with JADE. In proceedings of *The 7th international Workshop on intelligent Agents. Agent theories Architectures and Languages* (July 07 - 09, 2000) edited by C. Castelfranchi and Y. Lespérance, LNCS 1986, Springer-Verlag, London, pp. 89-103
- [9] FIPA-OS Website - [Available on Internet], <http://fipa-os.sourceforge.net>
- [10] FIPA Abstract Architecture - [Available on Internet] <http://www.fipa.org/repository/architecturespecs.html>
- [11] Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., and Perini A. 2004. TROPOS: An Agent-Oriented Software Development Methodology, *Journal of Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers 8(3), pp. 203-236
- [12] M. Cossentino, L. Sabatucci and S. Gaglio. 2007. An MDA Approach For Multi-Platform Agent Design Patterns. ICAR-CNR Technical Report n.02, 2007.