A High-level Reference Model for Reusable Object-level Coordination Support in Groupware Applications

Miguel A. Gómez-Hérnandez, Juan I. Asensio-Pérez, Eduardo Gómez-Sánchez, Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis
Collaborative and Intelligent Systems Group, University of Valladolid
School of Telecommunications Engineers, Camino Cementerio s/n, Valladolid, Spain
{miggom, juaase, edugom, migbot, yannis}@tel.uva.es

Abstract

The success of groupware software largely depends on its capability for being reused in different collaborative scenarios without requiring significant software development efforts or user technical involvement. During the last decade several toolkits, frameworks, and specialized middleware software layers have been proposed as potential solutions to the groupware reusability problem. Those solutions cover common groupware functionality such as group formation and management, group awareness, shared workspace management, etc. This paper is focused on the reusable support to one of these common functionalities: the so-called object-level coordination that deals with multiple participants' sequential or simultaneous access to the same set of shared objects in a collaborative setting. The paper compares existing object-level coordination proposals and analyzes their capabilities and level of reuse. Starting from that analysis, and using a bottom-up approach, the paper also proposes a high-level reference model that identifies the desirable set of required functional elements, as well as their relationships, that reusable object-level coordination support should contain. The purpose of this reference model is twofold: to be used as a comparison framework for analyzing existing or yet-to-come object-level coordination building blocks; and, to be used as a starting point for developing new object-level coordination solutions (or redesigning existing ones) satisfying the whole set of requirements implicitly collected in the model. In addition to detailing the proposed reference model, the paper also illustrates and discusses both potential ways of using it.

1. Introduction

Reusability is a recurrent problem in software engineering that is particularly difficult to solve in the groupware domain due to the great variability in the requirements posed by different collaborative scenarios: group size, distribution scale, type of shared resources, remote vs. face-to-face collaboration, etc. Different proposals can be found in the literature claiming to alleviate this problem. Most of them agree in dividing groupware applications in building blocks (objects, components, services, libraries, etc.), some of them covering key and common groupware functionality and thus becoming potentially reusable: shared workspace management and awareness, group management, etc. Significant examples include toolkits such as GroupKit [20] or JSDT (Java Shared Data Toolkit) [23], component frameworks such as JViews [9], and specialized middleware layers such as ANTS [8]. The choice of using toolkits, frameworks, middleware layers, etc. (based on objects, components, services, etc.) implies software engineering differences in the way groupware applications are built on top of them. But reusability does not only depends on which software engineering approach has been adopted but also on groupware domain issues such as the significance, from a functional point of view, of the offered building blocks [22]: if they are too coarse-grained they need to be configurable so as to adapt to the needs of the targeted scenarios (otherwise, they are difficult to be reused); if they are too fine-grained they do not solve important enough domain problems to the groupware developers so as to be worth its reuse. Also, the "size" of building blocks may have influence on the number of relationships (i.e. coupling) among them: too fine-grained building blocks might imply more tightly-coupled relationships thus making reusability of one single block more difficult.

This paper is focused on one particular groupware building block: object-level coordination. As defined in [6] object-level coordination "…deals with multiple participants' sequential or simultaneous access to the same set of objects…". 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.

The paper is motivated by research questions aimed at achieving reusable object-level coordination support in groupware applications: what are the functional differences among existing proposals for object-level coordination support that can be found in the literature? Which is the most proper "size" of the functionality of this building block so as to be significant from the groupware domain point of view and therefore increasing reusability? How is object-level coordination support coupled to other groupware building blocks? Might this coupling make reusability more difficult?

For answering these questions, the paper firstly analyzes (section 2) existing proposals of groupware conceptual models so as to identify potential relationships of object-level coordination with other building blocks. Secondly the paper analyzes and compares (section 3) some significant proposals for object-level coordination support so as to identify, following a bottom-up approach, an adequate set of common coordination functions for that building block. Then (section 4), the paper proposes a high-level reference model that combines the previously identified relationships with other building blocks and coordination functions thus providing a set of functional and high-level architectural requirements for potentially reusable object-level coordination support. In order to provide a better description of the proposal, the paper shows how the elements of the high-level reference model are used in order to support two example coordination scenarios (section 5). Finally, the paper discusses (section 6) on the twofold utility of the proposed reference model: on the one hand, the reference model might be used as a comparison framework for analyzing and evaluating existing and yet-to-come proposals of object-level coordination; on the other hand, the reference model might be used as the starting point for developing new object-level coordination support systems (or redesigning existing ones) satisfying the functional and architectural requirements prescribed by it. In this sense the paper also discusses how the reference model is also useful for anticipating software engineering problems that might arise during that development process and that would imply difficulties for achieving reusability. Software coupling among building blocks is one of those problems that is currently under study. Figure 1 tries to graphically summarize the goals, structure and scope of the paper.

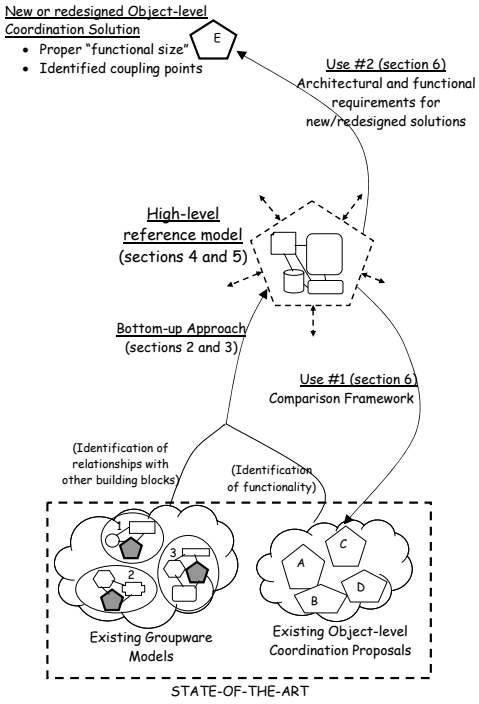

Figure 1. **Scope of the paper proposal**

2. Identifying object-level coordination relationships

Before analyzing existing proposals for object-level coordination support, it is important to identify its most relevant relationships with other "building blocks" of groupware applications, and with the applications themselves.

The identification of those relationships is a prerequisite for understanding mutual dependencies and coupling points that might eventually preclude the reuse of the object-level coordination building block in different groupware applications. Figure 2 sketches the following relationships that can be extracted from existing groupware conceptual models such as those proposed in [24], [2], [14]:

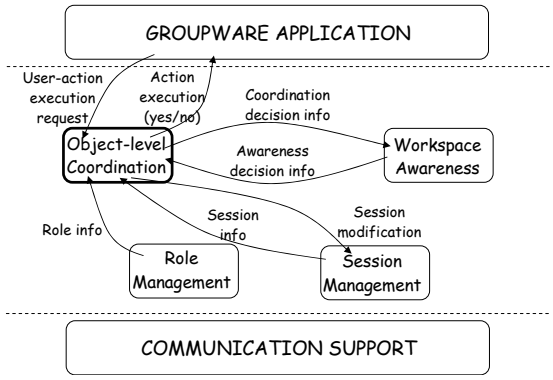

Figure 2. **Relations of object-level coordination with the groupware application and other groupware "building blocks"**

- **Application**: the groupware application requests the object-level coordination support to take the decision on whether a user-requested action on a shared object can be executed or not. It is important to point out that determining the way the object-level coordination building block will take that decision is a key factor for achieving reusability (as this factor largely influences the "size" of the functionality of this building block). For instance, if a user requests the modification (action) of a document (shared object) in a collaborative editor, object-level coordination might relay on a simply "owner rights" approach or use more complicated schemas such as the use of tokens, checking if other users are accessing the same document, etc. Also, another key aspect for fostering reusability is the way the applications interchange information with the object-level coordination block because if that communication is application-dependant, reusability will be very difficult to achieve.
- **Shared workspace awareness**: this building block comprises "…the up-to-the-minute knowledge a participant needs about other participants' interactions with the shared workspace…" [10]. Several authors recognize that coordination and awareness are mutually influencing factors in collaborations. For instance, [4] states that "…awareness information is always required to coordinate group activities, whatever the task domain…". Similarly, [11] stresses the usefulness of awareness information "…for many of the activities of collaboration – for coordination action, managing coupling …" Coming back to the collaborative editor example, awareness information (e.g. a user delivers a document section to another user in the role of "chief editor") could be used by the coordination support to take or change a decision (e.g. the other users collaborating in the writing of that document section are no longer allowed to modify it). On the other hand, coordination decisions (e.g. a participant loses its turn, in a scenario in which only users having the turn may request editing actions) 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).
- **Session Management:** this building block informs object-level coordination support on the set of participants that are interacting through shared objects in a collaborative setting. Decisions taken by object-level coordination support might imply, for instance and using the collaborative editor example, that a participant should abandon a session or momentarily stop collaborating if he has attempted more than four illegal modifications on a document section. These decisions should be communicated to Session Management to proceed accordingly. On the other hand, session information (e.g. a user possessing the turn abandons the session) might have influence on coordination decisions (the turn should be given to another user).
- **Role Management:** this building block informs object-level coordination support on the role adopted by the participants in a collaborative setting. Decisions taken by object-level coordination support on whether a user-requested action should be executed might be linked to the role that particular user is playing at that moment. This is the so-called Role-Based Access Control technique, RBAC [21]. For instance, the user playing the role of "chief editor" in the collaborative editor example is permitted to modify any document at any time independently of who else is making modifications at that moment.

All the described building blocks are supported in all models by some kind of "Communication" layer whose goal is hiding distribution complexity. Although not represented in Figure 2, there can also be a relationship among the object-level coordination building block and the communication support. The nature of that relationship depends on the distribution architecture of the object-level coordination support (e.g. from totally centralized to completely distributed among participants computing systems if they are not co-located). These distribution alternatives constitute a

key problem to take into account in the implementation of the coordination support. Nevertheless, distribution issues are out of the scope of this paper.

Any object-level coordination building block claiming to be reusable should be able to support the above relationships among building blocks in an as decoupled as possible way. Otherwise, reusing the object-level coordination block would imply "moving" some other blocks with it. Thus, and as it was explained in the introduction, too "big" functional blocks may pose difficulties for their reusability.

3. Identifying object-level coordination functionality

In order to identify the functionality that a potentially reusable object-level coordination "building block" for groupware application should provide, several significant existing proposals, that can be found in the literature, have been studied and analyzed. Those include DCWPL [1], COCA [15], Intermezzo [5], "A coordination model for secure collaboration" [25] and "A coordination framework and architecture for internet groupware" [3].

Although the analyzed proposals differ in their functional scope and targeted collaborative scenarios (as summarized in Table 1) they all agree in a key aspect that was also mentioned in the previous section: the goal of object-level coordination support is to take the decision on whether a user-requested action on a shared object should be executed or not.

The difference among the analyzed proposals is how they take those decisions. Basically, two main alternatives can be identified (these alternatives will be called "coordination functions" along the paper):

- **Access control to shared objects**: if a user requests an action on a shared object, the object-level coordination support allows it if the request satisfies a set of access rights prescribed by a predefined policy. For instance, in a collaborative editor, a policy may state that a user can only modify those parts of the documents (shared objects) created by him at any time.
- **Conflict resolution**: object-level coordination support detects whether several actions are simultaneously requested on the same shared object, and decides which ones should be permitted, according to a predefined policy. For instance, in the collaborative editor example, a policy may state that in the case several users are modifying the document title (shared object) at the same time, only the changes performed by the user in the role of "chief editor" will prevail. Also, while the "chief editor" is modifying the document title, other incoming modification requests will be denied.

Table 1. **Scope of the analysed object-level coordination support proposals**

Proposal	Scope
DCWPL [1]	Synchronous groupware applications. Validated with a shared document editor.
COCA [15]	Synchronous groupware applications with audio/video support and high scalability. Validated with a shared slide viewer.
Intermezzo [5]	Asynchronous groupware applications. Validated with a shared document repository.
"A coordination model for secure collaboration" [25]	Asynchronous groupware applications with security requirements. Validated with a shared documents reviewing tool.
"A coordination framework and architecture for internet groupware" [3]	Web-based and asynchronous CSCL (Computer-supported collaborative learning) applications. Validated with a web-based collaborative training tool.

Policies for access control and conflict resolution in the analyzed proposals evaluate different types of data (user, role, type of action, type of shared object, particular shared object, timing, etc.) at a particular instant or considering its evolution, thus defining what might be called "Coordination state".

Table 2. **Object-level coordination functionality supported by analysed proposals**

	DCWPL [1]	Coca [15]	Intermezzo [5]	[25]	[3]
Access control to shared objects	YES	YES	YES	YES	YES
Conflict resolution	YES	NO	NO	NO	YES (very limited)
Configurability	YES	YES	YES	YES	NO

Another interesting issue that arises when analyzing existing proposals for object-level coordination support is how those proposals define policies for their coordination functions. In some proposals, policies are predefined and limited in number. Other proposals provide support for defining new policies customized for the coordination scenarios in which they will be applied. This functionality, that will be called **"Configurability"**, introduces a new type of role in collaborative settings: the *"Coordination process designer"*. Configurability is a key functionality for achieving reusability as it allows object-level coordination support to fulfil the coordination requirements of a larger set of collaborative scenarios thus benefiting a potential larger set of groupware applications.

Table 2 summarizes the support of the identified coordination functions by the analyzed proposals.

4. A high-level reference model for reusable object-level coordination support

The previous two sections have identified a set of significant coordination functions for object-level coordination support, as well as relevant relationships with other groupware building blocks. This section details and combines both aspects into a single model that is intended to be used as a high-level reference that collects architectural and functional requirements for a potentially reusable object-level coordination support.

Reusability is promoted because the coordination functionality proposed by the model, supported by the model elements, is reportedly considered to be significant in a broad range of coordination scenarios (covered by the proposals analyzed in section 3). Also, the relationships of object-level coordination with other groupware building blocks identified by the model anticipate potential coupling problems that might make reusability more difficult.

Figure 3 depicts the proposed reference model by enlarging the object-level coordination block of Figure 2. Section 5 will discuss how the elements of his reference model would fulfil the object-level coordination requirements of two example applications in different collaborative scenarios.

The model contains a central functional element called **"Coordinator"** that is in charge of performing the coordination functions introduced in section 3.

The coordination functions receive input information coming from other building blocks (already identified in section 3) and the groupware application itself, as well as information provided by two additional data repositories (that belong to the coordination support). The purpose of these two repositories is to adapt the behavior of the coordination functions to different coordination scenarios without modifying those functions (thus fulfilling the goal of the "Configurability" capability introduced in section 3). They are:

- **"Coordination configuration"**: contains application-specific information that is needed to take coordination decisions. That includes:
  - *User list*: the enumeration of the users that might potentially request actions on shared objects.
  - *Shared object list*: the enumeration of the shared objects that might potentially be accesses by collaborating users so as to request an action on them. This list should also include the description of the actions that can be requested on each shared object.
  - *Access control list*: an association of which users have rights for accessing which shared objects so as to perform which actions.

  "Coordination configuration" information is complemented by role information provided by the role management building block. Thus, the "Access control list" may be enriched by associating roles to shared objects and actions (using the already mentioned Role-Based Access Control technique).

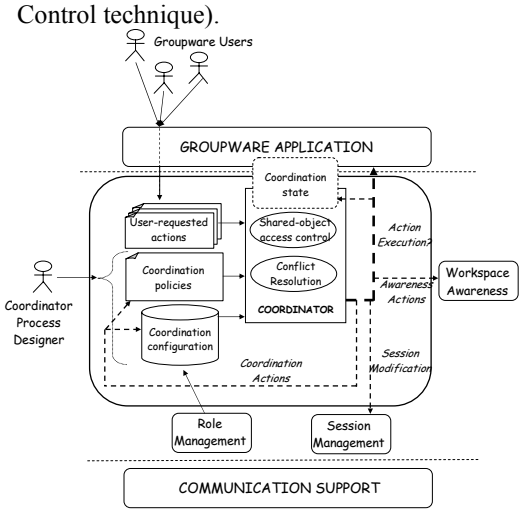

Figure 3. **High-level reference model for object-level coordination support in groupware applications**

- **"Coordination policies"**: contain instructions on how coordination functions should behave when receiving a user-request action on a shared object. According to section 3, this repository will contain two types of policies:
  - *Access policies*: indicate to the "Shared object access control" function what to decide when a particular user has (or hasn't) rights to perform an action on a particular shared object.
  - *Conflict resolution policies*: indicate to the "Conflict resolution" function what to decide when a coordination conflict has been detected and therefore needs to be solved.

  It is worth mentioning that when dealing with potential coordination conflicts (see section 3) "Access policies" play a proactive role as they may avoid that several users access the same object during the same time frame. Therefore, "Conflict resolution" policies are applied, in a

reactive way, only in those cases in which preventive measurements have failed. This same differentiation between proactive and reactive coordination policies is introduced in [17].

"Coordination state" is another important element of the reference model that influences the decisions taken by the coordination functions. The "Coordination state" provides information on previously taken coordination decisions. For instance, in the collaborative editor example, the coordination state might maintain a counter with the number of times user "John" has tried to modify the document title. An "Access control policy" might indicate to the "Share object access control" function to deny such access when the counter is greater than 3. Figure 3 shows how the coordination state is maintained by both the application and the object-level coordination building block. This "shared responsibility", observed in some of the existing proposals introduced in section 3, is due to the fact that some state information might be application-dependant (in the collaborative editor scenario example, the size of a particular paragraph might be a determining factor for precluding a particular user to modify it).

The final aspect of the proposed model to be described deals with the outputs of the coordination functions. Basically, four types of outputs can be identified:

- *Action-execution decisions*: this is the most important output and the basic goal of coordination functions. These functions, using the above described information sources, decide whether a user-requested action on a shared object should be executed or not.
- *Awareness actions*: as already explained in section 2, awareness actions are communicated to the awareness building block so as to let it know the decision taken on a user-requested action. Then, the awareness building block might inform the users on that decision using predefined techniques (visual gadgets, text messages, sounds, etc.)
- *Coordination actions*: some coordination decisions on user-requested actions may imply a modification of the input information that the coordination functions will use to process subsequent requests. Coordination actions are used with that purpose. They may be directed to the "Coordination state", to the "Coordination configuration", or even to the "Coordination policies". In the next section some examples of this kind of actions will be shown.
- *Session modifications*: as an additional side-effect of a coordination decision, the object-level coordination support might request the "Session management" building block to modify session configuration (for example, as mentioned in section 2, to remove a user from a session).

The next section will show examples of all the elements of the reference model that has been introduced.

5. Examples

This section is aimed at showing how all the elements of the reference model proposed in the previous section are needed in order to support the object-level coordination requirements of two different examples of groupware applications. In this way, the section will not only give a better explanation of the reference model, but also will provide indications on the model significance and potential applicability to different collaborative applications (a key prerequisite for reusability).

Two groupware applications have been chosen: an asynchronous share document repository (examples of real applications of this type are Basic Support for Collaborative Work – BSCW [18], Synergeia [12], .LRN [16], ...) and a synchronous collaborative application for the creation of conceptual maps (examples of real applications of this type are CMapTool [13], FreeMind [7]...).

In Table 3 and Table 4 each type of tool will be briefly described indicating potential "User-requested actions". Then, examples of "Coordination configuration" information will be provided. Also, some examples of "Coordination policies" will be described in order to show how the "Coordination functions" would take their decisions and how "Coordination actions", "Awareness actions", and "Session modifications" might be triggered. Those policy examples simply intend to cover cases in which all reference model elements are involved at least once.

Table 3. **Example #1: an asynchronous shared document repository**

Simplified Tool description
The application consists of a web-based repository of documents organized in folders. Users may upload/remove/view documents in each folder. Also, they may create/modify/remove folders. Users may add/modify/remove notes on each document.
Example "Coordination configuration"
• Users: John, and Bob • Shared objects types, examples instances, and available actions   ○ Folders (modify, remove, upload document):

Folder1
    - Documents (view, remove, modify description, add note): Doc1, Doc2
    - Notes (view, modify, remove): N1, N2, N3, N4
- Access control list
    - Folder1: both John and Bob can execute any available action on F1
    - Doc1:
        - John can view, add note.
        - Bob can view, remove, modify description, add note.
    - Doc2:
        - Bob can view, add note.
        - John can view, remove, modify description, add note.
    - Note1, Note2, Note3, Note4: both John and Bob can execute any available action on Note1, Note2, Note3, Note4.

Example "Coordination policies"

- Shared object access policy #1
    - "A user-requested action on a shared object is not allowed if the user does not have access rights for that action on that object unless the user has "administration" rights".
- Shared object access policy #2
    - "A user cannot remove a note if it has not been previously read by all users"
- Conflict resolution policy #1
    - "A user cannot remove a document if that same document is being read by another user, unless the user has "administration" rights".

Example coordination decisions and associated awareness actions, coordination actions and session modifications

- Scenario #1: "John requests Doc1 deletion"
    - Decision (based on shared object access policy #1): although John does not have access rights for that operation on Doc1, the "Shared-object access control" function permits the action because:
        - The "Role Management" block informs that John is playing the "Administration" role.
        - The "Coordination state" informs that there are no more users viewing Doc1 (according to conflict resolution policy #1).
    - Awareness action: the Awareness building block is informed on the decisions. That block sends an e-mail to all users indicating that John has deleted Doc1
    - Coordination action: none
    - Session modification: none
- Scenario #2: "Bob requests Note1 deletion"
    - Decision (based on shared object access policy #2): although Bob has rights for that action on that object, the "Coordination state" informs that John has not read that note yet. Therefore, the action is not permitted.
    - Awareness action: none
    - Coordination action: none
    - Session modification none
- Scenario #3: "John request Note1 viewing"
    - Decision (based on shared object access policy #1): John has rights for that action on that object so the action is permitted.
    - Awareness action: none
    - Coordination action: the "Coordination State" is updated indicating that the note has been read by John.

Table 4. **Example #2: a synchronous collaborative conceptual map application**

Simplified Tool description

The application provides a shared graphical panel (the conceptual map) in which users draw concepts and links among them synchronously. Concepts and links may be repositioned along the map.

Example "Coordination configuration"

- Users: Laura, and Daisy
- Shared objects types, examples instances, and available actions
    - Conceptual map (remove, create concept, create link): Map1
    - Concept (remove, add description, modify description, move, add link): Concept1
    - Link (add description, modify description, remove)
- Access control list
    - Map1: both Laura and Daisy can execute any available action on M1
    - Concept1:
        - Laura can remove, add description, modify description, move, add link.
        - Daisy can move, add link.

Example "Coordination policies"

- Shared object access policy #1
    - "Only a user with the TOKEN (i.e. turn) can execute an action on a shared object. After having requested an action, the user loses the TOKEN, which is assigned to another user".
- Shared object access policy #2
    - "A user-requested action on a shared object is not allowed if the user does not have access rights for that action on that object unless the user has "administration" rights".
- Shared object access policy #3
    - "After six times forbidding the same action on the same object to the same user, that user will be forced to leave the collaborative session."

Example coordination decisions and associated awareness actions, coordination actions and session modifications

- Scenario #1: "Daisy requests removing Concept1"
    - Decision: Daisy has the TOKEN (according to information provided by "Coordination State") but she does not have access rights on the shared object for that action so the request is denied (according to shared object access policy #2) taking also into account the "Role Management" block informs that she is not playing the "administrator" role.
    - Coordination action: the "Coordination State" is updated as Laura will now have the TOKEN (according to shared object access policy #1).
    - Awareness action: the Awareness building block is informed on the decisions. That block changes the color of the icon representing Daisy and Laura to let them know that now Laura has the TOKEN.
    - Session modification: Daisy leaves the session (according to shared object access policy #3) because "Coordination State" informs that this is the sixth time that Daisy requests the same denied action.

6. Discussion

As explained in section 4, both the functional scope of the proposed reference model, as well as the potential coupling points identified by it, imply an added-value for achieving the reusability of object-level coordination support in groupware applications.

Nevertheless, the proposed model is a bottom-up conceptual construction whose utility might be initially considered as simply a help for understanding coordination processes (as shown by the examples of section 5). Therefore, this section tries to discuss two additional potential uses of the proposed model that are being exploited by the authors: the model as a comparison/evaluation framework; and the model as a starting point for the development/redesign of reusable object-level coordination support for groupware applications.

6.1. Use #1: the model as a comparison/ evaluation framework

Table 5 details which elements of the model are supported by the proposals analyzed in section 3.

According to that table, and using the model proposed in the paper as reference framework, it can be said, for instance, that DCWPL, although able to resolve coordination conflicts, is not capable of enforcing coordination policies involving session modifications (capability that is supported by COCA).

Another example: Intermezzo, as it can be observed from its support to "Coordination Actions", is not coupled to other groupware building blocks (it only communicates with the groupware application itself). That might be understood as a positive aspect for improving reusability, but, also looking at the table, that decoupling implies that Intermezzo only supports coordination based on stateless access policies (probably enough for applications based on file sharing but insufficient for more complex ways of collaboration). This is an example of the trade-off between coupling and functional significance for fostering reusability.

These are just two examples of the kind of comparison/evaluation that can be done using the proposed model. Of course, a finer-grained comparison/evaluation would imply a more detailed model. But the model is still valid for getting an overall picture of the coordination capabilities of an analyzed proposal as well as for discussing the trade-off derived from the support of certain model elements.

Table 5. **Reference model elements supported by the proposals analysed in section 3**

	Dcwpl [1]	Coca [15]	Intermezzo [5]	[25]	[3]
Coordination Configuration					
User list	YES	YES	YES	YES	YESº
Share Object list	YES	YES	YES	YES	YES
Access Control list	YES	YES	YES	YES	NO
Coordination Policies					
Access policies	YES	YES	YES	YES	YES
Conflict resolution policies	YES	NO	NO	NO	YES
Coordination State	YES	YES	NO	YES	YES
Coordinator Actions					
Action-execution decisions	YES	YES	YES	YES	YES
Awareness actions	YES	YES	NO	NO	NO
Coordination Actions	NO	YES	NO	NO	NO
Session modifications	YES	YES	NO	NO	YES

6.2. Use #2: the model as the starting point for development/redesign

As mentioned in section 4, the proposed reference model tries to collect functional and architectural requirements to be taken into account for achieving reusability. Therefore, developers of coordination software solutions can use the model as a starting point for deciding which of those requirements are going to be incorporated in their designs and the derived implications (e.g. not supporting configurability would make the solution unsuitable for applications to be used in changing coordination scenarios). Also, the associated architectural requirements (e.g. incorporating the "Coordination State" for supporting policies that take into account the coordination "history") may help the developers to anticipate problems for achieving reusability (e.g. supporting the "Coordination State" might add a coupling point between the coordination support and the groupware application). The identification of potential coupling points is probably one of the most interesting aspects in this sense. Actually, the authors have proposed in [19] a solution for avoiding one of the coupling points identified in the reference model: that between the application and the object-level coordination support when requesting actions and executing them (the two

arrows between these two blocks shown in Figure 3). In that case the solution consisted of the application of the so-called "Command" software pattern. This is an example of how the model is just the starting point for achieving reusability: bad decisions during subsequent phases of the software development process may still preclude reusability.

7. Conclusions and Future Work

This paper, using a bottom-up approach, has proposed a high-level reference model for reusable object-level coordination support in groupware applications. Reusability is favored by collecting in the model a set of functional requirements that are significant from a domain point of view. Also, the model identifies potential coupling points between the object-level coordination support and other building blocks. The identification of those coupling points is intended to allow developers to anticipate reusability problems and to plan strategies for their solving.

The paper has also discussed on two potential uses of the proposed model: as a comparison/evaluation framework for assessing the functional scope and reusability level of existing and yet-to-come object-level coordination proposals; and, as the set of functional and architectural requirements that constitutes a starting point for developing (or redesigning) potentially reusable object-level coordination building blocks.

The presented proposal is the first step in a research work aimed at achieving reusable coordination support. Nevertheless, the experience of the authors working on particular aspects of the coordination reusability problem [19] indicates that this first step is very important because understanding of the overall coordination problem is crucial so as to assess the significance of partial proposals and their potential implications.

Therefore, future work includes: verifying the applicability of the model to the description of the other existing approaches to coordination support (and refining the model consequently); enhancing authors' proposals (specially [19]) according to the requirements of the model; identifying alternatives for supporting the model elements (the authors are particularly interested in the implications of the different proposals for formalizing and interpreting policies); and, exploring other coordination features, specially dynamism (the possibility of changing policies automatically depending on the collaboration context).

8. Acknowledgements

This work has been partially funded by the EU e-Learning project EAC/61/03/GR009, EU Kaleidoscope NoE FP6-2002-IST-507838, Spanish Ministry of Education and Science project TSI2005-08225-C07-04 and Autonomous Government of Castilla y León, Spain, projects VA009A05, UV46/04 and UV31/04. The authors would also like to thank the EMIC/GSIC research group at the University of Valladolid for their support and ideas.

9. References

[1] Cortes, M. and Mishra, P., "DCWPL: A Programming Language for Describing Collaborative Work", *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work (CSCW'96)*, Cambridge, MA, USA, 1996.

[2] Dewan, P., "Architectures for Collaborative Applications", *Trends in Software: Computer Supported Collaborative Work*, vol. 7, 1998, pp. 165-194.

[3] Dommel, H. P. and Garcia-Luna-Aceves, J. J., "A coordination framework and architecture for internet groupware", *Network and Computer Applications*, vol. 23, no. 8, 2000, pp. 401-427.

[4] Dourish, P. and Belloti, V., "Awareness and Coordination in Shared Workspaces", *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work (CSCW'92)*, Toronto, Canada, Oct. 1992.

[5] Edwards, W. K., "Policies and Roles in Collaborative Applications", *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work (CSCW'96)*, Cambridge, MA, USA, Nov. 1996.

[6] Ellis, C. and Wainer, J., "A Conceptual Model of Groupware", *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (CSCW'94)*, Chapel Hill, NC, July 1994.

[7] Freemind, http://sourceforge.net/projects/freemind/, last visited: June 2006.

[8] García López, P. and Gómez-Skarmeta, A. F., "ANTS Framework for Cooperative Work Environments", *IEEE Computer*, 2003, pp. 2-8.

[9] Grundy, J. C. and Hosking, J. G., "Engineering plug-in software components to support collaborative work", *Software, Practice and Experience*, vol. 32, no. 10, 2002, pp. 983-1013.

[10] Gutwin, C., Stark, G., and Greenberg, S., "Support for Workspace Awareness in Educational Groupware", *Proceedings of 1st International Conference on Computer*

Support for Collaborative Learning (CSCL'95), Bloomington, IN, Oct. 1995.

[11] Gutwin, C., Stark, G., and Greenberg, S., "The Effects of Workspace Awareness Support on the Usability of Real-Time Distributed Groupware", *ACM Transactions on Computer-Human Interaction*, vol. 6, no. 3, 1999, pp. 243-281.

[12] Synergeia, http://bscl.fit.fraunhofer.de/, last visited: June 2006.

[13] CMapTools, http://cmap.ihmc.us/, last visited: June 2006.

[14] Laurillau, Y. and Nigay, L., "Clover Architecture for Groupware", *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work (CSCW'02)*, Louisiana, USA, Nov. 2002.

[15] Li, D. and Muntz, R. R., "COCA: Collaborative Objects Coordination Architecture", *Proceedings of ACM CSCW'98*, Seattle, Nov. 1998.

[16] .LRN Home, http://www.dotlrn.org/, last visited: June 2006.

[17] Morris, M. R., Ryall, K., Shen, C., Forlines, C., and Vernier, F., "Beyond "Social Protocols": Multi-user Coordination Policies for Co-located Groupware", *Proceedings of ACM CSCW'04*, Chicago, Illinois, Nov. 2004.

[18] BSCW, http://bscw.fit.fraunhofer.de/, last visited: May 2006.

[19] Orozco, P., Asensio-Perez, J. I., Dimitriadis, Y., García López, P., and Pairot, C., "A Decoupled Architecture for Action-Oriented Coordination and Awareness Management in CSCL/W Frameworks", *Proceedings of 10th International Workshop on Groupware (CRIWG'04)*, San Carlos, Costa Rica, Sept. 2004.

[20] Roseman, M. and Greenberg, S., "Building Real Time Groupware with Groupkit, A Groupware Toolkit", *Transactions on Computer Human Interaction*, vol. 3, no. 1, Mar. 1996, pp. 66-106.

[21] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E., "Role-Based Access Control Models", *IEEE Computer*, vol. 29, no. 2, 1996, pp. 38-47.

[22] Sparling, M., "Lessons Learned-Through Six Years of Component-Based Development", *Communications of the ACM Journal*, vol. 43, no. 10, Oct. 2000, pp. 47-53.

[23] Java Shared Data Toolkit, http://java.sun.com/jsdt, last visited: June 2006.

[24] ter Hofte, H., "Working apart together: Foundations for Component Groupware", *Telematic Insituut Fundamental Research Series*, vol. 1, 1998, pp. 288.

[25] Tripathi, A., Ahmed, T., Kumar, R., and Jaman, S., "A coordination model for secure collaboration", *Process Coordination and Ubiquitous Computing*, 2002, pp. 1-20.