A Semantic Description For Agent Design Patterns

L. Sabatucci[1], M. Cossentino[2], and S. Gaglio[1]

[1] Dip. Ingegneria Informatica, University of Palermo, Italy,
sabatucci@csai.unipa.it, gaglio@unipa.it,
[2] ICAR-CNR, Consiglio Nazionale delle Ricerche, Palermo, Italy,
cossentino@pa.icar.cnr.it,

Abstract. In last years, multi-agent systems (MAS) have achieved a remarkable success and diffusion in employment for distributed and complex applications. A fundamental contribution has come by the adoption of reuse techniques and tools providing a strong support during the design phase. Even though design patterns have been widely accepted by industrial and academic organizations as a proper technique for reuse, their definition still imposes deep concerns on contemporary software engineers. Design patterns are largely sensitive to different contexts where they are employed, especially on how they are blended with each other. This work introduces a design language for describing fine-grained pattern formalizations and compositions based on structural semantics. This formalization as been used in order to describe a couple of design patterns for agents and their composition.

Key words: Design Patterns, Multi-Agent Systems, Agent Oriented Software Engineering

1 Introduction

In last decade design patterns have been widely accepted by industrial and academic organizations, even if their definition and reuse still impose deep concerns on contemporary software engineers.

The pivotal difficulty stems from the fact that reuse of design patterns in realistic software systems is often a result of blending multiple patterns together rather than instantiating them in an isolated manner. The composition of design patterns can results in an intricate twine of pattern participants and the target application [14]. Pattern blending may entail significant morphs of the original pattern solutions through the merge of structural and behavioral elements. Also, the lack of explicit documentation for recurring compound patterns leads to design rationale being irrecoverable [4]. Patterns should be systematically documented so that they can be unambiguously instantiated, traced and reused within and across software projects [11, 5].

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 intelligent behaviours. In the following, we are going to pursuit a specific goal: lowering the time and costs of developing a MAS application without forgetting the necessary attention for quality of the resulting software and documentation. We have always considered design patterns for agents as a fundamental contribution to the agent-oriented software engineering. In past works we have defined some reuse techniques and tools based on design patterns [9, 17]; this approach has been integrated with the PASSI design process [7], a step-by-step requirements-to-code methodology for developing multi-agent software. Design patterns have been conceived as a crosscutting design activity occurring during almost all the phases of PASSI.

Now we concentrate on another aspect of pattern reuse. This paper deals with formalization of design patterns for agents, by using a semantic approach. Patterns are represented with semantic networks that can be modeled and transformed by using a set of operators. This formalization is perfectly suitable to be integrated in a tool that support the designer during the development process of a multi-agent system.

Section 2 presents motivations for pattern formalization and composition. A design approach based on the semantic description of patterns is proposed in Section 3 and a graphic notation, the Pattern Semantic Description is discussed. Section 4 introduces an expressive yet simple set of operators for *unifying*, *conjoining*, *concealing* and *externalizing* pattern elements, illustrating a composition example. Section 5 reports an analysis on the reusability and expressiveness of our language. Finally, some concluding remarks are reported in Section 6.

2 Motivation

This section presents an analysis of heterogeneous forms of pattern blending, which are commonly found in real multi-agent system development. This analysis provides some motivations for the introduction of our language.

The Digital Business Ecosystem [10] consists of a research project supported by the European Commission's 6th Framework Programme IST Thematic Priority. In particular, the Sicilian DBE project supports Sicilian micro and middle size companies in order they can access to advanced information and communication technologies to grow their business. We have realized a multi-agent system, the *Sicilian Digital Business Ecosystem Simulator* (SDBE Sim), that simulates the business evolution of companies, in which several design patterns for agents have been used and combined to achieve system requirements of scalability and adaptation. Figure 1 shows two design slices in which two patterns have been used and combined. In Figure 1.a the internal architecture of the Company agent is shown. This agent simulate the core business of a company. The agent provides the user a GUI where simulation data can be edited. Other agents, for example the ISESearcher agent, may request company's business data by using a FIPA Request communication protocol [12]. According to agent autonomy feature the Company agent may decide whether to give back its data or to refuse.

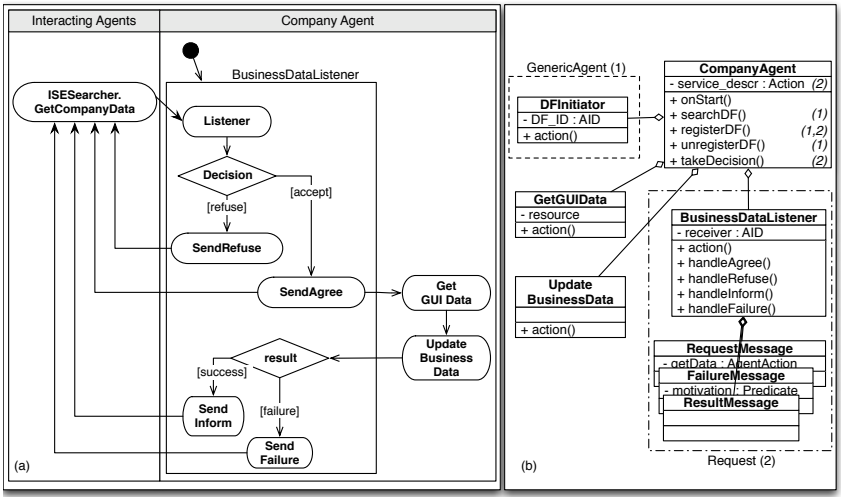

Fig. 1. Two PASSI design slices of the SDBE case study. (a) Task Specification for the Company agent. A number of agent's tasks and interacting agents were omitted for simplification reasons. (b) A Single Agent Structure Description diagram that illustrates the implementing class structure of the same agent. This diagram has been decorated with references to patterns that have been employed.

If the agent accepts the request, it must read user's data from the GUI and eventually update its core business. After this update, the agent may reply to the request message. Figure 1.b reports the Jade [2] implementing structure for the Company agent. This class diagram contains a class for the agent, and another class for each agent's task. In addition three message classes, are shown, that are used during conversations. This class diagram is useful to illustrate the employment of design patterns for implementing this agent. The *GenericAgent* pattern as been used to give the Company agent the abilities to interact with the system Directory Facilitator. This is fundamental for agents that want provide services to the community. This pattern has introduced the searchDF, registerDF, unregisterDF methods in the agent class, and a new DFInitiator task class. The *Request* pattern has been used, in order to give the agent the ability to participate to communications by using the FIPA Request protocol [12]. This pattern has introduced the service_description attribute in the agent class that contains a reference to the action to execute to provode the service. The registerDF is modified in order to register the agent's service and a new method takeDecision is introduced to encapsulate the decision process in order to evaluate whether accept or refuse incoming requests. Finally, the BusinessDataListener task class and three message classes are introduced in the structure in order to handle incoming communications.

The attachment of a number in the CompanyAgent class implies that the respective method or attribute is part of the implementation of the corresponding

pattern. Each number represents a specific pattern and the aim of this representation is to illustrate how various pattern realizations affect internal members of a single class. The result of the application of these two patterns, is a typical pattern composition that occurs when implementing an agent that offers services to a community. In the composition each pattern introduces specific agent abilities and features, and some of these are influenced by more than one pattern at the same time. This creates a synergy of pattern functionalities and responsibilities that are merged together in a new pattern structure. The remaining part of this pattern illustrates how pattern compositions may be formalized and employs in the agent paradigm.

3 Semantic Description For Pattern Structures

Last years revealed the importance of semantic description of data, especially in certain context, such as the web. The semantic descriptions are generally based on the use of ontologies in order to structure informal description in hierarchical relationships of concepts on which it is possible to operate logical reasoning [19]. The remaining sections describe a fine-grained design approach to support the pattern solution definition that is based on a set of constituents that can be combined in order to define the structure and the behavior of the solution. In order to discuss the language we give some definitions:

Pattern Description Element (PDE). An atomic constituent of a pattern that describes the structure or the behavior of the solution. They are: (i) participants, (ii) collaborators, (iii) events and (iv) actions.

Language Element (LE) Element of the target programming language used for implementing the pattern. Language Elements are expressed by using elements of the meta-model. In our repository for agents, examples of LEs are: agent, organization, communication, role and task.

Affected System Element (ASE) Element of the system that is influenced by the pattern application. A typical example of ASE is a business class that is assigned to a participant of the pattern. Its structure is modified because it must be compliant with PDE constraints.

PSEs are the constituents of a pattern solution. The definition of pattern solutions encompasses alternant levels of stability: some PDEs (Pattern Description Elements) are precisely described and do not require further details through the pattern instantiation, whereas some others are only sketched and their concrete definition is delayed to the pattern instantiation phase. The structure and behaviour of those PDEs depends on the application context and on the other patterns to which they are going to be composed. This kind of PDE supports the generalization and reuse of patterns in very distinct contexts where the nature of the problem may be different.

Participant. Participant are placeholders for assigning responsibilities to ASEs. This pattern constituent is similar to the classic "role" element, introduced

in [13] and detailed in several pattern formalization approaches [16][15]. Participant is a more general concept because roles can be played by classes only, whereas every element of the MAS meta-model (for instance an Agent or a Task, or even a Communication) may be a participant of the pattern. Responsibilities assigned to a participant will be taken by LEs that are assigned to this participant.

Collaborator. A collaborator is a concrete element of the pattern, totally defined in every its feature. It is used in order to introduce in the system an element that generally mediates other pattern's elements with a standard behavior. Its behavior is standard, and except for special situations, does not require a further specialization. A collaborator owns a type, which refers to a LE (Language Element). Therefore, a collaborator may be an instance of any element of the MAS Meta-Model.

Event. An event encapsulates an abstract circumstance that is the cause of triggering a specific behavior, involving one or more pattern elements. Typically the context that generates an even is external to the pattern. It is an non deterministic condition (from the point of view of the pattern) generated by the specific need of a participant. The event execution may be considered as a service request operated by the participant in order to produced the desired behavior.

Action. Together with events, actions have a fundamental role in the definition of the behavior of a pattern. An action encapsulates what happens when an event occurs. Actions must not be considered as merely methods. In the agent paradigm, actions may correspond to agent's abilities or tasks depending by implementing issues.

For problems of space, in this paper we mainly focus on the static structure of the paper. Even if the formalization approach also supports the dynamic description and composition this is only shortly discussed.

3.1 The Pattern Semantic Description Diagram

Semantic networks are often used as a form of knowledge representation. They represent declarative graphic representations that are expressed with models of interconnected nodes and arcs. Semantic networks generate machine-readable dictionary that can be used either to represent knowledge or to support automated systems for reasoning about knowledge [18].

Several graphical notations exist for representing a semantic network. The UML class diagram is often used in order to represent concepts and their relationships. This diagram is perfectly suitable for our aims, but we have introduced the two stereotypes *participant* and *collaborator* in order to immediately distinguish participants and collaborators. Relationships are used to connect participants and collaborators, thereby creating a semantic network.

The class diagram that uses these stereotypes to describe the semantic structure of a pattern is named Pattern Semantic Description (PSD) diagram. An example of this diagram is shown in Figure 2, that illustrates the *Request* pattern.

In this figure we have introduced a graphic notation in order to reduce the space: participants are shown by using ovals, whereas collaborators are shown by using boxes. This description is designed to assert propositions about the structure of pattern solutions (assertional networks [18]). Participants and collaborators are the *concepts* of this network. In particular, collaborators are concepts whereas participants are classes of concepts. Relationships express semantic connections among these concepts. In a PSD relationships can not freely connect any kind of concepts; they are precisely ruled by the MAS meta-model [1, 3, 8].

The information in a PSD is a set of conditions that should be contingently true in order to apply the pattern in the system where the problem occurs. The aim of PSD diagrams is twofold:

- they are human readable, so that designers can easily understand and apply the rationale of each pattern solution;
- the use of a limited set of concepts and relationships allows the realization of a parser for automatic interpretation of patterns, thus resolving syntactic ambiguities.

3.2 An Example of Pattern for Agent

The pattern, we consider here, is the *Request* pattern, from our repository, briefly introduced in Section 2. This pattern has been conceived for giving agents the ability to initiate and participate to communications that are compliant to the FIPA Request protocol.

This communication is useful in several circumstances. The delegation of a task is a typical scenario useful to illustrate the aim of this pattern. It occurs when an agent has to perform an action (for example, an interaction with a physical resource) but it is not able to do it or it has not sufficient permissions. So the agent can ask to another agent of performing that action (because of agents autonomy, the involved agent can refuse or accept the request according to its personal goals).

The analysis of the description of the protocol, reported below, has been the core for the identification of the participants of this pattern (Figure 2)[M1]. The elements that are included in the description of the protocol but are outside the definition of the pattern are the two agent roles: initiator and participant. Any

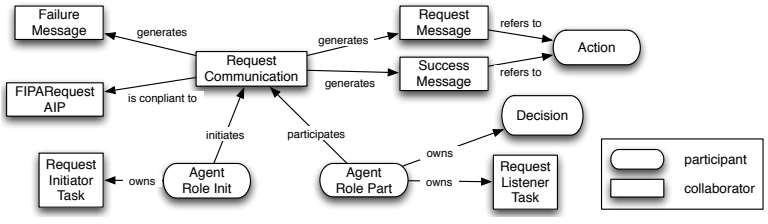

Fig. 2. Pattern Semantic Description diagram for the *Request* pattern

couple of agents may participate in this protocol by simply playing these two roles. This is the reason for which we defined these two roles as participants of the pattern: (i) *Agent Role Init* is responsible to begin the communication and (ii) *Agent Role Part* is responsible to maintain a listener for the request communication. Designer must specify the couple of agents to assign to these participants. Other elements that can not be encapsulated in the pattern definition are: (i) the *Action*, that is requested by the initiator to the participant by using the communication and (ii) the *Decision* task used to encapsulate the decision process to activate when the agent receives the request.

Figure 2 also shows the collaborators of this patterns. The *Request Communication* is the description of the kind of communication among initiator and participant. This must be compliant to the FIPA Request protocol [12], whose description is given in the *FIPARequest AIP* collaborator. Several *Messages* are included for information exchange. In addition, two tasks are defined as collaborators: (i) the *Request Initiator Task* is responsible to send the request message and wait for a reply, and (ii) the *Request Participant Task* is responsible to wait a request message and to reply with a result.

4 Operators for Pattern Blending

This section presents the operators for pattern composition based on the fine-grained pattern elements. These operators can be used in order to modify the structure of the pattern solution that is represented by a semantic graph in PSD diagrams. The following list briefly describes all the static composition operators. Their concrete usage will be later discusses in a detailed example.

Unification The unification is used to express overlapping compositions. The rationale behind this operator is to operate fusions of couples of static elements with a consequent merging of responsibilities. The result is to overlap the structure of two patterns using the two elements as pivots for the operation. This produces strong changes in the resulting pattern structure.

Conjunction The conjunction operates a conservative pattern blending. The rationale behind this operator is to create a synergy among the responsibilities of two patterns, by maintaining them separated. The two elements are linked by a new element, introduced in the structure. Only marginal changes are visible in the resulting structure of the involved patterns, promoting the traceability of the involved elements.

Concealing This unary operator has been conceived to modify the nature of a participant into a collaborator. The responsibilities assigned to a participant are imposed to the elements of the system that participates to the pattern. Concealing a participant means that all its responsibilities are delegated to a collaborator. They are no more visible outside the pattern. The visible effects of this operation are i) to allow mixed composition (unification and conjunction) among participants and collaborators ii) to internally set some responsibilities in order to assign a standard behavior and iii) to reduce the complexity of the pattern.

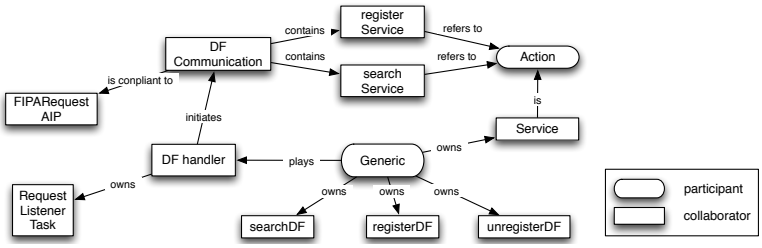

Fig. 3. The pattern semantic description for the *GenericAgent* pattern.

Externalization This unary operator has been conceived to modify the nature of a collaborator into a participant. The rational behind this operator is to delay the assignment of these responsibilities till the instantiation phase, exactly like for participants. The visible effects of this operation are i) to allow mixed composition (unification and conjunction) among participants and collaborators and ii) to change the standard behavior of a pattern, by delegating some aspects of its structure to elements of the system.

In order to discuss the usage of these operators in the static context, a composition of the *GenericAgent* pattern with the *Request* pattern is illustrated. The result of the composition is a new pattern, the *SequentialShareResource*, that solves the classical problem of providing a service to the remaining part of the society. A service is an action performed by an agent when requested by another agent. Service provisioning is subject to certain conditions (pre-condition, post-condition, grounding) to be verified. In particular this pattern provides the other agents with an access to a physical resource that the agent may manipulate; the specific characteristic of this access is that resource parameters are read/affected each time an access request occurs. A typical example of service that depends on a resource has been given in Section 2. The Company agent provides information on its business, but this depends on the user's data introduced by a GUI. The solution proposed by this pattern is to sequentially execute three activities: (i) the verification of the conditions under which the service may be provided, (ii)

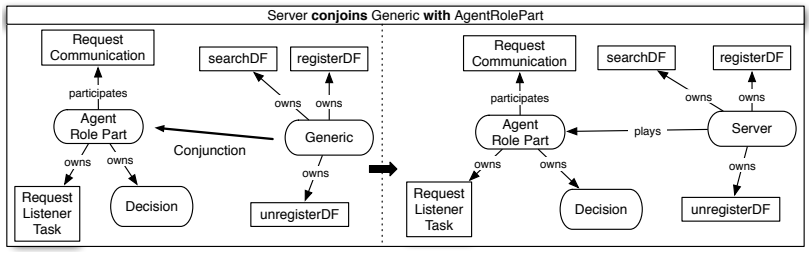

Fig. 4. An example of conjunction among participants for the *SequentialShareResource* pattern.

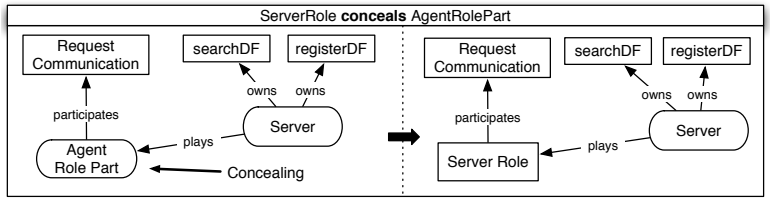

Fig. 5. An example of concealing of participant for the *SequentialShareResource* pattern.

the update of the status of the physical resource and (iii) the execution of the service-action.

The first component of this blend is the *GenericAgent* pattern, that has been conceived as a root for giving an agent the ability of interacting with the yellow pages service of the platform. The agent resulting from this pattern is able to register/unregister services to/from the yellow pages and to search them in.

The second component of the blend is the *Request* pattern, already discussed in Section 3.2. This pattern is used to implement a FIPA Request communication. By using this pattern an agent can request to another agent to perform some actions.

The composition process details (for the static part) are described in the following list:

- The first operation in this composition is a conjunction among the *Client* participant of the *GenericAgent* pattern and the *AgentPartRole* participant from the *Request* pattern. The rationale of this operation is to assign an agent, the *Server*, to play the *AgentPartRole*. Figure 4 shows the result of this operation: after the conjunction the *Server* participant plays the *Agent Role Part*.
- After the conjunction the designer can delegates a concrete agent of the system to be the *Server* of this new pattern (since it is a participant). This agent will play the *AgentPartRole* in the request communication. Therefore this role can be converted to a concrete element of this pattern: designer is

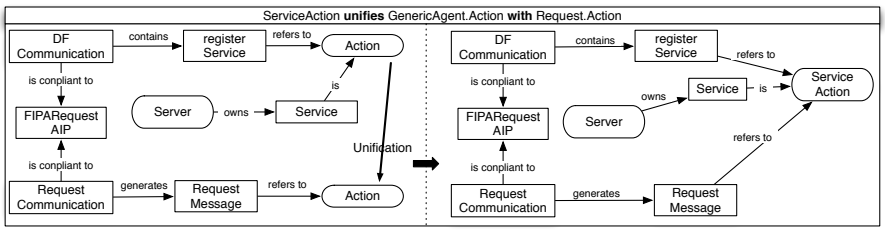

Fig. 6. An example of unification among participants for the *SequentialShareResource* pattern.

not required to specialize this element. For this reason the *AgentPartRole* is concealed by the *Server Role*, a new collaborator of the pattern. This operation is shown in Figure 5.

- The third operation is an unification between the *Action* from *GenericAgent* with the *Action* from the *Request* pattern. The meaning of this operation is to specify that the action registered to the yellow pages by the *Server* agent is the same action that the agent provide to the community. This unification is shown in Figure 6.
- Finally, we also introduced a new participant in the structure, the *UpdateResource* pattern, that is a task responsible for accessing the resource and update its state. Since the access to the resource is variable, depending by the nature of the resource, this element is defined as participant. Figure 7 shows the resulting structure of the new pattern after all these operations.

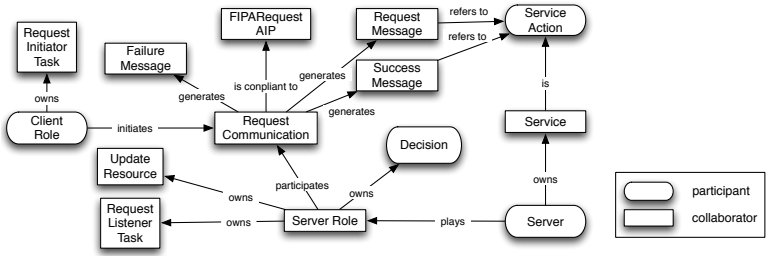

Fig. 7. The pattern semantic description for the *SequentialShareResource* pattern.

5 Discussion

This section discusses the pattern reuse process, and some results obtained by the application of the pattern composition technique to three different case studies are reported.
The reuse process encompasses four phases for introducing patterns in a system:

1. Meta-model definition or importing. This phase defines the domain where patterns can be employed. The Meta-Model defines a set of rules to be considered during next phases. Concepts and relationships in a PSD must be compliant to the Meta-Model. The definition of a meta-model is a complex activity but several reusable meta-models already exist.
2. Pattern structure and behaviour definition. In this phase the pattern is modelled by using a fine-grained description based on PDEs. Pattern modellers define the core semantics of patterns, in order to formalize their descriptions. This phase of the reuse process is supported by the Pattern Semantic Description diagram for representing the structure of the proposed solution.

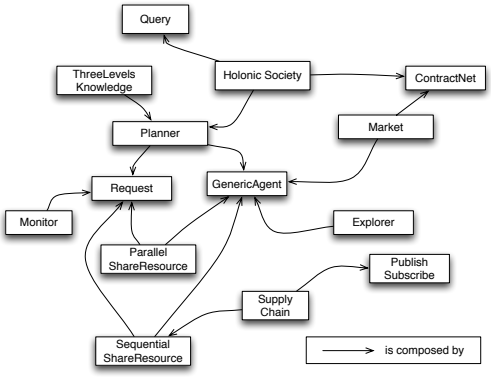

Fig. 8. Composition relationships among patterns in our repository.

3. Pattern composition. Patterns can be also defined by using pattern blending that allows for creating pattern synergies to solve more specific problems. This phase is supported by four operators: unification, conjunction, externalization and concealing.
4. Pattern instantiation. This is the final phase of the pattern lifecycle, where designers are involved in applying patterns to under development systems.

Discussion in this section is focussed on the pattern instantiation phase. The semantic approach for describing patterns, as introduced in Section 3, can be manually employed to introduce solutions in systems, or can be automatically interpreted by a tool that may generate the desired solution. Subsection 5.1 illustrates some statistics obtained by manual reuse of patterns from our repository. Finally, Subsection 5.2 introduces a tool, we are developing, that obtains benefits from the semantic approach.

5.1 Reusability of Pattern Blends

Our repository is by now composed of 22 patterns for agents. We have already formalized 14 of these patterns in order to evaluate our language. Patterns we have represented and composed with our approach are shown in Figure 8. Only 4 of these are atomic patterns: (i) *GenericAgent*, (ii) *Request*, (iii) *Query* and (iv) *ContractNet*. The remaining 10 patterns in Figure 8 are obtained by composition. This situation is represented by relationships "is composed by".

All patterns in this repository have been manually reused in three different case studies reported in Table 1. We have chosen these applications because they are from heterogeneous application domains: (i) *SBE Sim* is a software that simulates business evolution of Sicilian micro and mid-size companies, (ii) *CiceRobot*[6] is a robotic application able to give guided tours of the Agrigento's Regional Archaeological Museum and (iii) Iron Manufacturer is an application for supporting a B2B scenario involving an iron manufacturing company. The

number of different application contexts, proves the feasibility of our patterns and their compositions.

Table 1 reports that 8 patterns have been reused in at least two case studies. The most used patterns are the*GenericAgent*, the *Request*, and the *Query* that are generally easily to combine with other ones.

Table 1. Statistics of pattern usage in our case studies

	SDBE Sim	CiceRobot	Iron Manufacturer	TOT
ContractNet	1		1	**2**
Explorer			1	**1**
GenericAgent	4	5	5	**14**
Holonic Organization	1		1	**2**
Market	1			**1**
Monitor	1		1	**2**
Parallel ShareResource			1	**1**
Planner		3		**3**
Publish-Subscribe	1	1		**2**
Query	9	4	7	**20**
Request	9	7	21	**37**
Sequential ShareResource	3	1	2	**6**
Supply Chain		1		**1**
ThreeLevels Knowledge		1		**1**
TOT	**30**	**23**	**40**	

5.2 A Tool for Pattern Reuse

In previous section we proved that

- a framework is provided for representing declarative knowledge on design pattern, that is based on a semantic network,
- the syntax and semantics of the network are clearly defined

Under these conditions, pragmatics of the network are defined by several rules which are problem-independent. This allows us to formulate control algorithms to handle the described structures. In particular we are developing a tool with the following requirements: (i) complete control of the entire pattern reuse process, from meta-model definition to pattern definition, composition and instantiation; (ii) verification of pattern syntax and semantics; (iii) management of pattern applicability pre and post conditions; finally (iv) automatic generation of code and documentation for the multi-agent system generated by using pattern composition.

The latter requirement of this tool has been already completed, and discussed in [17]. This component of the tool uses an aspect oriented approach for weaving together different contribution to the final workproduct. These contributions are generated separately by *aspect weavers* who are specialized to realize code for a specific aspect of the system. Typical aspect weavers are: (i) the *architecture weaver* who is responsible to define the basic structure of an agent, its capabilities and the its basic life-cycle activities; (ii) the *role weaver* is responsible to manage complex agent activities, both internal processes than social behavior;

(iii) the *communication weaver* is responsible to give agents the ability to interact by using communications; (iv) the *protocol weaver* generates the structure for managing agent interaction protocols; and finally (v) the *ontology weaver* is responsible to generate agent knowledge structure. All these aspect weavers must collaborate in order to generate a unique source code.

The other requirements, we are facing for the development of the tool, are related to the pattern formalization problem. A semantic definition of patterns is easily representable in a formal language. We already defined this language, that we named POLaR (Pattern Ontology Language for Reuse) but it was not discussed in this paper for limit of space. Anyway the production of the parser for the POLaR language is under construction. A portion of the BNF code used to represent the syntax of this language is shown in the following:

```
<Pattern_Descr> ::= <Pattern_Header> "{" <Pattern_Definition> "}"
<Pattern_Definition> ::= { < Element_Clause > }
<Element_Clause> ::=  <Participant>
        | <Collaborator>
        | <Event>
        | <Action>
<Collaborator> ::= collaborator <Identifier> is <Element_Descr> ";"
<Element_Descr> ::= <Element>
        | <Operator>
<Operator> ::= <Unification>
        | <Conjunction>
        | <Promotion>
        | <Externalization>
```

6 Conclusion

This paper presented an innovative formalization technique for describing and composing design patterns for agents. The technique is based on a semantic analysis of the pattern solution and introduces a graphic notation to represent pattern's concepts and their relationships. The proposed formalization has been conceived for dealing with composition, presenting a set of operators to manage different pattern blending styles. The peculiarity of the approach is the fine grained level chosen for fronting with the composition, which makes it possible to combine pattern elements in the resulting composite pattern. We have applied our approach in three agent oriented applications, thus obtaining encouraging results in terms of reusability and expressiveness. In addition we introduce a tool we are developing in order to automatically compose and instantiate design patterns and to automatically generate implementing code and documentation.

References

1. C. Atkinson and T. Kiihne. The essence of multilevel metamodeling. *Uml 2001: The Unified Modeling Language: Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5, 2001: Proceedings*, 2001.
2. F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa2000 compliant agent development environment. In *Agents Fifth International Conference on Autonomous Agents (Agents 2001)*, Montreal, Canada, 2001.
3. C. Bernon, M. Cossentino, M. Gleizes, and P. Turci. A study of some multi-agent meta-models. *Agent-Oriented Software Engineering V: 5th International . . .* , Jan 2005.
4. J. Bosch. Specifying frameworks and design patterns as architectural fragments. In *Proceedings of TOOLS '98*, page 268, Washington, DC, USA, 1998. IEEE Computer Society.
5. F. J. Budinsky, M. A. Finnie, J. M. Vlissides, and P. S. Yu. Automatic code generation from design patterns. *IBM Syst. J.*, 35(2):151–171, 1996.
6. A. Chella, M. Liotta, and I. Macaluso. CiceRobot: a cognitive robot for interactive museum tours. *Industrial Robot: An International Journal*, 34(6):503–511, 2007.
7. M. Cossentino. From requirements to code with the PASSI methodology. In *Agent Oriented Methodologies*, chapter IV, pages 79–106. Idea Group Publishing, Hershey, PA, USA, June 2005.
8. M. Cossentino, S. Gaglio, L. Sabatucci, and V. Seidita. The passi and agile passi mas meta-models compared with a unifying proposal. In *In proc. of the CEEMAS'05 Conference*, pages 183–192, Budapest, Hungary, Sept. 2005.
9. M. Cossentino, L. Sabatucci, and A. Chella. Patterns reuse in the PASSI methodology. In *ESAW*, pages 294–310, 2003.
10. Digital Business Ecosystem. http://www.digital-ecosystem.org. onsite.
11. A. H. Eden, A. Yehudai, and J. Gil. Precise specification and automatic application of design patterns. In *Proceedings of ASE '97*, page 143, Washington, DC, USA, 1997. IEEE Computer Society.
12. Foundation for Intelligent Physical Agents. *FIPA Interaction Protocol Library Specification*, 2000.
13. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements od Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.
14. I. Hammouda and K. Koskimies. An approach for structural pattern composition. In *Proceedings of SC 2007*, Braga, Portugal, March 2007.
15. E. Kendall. Role modeling for agent system analysis, design, and implementation. In *IEEE Parallel and Distributed Technology*, volume 8, pages 34 – 41, IEEE, Apr-Jun 2000. IEEE Computer Society.
16. D. Riehle. Describing and composing patterns using role diagrams. In K.-U. Mätzel and H.-P. Frei, editors, *1996 Ubilab Conference*, pages 137–152, Zürich, Germany, June 1996.
17. L. Sabatucci and S. Gaglio. Separation of concerns and role implementation in the passi design process. In *5th International Conference on Industrial Informatics (INDIN 07)*, 2007.
18. S. Shapiro. *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, Inc. New York, NY, USA, 1992.
19. E. Sirin and J. Hendler. Semi-automatic Composition of Web Services using Semantic Descriptions. *WSMAI 2003*, 2003.