A Multi-Platform Architecture for Agent Patterns Representation and Reuse

Luca Sabatucci
Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
Consiglio Nazionale delle Ricerche(CNR)
Viale delle Scienze, 90128 -Palermo- Italy
sabatucci@csai.unipa.it

Massimo Cossentino
Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
Consiglio Nazionale delle Ricerche(CNR)
Viale delle Scienze, 90128 -Palermo- Italy
cossentino@pa.icar.cnr.it

***Abstract*— Design patterns already proved successful in incrementing the quality of object-oriented software; they are, therefore, candidate to play a similar role in the MAS (multi-agent systems) context. In this work we describe our proposal for patterns of agents and the tool to reuse them. With the support of this application the patterns can be applied to an existing agent or used to produce a new one. After the successful application of the desired pattern(s), the source code and the design diagrams (usually a structural and dynamic diagram) of the agents can be exported. In this paper we describe our approach together with the classification criteria and the documentation template we adopted. The experimental results we report demonstrate that during the realization of a FIPA-based MAS, an interesting percentage of code lines can be automatically produced.**

I. Introduction

In the last years, multi-agent systems (MAS) have proved successful in more and more complex duties; as an example, e-commerce applications are growing up quickly, they are leaving the research field and the first experiences of industrial applications are appearing. These applicative contexts require high-level qualities of design, affordable and well-performing implementation architectures and cost effective development processes.

In this context the design process assumes a fundamental role in order to properly model the system and the agents interactions. Several scientific works that address this topic can be found in literature; it is possible to note that they come from different research fields: some come from Artificial Intelligence (Gaia [17]) others from Software Engineering (MaSE [9], PASSI [14], Tropos [6]) but there are also methodologies coming directly from Robotics (Cassiopeia [7]). They give different emphasis to the various aspects of the process (for example the design of goals, communications, roles) but almost all of them deal with the same basic elements although in a different way or using different notations/languages.

In the proposed work, we pursue a specific goal: lowering the time and costs of developing a MAS application. In order to obtain this result, we think that a fundamental contribution could come by the automation of as many steps of the process as possible (or similarly by providing a strong automatic support to the designer).

In this scenario, the reuse of existing patterns has a great importance. We define a pattern as a representation and implementation of a frequently found system behavior (or portion of it). Each pattern is, therefore, composed of a model of the (dynamic) behavior, a model of the structure of the involved elements, and the implementation code.

In this paper we will illustrate our approach to the reuse of patterns for agents. In section II we define our view of design patterns for agents and we illustrate the documentation template used to catalogue them in a repository. In section III will show how our patterns can be used to lower the development time and the number of errors of multi-agent systems. In section IV we will illustrate an experiment realized with Jade and our tool, reporting the results obtained about automatic code generation. In section V we will discuss how our work is going on.

II. Patterns for Agents

The classical concept of design patterns has been well expressed by Christopher Alexander [3]:

> Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to that problem

According to this principle, the use of patterns can contribute to significantly enhance the quality of the software. This is one of the reasons of their great diffusion. In addiction to this argument, we think that agent patterns can also provide another important contribution to the development process, they could enhance the amount of code produced with a design tool.

As already stated, the problem we are facing consists in drastically lowering the cost of developing a multi-agent application; this is not so easy to obtain without simplifying the context. If we suppose that all the agents will be implemented with a specific FIPA-compliant [1] platform then we define the structure of the implementation, obtaining an useful simplification. In so doing, we selected two different platforms (JADE [5] and FIPA-OS [15]) that, together represent a greater part of the installed platforms in the Agentcities EU initiative [2].

A. Pattern of Agent Definition

Many works have been proposed (among the others [4] [13]) about patterns of agents. We now propose a concept of pattern that combines portions of design and the corresponding

implementation code. According to this definition a pattern is a portion of a multi-agent system demonstrating some kind of behavior that can be recurrently identified in designing MASs. This elementary piece, can be differently grained: for instance, we can find a pattern representing a couple of agents interacting in order to accomplish a cooperative goal, a single complete agent, a complex behavior of an agent or even a small piece of it. Then, the possibility of applying one pattern upon another will allow to iteratively build the application.

To maintain this level of generality, each pattern is composed by a model of the structure of the involved elements (an UML class diagram), a model of the dynamic behavior (an UML activity or sequence diagram) and the implementation code [8] that realizes them.

From the structural point of view, we consider four classes of patterns, described as follows:

- **Action pattern.** An atomic functionality of the system; it may be a method of either an agent class or a task class.
- **Behavior pattern.** A specific behavior of an agent; we can look at it as a collection of actions collected in a well defined structure (it represents a task in FIPA-OS and a behavior in JADE).
- **Component pattern.** An agent pattern; it encompasses the entire structure of an agent together with its tasks.
- **Service pattern.** A collaboration between two or more agents; it is an aggregation of components that involves always some type of communication between participants.

On the other hand, looking at the pattern functionalities, we enumerate four categories:

- **Access to local resources.** They deal with information retrieval and manipulation of data source.
- **Communication.** They represent the solution to the problem of making two agents communicate by an interaction protocol.
- **Elaboration.** They are used to deal with the agent's functionality devoted to perform some kind of elaboration on its knowledge.
- **Mobility.** These patterns describe the possibility for an agent to move from a platform to another, maintaining its knowledge.

B. Pattern Documentation

When the number of identified patterns grows the documentation assumes a great importance in order to:

1) simplify the identification of the right pattern for a specific problem;
2) support the software maintenance.

In Table I we report the documentation schema we used in our repository. While keys are sometimes derived by [11], their use and meaning is slightly different because of the specific needs of agent-oriented software. We also added/changed some entries in order to document our multi-platform code solutions.

TABLE I

THE DOCUMENTATION SCHEMA OF THE PATTERNS IN OUR REPOSITORY

Item	Description
Name	Name of the pattern
Classification	The classification of the pattern according to the categories reported in II-A
Intent	A description of what the pattern does and its rationale and intent
Motivation	A scenario that illustrates a design problem and how the agents and their tasks in the pattern solve the problem.
Pre-conditions	The initial situation in which the pattern can be applied.
Post-conditions	The consequences of the application of the pattern: what changes the pattern introduces into the system.
Structure	A graphical representation of the structure of the agent and its tasks (usually done with a class diagram)
Participants	A description of the agents involved in the pattern and their roles
Collaborations	A (graphical) representation of the collaborations of the agents involved in the pattern (if any)
Implementation availability	Availability of the implementation code for the FIPA-OS/JADE platforms. Availability of the UML diagrams of the solution (XMI) for importing them in the existing system design.
Implementation description	Comments on the most significant code fragments to illustrate the pattern implementation in the specific agent platforms
Implementation	Code FIPA-OS/JADE code of the solution
Related Patterns	Patterns that should be used in conjunction with this one

III. PATTERNS IN THE AGENT IMPLEMENTATION PROCESS

As already discussed, in a pattern we include its structure, dynamics and implementing code. All these constituents have to be totally compatible with the agent platforms that will host the resulting system.

In both of the implementation platforms we use (FIPA-OS and JADE), an agent is composed of the base agent class and a set of tasks/behaviors that are implemented as inner-classes (see figure 1). In dealing with this, we thought to adopt a hierarchical meta-language to describe the agent and its properties. The agent is the hierarchys root level entity; it contains inherent properties such as attributes, methods and the tasks/behaviors used to implement its dynamics.

If we consider an agent as an entity capable of pursuing specific goals, carrying out some operations (e.g. communication, moving across platforms, using some hardware resources), then we can also identify some agent patterns related to these agent's aspects.

The communication pieces of behavior are among the most common that we can identify. For example, if two agents communicate using one of the FIPA standard interaction protocols [10], parts of code devoted to dealing with this communication can be reused in other contexts. This is the case of the "Listener" task reported in figure 1, 2 where we describe a component pattern whose purpose is to share a resource that is continuously updated (task "Prepare Resource" of figure 2) and in parallel sent to other requesting agents (task

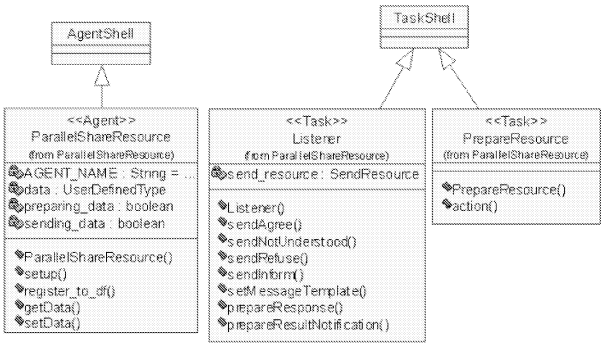

Fig. 1. ParallelShareResource pattern: structural model (Jade platform)

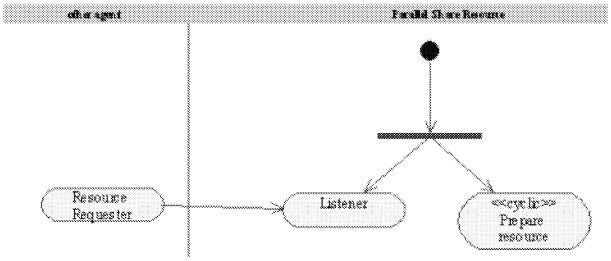

Fig. 2. ParallelShareResource pattern: behavioral model (Jade platform)

"Listener").

A. Meta-Patterns and Constraints

The key element of our multi-platform implementation of patterns consists in the introduction of a meta-level of representation of them.

A meta-pattern is platform independent and contains all the elements that are common to the different environments (JADE, FIPA-OS, most of other FIPA-compliant platforms) even if they have little differences. For example, meta-patterns refers to the *setup* method of tasks (a method usually devoted to implement the main duty of the task/behavior that is automatically invoked by the agent platform after the class constructor) using the abstract name "TaskSetup" while Jade uses the name "action" and FIPAOS the name "startTask".

When a pattern is applied to a project, it modifies the context in which it is placed, introducing new functionalities into the system. These additions may request to satisfy some preconditions or postconditions. For example in FIPA-OS, when we insert a communication behavior pattern (see section II-A) into an existing agent, the *Listener* task should have a *handleX* method to catch the different communicative acts (i.e.: Request, Agree, Inform, ...) of the chosen interaction protocol.

The relationship between the pattern and the existing agent elements could be expressed with a constraint. A constraint is a rule composed of two elements: a target and a content. The target specifies what agent/task will be influenced by the rule. The content expresses the changes that are to be applied when the pattern is inserted into the system; it could be an

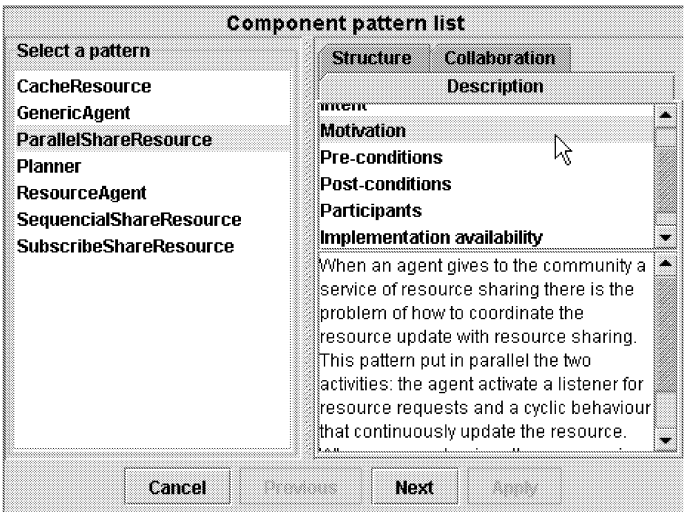

Fig. 3. AgentFactory: the selection of a component pattern from the repository

aggregation of attributes, constructors, methods and tasks.

B. A Tool for Patterns Reuse

In order to concretely use our pattern approach we developed a tool named AgentFactory[1]. It exists in three different versions: as a standalone application (see section 3), as a web-based applet and as component of the PASSI PTK tool (it introduces the support for agent patterns in the design process of the PASSI methodology [14]).

The main functionalities of this tool are:

- to maintain a repository from which the user can take patterns to be introduced in his/her project,
- to automatically generate code for a specific agent platform (FIPA-OS or Jade).

Our pattern meta-language representation is based on XML since it is oriented to tree data structure and it proved very useful for managing the repository of agents and tasks.

Moreover, agents source code can be automatically generated from their meta-language representations through the use of a series of XSL transformations that starting from the higher platform independent models bring to the actual code implementation.

C. Code Generation

The main purpose of the AgentFactory tool is to make the code generation process totally automatic. Starting from meta-pattern, in order to obtain the agent code for a specific platform, the tool executes the series of transformations illustrated in figure 6.

From the UML descriptions (class and activity/statechart diagrams) we obtain the XML meta-pattern (that is platform independent); from this, applying an XSL transformation, we deduct the platform (FIPA-OS or JADE) specific pattern. This first transformation specialize the meta-pattern for a

[1]Website: http://mozart.csai.unipa.it/af/

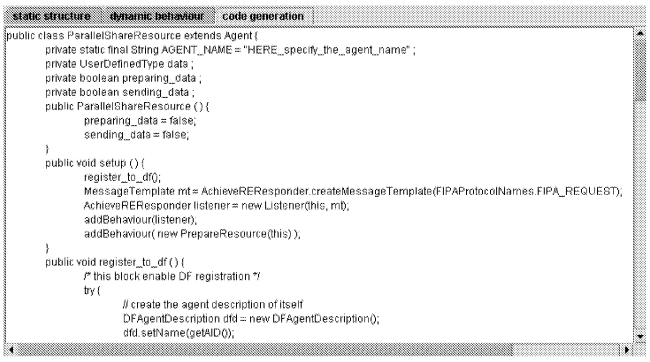

Fig. 4. AgentFactory: part of the code generated for the ParallelShareResource pattern

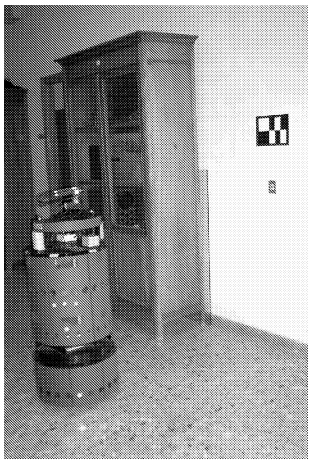

Fig. 5. The B21R robot performing the autolocalization looking at a wall-positioned marker during a surveillance mission.

specific environment. For example the "StartTask" (platform independent) methods are converted into the "action" (JADE) or "startTask" (FIPA-OS) ones according to the selected agent platform. Then, the obtained pattern is transformed in the JAVA skeleton of the agent with another XSL transformation. At this skeleton the tool adds the body of the methods (when available for the specific environment) taking them from the repository according to the dynamic behavior description. This produces a class that is complete both in the structure and in the inner code (see figure 4).

IV. Experimental Results

The experiment we are going to report consists in a robotic system devoted to surveillance tasks. More in detail the implemented functionalities comprehend the reconnaissance of the building, the detection of new objects in the environment (for example a bag that someone has forgotten) with the consequent update of the environmental knowledge and map description, the automatic detection of an intruder, the pursuit (and encirclement if more robots are available) of the intruder.

The software aspects of the application, are characterized by a multi-agent system implemented with a FIPA-compliant platform (JADE) with a massive use of patterns. The dimension of the whole project are quite interesting since more than 10 thousands of lines of code and 16 different types of agents have been produced (some of them have several instances at runtime). A particular effort has been dedicated to the vision subsystem that introduces a multi-level architecture allowing the dynamical introduction of new hardware (cameras) and services (agents performing different kinds of filtering and images manipulations) [12].

This MAS has been realized largely reusing the patterns of our repository. The AgentFactory tool has generated the initial version of the agent code of the project. This code has been manually completed adding the specific behavior that is not available in the pattern repository.

Table II summarizes the results obtained with this experiment: the whole project involved the development of about 10.6 thousands of lines of code (LOC) divided in: about 4 KLOCs of utility classes code, 4.5 KLOCs of agents code and almost 2 KLOCs of ontological descriptions. The time spent in developing the MAS and ontology (design, coding and testing phases) was about 7 man/months. About the 30% (1461) of the 4500 LOCs of the agents has been generated using pattens. Of these 1400 lines of code automatically generated, the 50% constitute the class skeletons while the remaining parts are method bodies. Therefore about 700 line of code generated with our patterns could not be generated by traditional UML CASE tools.

V. Conclusions and Future Works

In the previous sections we discussed a multiphase approach to the representation of patterns that using XML allows us to generate code for two different FIPA platforms (FIPA-OS and JADE) from platform-independent design patterns. In order to cover the entire process we use different representation languages (UML, XML and JAVA for the final code) and apply several transformations.

Our conviction is that pattern reuse is a very challenging

TABLE II
THE COMPARISON OF THE TWO DIFFERENT EXPERIMENTS (WITHOUT AND WITH PATTERNS REUSE) PERFORMED ON THE SURVEILLANCE ROBOTIC APPLICATION

	WITHOUT patterns		**WITH patterns**	
	LOC	**Months**	**Autom. gen. LOC**	**Months**
Utility classes (drivers, algorithms)	4150		0	
Study, Design, Implementation		9		9
Test and Tuning		2		2
Agents/Ontology	4557/1903*		1461/1903*	
Design		3		2
Coding		2		1
Testing		2		1
Total		**18**		**15**

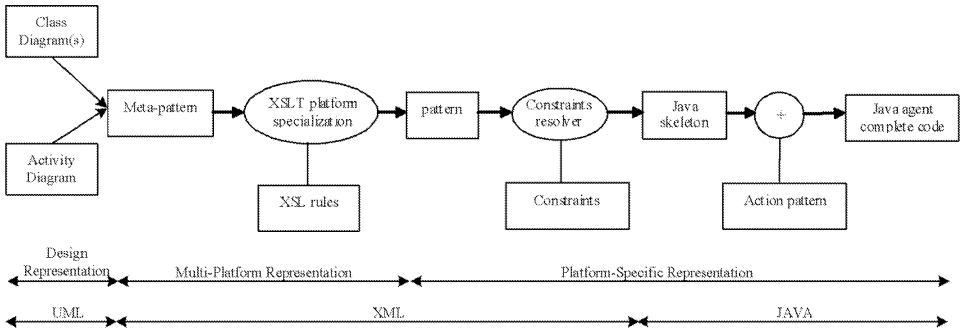

Fig. 6. The different phases of the Java code production for the proposed multi-platform pattern architecture

and interesting issue in multi-agent systems as it has been in the object-oriented ones. However we are aware that the problems arising from this subject are quite delicate and risky. Nonetheless, we believe, thanks to the experiences made in fields such as robotics and information systems, that great results could come with further works.

Experimental results encouraged us to continue on our path refining and improving pattern models and code. We are now interested in deepening the possibilities of design patterns code documentation [16] and we plan to include PCL (pattern comment language) entries in our patterns. In this way the automatically generated code will be more understandable by the users and the time to develop/maintain a system will be further decreased.

REFERENCES

[1] FIPA - Foundation for Intelligent Physical Agents, http://www.fipa.org.

[2] Agentcities.NET, http://www.agentcities.net.

[3] C. Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[4] Y. Aridor and D. B. Lange. Agent design patterns: Elements of agent application design. In *Second International Conference on Autonomous Agents*, pages 108–115, Minneapolis, 1998.

[5] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa2000 compliant agent development environment. In *Agents Fifth International Conference on Autonomous Agents*, Montreal, Canada, 2001.

[6] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: The tropos project. In *To appear in Information Systems*, Elsevier, Amsterdam, The Netherlands, 2002.

[7] A. Collinot and A. Drogoul. Using the cassiopeia method to design a soccer robot team. *Applied Articial Intelligence (AAI) Journal*, 12(2-3):127–147, 2000.

[8] M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing pattern reuse in the design of multi-agent systems. In *AITA'02 workshop at NODe02*, Erfurt, Germany, 8-9 October 2002.

[9] S. A. DeLoach, M. F. Wood, and C. H. Sparkman. Multiagent systems engineering. *International Journal on Software Engineering and Knowledge Engineering*, 11(3):231–258.

[10] Foundation for Intelligent Physical Agents. *FIPA Content Languages Specification*, 2001.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object Oriented Software*. Addison-Wesley, 1994.

[12] I. Infantino, M. Cossentino, and A. Chella. An agent based multilevel architecture for robotics vision systems. In *The 2002 International Conference on Artificial Intelligence*, Las Vegas (NV), USA, June 24-27 2002. ICAI'02.

[13] E. A. Kendal, P. V. M. Krishna, C. V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In *Second International Conference on Autonomous Agents*, page 9299, Minneapolis, 1998.

[14] M.Cossentino and C. Potts. A case tool supported methodology for the design of multi-agent systems. Las Vegas (NV), USA, June 24-27 2002. The 2002 International Conference on Software Engineering Research and Practice, SERP'02.

[15] S. Poslad, P. Buckle, and R. Hadingham. The fipa-os agent platform: Open source for open standards. In *5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, Manchester, UK, April 2000.

[16] L. Prechelt, B. Unger, M. Philippsen, and W. Tichy. Two controlled experiments assessing the usefulness of design pattern information during program maintenance, 1998.

[17] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–315, 2000.