Support for Coordination and Flexible Synchronicity in Large Scale CSCW

Henrique João L. Domingos | José A. Legatheaux Martins | Nuno Manuel Preguiça | Jorge P. Ferreira Simão

{hj,jalm,nmp,jsimao}@di.fct.unl.pt

DÁgora Project [Ψ]
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa

Abstract

This paper relates with the study and implementation of an integrated large scale distributed system support, for the development and operation of widespread groupware applications. We discuss the implementation of a flexible framework providing basic cooperation concepts and abstractions: communication support for group-oriented interactions requiring flexible synchronicity, coordination support at different levels and awareness-control mechanisms based on efficient event-notification and feedback at system support level. By "flexible synchronicity" we mean a flexible and complementary support for multi-synchronous interactions: peer-synchronous intergroup interactions alternated with periods of asynchronous and disconnected cooperative work.

Keywords: *CSCW/Groupware, Large Scale Distributed Computing System, Object-Group Oriented Distributed Programming, Reliable Group-Communication, Object-Replication*

1. Introduction

The Internet technology, the emergent global broadband high speed computer networks and the availability of ubiquitous internetworking solutions for net-centered stationary and mobile computing environments, will be adopted, by people and by organizations, as potential computing infrastructures for widespread collaborative-work environments. However, to take advantage of this potential, we need a new generation of software components, based on integrated platforms supported by large scale distributed computing systems.

These collaboration-oriented distributed systems need radically new approaches in terms of basic components: (1) group-oriented coordination control and system management; (2) new transparency criteria (or non-transparency criteria) to explicit the collaboration purposes and the support for collaboration-awareness; (3) reliable group-oriented communication protocols and services (or reliable multicasting channels) providing efficiency and flexibility in terms of intergroup interactions and multiple communication semantics; (4) scalable and realistic supports for object replication and related caching management policies, highly demand-driven by the specific requirements of different groupware; (5) distributed programming environments providing basic collaboration-oriented concepts and abstractions and (6) facilities to integrate orthogonal standard service-componen.s or gateways, for conventional communication tools, external awareness-control channels and other auxiliary services (ex: e-mail, telephone/fax, relay-chat services, mobile-sms services, video and audio conference tools, web based information dissemination services, etc.).

Moreover, other standard services for large scale distributed environments are relevant in terms of groupware integration and coordination, such as: directory and management services for cooperative sessions and users/groups registration and authentication; standard gateways for the re-use of information stored and

Ψ This research is partially supported by JNICT - "Junta Nacional de Investigação Científica e Tecnológica", under the PRAXIS XXI scholarship program

managed on conventional legacy systems, standard solutions for security and privacy in wide area internetworking, etc..

The paper discusses the implementation of a generic platform and framework, providing communication and coordination abstractions, for the requirements of widespread collaborative work. It is based on a large scale distributed system (LSDCS) architecture offering high degree of orthogonality and flexibility(1).

2. Support for Large-Scale Cooperation

Main motivations. The approach of generic platforms for CSCW [1][2], in particular those based on LSDCS supports [3][4][5], is our fundamental work direction. We believe that this approach has interesting advantages to build flexible infrastructures for large scale cooperative computing: *openness*, *standardization of components*, support for *heterogeneity/interoperability*, adoption of standard *interprocess communication protocols and mechanisms*, *multi-level concurrency control* and good conditions for the *scalability* criteria. Recently, requirements imposed by the "*large-scale factor*" are emphasizing the need of support for new requirements: *fault-tolerance*, *dynamic reconfigurability* and *high* availability. The research on new solutions for the support of all the above characteristics is an interesting trend, considering some limitations of today's CSCW technology. Most of the current groupware applications are largely incompatible with one another, implementing specific interoperability, cooperation and coordination control criteria from scratch. This is in general an important limitation in terms of large scale adoption. Despite the relevant research in the implementation of useful groupware applications on top of some available integrated toolkits, as well as the research results in areas like: reliable group-oriented communication systems and multicasting protocols, or scalable object-replication solutions, there are some recognized difficulties in the adoption of all those contributions to materialize generic and integrated platforms for large scale collaboration. Some requirements are apparently better supported and optimized at application specific level. However, the impact of scale imposes complex problems that would be directly better supported, transparently and with "standard" solutions, at system level. The balance between the applicational-dependent criteria and optimizations and the basic system support level mechanisms to be provided, is not easy to achieve.

(1) Our results are being tested building the DAgora system: a generic platform and framework to support large scale groupware". References are available at "http://dagora.di.fct.unl.pt/DAgora"

3. Building an integrated platform for large scale CSCW

3.1 Background

In our cooperation model, the participants involved in large scale collaborative work are organized in cooperation domains coordinated as *collaborative sessions*. The sessions are *coordination units* defined as *persistent cooperative system workspaces*, corresponding (eventually) to different *security logical domains*. To each session corresponds a well-known cooperative workgroup. The participation in the sessions can be *reserved* (when the group members are previously registered by an administration entity in a registration service) or *free* (when the participation is open and only based on simple session-binding mechanisms). Reserved or closed sessions require user-authentication services and well-coordinated formal logon procedures.

While in a session, users tend to plan and phase complex tasks into smaller sub-tasks. Each sub-task has a specific cooperative goal and a work methodology. In terms of the computational model, work methodologies are supported by specific tools or applications. The participants can adopt different *group-oriented collaboration-aware tools and applications* in the context of the same session. The tasks developed in different applications can be accomplished in parallel or in sequence, to achieve the expected common cooperative goal established for the global session. The session coordination/management information is shared by different applications. In addition, each cooperative application shares, particularly, specific data-objects, and possible specific coordination and awareness control information, in a sub-set of the global session workspace.

Each individual user should be allowed to participate in different sessions (or workgroups), independently that in specific moments she or he would normally concentrate on just one. For coordination purposes, there is a well-known notion of the collaborative workgroup entity. This entity has two levels of relevance: the static group configuration at session level and a dynamic group membership participation, variable in the time. The first level relates with the potential workgroup organization for organizational coordination issues and, in fact, is more relevant when the cooperation is organized in closed sessions. The second level relates with the dynamic group participation control and is very relevant for coordination awareness purposes.

In closed sessions, the group organization/configuration is supported in a well-known registration service (ex: a database or directory service). Each user registration can include all the relevant social information about each

participant (ex: name, nickname, E-mail address, pager/phone number, social role in the session, social identification session, etc.).

As described above, the dynamic group membership control is very relevant for the group-participation awareness control. If the session is open and there is no initial group organization/configuration, the participation control is only based on the dynamic membership information, which must be managed by the applications as an important coordination and awareness control. This is very important specially in the case of synchronous groupware. In the case of asynchronous interactions the same information can be provided by means of a previous "logon" service, which is a solution currently used by many well-known applications (ex: internet relay-chat services). Other solution is to disseminate, periodically, local configuration attributes (ex: user and machine nicknames) during the interactions.

The most relevant idea of the model is that each cooperative session is not modeled as a unique complex monolithic application, but as a collection of tools aggregated, managed and used in terms of a persistent coordination domain. From a software-engineering perspective, it is also preferable to use a generic multi-tool approach per session, rather than to provide all the functionality (coordination and communication facilities) from scratch, in every application.

3.2 Flexibility and adaptive synchronicity

In the "time vs. space" taxonomy [5], groupware applications can be most usefully classified in two classes: asynchronous or different-time and synchronous or same-time, with users distributed in the same place or in different places. In asynchronous settings, the interaction usually lasts for long periods of time, with different users working not necessarily in the same time-frame (ex: a cooperative development of software). In synchronous applications, users cooperate in a more closely-coupled manner, during a common and usually short time-frame (ex: a multi-user video-conference tool).

The time and space dimensions adopted in the "time vs. space" taxonomy, promotes an alternative support for synchronous and asynchronous applications, considering the communication requirements at system support level. However, the flexibility issues of our proposed session model are not compatible with the orthogonal vision of the time and space dimensions.

In our perspective, synchronous and asynchronous cooperation paradigms are not considered as alternatives, but rather as complementary. The cooperative work that takes place in a session is most often performed alternating asynchronous (or disconnected) work with synchronized moments. In fact, the synchronous and asynchronous paradigms are only edges of a continuum spectrum representing different time-frames, in which a user expects to be aware about the activities and operations performed by the other users. Many applications have requirements for adaptive synchronicity. In the bibliography, some authors refer these cases as multi-synchronous applications [23]. Finally, considering the support for mobile computing, users can also interact in each session with applications supporting autonomous and disconnected work.

The above requirements must be implemented by means of a flexible and generic communication service, encompassing the mechanisms for all the possible interaction modes involved. This is what we call the "flexibility or adaptive synchronicity at system-level".

3.3 The impact of "scale"

In large scale systems, the "space dimension" is characterized in terms of "time-distance" criteria. "Time-distance" depends on the quality-of-service warranties and fault-tolerance properties provided by the internetworking infrastructure and basic communication services. The "time-distance" criteria also relates with the communication semantics and reliability warranties provided by the intragroup communication protocols supporting the cooperative interactions.

Large scale internetworking infrastructures (such as the INTERNET) involve systems not directly fully connected and are typically quite sparse. Computers, execution processes and communication links may fail by crashing or by other operational problems. The load on different systems is highly variable in time and the overall response times of the systems involved are unpredictable. It is impossible to consider bounds on communication delays, in relative speeds or in the performance of different systems, applications and services. A LSDCS is always asynchronous and naturally unpredictable. As a consequence, there are fundamental limits on what can be achieved in presence of strange effects. The absence of an expected user action or the occurrence of a serious inconsistency or divergence cannot be attributed to its possible real cause. In practice, the state of a large scale collaborative application only can be represented trough a presumed reachability relation between all the distributed processes P1,P2, ... PN cooperating in the context of the global application. This situation forces that the support for high-availability and fault-tolerance must be present at system support level. This support is very important to prevent the applications from the manifestations of the strange properties of the "reachability presumption": *asymmetry*: (P1 can be accessible from P2 when P2 is not accessible from P1),

non-transitivity (P1 can access to P2, P2 to P3, but P1 cannot access to P3), and possible occurrence of *partitions* (inaccessibility between sub-sets of P1,P2,...PN). These are complex and hard problems to be solved at application programming level. We consider that the basic support must provide, at system-level, special reliable communication services, to lead with those problems in an uniform and standard way.

3.4 Coordination control criteria

We have an intuitive sense of the meaning of the word "coordination". If we participate in a well-run meeting or if we watch a winning football team, we notice, implicitly, that coordination is present (even it is hidden). Sometimes, we only discuss, explicitly, about the coordination needs, when it is lacking.

We define coordination control as the support for managing dependencies and conflicts in the group activities and tasks involved in a cooperative work. In our research, we are trying to characterize what kind of coordination mechanisms must be provided by the system-support, in a generic way, to help in the materialization of coordination and management policies at applicational-level. As in the case of communication support, coordination also requires flexibility. The coordination support must promote productivity criteria in cooperative work. However, the creativeness, the potential motivation and the cooperation-awareness control of the participants must not be obstructed by restrictions imposed by the "rigidity" of the coordination support. The coordination mechanisms and services must be dynamically reconfigurable and based on efficient group-oriented coordination awareness properties.

Coordination support. We characterize the coordination in CSCW in two main levels: the social coordination level and the system-supported coordination level.

The social coordination level is composed by a set of social rules, methodologies and policies applied autonomously by the participants. This kind of coordination is applied in two different ways: *implicitly* - without a specific informational or computational support, e.g., only based on the common background and social knowledge shared by the participants, and *explicitly* - based on the explicit dissemination of social-information (ex: in initial face-to-face meetings, in orthogonal information systems or trough the usage of informal communication channels). We understand the social coordination as a manifestation of auto-coordination psychological mechanisms.

The system-supported coordination level is provided, explicitly, by the computational support for collaborative work. In our perspective, this coordination must be structured in two levels: the internal or "built-in" basic system-level and the external or applicational-level.

The internal system-level coordination is composed by the low-level mechanisms that manage the dependencies and conflicts in the resource-sharing control: concurrency control mechanisms; communication management and group membership control; access-control mechanisms based on user-privileges (or social roles); and event-notification mechanisms or feedback about all kind of events managed at system support level. These notifications are the basis for the specific awareness control to be provided at application-level.

The external applicational-level coordination is provided in three fundamental supports: specific functionality provided inside each application, reusable system coordination components provided by the programming support and session management facilities.

The specific coordination inside each application is obviously application-dependent. Essentially it is composed by specific awareness control functions and by the materialization of specific access control policies in terms of object-sharing management. For standardization purposes, the access control policies must be applied accordingly with the social roles (user privileges) assigned to the participants by the orthogonal user registration/authentication services (re)used in the context of the session management.

System coordination components are essentially reusable object-components, adopted as building blocks for programming purposes, such as: standard APIs, shared libraries, object-adapters and implemented gateways to external/orthogonal relevant services. Examples of specific components are: an API implementing the interoperability with a user registration/authentication directory service, an SMTP-mail gateway or an object-adapter to send SMS-messages for mobile-telephones, pagers or palmtops.

Session management facilities are provided by means of special applications and services implementing session-managers. A session manager can provide a "browsing" oriented interface for searching/querying about the registered or configured sessions in a data-repository. Using the session-manager each participant can obtain, dynamically, coordination and informal management information about each session. Then, the session-manager can provide automatic binding mechanisms (ex: well-coordinated hyperlinks) for different applications that will be used in the session.

3.5 Workspace organization and topology

The fig. 1 represents a workspace organization and topology for large scale cooperation. We use this topology as a reference for the architectural model of the generic platform and integrated framework that we will describe.

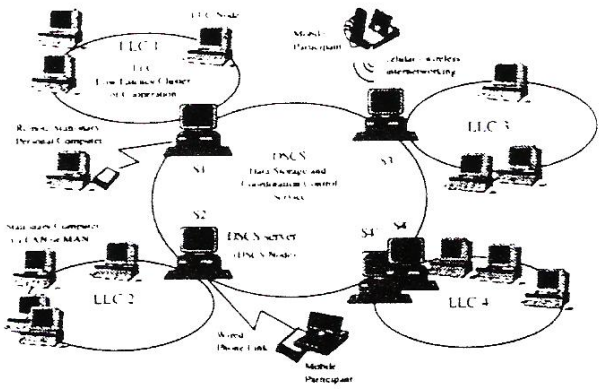

Fig.1 Large scale workspace topology and organization

As represented in this workspace topology we consider different domains of cooperation and coordination:

LLC - Low Latency Clusters - are typically cooperation domains supported by LANs (Local Area Networks) or MANs (Metropolitan Area Networks) or internetworking infrastructures based on LAN-TO-LAN connectivity solutions. In each LLC a set of workstations are used by different users to interact with possible "same time/different place" paradigms. An LLC offers a realistic environment for a cooperative workspace to support synchronous "soft-real-time" interactions among strongly-coupled workgroups. The fundamental criteria to define an LLC is only its potential low-responsiveness characteristic, low latency and good quality of the communication service infrastructure. We consider 1 sec. as the maximum round-trip-time involved between two participants in an ideal LLC.

The **DSCS - Data Storage and Coordination Service** - is provided by a distributed group of replicated servers (ex: S1, S2, S3, S4 or S4') managing a persistent global data-repository service. This service also provides basic coordination and management facilities for different collaborative persistent work sessions. Each server has support for asynchronous interactions. It also provides the support for dynamic (re)integration of mobile and partially connected participants in the context of possible synchronous applications running in each LLC. The DSCS also provides administration and configuration tools and a session-management service. The DSCS infrastructure offers high availability and is the essential support for dynamic reconfigurability and scalability criteria for large scale cooperation.

We define the *Permanent Core Membership Coordination Space* as the topological workspace composed by the set of DSCS servers. The *Stationary Membership Space* is the space where closely-coupled participants can interact from stationary computers operating in the context of each LLC. The *Variable Membership Space* is an highly dynamic space with variable geometry, where mobile users can interact in certain moments with a selected DSCS server, in the context of an existent session.

In the variable membership space, several mobile participants can work asynchronously and disconnected in an unpredictable way. In the most part of the time, they are doing disconnected work. Periodically they submit their contributions to the DSCS server. Each DSCS server provides the necessary support for asynchronous-group dissemination. Even disconnected, each participant can belong virtually to a well-coordinated group and session, where it is registered in a convenient way with the coordination rules established for the respective session.

The DSCS consolidates an high-available and flexible infrastructure providing inter-group communication (communication inside the permanent core membership coordination space) and participant-to-group communication (communication between each participant and the other group members). With this infrastructure, loosely-coupled groups can establish the basis for asynchronous cooperation and can store in a persistent way the partial results of the cooperation. The communication is provided at two different levels: in the first level, a group of DSCS servers (consolidating an LLC-Server Cluster) provide strong consistency criteria using a transparent replication mechanism (S4 and S4'). At the second level, each one of the DCSC cluster (cooperating in a LLC server cluster group) offers weak consistency criteria using non-transparent replication and non-transparent location (from the clients viewpoint). Availability and fault-tolerance is achieved in two different modes: strongly consistency models and lazy replication models. To realize the first model, we can use the virtual synchrony model [7], as in Isis [8] or Horus [9]. This can be the case of S4 and S4', which are two consistent server replicas offering interactive-consistency in the perspective of the LLC4 cluster. The default mode uses a local replication service implementing optimistic replication protocols based on epidemic algorithms, materializing a lazy replication strategy [10], as we will explain in the section 4 with more detail.

Whereas traditional group-oriented toolkits only define a single type of membership space [8],[9],[11],[12],[13] the architectural model proposed offers a hierarchic structure where, in a given moment, a sub-set of the cooperation is represented by a process group (or object group), while in other moments they are represented as a loosely-coupled or completely disconnected set of applications running in disconnected computers.

4. The Framework

Fig.2 represents the general model for the framework.

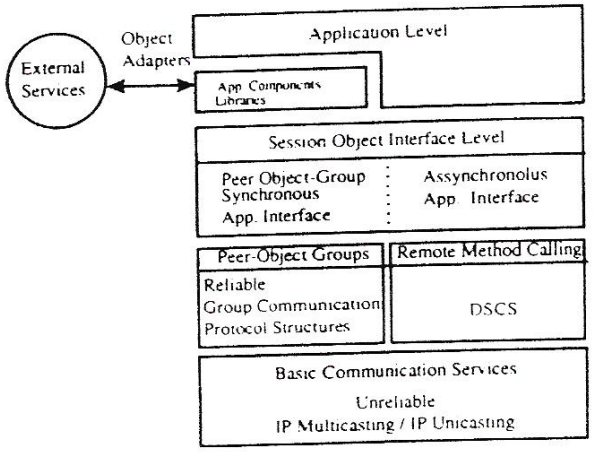

Fig.2 General Model for the Framework

The session-object implements the main relevant tasks related with the coordination control at session level. This object provides facilities for the re-use of session coordination and management information in the context of different applications. Most of this information is obtained from the DSCS by means of specialized coordination services (session registration/configuration, user/group registration and specific configuration attributes to be reused by the applications and tools. By means of a special session-manager service, each participant binds automatically to the different applications using, by default, the configuration and coordination globally configured for the session.

Synchronous closely-coupled cooperative applications will use the support provided by the peer-synchronous object-group programming interface. Asynchronous loosely-coupled collaborative applications use the remote-method invocation interface, interacting with the support provided by the DSCS, particularly the data-repository management facilities. Multi-synchronous applications requiring adaptive synchronicity can use both supports in a complementary way. Session manager applications can be implemented, basically, as asynchronous applications. Additionally, the framework provides specialized libraries providing system coordination object components for the applications. As described above, these objects implement reusable standard and specialized building blocks which can be adopted for coordination and awareness control facilities.

Support for peer-synchronous cooperation in LLCs. The synchronous cooperative applications running on different LLCs present strong requirements in terms of: response times for group interactions; low latency; fine-grain granularity in terms of built-in system coordination control; and relative strong consistency criteria. These applications are organized as specific synchronous sub-sessions scheduled in the context of each persistent work session managed in the DSCS.

Synchronous groupware applications are highly environment sensitive. User actions are strongly driven by unpredictable events. The system support should provide active upcalls representing reactive event notifications driving the awareness control needs for different kind of applications. These requirements are facilitated in terms of the programming support by providing peer-synchronous collaborative-oriented object-group abstractions on top of reliable group-communication protocols [14].

For this support, new requirements (not necessarily found in general settings provided by well-known group-communication technology [8],[9],[11],[12],[13]) are raised, namely: short or immediate responsiveness, short event-notification times and implementation of multi-consistency object-replication control criteria. A more detailed description about this support is available in [15]. The implemented group-communication protocols are specifically tailored for synchronous cooperative interactions, making certain particular assumptions and strongly benefit from them (in a way that some well-known group-communication technologies can not). In this, we include the ability to use the semantic knowledge of each application to choose the appropriate semantics of certain group communication protocols available. Other parallel communication channels, (e.g., video, sound and other bilateral communication facilities) can be used as *ad-hoc* synchronization mechanisms avoiding to pay the "price" and overhead of using a heavyweight protocol in certain cases. These considerations are addressed by a special group-communication support service in the framework, which provides basic object-group communication protocols and a specialized group membership management control, suitable for dynamic group-participation awareness-control.

This support is provided "atop" of a stackable and extensible runtime machine supporting dynamic compositions of multi-semantics and reliable group-oriented communication protocol layers, as in [9]. These layers are all implemented in JAVA. Basically, this support implements several fault-tolerant group communication protocols, enforcing adequate consistency criteria added to the only simple "best-effort" guarantees offered by the basic IP multicast protocol at operating system level.

This is one important direction of our work, consisting in actively devising new fault-tolerant lightweight object-group communication protocols and group view-membership management services, specially tailored for peer-synchronous groupware. The protocols implemented take an optimistic replication strategy since for many applications they better match the recognized requirements of groupware. As the protocols are available "atop" of a lightweight stackable JAVA runtime "machine", the framework provides facilities to allow dynamic loading of protocol stacks, "on demand", from cooperative peer-synchronous JAVA applications. This is an interesting issue to provide flexible options for fully replication object-oriented groupware and for the support of heterogeneity.

To develop a synchronous application, the idea is to replicate the application state, the execution context and the data in each user workstation. Fully replication promotes high availability, fault-tolerance and load-sharing in overall system. However, it must be granted a certain degree of consistency between the replicas used in each moment by different sites. As this is a complex question, the system offers basic level concepts and abstractions to deal with the consistency control, isolating the application programmers from such complex details.

The consistency control and the group binding are offered, transparently, as part of the object-group abstraction provided by the group communication sub-system. Starting from a session manager application it is not difficult to bind to a specific synchronous application just downloading the application as a JAVA class, as in the WEB. Parameters and attributes managed at session coordination level can be obtained as arguments for application session binding. The application can use the existent coordination object-components to manage the coordination information stored in the DSCS in a persistent way. Interactions with the DSCS are provided by means of remote method calling. Presently we use the basic and native RMI support of JAVA for this.

Asynchronous cooperation and disconnected work

Asynchronous groupware can benefit from the following features provided by the DSCS infrastructure:

- An appropriate data-object model for a persistent collaborative data-repository, allowing several users to modify the same data concurrently, not restricting their actions, besides usual access control coordinated mechanism or coordination rules enforcement;
- A weak consistency control system for persistent object-replication in large scale;
- A support to establish coordination information at session support level and binding functions for collaborative sessions and applications;
- A versioning control mechanism for asynchronous updates with support for disconnected operations. With this support, concurrent modifications on resulting shared data-objects are conjugated, enabling type specific resolution and detection of conflicting updates;
- Log-based management information about user activities, data-objects' updates and conflicts detection/resolution. This information is reusable for basic asynchronous awareness control purposes.

We use an object-oriented approach for the DSCS object-model, which is very suitable to fulfil the above requirements for the following two main reasons:

1) Data types fit well in the object-oriented paradigm, because they usually are well-structured, despite they may be saved as simple text files (ex. source programs, LaTeX and other documents, note-books, etc.); and have a well-known set of operations that can change their state (for a JAVA source file we can insert/remove a class, insert/remove modify a method/variable in a class, etc.). Thus, we can easily map data-types in class definitions.

2) As updates are applied by remote method invocations, it is possible to easily determine the contributions done in each work session. These contributions are recorded as the sequences of method invocations made by users. These sequences are replayed whenever wanted, thus making it possible to easily conjugate changes made by different users. This technique is also adopted in filesystems supporting mobility and disconnected asynchronous updates. For example, the Coda distributed filesystem [15] uses this approach successfully for directory management.

To gain access to the objects managed by DSCS a (client) user process creates a local copy of it; future operations are performed locally without requiring communication with the DSCS. As suggested before, update methods invocations are logged by clients, until later re-

integration with an object copy located at a DSCS server. Updates performed concurrently by different users are combined when logged updates are propagated to each DSCS server. For example, in figure 3 we have an object that implements an interface to manipulate a document type, and two users that concurrently modify two different chapters. After modifying a chapter, the correspondent method invocation information is propagated to the DSCS. When both updates reach a server, they will be reflected in the final document without any problem.

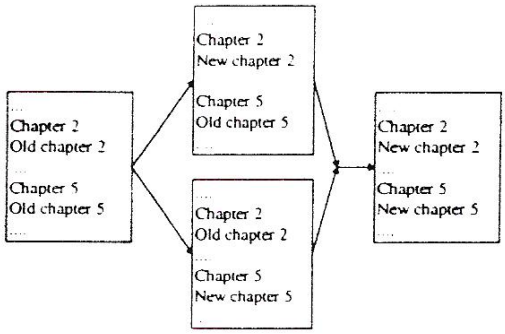

Fig.3 Expected evolution of a collaborative editing session (The middle versions represent two concurrent modifications in the same document)

In the DSCS design, there is a clear distinction between the base system and the specific data types implementations tasks. We have reduced the base system to a simple, but efficient core of operations. Its main tasks are: offer high-availability to clients; propagate clients' updates to servers; propagate objects' updates among all servers, replicating each object. The separation between the base system and data types implementations is very important to optimize the base system and to enable the easy implementation of new data types with different and new semantics, without modifications in the DSCS structure. A similar approach is adopted by the Rover system to tackle the same problem [17].

Data types implementations become responsible to log operations and apply them in a desired order. As suggested before, each server will apply all updates to the local copy of the object. The class programmers should select the ordering constraints for updates application that conforms to the expected objects' semantics.

To enable software re-use and easy data types construction, we propose that objects should conform to an object-model composed of several components: a capsule object that is responsible to aggregate all components that compose a data type. This capsule is the interface to the base system; an attributes object, that is used to maintain relevant meta-information related with the replication process and other; a log object that deals with logging details; a log ordering object, that orders updates' application in a type-specific way; and a data object, that represents the real data type with its own state and operations. From this components, the only one that needs to be built for a new data type is the data object, being others chosen from a set of available options.
In the case that some desired component semantics is not available, a new implementation or an extension of an existing one can be built. We currently have a default implementation for log, attributes and capsule objects and several ordering objects: no-order; causal order; total order based on a primary replica scheme (to enable fast update commit as suggested in the Bayou system [18]); and optimistic total order adopting an undo/redo scheme. We are building an optimistic total order object that uses operations semantic information to deal with concurrent updates and that would notify users when it is unable to automatically resolve conflicts. We also intend to develop a capsule object that enables to have a stable and a tentative data object, as it has been proposed in [18].

Data Storage Structure and Architecture. In the DSCS, data are organized in volumes, which are sets of related objects. Each collaborative session and workspace is supported by a volume in the DSCS. When a group of participants join together to reach some common objective and a new cooperation begins, a volume is created to store the data that will be produced in course of time. Then, the collaborative work usually comprises several synchronous and asynchronous sessions, that manipulate numerous data objects of different types (text documents, images, timetables, ...). For ease of access to data objects, a volume is internally organized as an hierarchical space of objects named by symbolic names (just like a common file system). The mapping between a collaborative work and a volume will also enable to establish a job-dependent coordination, protection and security domain, which is very suitable to the real work model (the set of individuals involved and their roles may differ for different jobs even in the same organization).

As we described in the topology, the DSCS is composed by a dynamically variable set of servers that replicate workgroups' volumes and sessions. Each volume is replicated only for some of the servers. The decision of where should servers be placed and which servers should replicate each volume should be, as usual, wisely taken (possibly varying in the course of time), in order to: minimize the communication requirements of the overall system (considering client needs also); evenly distribute the workload for all servers; and minimize the probability of client isolation from every servers. There is, additionally, a second level of replication, usually known as *cache*, where frequently used objects are stored in

client's machines to enable them to continue working even when they have no access to any server, due to a communication failure or voluntary disconnection. For the success of this technique, the cache should contain the objects that users will be interested in, which is not trivial to determine. Our current implementation uses a simple strategy, based on recent accesses, but a more complex and aggressive one will be considered based on pre-fetch, sets of related objects, and statistical analysis of user activity. These techniques are proposed in [19]. The figure 4 represents the data storage architecture.

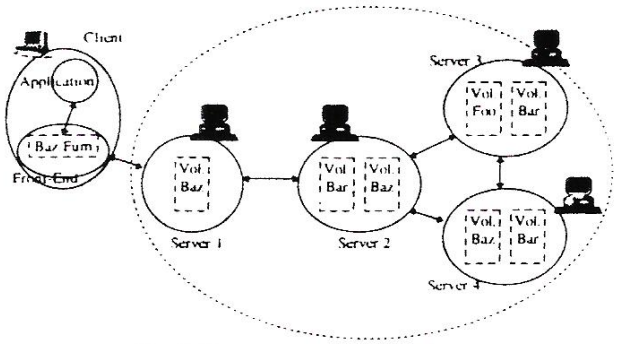

Fig 4. Data storage architecture.

Replication Support for Widespread Object Sharing

While caching is the key element to support disconnected users, server replication is the key to provide high-availability, fault tolerance and scalability. In DSCS servers the default policy is to replicate volumes using a lazy-replication approach using epidemic communication [20][21]. In this approach, pairs of servers communicate with each other, from time to time, to synchronize their states, exchanging updates not known by each other. As long as these communication pairs form a connected graph including all servers replicating a volume, all updates will reach all object's replicas and its contents will eventually converge. This characteristic in conjunction with the possibility of scheduling servers' interactions to the cheapest communication hours, makes the approach very attractive for wide area settings. Another advantage of this approach is that it can schedule server communication and the consequent resource-consuming reconciliation process to late hours, when there are few or there are no active clients. The scalability issue is tackled in DSCS using the characteristics of lazy-replication and the structuring of data in volumes. We think that this approach (combined with a hierarchical inter-volume organization for huge collaborative processes) enables to reduce the number of servers involved in the replication of each volume (or session) to a manageable size, thus succeeding to offer a good service to the desired scale.

When a pair of servers engages in a epidemic interaction, they must determine which updates need to be propagated to the other server. This is done using a summary of known updates represented as a time-vector [22], and comparing it with the time-vector reported by the peer. Update stability is checked using an acknowledge time-vector [20]. After server interaction, the newly received updates are delivered to servers' local copies of the objects, where they are logged and applied according to the ordering constraints selected.

Because the set of servers managing each volume may vary with time, a state transfer mechanism exists to allow new servers to contact an existing server to obtain the volume's content and join volume's replicating server set. Membership information is managed by a special volume object which is also synchronized during normal epidemic interactions. Servers automatically resolve conflicting views of the membership before proceeding with normal operation.

5. On going and future research

We have developed the complete support for the peer-synchronous cooperation support infrastructure and the DSCS. We are now in the development process of the session management and coordination abstractions. Until now, we have implemented a set of coordination classes and object-components that can be (re)used in a web-browser to the rapid prototyping of a session-management environment.

As an initial effort to test the suitability of the synchronous groupware support, we have implemented several demos, including a peer-synchronous collaborative editor and whiteboard. The system response has revealed to be quite acceptable and the performance did not suffer significant degradation when operating on replicated shared objects supported by the JAVA group-communication infrastructure. Additional experience and performance measures are required to analyze system behavior in more general environments.

To test the suitability of the asynchronous groupware support we have defined a simple class of structured persistent objects. These objects are composed of containers, that contain leaves and/or other containers, and leaves, containing data with (possibly) multiple versions. We have also implemented a cooperative editor that allows several users to asynchronously edit the same document. A document is supported by a persistent object and the document structure is mapped to the object's structure (e.g. a document is a container of chapters, a chapter a container of sections or a text leaf, and so on). When all users save their changes, the final document reflects all changes, and concurrent changes to the same leaves (chapters/sections text) are resolved by

creating multiple versions of the conflicting components. We use configurable settings to solve the conflicts as desired.

While we have made a clear distinction between the support for volatile and persistent objects, in medium or long-term real cooperation scenarios the two abstractions will be used in a better combination in terms of adaptive synchronicity . Persistent objects convey the durable part of the cooperative workspace, volatile objects provide the means for users to engage in closely-coupled synchronous interactions. The synchronous and asynchronous interaction modes also benefit from each other. Information related with synchronous sub-sessions (e.g. naming/binding, browsing/awareness and user information) can be persistently saved in the object store. On the other hand, conflicts updates to persistent objects detected by the object store and reported to users can be conveniently resolved using synchronous groupware merging tools. We are investigating issues related to the transparent mapping of the two kinds of objects for those cases in which that is required.

6. Conclusions

The paper discusses a generic platform and integrated framework for the support of large scale collaborative-oriented applications and services. The described work is developed in the context of the DÁgora project. Information and bibliography about the system is available in http://dagora.di.fct.unl.pt/).

The work described in the paper can be summarized in two main directions: (1) identify the communication abstractions required to support flexible and adaptive synchronicity (e.g.: synchronous, asynchronous and multi-synchronous interactions to be adopted by different widespread groupware applications) and (2) study and characterization of the coordination control abstractions to be provided at different levels of the support, to improve the productivity and help in the management of collaborative work sessions.

Bibliography

[1] Knister, M., Prakash, A., "DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors", Proceedings of the CSCW'90, Oct 1990

[2] Roseman, M. and Greenberg, S., "GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications", Proc ACM CSCW 92, Nov 1992

[3] Cosquer, F. J., "Large Scale Distribution Support for Cooperative Applications", Phd Thesis, Univ. Tecnica de Lisboa, Mar 1996

[4] Domingos Henrique J., Martins J. Legatheaux, Simão, J., "A Flexible and Integrated Middleware Framework for the support of Large Scale Cooperative Work", in proc. of the international conference on system sciences, vol. 1, pp 82-92 Hawai, Jan 1997

[5] Kindberg, T., Coulouris G., Dollimore J. and Heikkinen, "Sharing Objects over the Internet: the Mushroom Approach", IEEE Global Internet 96, London, Nov. 1996.

[6] Ellis, C.A., Gibbs, S., Rein, G.L., "Groupware: Some issues and experiences", Communications of the ACM, 34 (1) pp 38-58, 1991

[7] Birman, K.P., "The process Group Approach to Reliable Distributed Computing", Comm. of the ACM 36,12, Dec 1993.

[8] Birman, K.P., and Renesse, R.V., "Reliable Distributed Computing with the Isis Toolkit", IEEE Computer Society Press, 1994.

[9] Van Renesse, R.t, Takako, M. Hickey and Birman, K. P., "Design and Performance of Horus: A Lightweight Group Communication System", TR 94-1442, Cornell University, Aug 1994.

[10] Ladin, R., Liskov, B., Shrira, L.,"Lazy Replication: Exploiting the Semantics of Distributed Services", Proc. 4th ACM-SIGOPS European Workshop, Bologna - Operating Systems Review, (251):49-55, January 1991.

[11] Cheriton,D., Zwaenepoel, W., "Distributed Process Groups in the V Kernel"- ACM Transactions on Computer Systems, Vol.3, n.2 pp77-107, 1985.

[12] Kaashoek, F. amd Tanenbaum, A., "Group Communication in the Amoeba Distributed Operating System", IEEE Proc. International Conference on Distributed Computing Systems, pp.-230 1991

[13] Amir, Y., Dolev, D., Kramer,S., and Malki, D., "Transis: A Communication Sub-System for High Availability", porc. 22nd International Symposium on Fault-Tolerant Computing - IEEE, July1992.

[14] Simão, J., Martins, J.A:, Domingos, Henrique João L. and Preguiça N.M., "Supporting Groupware with Peer-Synchronous Object-Groups", proc. USENIX COOTS-97, Portland, Jun 1997.

[15] DÁgora TEAM (Simão J., Martins J., Domingos H. and Preguiça N.,"An Object Oriented Framework for Efficient Protocol Composition", TR UNL-DI-02/97, Dep. of Computer Science, FCT-UNL, Feb 1997 (http://dagora.di.fct.unl.pt/)

[16] J. Kistler, M. Satyanarayanan, Disconnected Operation in the Coda File System, in proceedings of 13th ACM SOSP, 1991

[17] Joseph, A. D:, et al., "Rover: A Toolkit for Mobile Information Access", in Proc. Fifteenth Symposium on Operating Systems Principles, (SOSP) December 1995

[18] Terry, Douglas B., et al., "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System", Proc. 15th Symposium on Operating Systems Principles, (SOSP), Dec 1995

[19] Kistler, J., Satyanarayanan, M., "Disconnected Operation in the Coda File System", in proceedings of 13th ACM SOSP, 1991

[20] Golding R., "Weak-consistency group communication and membership", Ph.D dissertation, University of California - Santa Cruz, Dec 1992

[21] Ladin R., et al., "Providing High Availability Using Lazy Replication", ACM Transactions on Computer Systems, 10(4):360-391, Nov 1992

[22] Parker, D., et al., "Detection of Mutual Inconsistency in Distributed Systems", IEEE Transactions on Software Engineering, vol. SE-9(3):240-247, May 1983

[23] Dourish, P., "A Divergence-Based Model of Synchrony and Distribution in Collaborative Systems", Technical Report - Rank Xerox EuroPARC / University College, London, 1996