In In D. Georgakopoulos, W. Prinz, and A. Wolf (eds.) *Proceedings of ACM Conference on Work Activities Coordination and Collaboration (WACC'99),* ACM Press, Feb. 22-25, 1999, San Francisco

Interoperability as a Means of Articulation Work

Carla Simone[1], **Gloria Mark**[2], **and Dario Giubbilei**[1]

[1]Department of Computer Science
University of Torino
Corso Svizzera 185, 10149 Torino, Italy
simone@di.unito.it

[2]GMD-FIT
Schloß Birlinghoven
D-53754 St. Augustin, Germany
gloria.mark@gmd.de

ABSTRACT

The interoperability of systems to support cooperative work requires moving beyond purely technical issues; it also concerns the means and practices that users adopt to articulate their cooperative activities. Articulation has to be supported by a technology which focuses on this higher level of interoperability. This claim is motivated by observing the articulation process of users in real cooperative work practice. Based on this study, the functionality for this technology was designed to help users reconcile different handling and perspectives on shared objects in their cooperative work. The paper presents the architecture of an application infrastructure centered on the identified interoperability issues and focuses on the design of a specialized module, called reconciler, which provides the above functionality. The current state of its implementation together with identifying open research problems conclude the paper.

Keywords

Interoperability, cooperative work, awareness, architectures, groupware conventions

1 INTRODUCTION

In the last decade, interoperability has become a critical topic in the development of cooperative applications due to the necessity to build systems that support a wide range of cooperation modes. Currently interoperability issues are common in the development of almost all types of systems and significant results have been achieved both at the level of applications and underlying infrastructure. However, especially when cooperative work is to be supported, the theme of interoperability reaches an additional level of complexity. In fact, the inherent distributed nature of cooperative work, as discussed e.g., in Simone and Schmidt [16], implies that interoperability must be taken into account not only at the level of the infrastructure or between loosely-coupled applications, but also within the same application supporting tight cooperative activities. This is true irrespective of whether the actors cooperate remotely.

The distributed nature of (tight) cooperative work implies that system design must account for the local malleability of the constituent parts of the application to guarantee autonomy, as well as suitable support for preserving their mutual alignment. Because changes are made locally, the alignment process is also a distributed activity. In turn, the local malleability of cooperative applications is made easier by a clean separation between *articulation work*, i.e., the work devoted to activity coordination and *coordinated work*, i.e., the work devoted to their articulated execution in the target domain. This distinction has been emphasized in the framework of both CSCW [13], and more abstract approaches to coordination models and languages as shown by an increasing research interest in these topics. Notice that the distinction between articulation work and coordinated work provides two decoupled, although connected, perspectives on cooperative work, and excludes any hierarchical relationship between the two perspectives. The above distinction implies that *interoperability has to be considered and managed in relation to articulation work as well as coordinated work*, in a framework which guarantees a smooth transition between the two.

While the second type of interoperability can be viewed in relation to more traditional IS issues, or more recently, within the so-called shared object approaches (e.g., see CORBA), the first type of interoperability is a quite open domain and requires a deep understanding of the nature of articulation work**.** The CSCW literature is rich with empirical studies showing that physically collocated actors achieve the mutual alignment necessary for their cooperation by using an unpredictable mixture of explicit and implicit conventions, according to their current needs. When articulation cannot rely on the information and cues readily available in face-to-face cooperation, then the design of a suitable support for facilitating and preserving mutual alignment becomes a crucial part of the articulation work and therefore of the application development supporting it. That is, the infrastructure supporting the application development must be equipped with suitable tools to allow both types of interoperability.

The above claims identify a challenging research area in which this paper intends to contribute by presenting requirements and design issues of a kind of cooperative system on the basis of empirical results from actual

cooperative work. We employ a bottom-up approach beginning from real, special situations and extrapolate a general solution applicable to a class of similar situations.

The paper is organized as follows. In the next section we describe experiences of real users of a groupware system which motivated us to think about interoperability in a new way: users had developed individual and incongruent conventions that adversely affected their cooperation. In section 3 we derive requirements for how a technical solution can promote the reconciliation of different perspectives. We present the architecture in section 4, its current implementation in section 5, and conclusions and summary in section 6.

2 INCONGRUENT VIEWS ON SHARED OBJECTS

The empirical research was conducted in the PoliTeam project [9, 12], whose aim is to supplement paper work processes with electronic work processes as German government ministries relocate from Bonn to the new capital of Berlin. The results reported here are part of a larger study of groupware use in real practice (see [10]).

Two different groups in a German ministry worked cooperatively using shared workspaces in the PoliTeam system: typists in a central typing pool and members of a ministry unit. The writing office members type electronic versions of documents for the unit members, whose job is to support the Minister through activities such as speech-writing, information dissemination, or answering citizens' queries. The two groups can be distinguished by differences in jobs, tasks, education levels, career path orientations, salaries, and computer experience, among other differences. The two groups were also spatially distributed in the same Ministry building.

An evaluation of the system usage provided a number of examples of how the two user groups, with different perspectives on handling shared objects, established unique views of the same shared workspace. By "view", we refer to a very broad description concerning the organization of a shared workspace: the information it contains, how it is visually displayed, and how it is internally organized. In the following, we have chosen three different aspects of shared object use to describe, which illustrate how different perspectives are used and how they influence cooperation.

1) *Personal information structuring*. The writing office members organize documents according to a scheme which is logical for their work process: documents are sorted by the name of the document owner and date of creation, in a two-level hierarchy. In contrast, the unit members structure their documents according to their work processes, in rather deep multi-level structures, an organization which is logical for them. To the typists, accessing a document by the owner makes much more sense to them than accessing a document by the subject which has little meaning to them. To the unit members, accessing a document by its subject has semantic meaning for them, e.g. a speech on a senior citizen initiative. The dates of the documents have less meaning for them, since they may work on multiple projects within the same time frame.

Unit members report that it is an overhead to find documents among the vast array of information in the shared workspace because the system supported only one view: the typists'. Further, the method that is used in exchanging documents between the unit members is basically ad hoc. It is an overhead for both typists and unit members to communicate about which folder finished documents must be placed into. For some unit members, communication is also an overhead when they exchange documents among themselves. For example, two unit members often exchange large numbers of documents. One member stores them in the same folder, but her cooperating partner uses the documents for different work processes as well and stores them by distributing them in different folders.

2) *Naming of documents.* The typists worked electronically before PoliTeam was introduced, and during this time, they established a convention for naming documents, using document type, unit member name, and date. Although after the system introduction a convention was set for all users to use this naming system for the shared documents, it remains confusing for the unit members. Again, their preference is to name documents according to the subject, which relates to their work processes. In fact, users began using the subject field within the document to elaborate the names, which led to even more different naming references. One user even used fantasy names, which were unintelligible to the others. A further complication arose since users also can display information by lists or by icons. When icons are used, document names are cut off, and only a cryptic name is visible. As a result, the users began to add their own keywords to understand the content of the document. Thus, the preference for different views led to a different semantic referral of the shared documents.

3) *Awareness information.* Each employee, depending on their work role, has a need for a different form of awareness information. For the typist, it is a benefit that a message is sent to the owners informing them when she places finished documents in the shared folder. For the unit members, automatic outgoing messages have less value, but notifications of finished texts have benefits for them. For the unit leader, he has a distinct requirement in his management function; he would like to ascertain who had made changes to, and who possesses a shared document. Some unit members would like to have information about the events and activities that occur in parts of the shared workspace; other unit members see it as an information overload.

Thus, awareness serves different purposes for different workspace members, depending on their task and perspective. Problems in interaction can arise when users expect that others have the same awareness information as themselves; i.e. when the awareness information is not symmetric. And users sometimes behave as though it is

symmetric, believing that another user will receive notification when they place a document in the workspace, when the user in fact, has constructed an awareness profile that only reports very general events.

3 RECONCILING INTRAGROUP AND INTERGROUP COOPERATION

The above section provides evidence that through *intragroup* cooperation, i.e. work within a long-established group, members of the same group develop congruent interpretations for some objects. Articulation work is about establishing conventions that are materialized in some artifact. More specifically, in the above scenario, artifacts are classification schemes which not only organize information according to specific relations but also govern the action on this information. Action can refer to the manipulation and retrieval of information, as well as sending awareness information. As discussed in Schmidt and Simone [13], classification schemes can be considered as a special class of coordination mechanisms (CM). While other artifacts, like the representation of the flow of activities, are widely recognized as means for the articulation of activities, classification schemes are in general not considered as such in the design of the technological support of cooperative work. As an exception, the use of hypertext technology is used as a support for argumentation, although this is a special kind of application.

3.1 Coordination mechanisms as explicit artifacts

Classification schemes are of course used in current technology; for example, they correspond to conceptual schemes in database applications. However in this case they are fully embedded, i.e. "wired" in the application. In applications based on shared information spaces, they are often left unspecified, or just implicitly specified through conventions, e.g., by organizing objects into certain containers or by just arranging them according to some spatial scheme carrying a conventional semantics. In any case, either wired or implicit, those classification schemes can hardly serve coordination purposes; the first solution hinders flexibility while the second one does not make the conventions manifest, and therefore inhibits the development of a shared understanding among the users. Users' individual and unique patterns of technology usage can become ingrained as we have seen, hindering the transactions of shared objects between partners.

What we propose here (see figure 1) is to give classification schemes, and more generally to all sorts of coordination mechanisms, a status of explicit artifact, which makes conventions visible and therefore more easily modifiable by being decoupled from the field of work. This approach is consistent with the idea of "factoring out control" from information systems [5]. However, factoring out per se is not sufficient; what has been factored out must be made available and modifiable by the users.

On the other hand, the contrasting case is when cooperation occurs *intergroup*, i.e. between members of heterogeneous groups; here perspectives on shared objects can be quite diverse. The boundary objects shared by heterogeneous groups are vehicles for communication about work and must be managed [17]. As found with the PoliTeam users, the difficulty in reconciling different perspectives is that they are generally logical for their users' roles and tasks.

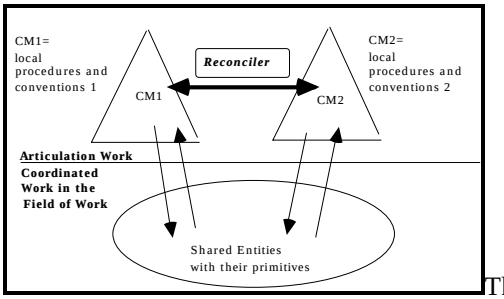

Figure 1. A logical framework supporting articulation work across coordination mechanisms

These results are likely to be a general problem with cooperative systems. Hence, the technology supporting intergroup cooperation has to deal with the tension between maintaining the needed individual perspective and a shared meaning to interpret boundary objects and actions characterizing intergroup cooperation. The technology has to provide a means to manage this tension, that is, a kind of interface whose goal is to aid actors in reconciling differences in their perspectives, which may be reflected in different views and organizations of a shared workspace. We call this special kind of interface a *reconciler* (see figure 2). The reconciler can be distinguished from personalized groupware interfaces which serve to meet the requirements of unique working styles of group members or subgroups, by tailoring an interface to an individual perspective, e.g. [18]. In contrast, a reconciler is intended to provide relevant information to users about how other users are handling shared objects in order to promote the convergence of perspectives. Another approach to handling individual and group working styles was taken recently by Gutwin and Greenberg [7] who promote flexibility for individuals and groups through split views, symbolic manipulation techniques, and gestural communication to align different representations. Their approach is less active in promoting an alignment of views as compared to our proposed reconciler interface.

3.2 Requirements for aligning incongruent views

Cooperating partners switch back and forth between individual and shared work. An important requirement of such a reconciler is that it must be flexible to adapt to these dynamic needs. In some cases, it should enable users to retain individual views; in other cases, it should identify incongruencies to promote the merging of perspectives. In the former case, the reconciler saves overhead by working "behind the scenes", in the latter case, it actively supports an articulation process among the group members. We need to consider that a learning curve exists with

groupware and the group needs time to learn the system supporting their cooperative activities [10]. Here, learning implies the understanding of both system functionality and work practices developing from, and changed by, the system usage. A reconciler interface can promote learning within and across groups about their work practices since users can cooperatively process the contents of their work: interpreting, modifying, and aligning them. Of course, the effectiveness of a reconciler depends on both its contents and its active behavior. Figure 2 gives a schematic view of the functionality we envisage using.

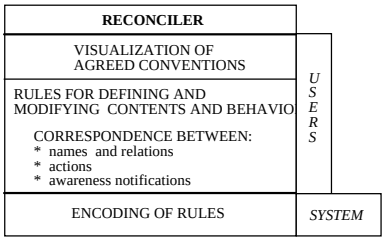

Figure 2. A schematic view of the functionality of a reconciler.

First of all, conventions are not always (or at least not immediately) "algorithmic", that is, directly translatable in a system behavior. Then, the reconciler contains a section which serves as a shared space where users can record, possibly in a free format, such type of conventions. This section can be accessed intentionally by users in order to become aware of, interpret, and apply established conventions as the need arises. In addition, users can send this, or appropriate sections of it, to others as reminders when conventions may be violated. Secondly, when conventions become algorithmic (possibly due to the above mentioned learning process) then users can decide to delegate their management to the reconciler. Then the reconciler should support the definition of the rules to reconcile the different perspectives and the associated active behavior, that is, the correspondence between names and relations, between actions and the related awareness behavior in the two perspectives. Such rules, once defined, can be incorporated and presented to the users in the above shared space. Finally, the third section contains the code implementing the behavior delegated to the reconciler; such code is automatically generated from rule definition.

The scenario considered in section 2 provides an example of a correspondence between incongruent actions: sending a document to a unit member corresponds to depositing it actually, to a specified folder related to him/her. Moreover, this induces a correspondence between the awareness information of actions in the source and target perspectives; the awareness information generated by a typist is to be interpreted so as to send a signal to the unit member (recipient) when the document is deposited in his/her specified folder. Rules can define also a correspondence of different ways to identify objects in the two perspectives. The two different ways of organizing documents described in section 2 allows a unit member to ask a typist, "send all documents in folder X to John". Those documents can be identified and contextualized in the typists' workspace using their system of organization, if the appropriate rule is defined. The same approach can solve the problem of different naming conventions, e.g. task/work process names used by unit members vs. owner-id/date names used by typists.

To sum up, the reconciler dynamically contains the relevant information one cooperating partner needs to know about another partner's system usage. All the rest of the information can be hidden, or accessed through other means. In addition, the rules can also define how and when the reconciler should convey this information to the users, by defining subscribe and publish policies. Then the reconciler is able to make users aware of the fact that awareness information is not always reciprocal, by visualizing and comparing the above policies. Moreover, suitable policies can make users aware of the extent to which their changes and related actions, as a result of their view, impact other users, and vice versa, e.g., changing access rights to a shared folder affects all who are cooperating with it. Awareness information can be modulated in strength and mode of support depending on the relevance of the event triggering it. The reconciler can thus serve as a catalyst for the group to identify congruency problems. There will of course be cases where the process of forming congruencies is too complex for the reconciler to handle; the group can then use other strategies, such as social methods using negotiation.

4 INTEROPERABILITY AT DIFFERENT SEMANTIC LEVELS

The theme of interoperability and individual perspectives is not new, at the technological as well as the conceptual levels. For example, the relational database technology supports the notion of view to select items according to specific user needs. On the other hand, interoperability is the main goal of the CORBA standard [1] which in addition provides services and facilities that make the compliant infrastructure at the same time very rich and yet still not sufficient for our purposes, as we will see later on.

At the conceptual level, interoperability and individual perspectives are considered in the frameworks of cooperative IS and agent-based systems. It is beyond the scope of this paper to account for the rich set of proposals originated from these frameworks. It is sufficient to notice that their main application domain is the design of systems where cooperation is among artificial entities: the so-called federated databases in the first case (see, e.g. [11]), and the artificial agents of Distributed AI in the second one (see, e.g. [4]). Then, the goal of the proposed approaches is to define, on the basis of the knowledge of a human designer, ways to let cooperation happen, in the best possible way, without each component being aware of the problems cooperation raises in the other components. Hence, the techniques proposed by both the technological and conceptual frameworks, although quite sophisticated and inspiring, cannot be directly used in the design of a

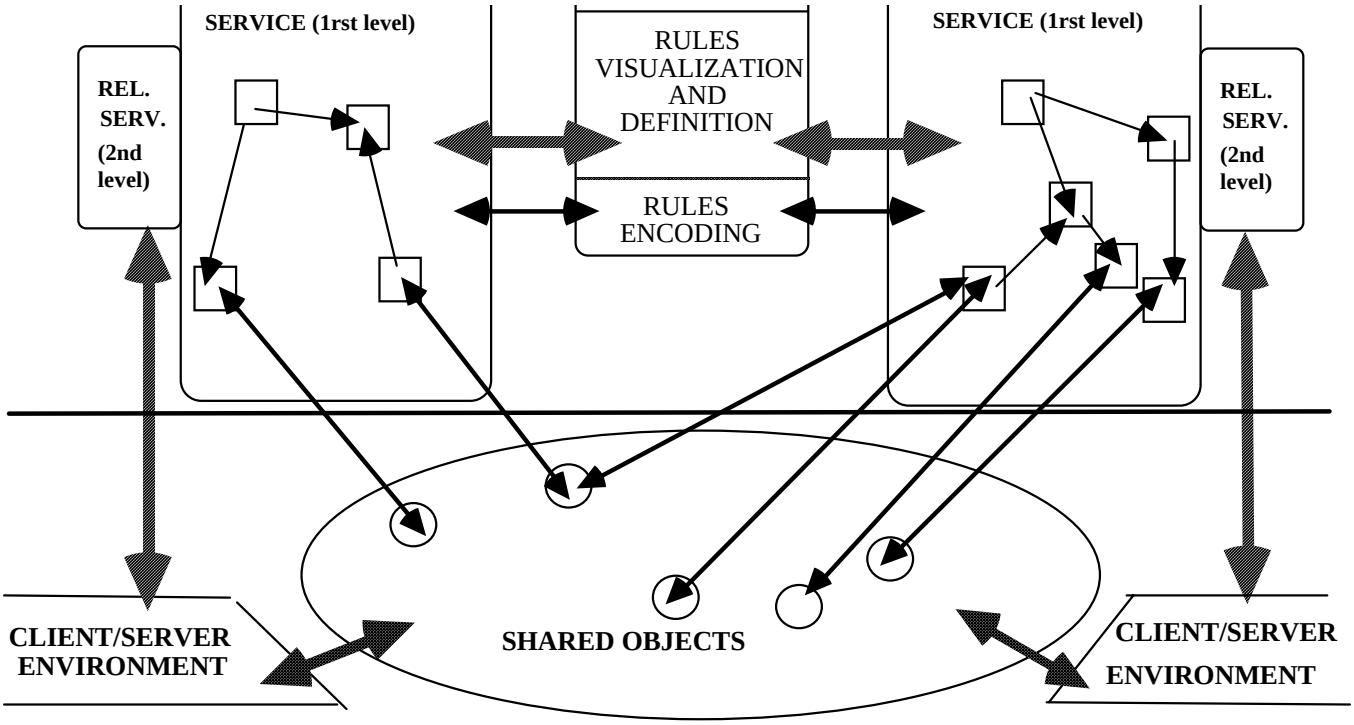

Figure 3. An architecture of an infrastructure supporting articulation work and interoperability across CMs.

CSCW system, and specifically of the reconciler, for a combination of different reasons: the lack of flexibility and visibility by the users, the inappropriate semantic level, or finally the lack of support for mutual and reciprocal awareness. All these requirements can be fulfilled just by considering a comprehensive framework where all of them can be dealt with in an integrated way to support articulation work. This "ideal" architecture is described in figure 3 as an infrastructure for cooperative CSCW applications, which reflects the logical framework illustrated in figure 1. This architecture deals with the above mentioned ways to look at interoperability. Therefore, we will adopt CORBA's lexicon to describe it as CORBA as a common point of reference when speaking about interoperability, at the technological level. Let's consider the various components in turn (from the bottom) together with their mutual relations.

The two (heterogeneous) Client/Server environments interact through the Shared Objects, according to CORBA's approach based on IDL and the supporting services at the lower ISO/OSI levels (thick short double-arrows). So, for our purposes, it is sufficient to think of the shared objects as specified according to the notion of CORBA's interface, in terms of attributes and operations, to guarantee the interoperability, say, at the system level. Client/Server environments and Shared Objects correspond to the space of field of work of figure 1. Going up, the next components are the two Contexts defined through a CORBA-like Relationship Service which allows the definition of proxy objects for the shared ones (roles and nodes in CORBA's terms) together with their relationships. The two levels of the service refer to the definition and manipulation of the resulting graphs, respectively. The latter materialize the protocols and classification schemes (all together, Coordination Mechanisms) of figure 1. Hence, the architecture preserves the distinction between articulation work and coordinated work in two separated modules which are, consequently, also not hierarchical.

Moreover, the way in which the Relationship Service is interpreted in our architecture goes far beyond its standard use within CORBA. In fact, there it is mainly reduced to a feature allowing for (minor) user interface adaptability. On the contrary, here it plays the central role to let the related graphs become visible to and manipulable by the application users (vertical thick shaded arrows). They are the user access point to the Shared Objects. The Relationship Service has thus to maintain a connection between these manipulations and the related ones at the Shared Objects level (thin double-arrows).

At the top-most level, we have the reconciler to manage how the two Contexts interact. The term Context is used in accordance with CORBA's terminology because Contexts define the distributed space where the Naming Service operates. Obviously, the service provided by the reconciler is by far more articulated than CORBA's Naming Service. However, we keep the same name as the former can be seen as a generalization of the latter, as discussed in the next section. According to the schema of figure 2, the reconciler has both an information content and an (autonomous) behavior. Hence, it is an object, specifically the privileged shared object that allows for the interoperation of the two Contexts as it functions as their mediator. Again, it can be realized in terms of a CORBA-like interface operating at the semantic level of articulation

work. This basically implies two things: first, it has to be explicitly programmed, as part of the whole cooperative application, to incorporate the conventions supporting intergroup cooperation; secondly, it has to be implemented so as to be accessible and malleable by the users.

We conclude this section by clarifying why the architecture based on the 'pure' CORBA standard (as well as any analogous solution) is not adequate, per se, to support the above ideal architecture. The answer is that CORBA's features (IDL and Services) are at the programming level and not at the level of the articulation work. Therefore they are not adequate and usable by users for this purpose. Moreover, the assumption underlying their design is typical of Software Engineering approaches, that is, to base interoperability on information abstraction and hiding. On the contrary, the field studies show that interoperability as an articulation support has to be based on a flexible non-obtrusiveness (see section 2) and on the promotion of mutual awareness, which in turn, are based on the accessibility and malleability of the related linguistic constructs.

5 IMPLEMENTING THE RECONCILER

Strictly speaking, the reconciler can be viewed as an autonomous component that can be constructed independently of different applications supporting articulation work, and used to mediated among the related coordination mechanisms. However, its effectiveness depends greatly on how the latter are implemented in terms of accessibility and malleability. For this reason we prefer to present briefly how our previous work defined a framework to construct mechanisms of this sort. In fact, some of the available features are suitable for the implementation of a reconciler naturally interacting with them. These features can be seen as a kind of services that any other approach has to 'simulate' in order to incorporate a reconciler as a mediating component.

5.1 Supporting intragroup cooperation

The requirement of accessibility and malleability of coordination mechanisms and, by consequence, all their constituent components, is the main concern of a research effort started within the COMIC European project. This led to the definition and implementation of two prototypes: ABACO, implementing a notation (called, Ariadne) for the definition and the manipulation of coordination mechanisms [15], and the AW-Manager supporting the promotion of awareness [14]. The definition of Ariadne is based on empirical studies of how actors articulate their cooperative work. We derived from them a set of basic categories together with their relationships to express the mechanisms actors define to obtain articulation [13]. Among others, the most relevant category here is the Active Artifact. Its role in articulation work is to materialize and to make visible to the cooperative actors the state of the distributed procedures and conventions incorporated in the coordination mechanism. The Active Artifact, through its information structure and communication behavior, is one of the sources to promote mutual awareness among actors. Being at the semantic level of articulation work, the basic categories together with their relationships are immediately usable by the actors who can represent procedure and conventions in a quite flexible way, according to their culture and habits, by defining and modifying combinations of categories and relationships in a practically open-ended set of modeling approaches.

Due to the central role of awareness, this functionality has been specifically considered and implemented in a dedicated component, the AW-Manager, again characterized by a set of linguistic features (called Awareness Language, AL) that can be composed to realize awareness capabilities. The latter aim at propagating awareness information generated by specific components to the other ones by taking into account their spatial location within the cooperative application, both at the physical and logical levels. This idea was first proposed in [3]. To guarantee the high modularity required by malleability and interoperability, Ariadne and AW-Manager are implemented as a multi-agent architecture. AL is the agent language of the latter, while ABACO, the agent-based abstract machine [6] defining Ariadne's operational semantics, uses an agent language called (not surprisingly) Interoperability Language (IL). Its basic primitives allow for the coordination (*tell/ask*), the subscription (*activate*), and the malleability (*overwrite*) of mechanism behaviors.

In relation to the above ideal architecture, the two mentioned prototypes cover the definition and activation of individual coordination mechanisms able to support *intra-mechanism* coordinating protocols and awareness capabilities (corresponding to the Contexts of figure 3). The *system level interoperability* is solved by having homogeneous JAVA™ client/server environments. Moreover, IL and AL are implemented in JAVA™ too by exploiting RMI capabilities.

5.2 Supporting intergroup cooperation

Supporting intergroup cooperation requires defining how the related mechanisms interact, that is, an *inter-mechanism interoperability*. An initial inter-mechanism interoperability was achieved by using the same primitives of IL and AL across coordination mechanisms. This solution is inadequate for the scope of a reconciler as it focuses only on a standardized inter-mechanism communication and disregards its contents and pragmatics. However, since Ariadne allows one to construct a coordination mechanism as the composition of more elemental ones, and associate to each its Active Artifact, it is natural to incorporate a reconciler in a compound coordination mechanism. In fact, the reconciler can be viewed as the Active Artifact characterizing a compound coordination mechanism. In this way, one can use the features of Ariadne to define it and inherit the related capabilities in terms of malleability and visibility. Moreover, the reconciler's communicative behavior is directly implemented through AL and IL. What is still

needed is a way to let this Active Artifact represent the information contents shown in figure 2 and the related action towards the reconciled mechanisms.

We are currently implementing the reconciler according to a bottom-up strategy which considers the special cases we have identified in the described field study and in our experience in our own work environments. This means that the current solution is not complete; rather it can be seen as a test of feasibility for an incremental construction of an articulation support based on interoperability. The bottom-up approach is crucial since it must support a learning process which also occurs bottom-up. Once new cases are recognized, the related rules are added and a consistency check is made in order to avoid contradictions; the same happens in actual work practice when ambiguities are to be kept under control. This is fully in agreement with the incremental approach to the construction of coordination mechanisms characterizing both Ariadne and the AW-Manager.

The basic idea is to construct a user interface guiding the cooperative actors in defining the correspondences (see figure 2) on which the reconciler functionality is based. Then, according to the above requirements for a reconciler, the first goal is to make the reconciler a shared information space where actors progressively record (and access) the outcomes of the incremental learning process discussed in section 3.2. In this paper we focus on the construction of the formalized correspondences which support the reconciler's active behavior. To this aim, the interface proposes a dialogue to capture the necessary "knowledge" according to the following two steps; their order is not prescriptive, except that the process possibly starts out with a partial accomplishment of the first step:

1) identification of the entities which are involved in the intergroup communication. Those entities are mainly (but not exclusively) selected from the lexicon used within the coordination mechanisms (here, conventionally called A and B) to be reconciled.

2) iterative check and a solution for different types of conflicts. To improve understandability, the latter are classified here according to the terminology proposed in [2] and [8]. Two observations are in order: first of all, these technical terms are not (all) used in the user interface as users are not supposed to be experts in information systems; secondly, adopting this terminology does not mean adopting the related prescriptive constraints in terms of consistency. In fact, on the one hand, consistencies have to be identified together with their follow-up in terms of possible new inconsistencies. On the other hand, solutions can just be negotiated, suggested, and possibly solved, but never be imposed on the basis of some abstract reconciliation strategy. The types of conflict currently taken into account are:

Terminology conflicts: identification of synonyms, homonyms and local terms. In the first case, a correspondence table is constructed to make equivalent terms visible. In the second, the homonyms are distinguished by giving them an extension associated to the source context and possibly inserted in the synonyms table, if needed. Terms local either to A or B are identified and recorded as a means to promote awareness of the possible communication problems they invoke.

Category conflicts: here the reconciler considers concepts identified by equivalent terms and highlights the conflicts arising from the sets of attributes of the related entities. Possible communication problems arise when an entity of A corresponds to a super/sub-concept of some concept of B (i.e. the set of attributes of the first entity is contained by/or contains the attributes of second one) or when the two sets simply overlap. In fact, the nonintersecting attributes, when referred to during communication, could not be understood by the receiver.

Type conflicts: here the reconciler considers concepts identified by equivalent terms and highlights the conflicts arising from the different roles they could have in A and B. A recurrent case in our field studies is when a term identifies an attribute in A and the equivalent one identifies an entity in B (see figure 4). In this case, the reconciler represents and records the concept (represented as an attribute) in A as an entity identified by the same term and carrying a single attribute (see figure 4). The general idea is to reduce Type conflicts to Category conflicts. Simple cases can be managed almost automatically; more complex cases require a combination of the proposed techniques with alternative solutions such as social methods discussed in [10]. However, users can decide to avoid this effort and be content with minimal support. In any case, the conflicts are recorded.

Dependency conflicts: here the reconciler considers pairs of concepts identified by equivalent terms and highlights the conflicts arising from the different relations linking the elements of those pairs. A recurrent case in our field studies is when in context A it is necessary to keep a record of a sequence of values, and in context B just one of them makes sense. Here the reconciler asks the users to define a criterion for selecting the appropriate value out of the sequence. Again, a recurrent criterion is to use order/time as a parameter (the last, the first, the more recent value). However, the following example proposes a different criterion.

Unit conflicts: here the conflicts arise from the different domains associated with attributes identified by equivalent terms. A typical example is the attribute *name* whose domain depends on the naming convention of the pertinent context. Examples of this type of conflict are given in section 2. The reconciler asks the users to make explicit the naming conventions and records the related correspondences.

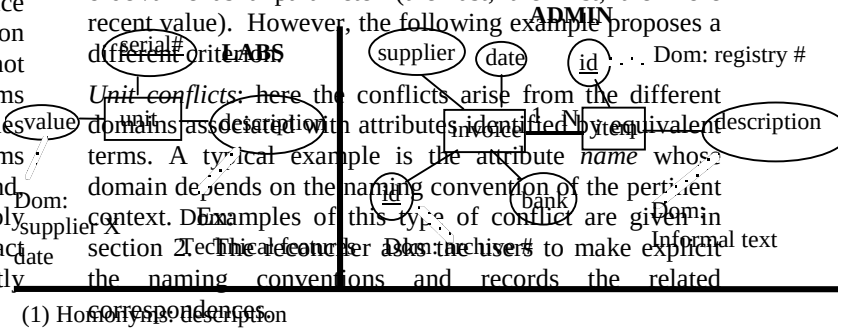

Figure 4. Examples of correspondences, identification of conflicts, and solution

(1) Homonyms: description
(2) Synonyms: unit <-> item; value <-> invoice;
description.LABS <-> description.ADMIN
(3) Unit conflict: Domain: description.LABS ° Domain: description.ADMIN
Resolution criteria: [common set of keywords in both descriptions]
(4) Type conflict: Attribute: value ° Entity: invoice;
New schema:

LABS.reconciler

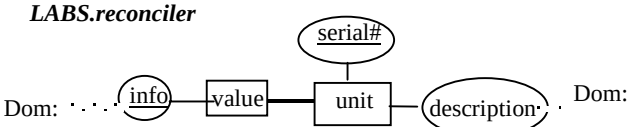

Figure 4 shows examples of conflicts derived from the analysis of some communication problems which arose in the cooperation between the Laboratories and the Administration of our Computer Science Department in Torino. The concepts they use are expressed by an Entity-Relationship formalism: rectangles represent entities, ovals represent their attributes, primary keys are underlined; dotted lines link attributes with the domain of their values. In the following we give a hint of how the reconciler interactively builds the correspondences on which its Naming Service is based (numbers in brackets refer to steps in figure 4).

First of all, homonyms are considered (1). At first glance, actors require distinguishing between the two occurrences of the attribute 'description', since they don't agree about its meaning and use. They require the reconciler to add the related extensions and put them in the list of synonyms to be treated in the next step (at the end, they refer to the same thing, but they are not completely equivalent). Other synonyms are recognized (2); each pair gives rise to a different type of conflict as follows. The two views of the attribute 'description' can be reconciled by solving a unit conflict (3): a resolution criteria is needed. Actors decide that the two descriptions are equivalent if they contain a predefined set of keywords, irrespective of their format and position in the text. The equivalence of 'unit' and 'item' raises a category conflict; however this is not a serious problem as the additional attribute of 'unit' is not used by the Administration. One relevant aspect is that the two groups define a different primary key for unit and item, respectively. They agree to avoid using these keys in their communication since the true link between them is based on the information about the description and the suppliers. This aspect refers to the last pair of synonyms: 'value' and 'invoice'. Their equivalence give rise to a type conflict (4), since the first is an attribute while the second is an entity. A new schema is proposed by the reconciler, in order to solve this inconsistency. 'Value' is raised to the role of entity with the new attribute 'info'; the latter takes the old domain of 'value'. This new schema raises a dependency conflict (5) with the ADMIN schema, since the cardinality of the relationships between the pairs of equivalent entities (value <--> invoice and unit <--> item) are different. Again, a resolution criteria is required. The reconciler proposes to build a correspondence between the value domains of the primary keys (serial no. and id). The actors refuse this solution as it does not provide enough semantics to solve possible critical situations; the attribute description makes much more sense to them. The actors decide to use the attribute description as follows: the pair <info, description> of 'unit' selects the corresponding item out of the list of items associated to 'invoice'. In this way, as agreed, the primary keys are not involved.

Once the goal of highlighting the conflicts and building the correspondences is achieved, the reconciler can then reward users' effort by activating the services supporting their interaction through the communication capabilities of the reconciled coordination mechanisms. Since this communication is translated into the IL/AL languages then the services are based on their syntax. Let us consider the case of a request by someone in context A to perform an action on some shared objects. In a semi-formal representation of IL this corresponds to:

a of A *asks* b of B to *perform* action X on E_1^A ... E_k^A

The reconciler tries to identify the entities of B corresponding to the ones mentioned in the request by using the information collected in the above constructed correspondences. Let's suppose that this naming problem can be solved successfully and that the reconciler is able to identify a correspondence between E_1^A ..... E_k^A and some E_1^B ..... E_h^B. Then the reconciler can use the way in which entities are organized in B to translate the above command into one or more commands on the identified entities in B. Let's consider an example similar to the one discussed in section 3.2. The command could be:

John of the Typist group *asks* Mary of the Unit Member group to *perform* a check on documents D_1, ...,D_k.

Then the first action of the reconciler is to identify the corresponding documents in the Unit Member context. Since the document organization in the latter is different from the other one, and the documents can belong to different sub-contexts (in the example, to different folders) the reconciler translates the single command to a set of commands that can be contextualized in the sub-context of each corresponding document in the Unit Member context. The case of folders is simple and is just a support for document retrieval. A more interesting case is when the target context is organized into a structure of working spaces, equipped with significant contextual information. Then the reconciler's support is much more rewarding since the activation of one of the commands automatically accesses the pertinent working space. Let's consider the following example which is based on the situation depicted in figure 4. The message:

Bruno of LABS *asks* Claudia of ADMIN to *perform* payment of unit

where info = <'XY', 1998> and description = PC

is transformed by using the new schema LABS.reconciler into the following:

Bruno of LABS *asks* Claudia of ADMIN to *perform* payment of {unit, value}

where info = <'XY', 1998> and description = PC

and finally, by using the dependency conflict resolution, it is transformed into:

Bruno of LABS *asks* Claudia of ADMIN to *perform* payment of {item,invoice}

where supplier = 'XY' and date = 1998 and description = PC.

Moreover, since the Administration organizes the invoices according to the bank dealing with the payment, the command is split into a set of commands accordingly. Specifically, the latter are directed to the person responsible for each set of invoices. When this person wants to execute the command, the related working space is open. It contains the procedure to be used with the pertinent bank. For example, one bank provides on-line payments, another one has special payment policies, and so on. In the real situation, this is done manually by Claudia and her colleagues. This working habit could be naturally incorporated into a coordination mechanism and then the related communication directly managed by the reconciler, possibly notifying Claudia, the ADMIN manager, of both the request and the carrying out of the whole command by her colleagues.

If the naming problem cannot be fully solved by the reconciler, then the latter activates a dialogue with the source context in order to clarify the ambiguities. Again, the involved user can refuse this cooperation and delegate the related overhead to the receiver. The receiver can use the information managed by the reconciler in order to understand the request and in so doing, can reduce the number of additional questions to the sender. Finally, if in the source coordination mechanism a command goes together with some awareness communication, then a similar process is activated on the latter by using the syntax of the commands belonging to AL.

We conclude this section by noticing that if a framework as Ariadne/ABACO and the AW-Manager is used, then actors can increase the reconciler's active behavior by using their standard interface for the definition of a generic Active Artifact. The type of support provided by the reconciler can therefore be tailored to the type of investment the two groups want to make to improve the interoperability of their coordination mechanisms.

6 SUMMARY AND CONCLUSIONS

In this paper, we began by presenting empirical evidence showing the consequence of multiple perspectives in a cooperative situation: the development of different views of shared workspaces resulting in incongruent conventions. Based on these observations, we developed requirements for a *reconciler* that can serve to aid users in managing these different views. The reconciler incorporates only that information which is relevant for the cooperative task at hand. The goal is to help users gain an intergroup perspective of the shared objects, in addition to enabling them to retain their individual views.

The reconciler can be seen as an extension of an architecture for cooperative applications, that is, as a component specialized to manage interoperability at the semantic level of articulation work. Although the current implementation manages simple cases of conflicts and supporting services, it shows the technical feasibility of solutions to problems identified in the actual situations that were described.

The functionality currently envisaged for the reconciler has to be further investigated through empirical studies to better identify the source of problems in inter-group cooperation and to develop ways for providing views at different levels of abstraction of the different perspectives to improve its effectiveness and flexibility. This will make the reconciler more adaptable to the users' practices and to the evolution of the coordination mechanisms to be reconciled. This is one of the most challenging efforts of our ongoing research.

ACKNOWLEDGMENTS

We thank Wolfgang Gräther and Piercarlo Giolito for their valuable comments.

REFERENCES

1. Baker, S. (1997): *CORBA distributed objects using Orbix*. Harlow, UK: Addison Wesley.
2. Batini, C., M. Lenzerini, and S.B. Navathe (1986): A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, vol. 18, no. 4, pp. 323-364.
3. Benford, S. and L. Fahlén (1993): *A Spatial Model of Interaction in Large Virtual Environments, Proc. of the Third European Conference on Computer Supported Cooperative Work,* G. De Michelis, C. Simone, and K. Schmidt, Editor. 1993, Kluwer Acdemic Publishers: Dordrecht, p. 109-124.
4. Bordini, R. H., Campbell, J.A. and Vieira, R. (1997): Ascription of intentional ontologies in anthropological descriptions of multi-agents systems. In *CAI 97*. Springer Verlag- Berlin, vol. LNCS 1202, 235-247.
5. De Michelis, G., E. Dubois, M. Jarke, F. Matthes, G. Mylopoulos, M. Papazouglou, K. Pohl, J. Schmidt, C. Woo, and E Yu (1997): Cooperative Information Systems: a manifesto. In M. Papazouglou and G. Schlageter (eds.) *Cooperative Information Systems: Trends and directions*, Academic Press, 315-363.
6. Divitini, M., Simone, C. and Schmidt, K. (1996): ABACO: coordination mechanisms in a multi-agent perspective. In *COOP '96 International Workshop on the Design of Cooperative Systems, Antibes-Juan-les-Pins, France, 19--22 June1996*.
7. Gutwin, C. and Greenberg, S. (1998): Design for individuals, design for groups: Tradeoffs between power and workspace awareness. *Proceedings of CSCW'98*, Seattle, Nov. 14-18, 1998, 207-216.
8. Kahng, J. and D. McLeod (1998): Dynamic classification ontologies: mediation of information sharing on cooperative federated database systems. In M. P. Papazouglou and G. Schlagter, (eds.), *Cooperative information systems*, San Diego: Academic Press, 179-203.
9. Klöckner, K., P. Mambrey, M. Sohlenkamp, W. Prinz, L. Fuchs, S. Kolvenbach, U. Pankoke-Babatz, and A. Syri (1995): PoliTeam - Bridging the gap between Bonn and Berlin for and with the users. *Proceedings of*

ECSCW '95, Stockholm. Kluwer Academic Publishers, 17-31.

10. Mark, G., L. Fuchs., and M. Sohlenkamp (1997): Supporting groupware conventions through contextual awareness. *Proceedings of ECSCW'97*, Lancaster, Kluwer Academic Publishers, 253-268.
11. Papazoglou, M.P. and G. Sclageter, ed. (1997): *Cooperative Information Systems: trends and Directions*, San Diego: Academic Press.
12. Prinz, W. and S. Kolvenbach (1996): Support for workflows in a ministerial environment. *Proceedings of CSCW'96*, Boston: ACM Press.
13. Schmidt, K. and Simone, C. (1996): Coordination Mechanisms: towards a conceptual foundation for CSCW systems design. *CSCW*, vol. 5, no. 2/3, 155-200.
14. Simone, C. and S. Bandini (1997): Compositional features for promoting awareness within and across cooperative applications. In S. C. Hayne and W. Prinz (eds.) *Proc. of GROUP'97*, Phoenix, AZ, ACM Press, 358-367.
15. Simone, C. and M. Divitini (1998): Ariadne: Supporting Coordination Through a Flexible Use of Knowledge Processes. In *Information Technology for Knowledge Management*, U.M. Borghoff and R. Pareschi (eds.) Berlin-Heidelberg: Springer, 121-148.
16. Simone, C. and K. Schmidt (1998): Taking the distributed nature of cooperative work seriously. In *6th Euromicro Workshop on Parallel and Distributed Processing*, Madrid (Spain), January 21-23, 1998. IEEE Computer Society, 295-301.
17. Star, S. L. and J. R. Griesemer (1989): Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, vol. 19, 387-420.
18. Wasserschaff, M. and R. Bentley (1997): Supporting Cooperation through Customization: the Tviews Approach. *CSCW*, , no. 6-4, 305-325.