Separation and Modularization of Crosscutting Social Patterns in Detailed Architectural Design

Carla Silva[1], Jaelson Castro[1, 2, γ], João Araújo[3], Ana Moreira[3], Fernanda Alencar[4, *] and Ricardo Ramos[1]

[1] Centro de Informática, Universidade Federal de Pernambuco, 50732-970, Recife, Brazil
{ctlls, jbc, rar2}@cin.ufpe.br

[2] Istituto Trentino di Cultura, Istituto per la Ricerca Scientifica e Tecnologica, Trento-Povo, Italy
jaelson@itc.it

[3] Dept. Informática, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
{ja, amm}@di.fct.unl.pt

[4] Dept. Eletrônica e Sistemas, Univ. Federal de Pernambuco, 50732-970, Recife, Brazil
fmra@ufpe.br

Abstract. This paper outlines an aspect-oriented approach to support separation and modularization of crosscutting concerns in multi-agent systems. Aspects are used as abstractions to capture social patterns as concerns that crosscut software agents in multi-agent systems, whose separation and modularization are not taken into account in current agent-oriented software engineering. Social patterns are described using a template and UML-based diagrams to represent the pattern's structure and behaviour in an aspect-oriented context.

1 Introduction

Agent-oriented design patterns have been proposed to support the development of more reusable, flexible, understandable and maintainable multi-agent systems (MAS) [1]. In particular, the Tropos framework [2] has defined a set of design patterns, named social patterns [3], which includes *booking*, *subscription*, *monitor*, *broker*, *matchmaker*, *mediator* and *wrapper*. Traditional software development paradigms do not address the crosscutting nature of some design patterns which are scattered among different functional modules, making these tangled concerns [4]. Design patterns concerns are, therefore, *crosscutting* issues that can be better addressed by adopting aspect-oriented software development (AOSD) techniques [5].

In our previous work we proposed an approach to describe social patterns in the context of the Tropos project [1]. However, we did not take into account the benefits of separating pattern concerns from application concerns. Hence, in this paper we advocate the use of aspects as abstractions for cleanly separating the social patterns

[γ] Currently on leave of absence from UFPE.

[*] Currently on leave of absence at FCT / Universidade Nova de Lisboa, Portugal.

concerns from the core system functionality modules (i.e. agent roles) in MAS. We start by introducing an approach to describe social patterns (in the next section) and finish by summarising the main contribution of our work and pointing out open issues that need to be investigated further.

2 Describing Social Patterns

Tropos offers some social patterns [3] described using dimensions, such as social, intentional, structural, communicational and dynamic, which reflect particular aspects of MAS architectures, but do not provide a detailed description of each pattern. To address this issue, in [1] we present a template based in GoF's [6] to describe in more detail the social patterns. This template includes the description of pattern's *Intent, Applicability, Motivation Example* and *Participants*. In this work, we present a complement of that template (Table 1) which illustrates the description of the Matchmaker pattern.

Table 1. The Partial Template for a Pattern Description

Element	**Description**
Name	Matchmaker Pattern
Problem	How can clients locate unknown providers which offer a specific service?
Solution	The solution involves an intermediary agent (matchmaker) that receives requests from service providers to subscribe/unsubscribe its services into the yellow pages maintaned by it. An agent (client) may need a specific service provided by an unknown agent (provider). The Matchmaker also receives requests from client agents to locate some provider agent which offers a specific service. If there is some provider for the requested service, the Matchmaker informs that provider's ID to the client which, in its turn, can directly interact with it.

Furthermore, to achieve a more complete description of a pattern we must provide a description of its structure and behavior. In this paper, we propose an approach which uses abstractions and mechanisms of AOSD to describe social patterns in order to promote separation and modularization of social patterns' concerns. In particular, we adopt an extension of the aSideML notation [7].

The aSideML class diagram has been extended to support organizational architectural features [8] and agency features [9], as well as the notion of model roles [10]. In our approach we support the following modeling elements: *agent role classes*, *plans*, *actions, ports* and *attributes*. The agent role class, which is our base unit, is stereotyped by <<Role>>. Simmilarly, a plan the agent has to achieve a goal is stereotyped by <<Plan>>, while an action, which composes a plan, is stereotyped by <<Action>>. Organizational architectural ports are stereotyped by <<Port>> [8]. Thus, in our extended aSideML class diagram (Fig. 1), model element roles are added for two main reasons: to define generic model elements and to facilitate aspectual composition.

In Fig. 1, Matchmaking is an "*aspect*" that describes the Matchmaker design pattern [3] and includes three crosscutting interfaces (CI) [7] which are: Matchmaker, Client and Provider. Crosscutting features are listed in different compartments of CIs: *Additions* lists data and operations to be introduced in classes; *Refinements* lists crosscutting operations to be combined before, after or before/after class operations; *Redefinitions* lists crosscutting operations that override class operations. For example, the Client is a CI modularizing features that affect arbitrary base units in such a way that they become clients. The Client CI declares six additions: the *matchmaker* attribute, the *portZ* and *portA* ports, as well as *locateProvider*, *getMatchmaker* and *setMatchmaker* actions. It also declares one refinement: the *_requestService()* operation, which denotes a behaviour to be executed before the |*performAction* agent behaviour.

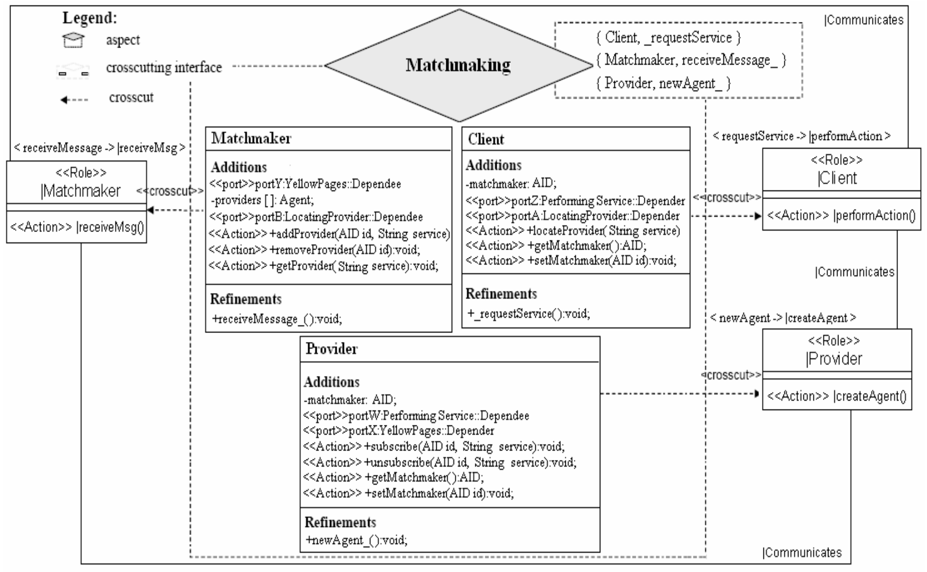
Legend:
aspect
crosscutting interface
crosscut
Matchmaking
{ Client, _requestService }
{ Matchmaker, receiveMessage_ }
{ Provider, newAgent_ }
|Communicates
< receiveMessage -> |receiveMsg >
<<Role>>
|Matchmaker
<<Action>> |receiveMsg()
<<crosscut>>
Matchmaker
Additions
<<port>>portY:YellowPages::Dependee
-providers [ ]: Agent;
<<port>>portB:LocatingProvider::Dependee
<<Action>> +addProvider(AID id, String service)
<<Action>> +removeProvider(AID id):void;
<<Action>> +getProvider(String service):void;
Refinements
+receiveMessage_():void;
Client
Additions
-matchmaker: AID;
<<port>>portZ:Performing Service::Depender
<<port>>portA:LocatingProvider::Depender
<<Action>> +locateProvider(String service)
<<Action>> +getMatchmaker():AID;
<<Action>> +setMatchmaker(AID id):void;
Refinements
+_requestService():void;
< requestService -> |performAction >
<<Role>>
|Client
<<crosscut>>
<<Action>> |performAction()
|Communicates
< newAgent -> |createAgent >
<<Role>>
|Provider
<<crosscut>>
<<Action>> |createAgent()
Provider
Additions
-matchmaker: AID;
<<port>>portW:Performing Service::Dependee
<<port>>portX:YellowPages::Depender
<<Action>> +subscribe(AID id, String service):void;
<<Action>> +unsubscribe(AID id, String service):void;
<<Action>> +getMatchmaker():AID;
<<Action>> +setMatchmaker(AID id):void;
Refinements
+newAgent_():void;
|Communicates

Fig. 1. Pattern Structure

The "crosscuts" relationships (stereotype <<crosscut>>) connect the Matchmaking aspect to |*Matchmaker* (binding *receiveMessage* to |*receiveMsg*), |*Client* (binding *requestService* to |*performAction*) and |*Provider* (binding *newAgent* to |*createAgent*). We also need to describe the composition of the aspect (i.e. the pattern's features) with the agent roles, which are going to be improved by the pattern's features.

An aspectual interaction [7] is a behavioural specification which incorporates a communication sequence exchanged by a set of instances of base units (agent roles) and an aspect instance in order to accomplish the implementation of a crosscutting behaviour (see Fig. 2). A small gray diamond symbol shaded in the base unit (agent role) instance lifeline denotes the weaving point [7].

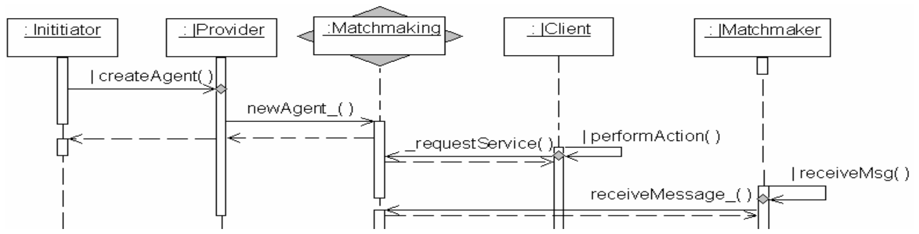

Fig.2. Aspectual Interaction of the Matchmaker Pattern

3 Conclusion

Our approach supports both the separation of social patterns concerns and its later composition with the agent roles present in the MAS. By doing so, we promote an easier way to apply the social patterns to the MAS design, since we only need to specialize the model roles present in the pattern description with specific agent roles of the MAS under development. However, further work is required to define a process for guiding the selection of proper social patterns to refine the MAS architecture. In fact, work is underway to describe a process that considers non-operationalized croscutting concerns as criteria to choose the patterns to be applied to a specific MAS.

Acknowledgements

This work was supported by several research grants (CNPq Proc. 304982/2002-4, CAPES Proc. BEX 1775/2005-7, Proc. BEX 3003/05-1, Proc. BEX 3014/05-3 & CAPES/ GRICES Proc. 129/05).

References

1. Silva, C., Castro, J., Tedesco, P., Silva, I.: Describing Agent-Oriented Design Patterns in Tropos. In Proceedings of the 19th Brazilian Symposium in Software Engineering. Uberlandia, Minas Gerais, Brazil (2005) 10 – 25
2. Giorgini, P., Kolp, M., Mylopoulos, J., Castro, J.: Tropos: A Requirements-Driven Methodology for Agent-Oriented Software. Book Chapter in Agent-Oriented Methodologies. ed.: Idea Group (2005) 20 – 45
3. Kolp, M., Do, T. T., Faulkner, S., Hoang, H. T. T.: Introspecting Agent Oriented Design Patterns. In S. K. Chang (Eds), Advances in Software Engineering and Knowledge Engineering, vol. III, World Publishing (2005)
4. Noda, N. and Kishi, T.: Implementing Design Patterns Using Advanced Separation of Concerns. In Proceedings of OOPSLA 2001, Workshop on Advanced Separation of Concerns in Object-Oriented Systems. Tampa Bay, FL (2001)
5. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J.: Aspect-Oriented Programming. In Proceedings of the 11th European Conference on Object-Oriented Programing. Springer-Verlag, Finland (1997) 220 – 242
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
7. Chavez, C.: A Model-Driven Approach to Aspect-Oriented Design. PhD Thesis, Computer Science Department, PUC-Rio (2004)
8. Castro, J., Silva, C., Mylopoulos, J.: Detailing Architectural Design in the Tropos Methodology. In Proceedings of the 15Th CAiSE. Klagenfurt/Velden, Austria (2003) 111 – 126
9. Silva, C., Castro, J., Alencar, F. and Ramos, R.: Extending UML to Support Both Agency and Organizational Architectural Features. In IX Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Softare (IDEAS'06) (to appear).
10. Kim, D. K.: A Metamodeling Approach to Specifying Patterns. PhD Thesis, Colorado State University (2004)