A Decoupled Architecture for Action-Oriented Coordination and Awareness Management in CSCL/W Frameworks

Pablo Orozco[1], Juan I. Asensio[1], Pedro García[2], Yannis A. Dimitriadis[1], Carles Pairot[2]

[1] School of Telecommunications Engineering, University of Valladolid
Camino Viejo del Cementerio s/n, 47011 Valladolid, Spain
{paboro@ulises, juaase@, yannis@}.tel.uva.es
http://ulises.tel.uva.es

[2] Dep. of Computer Engineering and Mathematics, Rovira i Virgili University
Avinguda dels Països Catalans 26, 43007 Tarragona, Spain
{pgarcia, cpairot}@etse.urv.es
http://www.etse.urv.es

Abstract. This paper introduces AORTA, a software architecture that provides object-level coordination and shared workspace awareness support to synchronous and distributed collaborative applications. AORTA is motivated by the need to enhance current coordination and awareness capabilities of existing software component frameworks for the domains of CSCL (*Computer-Supported Collaborative Learning*) and CSCW (*Computer-Supported Cooperative Work*). AORTA is characterized by the use of *actions* as its key abstraction instead of low-level events, the support for mutual influence between coordination and awareness, the use of coordination and awareness policies for supporting complex and dynamic collaboration scenarios, and the use of software design patterns in order to decouple coordination and awareness from the development of other aspects of CSCL/W applications. The paper motivates, justifies, and describes the main functional features of AORTA as well as its proposed software architecture. The paper also introduces a prototype of AORTA that adds coordination and awareness support to an existing groupware framework called ANTS. Finally it describes a CSCL application developed on top of both AORTA and ANTS that has been used to validate some of the presented contributions: application development is decoupled from coordination/awareness aspects, application development is facilitated by the use of action-orientation, and application coordination/awareness behavior can be configured and changed without modifying the application itself.

1 Introduction

Computer-Supported Collaborative Learning (CSCL) is a research paradigm within the field of educational software that underlines the key role that social interactions play in the process of learning [13]. CSCL applications promote and give support to *collaborative* ways of learning. From a technological point of view, CSCL has its roots into the field of *Computer-Supported Cooperative Work* (CSCW) [5].

The success of CSCL applications largely depends on their capability to be *reused* and *adapted* to different and dynamic collaborative learning scenarios. A change in the learning objectives, the involved participants, the group structure, etc. of a learning scenario might make a previously successful CSCL application unsuitable for the new situation. For example, consider a CSCL application for the collaborative edition of an electronic magazine in which each student is in charge of a particular section. If there is just a change in the way participants interact (e.g. the teacher desires that the students review the whole magazine and make modifications by turns), and the collaborative editor cannot *adapt* to that change, it could not be used anymore. On the other hand, the effort (and therefore the cost) devoted to the development of the collaborative editor might not be justified if it can only be used in that particular learning scenario.

The potential solution to this *reuse* and *adaptation* problem is a traditionally claimed benefit of the Component-Based Software Engineering (CBSE) [20]: first of all, it is easier to reuse a software component that supports a common functionality of several applications than reusing complete applications across different scenarios; and secondly, adaptation can be achieved by replacing one or several application components in order to change just that part of the application that did not fit the characteristics of a new scenario.

CBSE concepts and principles have been successfully applied to the development of non-CSCL educational software (see e.g. [18]). These experiences (and other related to the use of CBSE in other domains) stress the importance of a proper identification and dimensioning of software components in order to guarantee reuse and adaptability. Those potentially reusable components (and the set of design patterns that guide their use) could be grouped into a so-called *component framework* [12]. Such frameworks constitute a starting point for the development of new applications. Even more, the assembly of a set of existing framework components could eventually generate fully functional new applications.

Now, the construction of a CSCL framework is not an easy task: the identification and dimensioning of components implies that software engineers have a proper understanding of the main collaborative learning concepts [2]. Obviously, some of these concepts, namely those related to the support for collaboration, also belong to the CSCW domain.

Although with different motivations [21], CBSE principles have also been applied to the CSCW domain and, consequently, several proposals for CSCW component frameworks can be found in the literature (e.g. [1],[8],[9]). These component frameworks for the CSCW domain should be considered as a starting point for the achievement of those for the CSCL field. Framework components related to functionality such as group management, collaborative sessions management, shared workspaces management, coordination support, awareness management, etc. are also useful for CSCL applications.

This paper is particularly focused on two common functional aspects of CSCL/W applications suitable for being included into the corresponding component frameworks: coordination and awareness. Although coordination and awareness are broad concepts, this paper copes with: *object-level coordination* and *shared workspace awareness*.

Object-level coordination "…deals with multiple participants' sequential or simultaneous access to the same set of objects…" [6]. For example, in a collaborative editor where participants modify a document by turns, object-level coordination decides who has the following turn. If a participant does not own the turn and tries to

modify the document, the object-level coordination system should forbid that operation. *Object-level coordination* "…deals with the organization of activities to be performed by the users, and not the organization of processes to be performed by the system…" [6]. *Object-level coordination* should not be confused with *activity-level coordination* that manages the flow of collaborative tasks that participants perform when using a collaborative tool. *Object-level coordination* is a very important aspect in collaborative tools as "…it can enhance close inter-working of groups and the synergy that makes groups productive and energized…" [6].

S*hared workspace awareness* comprises "…the up-to-the-minute knowledge a participant needs about other participants' interactions with the shared workspace…" [11]. In the previous example regarding a collaborative editor, *shared workspace awareness* would be in charge of informing all participants about who is modifying what part of the document at every moment. There are other types of awareness such as social awareness, task awareness, and concept awareness but they are outside the scope of this paper. Shared workspace awareness is of crucial importance in collaborative tools as it "…allows groups to manage the process of collaborative working" [3]. In this paper, the term awareness refers, by default, to shared workspace awareness.

The analysis of existing CSCW frameworks shows that coordination and awareness support is: very limited or non-existent in some cases; and very biased to particular applications in others (what implies that it can hardly be reused). There are also some proposals of isolated coordination support systems [14] but they do not take into account relationships between coordination and other aspects of collaboration support (mainly awareness).

In this context, this paper introduces AORTA (Action-oriented decOupled aRchitecture for coordinaTion and Awareness) a software architecture for a set of components potentially integrable into existing CSCL/W frameworks in order to support object-level coordination and shared workspace awareness. AORTA is focused on synchronous and distributed collaborative applications according to the well-known taxonomy proposed in [5].

The main goal of AORTA is hiding, to the developers of CSCL/W applications, the complexity associated to the management of coordination and awareness tasks. Those developers should simply include AORTA's components into their applications and perform a limited set of configuration steps. In this scenario, the development of CSCL/W applications would be greatly simplified although it also poses important challenges to AORTA design.

From a functional point of view, AORTA supports a broad range of potential behaviors regarding coordination and awareness. That generality is achieved by the use of coordination and awareness configurable policies. Such policies specify a particular coordination or awareness behavior to be enforced by AORTA. A CSCL/W application developer should only select the most appropriate existing policy or even specify new ones. AORTA coordination and awareness policies are based on the information that characterizes collaborative interactions, and can support dynamic collaboration scenarios in which one particular policy might not be valid for the whole duration of the collaboration.

Another functional feature of AORTA is that it provides joint support to coordination and awareness, enabling the relationships between both aspects.

From a software engineering point of view, AORTA is designed in order to decouple coordination and awareness support as much as possible from the development of collaborative applications. That decoupling is achieved by means of

software design patterns that reduce the learning curve for developers that use AORTA and provide specific cutpoints for collecting information required for coordination and awareness purposes.

The paper is structured as follows. Section 2 motivates and describes the above key functional features of AORTA: joint coordination/awareness support whose behavior depends on configurable policies based on collaborative interactions. Section 3 introduces and discusses the software architecture of AORTA and its implications from a software engineering perspective. Section 4 describes our current prototype of AORTA that complements coordination and awareness aspects of the ANTS component collaborative framework [8]. Some features of that prototype have been tested using a CSCL application, also described in section 4, for the collaborative resolution of puzzles. More concretely, section 4 shows how, by using AORTA, application development is decoupled from coordination/awareness aspects, application development is facilitated by the use of action-orientation, and application coordination/awareness behavior can be configured and changed without modifying the application itself. Section 5 compares AORTA with other related proposals that can be found in the literature. Finally, section 6 concludes the paper and introduces some potential future research lines.

2 AORTA: Functional Characteristics

The previous section has introduced the main functional features of AORTA. Why are those features important for CSCL/W applications? This section motivates, justifies and describes two of the most important ones, namely joint coordination and awareness support, and the use of action-based coordination and awareness policies. These functional features have obviously influenced AORTA design decisions (that will be explained in section 3).

2.1 Joint Coordination/Awareness Support

Several authors recognize that coordination and awareness are mutually influencing factors in collaborations. For instance, Dourish and Belloti state that "…awareness information is always required to coordinate group activities, whatever the task domain…" [3]. Similarly, Gutwin and Greenberg stress the usefulness of awareness information "…for many of the activities of collaboration – for coordination action, managing coupling …" [10]. Coming back to the collaborative editor example, awareness information (e.g. a participant makes more that four illegal modifications during its turn) could be used by the coordination support to take or change a decision (e.g. that participant loses the turn). On the other hand, coordination decisions (e.g. a participant loses its turn) could be used as awareness information to all or part of the participants (e.g. the name of the punished participant is coloured in red in the others participants' list of collaborators to indicate the turn lose).

All these factors have influenced the decision to support, in AORTA, information exchange between coordination and awareness tasks.

2.2 Action-Based Coordination and Awareness Policies

What does AORTA coordinate? What does AORTA make participants aware of? These are not obvious questions as the literature contains several and different approaches to coordination and awareness that reflect each author' own conceptualization on these topics which is not always made explicit.

AORTA coordinates and makes participants aware of *indirect collaborative interactions*. This is a term that comes from the CSCL domain although it could be also consequently translated to the CSCW field. A *collaborative interaction* is "…an action that affects, o may affect, the collaboration process. That action, or its effects, must be perceived by at least a member of the collaborative group or community, different from the emitter" [15]. In this context, an action is an application event (observable from the exterior of the application) expressed in terms of application usage. For instance, an event (a change in the text shown by an editing application) could be an action (a user has modified a document) or not. This action becomes an interaction if another collaborative user becomes directly or indirectly aware of it. There are several types of *collaborative interactions*. AORTA deals with *indirect collaborative interactions* that are mediated by an object (a CSCL application in this case). They usually take place in shared workspaces [16].

The above distinction has two effects on AORTA: first of all, AORTA policies are triggered by *actions* that can potentially become *indirect collaborative interactions*. A coordination policy should evaluate whether an action requested by a participant is allowed or not according to the current *coordination state* (e.g. whether this participant owns the turn or not). If the action is finally executed, it becomes an *indirect collaborative interaction*. The awareness policy decides whether the result of a coordination decision should be communicated to other participants; secondly, *actions* constitute the point of junction between AORTA and the collaborative applications that use it. Therefore, the properties of an action determine what information an application should communicate to AORTA. According to [16], an *indirect collaborative interaction* (and therefore the action that generates it) is characterized by a role (that the participant that generates the action plays), a shared object (over which the action is performed), an operation (over that object), and a time stamp (that indicates when the action is performed).

3 AORTA: The Proposed Software Architecture

AORTA is a layered and replicated architecture that provides *object-level* and *action-based coordination* and *shared workspace awareness* services to synchronous CSCL/W applications.

AORTA is designed so as to offer its services to collaborative applications that follow the replicated or hybrid variants of the MVC (Model-View-Controller) architectural pattern [19]. The integration between the applications and AORTA is achieved by means of the *Controller* that must be totally or partially replicated in all the applications of the participants in the collaboration (see Fig. 1).

In this context AORTA takes decisions about coordination and awareness tasks that are resolved locally at each participant's application (using information previously received from other applications) thus avoiding performance degradation in the synchronous collaboration.

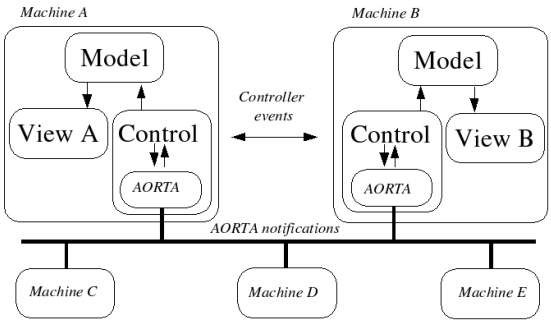

Fig. 1. AORTA within a CSCL/W application with a replicated MVC architecture

Coordination and shared workspace awareness services provided by AORTA are guided by *policies*. The definition of new *policies* implies the adaptation and extension of AORTA's behaviour without modifying AORTA's architecture.

AORTA is action-oriented. As it was explained in section 2.2, actions are the basic unit of information exchanged among applications and AORTA. Applications request the execution of actions to AORTA. Then AORTA processes them, and decides (coordination) whether an action must be executed (thus becoming an *indirect interaction*, see section 2.2) or not. AORTA afterwards notifies (awareness) the result of this decision to all or part of the other collaborating applications.

The replicated architecture of AORTA is composed of four functional blocks that are located in three software layers that constitute the AORTA layered architecture: application layer, collaboration layer, and communication layer (see Fig. 2).

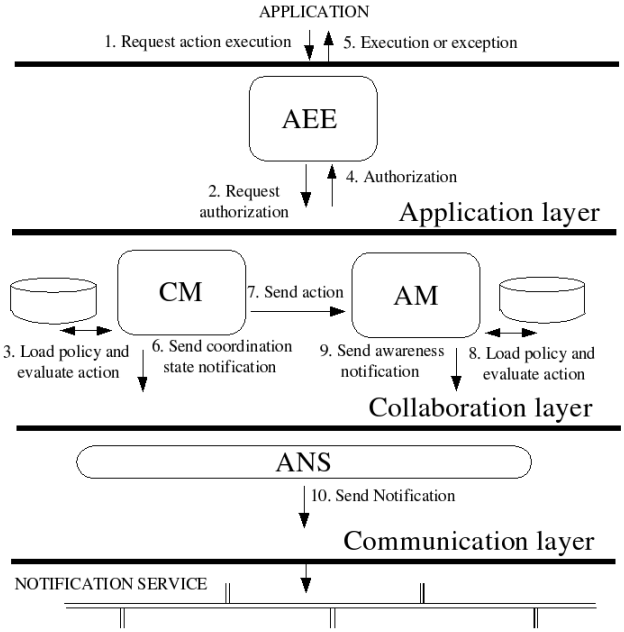

Fig. 2. AORTA layered architecture

3.1 Application Layer

The *application layer* constitutes the point of contact among applications and the *ActionExecutionEngine* (AEE) functional block of AORTA. The AEE is responsible for the execution of actions. It receives requests from the application for the execution of specific actions and responds with their actual execution or with an exception (if the action cannot be executed).

The relationship among applications and the AEE is mediated by means of the *Command* software design pattern [7] that encapsulates the request for the execution of an action within an object. By means of this object, the AEE knows the action whose execution is being requested by the application user. The execution of that action depends on the fulfilment of a set of rules (contained in a *policy*) that are associated to the request.

The use of the *Command* pattern enables the decoupling of the action execution request from the logic that determines whether it can be executed or not. The advantage of this decoupling is threefold: that logic can be separated from the application itself thus eventually becoming transparent to the developer of the application (e.g. it can be provided by AORTA); action execution can be made dependant on collaboration services (e.g. coordination); and it facilitates the maintenance and reuse of collaboration services.

If the developer of a CSCL/W application wants to use AORTA services, he only has to represent application-level actions according to the prescriptions of AORTA and to request their execution to the AEE (Fig. 2/Task 1). Those prescriptions simply indicate that an action type is a class that implements a predefined interface and that the instances of that class must contain information regarding the identity of the user that requests the action execution, the identity of the shared object affected by the action, and the operation that the user wants to perform on that object. Fig. 3 contains a UML class diagram in which the relationships among all those elements are detailed.

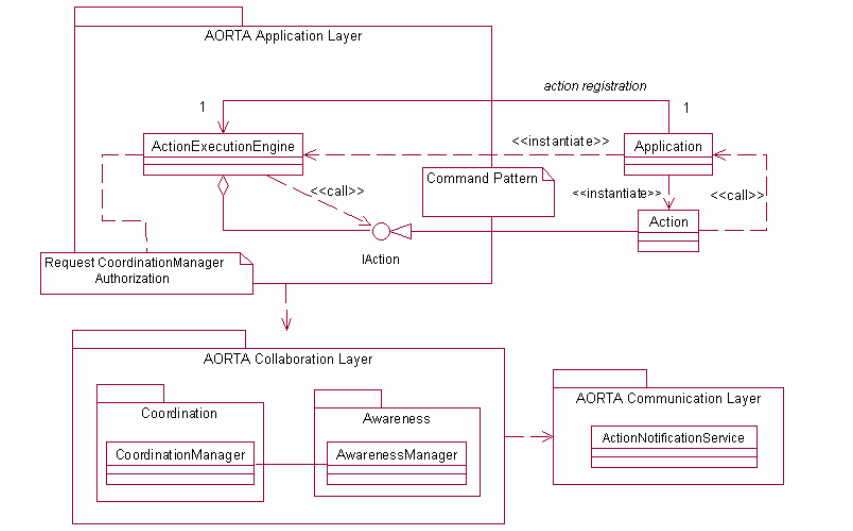

Fig. 3. UML class diagram of the AORTA application layer and its relationship with the supported CSCL/W application

When the AEE receives an action execution request, it also adds to that action complementary information such as a timestamp (also needed for characterizing a collaborative interaction, see section 2.2). After that, the AEE sends the resulting action object to the *collaboration layer* for its evaluation (Fig. 2/Task 2). Fig. 4 shows a UML sequence diagram detailing these interactions.

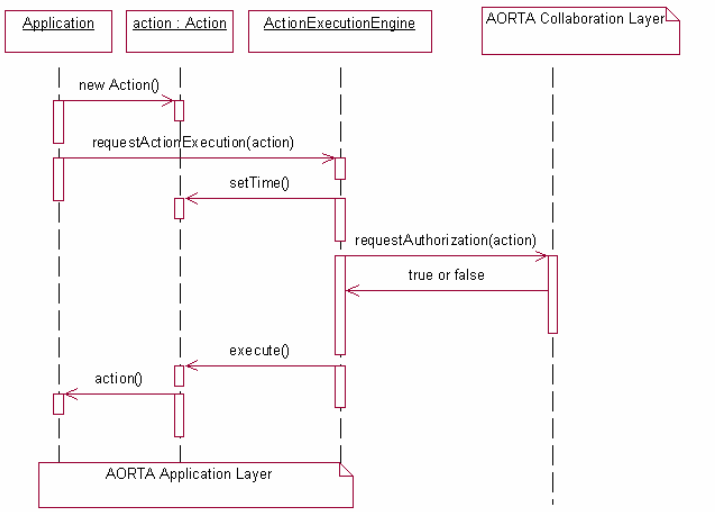

Fig. 4. UML sequence diagram describing the behaviour of the AORTA application layer and its relationship with the AORTA collaboration layer

3.2 Collaboration Layer

This layer offers coordination and shared workspace services by means of the *CoordinationManager* (CM) and the *AwarenessManager* (AM) functional blocks.

The *CoordinationManager* (CM) is responsible for coordination-related decision making. It evaluates whether requests for action execution can be performed or not. If the evaluation is positive (the action can be executed), the CM also informs other CMs (thus maintaining a consistent *coordination state*).

The *AwarenessManager* (AM) is responsible for shared workspace awareness. It receives decisions about the execution of actions from the CM and decides whether they have to be notified to other participants' applications or not. The AM is also responsible for performing those notifications.

The AEE requests authorization to the CM for the execution of an action. The CM receives an action object and evaluates its contents. That evaluation is based on the contents of a coordination policy that has been previously loaded from a *policies repository* (Fig. 2/Task 3). The decision on which policy should be loaded is dictated by an administrative interface (although a loaded policy may dictate the loading of a new one). Coordination policies may dictate rules for turn management, concurrency conflicts, resource access, etc. Furthermore, these policies may access information regarding shared workspace awareness and adapt their state consequently in a dynamic fashion.

Once a coordination policy has been selected and loaded, the CM uses it for evaluating an action. The policy defines rules that can be used to determine whether

an action can be executed or not. Those rules are based on information provided by the action (operation, user, object, timestamp), and in the actual *coordination state* associated to that policy. That *coordination state* is determined by the sequence of previously made coordination decisions. The *coordination state* is replicated in every involved CM thus enabling the local making of further coordination decisions. *Coordination state* is only updated when actions are executed (Fig. 2/Task 6).

The CM uses the result of a coordination-related evaluation to decide whether it allows the AEE to execute an action (Fig. 2/Task 4). If the action execution has been authorized, the AEE performs that execution. If not, the AEE sends an exception (Fig. 2/Task 5).

Finally, the CM updates the action object with the result of the coordination decision and sends it to the AM (Fig. 2/Task 7). The AM evaluates the content of the action object according to a previously loaded awareness policy (Fig. 2/Task 8) from a *policies repository*. Awareness policies determine what actions should be notified and what other AMs should receive those notifications. Once the policy is loaded, the AM performs the corresponding notifications by means of the NS (Fig. 2/Task 9).

3.3 Communication Layer

The communication layer only contains a functional block: *ActionNotificationServices* (ANS). The ANS is an action-based notification service responsible for propagating the occurrence of an action to all AORTA replicas.

The ANS is accessed by the AM and CM for requesting notifications sending. In the AM case, awareness-related notifications are sent to other applications interested in them. In the CM case, the ANS is used to exchange notifications regarding *coordination state* changes. In other words, CMs from the collaborating applications are synchronized through the ANS.

The ANS decouples AORTA from the use of a particular MOM (message-oriented middleware) technology (Fig. 2/Task 10).

3.4 Discussion

As presented before, AORTA bases its overall architecture in a modification of the *Command* pattern that uses actions as its key abstraction. Actions thus represent a higher level abstraction than events, and they create a seamless model for both coordination and awareness services. This is a clear contribution of this paper that can also lead to other two interesting ideas:

- Importance of patterns for CSCL/W development: Applications that use well-known patterns such as *Command* can be more easily adapted to a CSCL/W framework. One key problem in collaborative settings is to convert single-user to multi-user collaborative applications. Design patterns offer clear pointcuts for aspect oriented programming (AOP) that can help to automate such code conversions. As a clear consequence, instructing developers to use patterns help to reduce the learning curve and development cycle of collaborative applications.
- Integrate Actions as a first level service in CSCW/L frameworks: In this paper we propose a decoupled extension to a CSCW component framework. But, we can even go further and propose an action service as a first class member of any CSCL/W framework. The importance of actions justifies such service that can

also help to smooth the learning curve of new applications. Furthermore, an Action service is mandatory for creating a fully reflective system. A reflective system must support both introspection and intercession. Intercession is now supported with our decoupled service, but introspection requires action metadata and policy metadata that should be provided by the proposed action service.

In conclusion, we foresee interesting research in the future regarding coordination and awareness models. The joint use of higher level abstractions, designs patterns and CBSD can lead to more advanced models enabling innovative collaborative applications.

4 Proof of Concept and Validation

In order to validate AORTA, we decided to implement a prototype on top of an existing Java-based CSCW component framework. As we mentioned before, CSCW component frameworks represent a good starting point for the achievement of those in the CSCL field.

The ANTS component groupware framework [8] was selected as the basis for the AORTA prototype because it provides interesting collaborative services and because it permits third-party extensions that clearly fit with AORTA's architectural requirements.

Section 4.1 describes the ANTS framework whereas section 4.2 introduces our AORTA prototype and *MagicPuzzle*, a CSCL applications developed on top of ANTS and AORTA that has been used to test and validate some of the AORTA's features.

4.1 ANTS Framework

ANTS framework aims to facilitate development of CSCW components by facading complex distributed services in an easy and comprehensive fashion. It follows a CBSD approach and a layered services model that eases third party extensions. ANTS comprises three main layers (see
Fig. **5**):

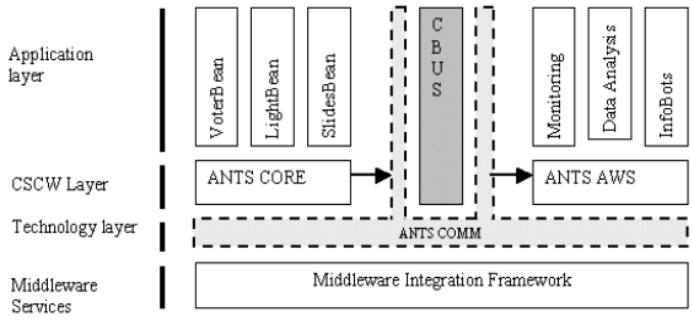

Fig. 5. ANTS framework architecture

- At the application layer, it provides a client-side container for JavaBeans components, transparently accessing remote properties and a distributed event service.
- At the conceptual or CSCW layer, it provides essential collaborative services: shared sessions, support for synchronous and asynchronous components, security, basic coordination, and a server-side awareness infrastructure.
- At the technological level, the framework is constructed on top of a middleware integration platform (Java 2 Enterprise Edition) and facades, by means of the CBUS (*Collaboration Bus*) different notification services like Sun *Java Message Service* (JMS) or DSTC Elvin.

ANTS heavily relies on the JavaBeans component model as a basic abstraction in order to simplify development of collaborative applications. In this line, ANTS leverages and extends the JavaBeans model to provide distributed properties and events, and to access the underlying collaborative services. Furthermore, the so called collaboration bus (notification service) creates the glue between component's state propagation and awareness and actuator services listening to events in the bus.

Nevertheless, ANTS offers very low-level abstractions to deal with awareness and coordination services. ANTS directly works with events and subscriptions in the awareness service. This approach is very flexible but also more intricate for framework users. Our AORTA action model thus represents a step forward compared to previous low-level abstractions such as events.

Furthermore, ANTS aims to integrate its coordination model with the JavaBeans Vetoable properties and listener approach. This idea is coherent with the overall architecture but it also hinders development of more complex and flexible coordination models. We again outline our coordination model based on actions as an interesting contribution for both CSCW and CSCL environments.

Finally, we outline that ANTS is a suitable platform to work with, mainly because its component based architecture, and its layered services model enabling extensions at all levels of the framework.

In the next subsection, we will present how our current prototype of the AORTA architecture provides an extension to the ANTS framework in terms of coordination and awareness, and a CSCL application built on top of AORTA and ANTS that validates our proposal.

4.2 Proof of Concept: the Magic Puzzle Application

MagicPuzzle is a synchronous application that supports the collaborative resolution of puzzles by groups of primary education students. This type of CSCL applications, that are oriented to the collaborative assembling of small pieces of knowledge, provides important advantages from an educational point of view: they promote the acquisition of social abilities and they also are capable of reflecting the process of learning.

Our current prototype of *MagicPuzzle* is a Java application that uses services provided by the ANTS CORE module of the CSCW layer of ANTS as well as coordination/awareness services provided by AORTA through its application layer (see Fig. 6, which also shows the look and feel of the *MagicPuzzle* GUI). ANTS provides *MagicPuzzle* with collaborative services such as session management, shared objects management, and security. On the other hand, AORTA is responsible for object-level coordination and shared workspace awareness services. As it can

also be appreciated in Fig. 6, AORTA uses the services provided by the ANTS COMM module of the technology layer of ANTS. It is important to point out that this is not a requirement of our current AORTA prototype as it might use a different notification service. If we compare Fig. 5 and Fig. 6, it is possible to appreciate how AORTA can be considered, in this particular case, as an enhancement to ANTS that provides coordination/awareness improvements: left-hand part of Fig. 6 includes those ANTS elements of Fig. 5 used by *MagicPuzzle*.

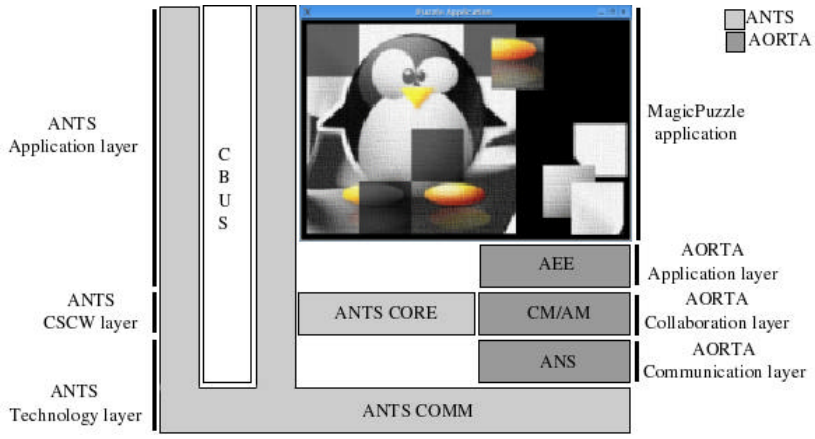

Fig. 6 Joint ANTS-AORTA support for the *MagicPuzzle* application

The action-orientation of the AORTA architecture had several implications in the development of *MagicPuzzle*. First of all, those actions that characterize the *MagicPuzzle* behaviour were identified and subsequently coded according to the prescription summarized in Fig. 3. Two action types were considered in this first *MagicPuzzle* prototype: *SelectPiece* and *DropPiece*. A second implication consisted of a proper coding of the logic associated to the identified actions so that it could be triggered by requests generated by the AEE module of the application layer of AORTA (see Fig. 2/Task 5).

Once the *MagicPuzzle* prototype was coded (taking into account the above considerations), its behaviour with respect to coordination and awareness could be changed without requiring further modifications. These behavioural modifications were possible by just changing AORTA coordination and awareness policies. In our current AORTA prototype policies are Java classes that can be dynamically loaded by means of an administrative interface.

Several well-known coordination/awareness policies were tested in order to validate the flexibility that AORTA provides. The following paragraphs describe some of the most representative (it is a non-exhaustive list):

1. *Token* coordination policy: it restricts concurrent accesses to shared objects (puzzle pieces in the *MagicPuzzle* case). The policy registers the action that a participant wants to perform over a shared object (*SelectPiece* or *DropPiece*) and prevents accesses to that object until the requested action is performed.
2. *Leader* coordination policy: this policy contains a list of participants and the identity of the so-called *leader*. Only one participant can perform an action on any of the shared objects at a time. The *leader* decides, once a previous action has been performed, who has the *turn* for performing a new action.

3. *Everybody* awareness policy: this policy dictates that all participants have to be aware of other participants' attempts to perform actions (even when those actions cannot be performed due to coordination decisions). Several variants of this policy (e.g. only action requests generated by a subset of participants are notified, only performed actions are notified, etc.) were also tested.

This flexibility provided by AORTA enabled the handling of complex collaboration scenarios by just changing coordination and awareness policies for *MagicPuzzle*.

Also, the decoupled nature of AORTA facilitated the development and maintenance of *MagicPuzzle*. In fact, after *MagicPuzzle* was finished, several changes that were performed into the AORTA prototype did not affect *MagicPuzzle*, thanks to the decoupling provided by the *Command* pattern.

Finally, and taking into account our previous experience developing collaborative applications (including other versions of *MagicPuzzle* using different underlying technologies), working with actions (which is the key AORTA abstraction) appeared as a more natural and easy way of developing CSCL tools: first of all, it is simpler, for developers, to update the model (or even the view) of an application from information provided by actions than from lower-level events; secondly, actions are abstractions more closely related to the CSCL domain, as it was already explained in section 2.

5 Related work and discussion

Many collaboration frameworks address the problem of coordination control over shared structures. Most of them also offer suitable abstractions to develop concurrent multi-user applications. We will however outline at this point that most approaches still focus on low level abstractions such as events, event ordering, locks and shared structures. Furthermore, a joint support for both coordination and awareness services is almost inexistent in many platforms.

In this context, this section analyses five existing collaboration frameworks and compares them with AORTA. Three of them, GroupKit, ANTS, and Groove, are general purpose frameworks whereas COCA and Intermezzo are particularly focused on coordination and awareness support.

GroupKit [9] is a classic toolkit for CSCW development that pioneered many advances in CSCW architectures. It offers seamless coordination support within the so called environments (shared structures) and permit third-party extensions through the "open protocols". More specifically, open protocols have three components: a controlled object (server) that maintains state, a controller object (client), and a protocol describing how they two communicate. GroupKit authors showed three examples like floor control, conference registration, and brainstorming that benefit from open protocols.

In conclusion, GroupKit provides clean and extensible support for coordination but it focuses on data structures, events and user interface widgets. Regarding awareness, they also permit event monitoring and workspace awareness widgets. We however believe that our action model represents a higher level abstraction than events and that both coordination and awareness services achieve interconnection in a more comprehensible fashion.

ANTS was briefly described in section 4.2. We outlined that ANTS is built on top of the JavaBeans component model and provides additional services like shared data structures (bean properties) and a distributed event system (bean events and listeners). ANTS coordination model is based on *VetoableActionListeners* and distributed locks and also permits third party extensions. State propagation, coordination and the awareness service are seamlessly interconnected by the collaboration bus (notification system). As we mentioned before, ANTS also bases its overall model on events in the Collaboration bus instead of actions.

Groove (http://www.groove.net) is a peer computing framework for developing collaborative applications. It encompasses a broad set of APIs and services that devise the biggest CSCW framework ever developed. Coordination services in Groove are handled with the so-called *deltas*.

Tools in Groove transform a user gesture into a transactional unit of change called a *delta*. A *delta*, which is a container that houses one or more commands created by an engine, indicates that something has changed in a shared space. When the tool has completed writing the engine's commands into the *delta* container, it submits the *delta* to the dynamics manager for execution and eventual dissemination. The dynamics manager is responsible for executing changes to shared space data requested by tools. It is the mediator in Groove's mediated model-view-control architecture.

Despite the nice and huge Groove's architecture, it does not provide hooks for developing coordination policies. Its main framework hotspots are the Tools, and coordination control is restrained to *deltas* and transactional services. Regarding awareness, Groove also provides subscriptions for user presence and activity awareness. We understand that Groove's main goal is to offer rich services to the main hotspot of the framework: the Tools in the Workspace. They do no offer a clear extensible model for coordination and awareness, and they also focus on lower level abstractions such as user gestures. We however think that Groove's model could easily evolve towards a fully integrated action support in *deltas* and create more extensible coordination and awareness services.

Intermezzo [4] is a client-server architecture which offers collaboration services to groupware applications. Intermezzo is related to AORTA in the sense that both of them specifically provide coordination and awareness support. However, Intermezzo coordination support is based only on user access control rights on shared objects. Action information is never considered as a part of the semantics that is evaluated by coordination rules. In the case of awareness, although Intermezzo does provide workspace information, coordination services never take advantage of it.

COCA [14] is a coordination framework for developing collaborative applications. COCA provides a powerful specification language for defining coordination policies that are interpreted at run time. COCA is similar to AORTA because all distributed peers are connected by a collaboration bus. Also, coordination policies can be changed and loaded at run time to handle unexpected collaboration states. Nevertheless, it does not provide awareness services. We truly believe awareness is an important part of synchronous collaboration and that support for workspace awareness can greatly improve coordination [10].

6 Conclusions and future work

This paper has presented AORTA, a software architecture for enhancing CSCL/W component frameworks with object-level coordination and shared workspace awareness support. The use of action-oriented policies in AORTA in order to determine coordination and awareness behaviour enables synchronous and distributed collaborative applications to achieve a better adaptation to dynamic collaborative scenarios. Also, the use of a modification of the *Command* pattern provides a very convenient decoupling between AORTA and the applications developed on top of it. Finally, the use of action orientation provides the developers of CSCL/W with abstractions more closely related to those of the collaboration domain.

The paper has described a prototype of AORTA developed to enhance some of the capabilities of the ANTS groupware framework. Also, it has shown how a prototype of a CSCL application for the collaborative resolution of puzzles has been used to test the main features of AORTA and its software engineering implications.

Of course, there are still a lot of open research issues. In terms of validation, more CSCL application types should be developed on top of AORTA to have a better idea of its applicability. Furthermore, our current AORTA prototype has only been tested in conjunction with the ANTS framework. One of our future goals is checking its potential use in other collaborative component frameworks.

We also recognise that the availability of a policy specification language, similar to that proposed by COCA, would greatly improve AORTA flexibility. Also, it would be very useful to provide educators and even learners with tools for edition and management of their own policies.

Other interesting research lines include: the use of design patterns for providing pointcuts for aspect oriented programming so as to facilitate the conversion of single-user to multi-user collaborative applications; and the inclusion of actions as first class members of any CSCL/W framework.

Finally, we foresee promising research work in new middleware services for collaborative work. More concretely, new decentralized peer to peer abstractions like DERMI's multicalls, anycalls and manycalls [17] can help to devise more flexible and autonomous collaborative scenarios. We are also studying how to leverage existing work in collaborative systems in order to permit a smooth transition to such decentralized scenarios. Furthermore, AORTA's decoupled and replicated model considerably help us to transition to the aforementioned p2p setting.

Note that ANTS, AORTA and *MagicPuzzle* are freely available, including source code, in the ANTS web site: http://ants.etse.urv.es.

References

1. Beca, L., Fox, G. C., Podgorny, M.: Component Architecture for Building Web-Based Synchronous Collaboration Systems. Proceedings of the 8th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '99) (1999)
2. Dimitriadis, Y. A. , Asensio, J. I., Martínez, A., Osuna, C. A.: Component-Based Software Engineering and CSCL in the Field of E-Learning. Upgrade (digital journal of European Professional Informatics Societies), special issue on e-learning - boarderless education. 4 (5) (2003) 21-28

3. Dourish, P., Belloti, V.: Awareness and Coordination in Shared Workspaces. Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work (CSCW'02) (1992)
4. Edwards, W. K.: Policies and Roles in Collaborative Applications. Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work (CSCW'96) (1996)
5. Ellis, C. A., Gibbs, S. J., Rein, G. L.: Groupware: Some Issues and Expeciences. Communications of the ACM. 43 (1) (1991) 39-58
6. Ellis, C. A., Wainer, J.: A Conceptual Model of Groupware. Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW'94) (1994)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
8. García, P., Gómez-Skarmeta, A.: ANTS Framework for Cooperative Work Environment. IEEE Computer. (2003) 56-62
9. Grundy, J., Hosking, J.: Engineering Plug-in Software Components to Support Collaborative Work. Software Practice and Experience. 32 (2002) 983-1013
10. Gutwin, C., Greenberg, S.: The Effects of Workspace Awareness Support on the Usability of Real-Time Distributed Groupware. ACM Transactions on Computer-Human Interaction. 6 (3) (1999) 243-281
11. Gutwin, C., Stark, G., Greenberg, S.: Support for Workspace Awareness in Educational Groupware. Proceedings of the 1st International Conference on Computer Support for Collaborative Learning (CSCL'95) (1995)
12. Jonsson, T., Crnkovic, I., Hnich, B., Kiziltan, Z.: Specification, Implementation and Deployment of Components. Communications of the ACM. 45 (10) (2002) 34-40
13. Koschmann, T.: CSCL: Theory and Practice of an Emerging Paradigm. Lawrence Erlbaum, Malwah, NJ, USA (1996)
14. Li, D., Muntz, R.: COCA: Collaborative Objects Coordination Architecture. Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work (CSCW'98) (1998)
15. Martínez, A., Dimitriadis, Y. A., de la Fuente, P.: Contributions to Analysis of Interactions for Formative Evaluation in CSCL. In: Llamas, M., Fernández, M. J., Anido, L. E. (eds.): Computers and Education. Towards a Lifelong Learning Society. Kluwer Academic (2003) 227-238
16. Mühlenbrock, M.: Action-Based Collaboration Analysis for Group Learning. IOS Press, Amsterdam, The Netherlands (2001)
17. Pairot, C., García, P., Gómez, A. F.: Dermi: a Distributed Hash Table-Based Middleware Framework. IEEE Internet Computing (to appear). (2004)
18. Roschelle, J., Kaput, J., Stroup, W., Kahn, T. M.: Scalable Integration of Educational Software: Exploring the Promise of Component Architectures. Journal of Interactive Media in Education. (1998)
19. Suthers, D.: Architectures for Computer Supported Collaborative Learning. Proceedings of the IEEE International Conference on Advanced Learning Technologies, Madison, Wisconsin, USA (2001)
20. Szyperski, C.: Component Technology - What, Where and How? Proceedings of the 25th International Conference on Software Engineering (ICSE'03) (2003)
21. Teege, G.: Users As Composers: Parts and Features As a Basis for Tailorability in CSCW Systems. Computer Supported Cooperative Work. Kluwer Academic Publishers (2000) 101-122