Separation and Modularization of Design Patterns-Specific Concerns in Detailed Architectural Design

Carla SilvaΨ, Jaelson CastroΨ, João AraújoΩ, Ana MoreiraΩ, Fernanda AlencarЖ and Ricardo RamosΨ

Ψ Centro de Informática, Universidade Federal de Pernambuco, 50732-970, Recife, Brazil {ctlls, jbc, rar2}@cin.ufpe.br
Ω Dept. Informática, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal {ja, amm}@di.fct.unl.pt
Ж Dept. Eletrônica e Sistemas, Univ. Federal de Pernambuco, 50732-970, Recife, Brazil fmra@ufpe.br

Abstract. In this paper we outline 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. Unfortunately, traditional software development paradigms do not use design patterns properly, causing design pattern concerns being scattered and, consequently, producing tangled concerns in application functional modules [4]. Design patterns concerns are, therefore, crosscutting issues that can be better addressed by adopting aspect-oriented software development 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 in account the benefits of separating patterns concerns from application concerns. Hence, in this paper we advocate the use of aspects as abstractions for cleanly separating the social patterns concerns from the core system functionality modules (i.e. agent roles) in MAS.

This paper is organised as follows: Section 2 introduces our approach to describe social patterns. Finally, Section 3 summarises our work and points out open issues.

1 Currently on leave of absence at ITC / IRST, Trento, Italy.
2 Currently on leave of absence at FCT / Universidade Nova de Lisboa, Portugal.
2 Describing Social Patterns

Currently, Tropos provides some social patterns [3] but it does not provide a detailed description of them. To address this issue by promoting a clearer separation of social patterns concerns, in this work, social patterns are described using a subset of GoF’s template [6] and aspect-oriented UML-based diagrams. In [1], we have presented a more complete template of the Matchmaker pattern [3] (Table 1), which includes the description of pattern’s Intent, Applicability, Motivation Example and Participants.

Table 1. The PartialTemplate for a Pattern Description

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Matchmaker Pattern</td>
</tr>
<tr>
<td>Problem</td>
<td>How can clients locate unknown providers which offer a specific service?</td>
</tr>
<tr>
<td>Solution</td>
<td>The solution involves an intermediary agent (matchmaker) that receives requests from service providers to subscribe/unsubscribe its services into the yellow pages maintained 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.</td>
</tr>
</tbody>
</table>

To support the specification of the social patterns separately from software agent roles specification, 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>>. Similarly, 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 with three crosscutting interfaces (CI) [7]. Matchmaker is a CI modularizing features that affect arbitrary base units in such a way that they become matchmakers. The Matchmaker CI declares four additions: the providers attribute; as well as addProvider, removerProvider and getProvider actions. It also declares one refinement: the receiveMessage(_()) operation, which denotes a behavior to be executed after the base unit behaviour (i.e., after receiveMsg). Client is a CI modularizing features that affect arbitrary base units in such a way that they become clients. The Client CI declares four additions: the matchmaker attribute, 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 base unit behaviour (i.e, before IperformAction). Finally, Provider is a CI modularizing features that affect arbitrary base units in such a way that they become providers. The Provider CI declares five additions: (i) the matchmaker attribute, as well as (ii) subscribe, unsubscribe, getMatchmaker and setMatchmaker actions. It also declares one refinement newAgent(_()), which denotes the behavior to be executed after the base unit behaviour (i.e., after IcreateAgent).
The “crosscuts” relationships (stereotype <<crosscut>>) connect the Matchmaking aspect to \textit{Matchmaker} (binding \texttt{receiveMessage} to \texttt{receiveMsg}), \textit{Client} (binding \texttt{requestService} to \texttt{performAction}) and \textit{Provider} (binding \texttt{newAgent} to \texttt{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.

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 (Fig. 2). A small gray diamond symbol shaded in the base unit (agent role) instance lifeline denotes the weaving point [7]. For example, as the aspectual interaction for the \texttt{<<Action>> newAgent} is shown in the beginning of the sequence diagram. It specifies that the crosscutting behaviour (newAgent) refines the behaviour of the agent role \texttt{Provider} after the execution of the base unit behaviour (\texttt{createAgent}).
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 crosscutting 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