An Aspect-Oriented Approach for Interaction Policies in Multi-Agent Systems

Chiara Di Francescomarino
dfmchiara@itc.it

Abstract

An agent is an autonomous entity whose behaviour is neither visible nor contrallable outside from its black-box. On the other hand, the MAS behaviour, determined by the agents' and artifacts' ones, should be shaped according to its security, organizational and coordination policies. The only way a MAS (organization or application) can control its internal interactions, in order to apply its policies, is through coordination artifacts. But, indeed, MAS policies crosscut the coordination artifacts, by being scattered across multiple unrelated artifacts and artifacts interfaces crosscut agents. So the idea is to use aspects and coordination artifacts for guiding MAS internal interactions in order to apply MAS policies. In this way both artifacts in MAS can be replaced without modifying MAS policies and policies can be easily changed without involving the general internal structure of the organization or application.

1 Introduction

Multi-agent systems are computational systems made up of agents and artifacts. An agent is an autonomous entity that can try to reach its goals by exploiting artifacts and communicating with other agents. An artifact, instead, is just a reactive entity that provides functions to agents. Therefore, the MAS behaviour is in general determined by interactions between agents and artifacts. On the other hand, MAS can also be seen as organizations or applications with their own rules and policies to apply to agents. To find a balance between these two positions is an hard problem to face in MAS. One of the most common approaches is the use of artifacts as mediating tools between agents and environment. In particular coordination artifacts, infrastructure abstractions meant to improve coordination entities automation, represent one of the best way to create a collaborative (according to MAS's policies) working environment[3]. Examples of coordination artifacts are tuple spaces[10] and tuple centres[6].

This paper proposes an aspect oriented approach for both coordination artifacts and agents. It can be noticed that, indeed, MAS policies crosscut the coordination artifacts by spanning multiple unrelated artifacts and artifacts' interfaces crosscut agents. With the adoption of Aspect-Oriented Programming, we can modularize the coordination parts of applications thus allowing both the artifacts' and MAS's reuse and the agents' partial unawareness to MAS environment.

After an overview about coordination artifacts in Multi-Agent Systems and Aspect-Oriented Programming in Section 2, Section 3 describes the Aspect-Oriented approach for both coordination artifacts and unaware agents. Finally Section 4 compares this work with other Aspect-Oriented approaches and Section 5 is a brief conclusion that proposes a future issue.

2 Overview

2.1 Coordination artifacts in Multi-Agent Systems

Multi-agent systems are computational systems in which more than one agent live together. Even if

the name can confuse, agents are not the only components of a MAS. Artifacts, any kind of resource or tool that agents can dynamically create, share, use and manipulate, also constitute important entities for MAS building. In particular, their importance is evident as portions of the MAS environment that can be designed and controlled to support MAS activities. In fact, as mediating tools, they have both an enabling (they allow agents to act in the environment) and a constraining (they limit the agents' perception and manipulation of the environment) functionality. In particular, in order to be used, they exhibit their interface to agents as the collection of available operations.

Agents are autonomous entities which cannot be controlled from outside: they have their own control and rules governing themselves. But agents are also social subsystems of a whole and, as such, they need to interact. Moreover, a MAS can also be interested to enforce some rules in the space of the system interactions to apply its organizational, security and coordination policies. In order to find a solution to the interaction issue, we have two possibilities: the endogenous and the exogenous one [15]. Whereas in the first case agents' computational, normative and communication aspects are encapsulated in agents themselves, in the second approach, the concept of coordination media, proposed in [1] is used. The paper defines a coordination model as a framework for expressing the interaction of agents (coordinables) through coordination media. Coordination media are abstractions which govern, by means of their behaviour and according to some coordination laws, the admissible and desirable interactions among coordinables. In the MAS case, the coordination media are just coordination artifacts: runtime persistent entities specifically designed and programmed with suitable coordination laws to improve automation of agents' interaction activities [2, 3].

Therefore, MAS are able to impose their own rules about:

- organizational and security concerns, in the case of single agent's interactions within a MAS, through individual artifacts;
- coordinative concerns, in the case of agents' mutual interactions, through social artifacts.

An example of individual artifact is provided by the Agent Coordination Contexts (ACC) [4, 5]. It is an infrastructural abstraction associated to each agent entering a MAS, modelling the presence or position of an agent within an organization in terms of its admissible actions. An example of social artifact, instead, is provided by tuple centres [6, 7], programmable tuple spaces for MAS coordination. The base tuple space is enhanced with a behaviour specification, expressed in terms of reactions to events.

2.2 Aspect-Oriented Programming

Aspect-Oriented Programming (AOP)[17] is a programming paradigm whose main goal is the separation of crosscutting concerns. A crosscutting module is a unit that crosscuts the modularization structure of the system. It cuts across others modules' boundaries: it is tangled in single unit and scattered among multiple modules of the system. The new modularization does not replace the old one; it just tries to provide an abstraction that is able to separate tangled concerns in order to facilitate the reuse and to make easier the modules' understanding.

An aspect in AOP is made up by three kinds of entities: join points, locations and advices. The first is the precise points in the program flow affected by crosscutting concerns; locations are predicates on the joinpoints' attributes that allow to select a set of join points and; advices are the actions to be taken when join points are reached.

AOP has recently been widely used in topics relative to patterns, authentication and authorization and business rules[16]. All these topics, in fact, have a meeting point: crosscutting concerns.

Patterns solution structure often involves crosscutting concerns, including one entity playing multiple roles, or many entities playing one role or an entity playing roles in multiple patterns instances. In particular, in the case of Java and AspectJ (the most popular aspect-oriented language), it has been showed that the modularity introduced by aspects greatly improves patterns [8].

Similarly, authentication and authorization concerns crosscut the core system logic, so they can be taken away from the core parts of the system and considered as aspects, thus improving reusability.

Finally, business rules also crosscut the core system functionality. Implementing the rules/policies as aspects make them separated from the main modularization structure of the system, thereby allowing modules' unawareness about policies in which they are involved[9].

3 MAS policies and crosscutting concerns

The importance of coordination artifacts for modelling coordination, organization and security in MAS is double:

- they allow a MAS to control the external behaviour of its agents without knowing anything about their internal computation;
- they separate the computational and coordination part of agents, by taking off a burden from agents' internal structure.

From one hand, the adoption of this coordination model allows to take out from agents the burden of MAS coordination, organization and security policies. From the other side, it strictly bounds policies to artifacts. Policies crosscut the ad-hoc artifacts rather than agents, thus preventing both the coordination artifacts's and MAS's reuse. For example, every time a policy embodied in more than one individual artifacts has to be changed, all the involved artifacts have to be modified.

Moreover, each agent, in a certain way, autonmously or by means of the programmer, has to catch the artifacts' interfaces/communication primitives in order to use them for its interactions. Again, every time agents have to be moved in different MAS, with the same shared resources and policies, but with a different communication protocol (communication primitives) or coordination artifacts' interface has to be modified, agents also have to be changed.

Decoupling the invocation protocol and, potentially, the basic functionality of artifacts from the real implementation of the MAS policies could avoid the aforementioned problems.

Driven by the above considerations, the solution proposed in this paper suggests an aspect-oriented approach for both coordination artifacts and agents.

In the first case, the scenario is similar to those involving authentication and authorization modules or business rules: a main core functionality and some policies (individual or social) to apply. Therefore, a base policy and the protocol of communication (the artifact's interface) are encapsulated inside the artifact, while all the other policies are put inside aspects. From the agents' point of view, instead, we need to adapt agents' communication primitives. Again it's possbile to catch sight of an adapter pattern: we just use aspects to fit communication primitives to the new protocols of the agents when they are moved into different MAS or when an artifact's interface is changed, so that they can remain more independent and heterogeneous as possible.

In order to better understand these concepts, we can consider, for example, a set of autonomous printers sharing requests from different computers. A shared queue, polled by printers asking for new documents by means of a primitive (for instance `getDocument`), can be chosen as coordination artifact. Every time a printer is free, or its local queue is not so full, or some event occurs according to one of its internal policy, it just uses the `getDocument` primitive to interact with the MAS. Let's suppose that at design time, the only policy the system wants to apply is the base coordination one: avoiding that more than one printer take the same document.
During the system's life could happen that:

- a more specific policy becomes necessary in the MAS: some documents can be printed only by some privileged agents; for example, colourful documents have to be printed only by colour printers. An aspect can be applied to the `getDocument` invocation, intercepting only requests coming from the color printers.
- an agent is moved to a different MAS with the same structure but with different artifacts

or an artifact is changed with a new one. In both cases it's likely that the old primitive `getDocument` is replaced by a new one, for example `documentRequest`. In this case, a new kind of aspect can be applied to agents in order to adapt the old `getDocument` primitive to the new one.

In summary, the use of coordination artifacts and aspects is a quite straightforward approach for applying coordination, organization and security policies in MAS.

4 Related works

Several works in between Agent-Oriented and Aspect-Oriented Programming have been presented.

Some aspect-oriented approaches for crosscutting concerns relative to agents and their internal modeling elements (actions and goals) have been proposed by [11, 12]. Examples of these concerns are learning, mobility, error handling and security, so that, differently from this work, aspects are used above all as a way for modularizing agents' specific features.

Another kind of perspective has been suggested in [13] and [14]. They both focuse on the advantages provided by an aspect oriented approach in the coordination within a MAS. In the first work the authors, though trying to modularize the coordination actions, thus separating computation and coordination concerns, insist on the importance of the naturality of endogenous models. Unfortunately, these models embodied in agents, impose on them the awareness of coordination policies and makes easier for them to avoid MAS rules. The second and the current work, instead, propose to use aspects on exogenous models but, whereas [14] just focuses on middleware infrastructure this paper extends the particular middleware approach to a more general MAS environment, combining it with some more specific concepts of coordination in MAS.

5 Conclusions

In Multi-Agent Systems, artifacts, agents and MAS policies should be as much modularized and independent as possible, even if their interactions are really necessary. This paper has suggested a way to reach this aim by means of coordination artifacts and aspects. The MAS policies regulating internal MAS interactions have not only been taken away from agents but also from artifacts and embodied into artifacts' aspects.

A current limitation for this approach could be related to the lack of aspects' dynamism at runtime in some languages' implementation, but it's just an implementation issue.

Taking into account that an agent can potentially change artifacts to adapt their functions to its own goals, a future direction for these ideas could be an evaluation of possible advantages and/or disadvantages deriving from agents' ability to modify aspects at runtime.

References

[1] Ciancarini P.(1996) **Coordination models and languages as software integrators.** *ACM Computer Surveys, 28(2):300-302.*

[2] Morini S., Ricci A., Viroli M. **Integrating a MAS Coordination Infrastructure with Web Services**. *In Zakcria Maamar, Cavedon Lawrence, David Martin, Boualem Benatallah, Katia Sycara, and Tim Finin, editors, Workshop on "Web Services and Agent-based Engineering"(WSABE 2004), AAMAS 2004, New York, USA, 19 July 2004. Proceedings.*

[3] Andrea Omicini, Alessandro Ricci, Mirko Viroli, Cristiano Castelfranchi, Luca Tummolini. **Coordination Artifacts: Environment-Based Coordination for Intelligent Agents** *aamas, pp. 286-293, Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS'04), 2004.*

[4] Andrea Omicini, Alessandro Ricci, Mirko Viroli. **Agent coordination contexts for the formal**

specification and enactment of coordination and security policies *Science of Computer Programming 63(1):88 - 107. Special issue on security issues in coordination models, languages, and systems.*

[5] Alessandro Ricci, Mirko Viroli, Andrea Omicini. **Agent coordination contexts in a MAS coordination infrastructure** *Applied Artificial Intelligence 20 (2-4):179-202.*

[6] Andrea Omicini, Enrico Denti. **From tuple spaces to tuple centres** *Science of Computer Programming 41(3):277-294.*

[7] Andrea Omicini. **Formal ReSpecT in th A&A perspective** *In Canal, C. and Viroli, M., editors, 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA06):93115, CONCUR 2006, Bonn, Germany. University of Malaga, Spain. Proceedings.*

[8] Jan Hannemann, Gregor Kiczales. **Design pattern implementation in Java and AspectJ.** *Conference on Object Oriented Programming Systems Languages and Applications archive: 161 - 173. Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.*

[9] M. A. Cibràn, M. D'Hondt, and V. Jonckers. **Mapping high-level business rules to and through aspects.** *In 2me Journe Francophone sur le Dveloppement de Logiciels Par Aspects (JFDLPA 2005), Lille, France, 2005.*

[10] D.Gelernter, N. Carriero. **Coordination languages and their significance** *Communications of th ACM, 35(2):97-107, February 1992.*

[11] Alessandro Garcia, Christina Chavez, Ricardo Choren. **Aspects in Agent-Oriented Software Engineering: Lessons Learned.** *Proc. 6th Workshop on Agent-Oriented on Software Engineering, AAMAS 2005, July 2005.*

[12] Alessandro Garcia, Christina Chavez, Ricardo Choren. **Enhancing agent-oriented models with aspects.** *Source International Conference on Autonomous Agents archive.Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems table of contents (2006):1332- 1334*

[13] Sirio Capizzi, Riccardo Solmi, Gianluigi Zavattaro. **From Endogenous to Exogenous Coordination Using Aspect-Oriented Programming.** *COORDINATION 2004: 105-118*

[14] Amor, M., Fuentes, L., Pinto, M. **Coordination as an aspect in middleware infrastructures.** *5th Int. Workshop on Aspects, Components and Patterns for Infrastructute Software, 5th Int. Conference on Aspect-Oriented Software Development, Bonn (Germany) (2006)*

[15] F. Arbab. **What Do You Mean, Coordination?** *In the March '98 Issue of the Bulletin of the Dutch Association for Theoretical Computer Science (NVTI).*

[16] Ramnivas Laddad. **AspectJ in Action. Practical Aspect-Oriented Programming**

[17] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes,Jean-Marc Loingtier, John Irwin. **Aspect-Oriented Programming** *In Proc. of European Conference on Object-Oriented Programming(ECOOP'97), 1241, Lecture Notes in Computer Science:220-242. Springer-Verlag, Berlin,1997.*