

Coordination as an Aspect in Middleware Infrastructures

Mercedes Amor, Lidia Fuentes, Mónica Pinto

Departamento de Lenguajes y Ciencias de la Computación,
University of Málaga, 29071 SPAIN
{pinilla,lff,pinto}@lcc.uma.es

ABSTRACT

In this paper we discuss the shortcomings derived from having coordination and computation tangled in the same software entities and from having coordination protocols scattered through the several components participating in an interaction. We show a possible solution to this problem by using aspect-oriented techniques to separate coordination as an independent entity in middleware infrastructures.

Keywords

Coordination, AOSD, MultiTEL, CAM/DAOP, MALACA

INTRODUCTION

Developing distributed applications can be seen as the combination of two distinct activities: a computing part that comprises programming a number of entities (objects, components, agents, web services) involved in manipulating data, and a coordination part responsible for the communication and cooperation between these entities.

Given a set of possibly heterogeneous computational entities, the purpose of the coordination paradigm is to provide mechanisms and primitives to specify the synchronized interaction for putting all these components together, and make them interact in such a way that form a single application. This paradigm provides a clean separation between individual software components and their interactions within their overall software organization. This separation, together with the high-level abstractions offered by coordination models, allows viewing computational entities as black boxes, promoting their reuse in different applications.

Most standard middleware infrastructures directly support some kind of coordination mechanisms based on traditional coordination models such as the publish-and-subscribe, the tuple-based and the blackboard models [1]. However, such mechanisms are not sufficient for managing complex interaction protocols, typically comprising several lines of code. For instance, in the publish and subscribe coordination model, the support for expressing patterns about distributed events and algorithms for detecting correlations among these events are still largely unexplored [2]. By failing to support separate abstractions for representing such complex protocols, most standard middleware infrastructures force programmers to distribute and embed them inside the interacting components..

In this paper we propose a possible solution to this problem encapsulating the coordination among a set of software entities in an aspect. The coordination aspect is defined as an entity that encapsulates the interaction pattern or coordination protocol that governs the communication and interchange of information among two or more software entities. A coordination protocol can be defined as the list of messages and/or events that a software entity is able to send and receive, along with a set of coordination rules that state the order in which messages and events must be interchanged by the participant entities of the interaction. A coordination protocol can be specified by a STD (State Transition Diagram), or any other similar formalism. Other works for enhancing traditional coordination model by embedding the description of coordination information in a third-party entity are starting to appear [2,3,4,5].

After this introduction the paper is organized as follows. Next section briefly describes the state of the art of coordination in component-based and multi-agent domains. We continue with our motivation to separate coordination as an aspect and with the solutions we have adopted to separate coordination in MultiTEL [6], CAM/DAOP [7] and MALACA [8]. Some initial discussion about how coordination might need to interact with other aspects in the middleware infrastructure is presented in the following section. Finally, we present the conclusions of the paper.

STATE OF THE ART

Coordination plays a central role in technologies used to develop distributed applications such as component technologies, web services and agent technologies.

With respect to component-based technologies, they usually provide coordination mechanisms based on the publish-and-subscribe coordination model. Examples of these are CORBA, CCM/CORBA and J2EE. The main advantage of this mechanism is that communication is anonymous. Therefore, suppliers and consumers of events are decoupled among them due to the use of a third-party object that acts as intermediary between them. However, by using this mechanism software developers do not achieve a complete separation between computation and coordination for two main reasons: (1) the coordination model is spread throughout the objects acting as suppliers, consumers and event channels. That is, suppliers do not simply throw events that are intercepted by the middleware infrastructure and managed by the coordination entity. Instead, suppliers and consumers need to create and/or

localize the event channels, including code related to the particular coordination model they follow as part of the code of the objects that implement consumers and suppliers; (2) the event channel does not usually provide support to encapsulate complex interaction protocols. In consequence, this service is mainly suitable for applications that just need to be aware of “change notifications”. Basically, a supplier produces an event and all the consumers registered in the event channel will receive the notification that such event was produced. The third-party objects in the Jini Distributed Event Service are an exception to the latter shortcoming. The reason is that Jini does not impose anything about the implementation of third party objects. They just need to implement the interfaces to register or to notify events but do not need to extend a particular object of a specified type. Therefore, Jini provides an event based programming model plus the possibility of encapsulating a coordination protocol in the third party objects. As commented before, other proposals that extends the publish and subscribe model to address the coordination of activities between decoupled components are [2,3,4]. Finally, ObjectPlace [5] is another proposal extended the traditional tuple space coordination model with support for role-based coordination between individual components.

Coordination also plays an important role in web services that use a loosely coupled integration model to allow flexible integration of heterogeneous systems in a variety of domains including business-to-consumer, business-to-business and enterprise application integration. However, the use of standards such as SOAP, WSDL and UDDI and interaction following a loosely coupled is not enough for such integration. Systems integration requires an additional layer able to describe and perform web services composition and orchestration. The latter term is the description of interactions and message flow between services in the context of a business process. This concept is not new; in the past it has been called workflow. Until now, different XML-based languages have been introduced to cover web services orchestration where two of the more well-known are the Web Services Choreography Description Language (WS-CDL) [9] and the Web services Business Process Execution Language (WS-BPEL) [10]. WS-CDL describes the set of rules that explains how different web services may act together, and in what sequence, giving a flexible and integral view of the process. WS-BPEL enables the composition of existing web services into a more complete web service. The composition is defined in terms of a workflow process consisting of a set of activities and the composition itself is exposed as a web service.

Nowadays, Multi-Agent Systems (MAS) are an effective paradigm for the design and implementation of complex software applications. Agent-based applications are understood as a system consisting of autonomous agents whose interactions are coordinated through an

organizational structure [11]. The necessity of coordinating agents in MAS relies on the common idea that an agent should be able to engage in, possibly, complex communications with other agents in order to exchange information or to ask their collaboration in pursuing a goal. Currently two coordination models are the most representative and effectively used to realize agents coordination mechanisms: Interaction protocols and Shared Dataspaces. A large portion of the agent community, which includes the standardizing organism FIPA, considers coordinating agents using interaction protocols, i.e. predetermined patterns of interaction. This approach is based on considering coordination essentially as a problem of communication. Thus the Agent Communication Language (ACL) plays a fundamental role, providing a means to achieve a higher-level of interoperability between agents. We can find in the agent models supporting this coordination model that the functional part of the code of an agent is interleaved with code that is there only to support coordination. This dependency derived from the intermingled code makes difficult the reuse of the agent functionality in other applications and platforms.

MOTIVATION

The conclusion is that coordination models provide support for loosely coupled interaction, mainly focusing on decoupling the source and the target of the communication. However, they do not achieve a complete separation among computation and coordination, since they do not provide support for the description of complex interaction protocols. As stated before, a complex interaction protocol specifies not only the information interchanged by coordinated entities but also the coordination rules.

To illustrate this shortcoming we will focus on an auction system and, concretely, in the simplified interaction between the seller and the buyer. In this example, Figure 1 shows the entities participating in an auction, while Figure 2 describes a scenario of the interaction among them. However, when the system is finally implemented using, for instance, a component based approach, the coordination information shown in Figure 2 is usually lost. That is, the coordination protocol among buyers and sellers is directly hard coded as part of the implementations of the *Seller* and the *Buyer* components. As a consequence, there may be situations in which a change in the interaction protocol requires changing the component implementations.

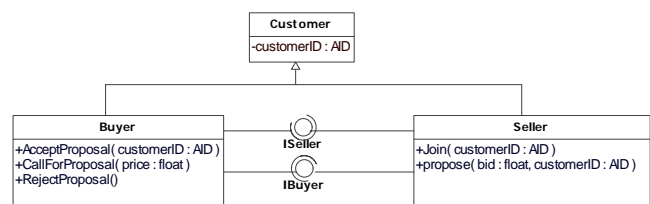


Figure 1. Part of the design of an Auction System.

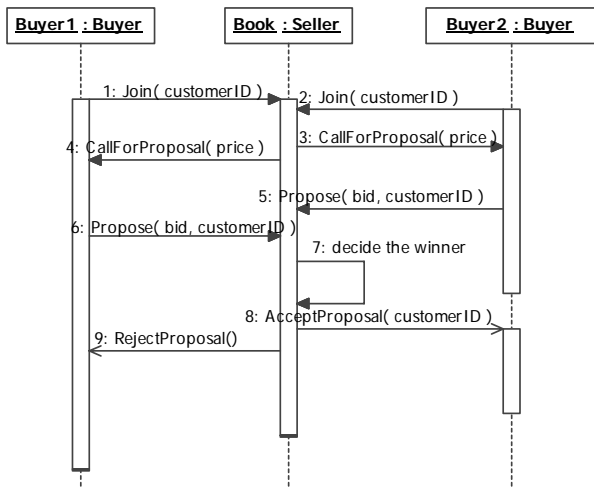


Figure 2. Buyer – Seller Interaction Protocol.

In Figure 2 when an auction finalizes, the *Seller* component decides the winner bid (step 7) and notifies to just one of the *Buyer* components that his/her bid won, using the *AcceptProposal(customerID)* operation (step 8). The other buyers are notified that they did not win invoking the *RejectProposal(customerID)* operation (step 9).

Let us suppose now that all the buyers have to be notified about which is the winner. This is a change in the coordination among the components and, with the current approach, implies to modify the implementation of the *Seller* component. Thus, the seller has now to invoke the *AcceptProposal(customerID)* operation over all the buyers.

This shortcoming may be overcome if the seller and the buyers use, for instance, the publish-and-subscribe interaction style. In this case, components register in a common communication channel. The use of a third-party object facilitates the 1-to-many communication. This means that the seller just publishes an anonymous *AcceptProposal(customerID)* event and all the buyers subscribed to that event will receive the notification.

Another change may be to make the auction *public* (one in which all bids are shared among the participants). We desire to (re)use the *Buyer* and *Seller* components in Figure 1 and Figure 2. In this case the *Seller* component should have to notify the *Buyer* components that a new bid is proposed, and this requires modifying its implementation. Concretely, after receiving the *Propose(bid, customerID)* operation invoked by *customerID* buyer, the Seller has to notify the bid to the rest of buyers using the *CallForProposal(bid)* operation. Once again this is part of the interaction protocol codified crosscutting the functionality of the Seller component. In this case, even using the publish-subscribe interaction style it is needed to change the implementation of the seller component. The reason is that this change modifies the interaction protocol, which is tangled with computation and scattered

throughout the participant components. The solution to this problem comes across not including the coordination rules as part of the components. This information should be included in the third-party object of coordination models. However, traditional coordination models do not provide support for this.

The extension of a coordination model with the necessary support to encapsulate complex interaction protocols may avoid to hard code this information as part of the coordinated entities. However, most of the realizations of a coordination model for particular technologies introduce some non desirable dependencies among the coordinated entities. For instance, the event service in CORBA follows a publish-and-subscribe coordination model. However, its implementation uses RPC (Remote Procedure Call). This implies that the channel is a CORBA object and it has to be located by the components before using it.

This problem can be alleviated if the responsibility of locating the adequate channels falls to the middleware infrastructure. This approach is followed in the CCM model of CORBA, where components just throw events and it is the container the responsible of managing channels.

The next section will show how AOSD mechanisms provide a natural solution to solve the limitations of the coordination models discussed in this section.

SEPARATING COORDINATION IN MIDDLEWARE INFRASTRUCTURES

In this section we describe our experience separating coordination as an aspect in three different middleware infrastructures we have developed: MultiTEL, CAM/DAOP and MALACA. Readers can obtain detailed information of these middleware infrastructures in [2,3,4].

A common feature of these infrastructures is that coordination is separated from computation and moved into a coordination aspect (Figure 3, label 1). Other important feature is that the weaving information is not part of the definition of components or aspects. Instead, it is explicitly described and allocated in the middleware infrastructure (Figure 3, label 2), which is the responsible of weaving components and the coordination aspect at runtime. This is very useful for implementing applications in open systems, in which components evolve over time. Thus, components do not need to choose or localize the proper coordination aspect, since this is the middleware infrastructure responsibility. This characteristic increases the reusability of components in different contexts. As shown in Figure 3, both load-time weaving and runtime weaving mechanisms may be provided, with divergent outcomes on flexibility and performance.

Other advantage of separating the coordination aspect is that it is easier to control the different states of a complex interaction (Figure 3, label 3).

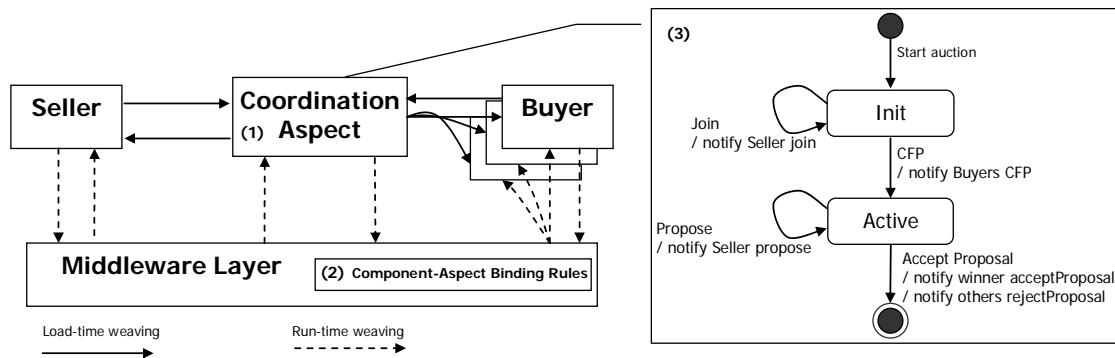


Figure 3. Separation of coordination in Auction System

Going back to our example, let us suppose that it is needed to control that once the auction begins (the *Seller* component sends the first *CallForProposal()* operation), no new buyers can join the auction. This kind of constraint can be easily satisfied with a coordination aspect that encapsulates a protocol with different states.

Coordination in MultiTEL

MultiTEL (Multimedia TELEcommunication services) is a complete development solution that applies component and framework technologies to deploy multimedia and collaborative applications. Essentially, MultiTEL provides separation of concerns between computation and coordination by encapsulating computation into Components and locating coordination into Connectors. Therefore, the architecture of a MultiTEL application can be seen as a collection of components that interact through connectors.

In MultiTEL, components are passive entities that react to external stimuli by propagating events. Connectors are abstractions of complex coordination patterns that implement coordination protocols. By handling the events propagated by external components a connector can coordinate the execution of the components participating in the same interaction or task. The interaction protocol encapsulated in connectors is specified as a STD and it is implemented using the State Design Pattern in Java.

Another important characteristic of MultiTEL is that the weaving information to connect components and connectors is specified separately from components and connectors and stored in the middleware infrastructure. Weaving information is specified in a Service Description Language (LDS). Afterwards this information is provided to the distributed middleware infrastructure that will use it at runtime to weave components and connectors dynamically. This increases both components and connectors reusability and evolution and it provides a powerful mechanism for late binding among them.

Coordination in CAM/DAOP

CAM/DAOP (Component-Aspect Model/Dynamic Aspect-Oriented Platform) is a model and a middleware infrastructure that combines the benefits of both

component-based and aspect-oriented disciplines in the development of complex distributed applications.

CAM/DAOP is an extension of MultiTEL and shares with it features such as the explicit description of the application architecture and the use of this information by a middleware infrastructure to perform the late binding between components and aspects. Therefore, CAM/DAOP also shares with MultiTEL the advantages introduced by these features. There are also some differences. The main difference is that the main entities of CAM/DAOP are Components and Aspects. This makes possible to separate any extra-functional properties that result to be tangled or scattered throughout the components in an application. This has the advantage that we increase even more the reusability and evolution of both Components and Aspects.

Another difference is that in CAM/DAOP Components can interact using both a message-based and an event-based programming models. The handling of events is resolved at runtime by a coordination aspect that plays the same role than the Connector in MultiTEL. Additionally, the coordination aspect can intercept messages among components. In this case, the coordination aspect introduces an additional advantage since it can be also used as an *adapter* for resolving component incompatibilities and integration problems.

A final advantage is how the coordination aspect is implemented. Concretely, the interaction protocol encapsulated in the aspect is XML-based specified conforming to an XML Schema. Thus, the implementation of the aspect consists on building an interpreter for these protocols, instead of providing different implementations of the aspect for each interaction protocol. Using this approach the application can evolve at runtime only by changing the XML description of the coordination protocol.

Coordination in MALACA

MALACA provides separation of concerns in the scope of MAS and more concretely in the context of agent internal architecture. MALACA is a component-based and aspect-oriented agent model. Inside the agent architecture, agent functionality is provided by components and it is separated from its communication, which is modelled by aspects.

Three main aspects are recognized as mandatory since they cover the basic agent communication functions in a FIPA-platform: Distribution, Coordination through an interaction protocol and ACL representation. Aspects are weaved at runtime by a third party component (internal weaving infrastructure) of the architecture following a set of aspect composition rules. Weaving is performed when the agent communicates, that is when it sends and receives messages.

As said before, agents of MAS mostly use interaction protocols to coordinate. This way they achieve a separation between how agent coordinates and how it performs its tasks. However, it has an impact on the internal development of agents, since most object-oriented infrastructures tangle in the same architectural unit computation and coordination, this time inside the agent.

In the MALACA architecture, each ongoing conversation is coordinated by a coordination aspect. To perform protocol-compliance the coordination aspect uses a XML-based description of the interaction protocol, which is given to the coordination aspect during instantiation. Currently MALACA provides a common implementation of the coordination aspect that is able to parse and use the XML description of an interaction protocol that follows a concrete XML Schema. This feature avoids implementing a coordination aspect for each interaction protocol the agent has to support. Each role participating in the interaction is described as an STD, in terms of states and transitions. The transition from a state to another carries out the execution of the agent functionality, which is also included as part of the XML-based STD description.

INTERACTION WITH OTHER ASPECTS

The most relevant relationship of the coordination aspect with other aspects at the middleware infrastructure level is with the distribution aspect. Let us consider that a distribution aspect encapsulates both localization and communication sub-concerns [12]. Therefore, when an anonymous interaction occurs – i.e. a component throws an event: (1) the coordination aspect is the responsible to manage the events thrown by components to determine which component(s) is/are the recipient of the information and forwards a message to it/them, and (2) once the targets are decided, the interaction is considered as an explicit component to component interaction and thus requires interacting with the localization and the communication sub-concerns of the distribution aspect. The relationship between the coordination and distribution aspects at the infrastructure level is also present in the agent domain. In the scope of the agent internal infrastructure, the distribution aspect encapsulates the access to a concrete agent platform through its done message delivery.

CONCLUSIONS AND FUTURE WORK

In this paper we have shown that it is possible to use aspect-oriented techniques to improve the separation between computation and coordination in middleware infrastructures. The main benefits of the proposed solution are: (i) facilitate the reusability of components in different contexts; (ii) well-designed support for the definition of complex interaction protocols; (iii) increase the maintainability and adaptability of final applications.

Comparison with other approaches enhancing traditional coordination models to provide support for the definition and management of the orchestration among a set of interacting components is needed and it is planned as part of our future work. Performance evaluation of the presented approach will also be made.

ACKNOWLEDGMENTS

This work has been funded in part by the Spanish CICYT project with number TIN2005-09405-C02-01 and in part by the AOSDEurope NoE with number IST-2-004349.

REFERENCES

1. Papadopoulos, G. A., et al. "Coordination models and languages", *Advances in Computers* 46 pp. 329-400, 1998.
2. Li, G et al." Composite Subscriptions in Content-Based Publish/Subscribe Systems", *Middleware 2005, LNCS 3790*, pp. 249–269, 2005.
3. Pietzuch, P. R. et al. "Composite event detection as a generic middleware extension", *IEEE Network Magazine, Special Issue on Middleware Technologies for Future Communication Networks*, January/February 2004.
4. Ulbrich, A. et al. "Programming abstractions for contentbased publish/subscribe in object-oriented languages". In *CoopIS/DOA/ODBASE (2)*, pages 1538–1557, 2004.
5. Schelfhout, K. et al. "Middleware for protocol-based coordination in dynamic networks", in 'MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing', ACM Press, New York, NY, USA, pp. 1–8, 2005.
6. Fuentes, L., et al. "Coordinating distributed components on the web: An integrated development environment", *Software Practice & Experience*, 31(3):209-233, 2001.
7. Pinto, M., et al. A dynamic component and aspect oriented platform, *The Computer Journal*, 48(4):401-421, 2005.
8. Amor, M., et al., Training compositional agents in negotiation protocols using ontologies, *Journal of Integrated Computer-Aided Engineering*, 11(2):179-194, 2004.
9. W3C. "Web Services Choreography Description Language Version 1.0", <http://www.w3.org/TR/ws-cdl-10/>
10. OASIS Open, Inc. "Web Services Business Process Execution Language Version 2.0.", <http://www.oasis-open.org/>
11. Omicini, A. et al, eds. *Coordination of Internet Agents. Models, Technologies, and Applications*. Springer, 2001
12. Loughran, N., et al. *Requirements and Definition of AO Middleware Reference Architecture*, <http://www.aosd-europe.net/documents/d21.pdf>, 2005