A Flexible Object-Group-Oriented Framework to support Large Scale Collaborative Applications

Henrique João *Domingos* *hj@di.fct.unl.pt*
José Legatheaux *Martins* *jalm@di.fct.unl.pt*
Jorge Paulo *Simão* *jsimao@di.fct.unl.pt*

Abstract

Recent results of distributed systems research, in multicasting and group oriented communication systems and protocols, are very attractive to implement multiparticipant and collaborative applications.
In this paper, we examine the requirements and criteria placed by large scale cooperative applications, namely those alternating multiparticipant synchronous work sessions with asynchronous interactions and disconnected work. We propose a topology organization, an architectural model and the materialization of an integrated generic platform to support those requirements. We argue on the necessary evolution from the well known client/server model to new paradigms involving collaborative-object-group orientation. We describe the implementation of an object-group-oriented infrastructure as well as a generic programming framework offering high flexibility(1).

1. Introduction

Distributed systems and infrastructures providing efficient information dissemination for collaborative applications and services, and high availability, are a natural way to support today's sociological reality characterized by (1) the worldwide nature of the information used by organizations, (2) the need for cooperation and the increasingly inter-relation between groups of people and (3) the gradual decentralization of activities and specialization levels in different decision processes. Several contributions in the recent research work aim a new generation of collaborative applications, emphasizing the need to re-evaluate structuring concepts suited for adequate distributed system support for groupware: *flexibility* seems to be more than ever a pre-requisite; the *scalability* criteria involve new concepts in variable geometry systems with the participation of mobile users and support for disconnected work.

The relevant research results of distributed systems in multicasting, group-communication and object-group-oriented programming are being introduced in several application fields including CSCW - computer supported collaborative work.

Although the support for large scale multiparticipant cooperation and coordination emerges as one of the most interesting trends in the current distributed systems research, some recognized relevant research results of group-oriented systems has not been applied, as quickly as one could initially expect. Possible reasons are: (1) the absence of generic integration platforms and frameworks implementing and structuring, with flexibility, well-accepted collaborative-oriented models with real scalable properties, (2) the lack of realistic programming environments with the necessary expressiveness to use the potential of group-oriented communication protocols and (3), the gap originated by the insufficient validation of complex social and organizational models and their related requirements in terms of group coordination and collaborative distributed system supports.

Several research projects have studied some common well-accepted requirements for collaborative systems. We summarize briefly all them in the following three classes: (i) fast and reliable group-oriented channels granting good responsiveness times and feedback for synchronous cooperation, (ii) efficient event notification mechanisms and membership management control among all the participants and (iii) replicated and effectively scalable data-storage infrastructures and coordination services, providing the basis for an interoperable global middleware infrastructure.

These considerations have motivated our interest in providing the support for cooperation and coordination in applications involving multiparticipant synchronous sessions (like interactive electronic meetings, cooperative editing or brainstorming sessions) alternated with asynchronous sessions with unpredictable duration. During the asynchronous sessions, disconnected work

(1) Our results are being tested, building a prototype called "Ágora - a generic platform and integrated framework to support large scale multi-participant cooperative and coordinated applications."
This research is partially supported by JNICT - "Junta Nacional de Investigação Científica e Tecnológica", program PRAXIS XXI, BM-1566/94

can take place, to prepare documents or to structure some ideas to submit later. Depending on the coordination criteria imposed by each application, users can join (or abandon) dynamically different collaborative tasks. They can have different organizational or social roles. We will describe a model and the implementation of an integrated scalable generic platform and programming framework, where the two interaction modes (synchronous and asynchronous) should be supported alternatively in a complementary way. In the following section, we will describe the identified requirements of CSCW. In the section 3 we propose an architectural model for the generic platform and a topology organization of the cooperation space where synchronous and asynchronous sessions take place. The section 4 is dedicated to the structuring model of the framework proposed. In the last sections we summarize some relevant references about the related research and the fundamental directions of our future work.

2. CSCW requirements and the distributed system support

CSCW is defined generally as a computer-based system that support groups of people engaged in a common goal, providing an interface to a shared environment. Large scale cooperation requires that the system must have additionally scalable properties. In this simple definition, we recognize many hidden obstacles and a lot of old and new complex problems, from the sociological and organizational aspects to the technical ones. Moreover, people can collaborate in a common task at the same time (synchronously) or at different time (asynchronously) in the same place or at different places, as represented in the table 1.

Table 1. Taxonomy and dimensions of CSCW applications

	Same Time	**Different Time**
Same Place	Synchronous classroom; Face-to-face meetings;	Shared file-server;
Different Place	Teleconferencing; Shareable whiteboard; Group meetings for decision support; Synchronous cooperative editing;	Electronic mail; Newsgroups; Asynchronous cooperative editing;

There are two essential approaches for CSCW design. Some applications are in fact single user applications extended with specific and limited collaborative tools. Others are focused in full group-oriented application environments of replicated objects and functions, representing a more recent approach.

The later approach, based on a common platform provided by a distributed system support has several potential advantages like: standardization, integration, re-usage of components, openness, resource sharing, global availability, reliability, fault-tolerance, flexibility and scalability. These are fundamental advantages, in despite of some difficulties related with guarantees in consistency control (that are obviously more easy to implement by means of simple multiuser interface extensions using classical windowing systems and a centralized coordination client/server model).

To take advantage of the current distributed systems research and the technological stability achieved in fields like: group communication and process-group abstractions [1], reliable group-communication protocols, group-oriented systems or toolkits [2] and object-group programming paradigms [3], several contributions are trying to build and validate general purpose platforms integrating and lying on those group concepts and abstractions. Several authors have stressed the need to offer integrated frameworks providing facilities to use the potential of those concepts in distributed programming environments [24], [23], [21], [20], [19]. However, we still recognize some limitations in several approaches:

- Many cooperative applications take place in different phases, different places and different moments, involving long-time cooperating processes with a mix of synchronous and asynchronous periods of interaction. This flexibility it is not easy to achieve.
- Different approaches for group-oriented synchronous CSCW use well established group-communication technologies and toolkits as a basic support. However, state-of-the-art toolkits offering process-group and reliable order-preserving multicast like [2], [5], [6], [9] and [12] aim, primarily, the support for fault-tolerance in a LAN environment, and not necessarily the support for interactive and collaborative requirements. Moreover, these systems offer too low-level primitives and abstractions. This originates a considerable gap between the abstractions needed to describe collaborative processes and the strict basic group-communication abstraction provided by those systems.
- The lack of support for adaptive collaboration, awareness control and coordination is a visible problem when we consider which transparency properties the system must preserve. In most collaborative operations, user actions are reactive, requiring system feedback. Cooperative and interactive operations are driven by control mechanisms of event-notification and membership management control. Many group-oriented systems do not support the idea that these mechanisms must be closed to the application semantics and that the system must provide “feedback” to the application. In

addition, the well known group-communication protocols do not have properties capable of distinguish different group membership roles.

• Considering synchronous groupware, a rapid feedtrough and good response-times are fundamental to avoid possible side-effects resulting from the physical separation between the components of an application. This must be achieved with certain communication-semantics guarantees, not generally available in several proposed group-oriented communication protocols. This motivates the research on new lightweight group communication and membership management protocols.

• The growing need to integrate mobility and support for disconnected operations in asynchronous environments has been stressed more recently in several research projects [7], [8]. Although mobile computing can profit from process groups and reliable group communication to maintain replicated data, only a small number of research work is focusing on this trend [4]. In addition, most operating systems available today, fail in providing basic primitives and abstractions needed by CSCW applications for mobile environments.

On the other hand, the support usually found in applications aimed for asynchronous interactions and large scale scenarios, are generally based on persistent data repository services. The research work has been done considering different solutions to the structure and organization of data-repository systems and data-object storage models. This research, in our opinion, is very relevant to the needs of large scale CSCW [15], [17].

Group-communication sub-systems implementing efficient multicasting channels to disseminate messages are a fundamental technological basis for CSCW. Group membership control provided by the group-communication sub-system is also fundamental to achieve the necessary collaboration-aware basic support. However, warranties on reliable semantics of group communication protocols is only one part of the problem.

Groups of people may work with different social roles using different organizational and coordination models. This flexibility seems to be a necessary condition to grant a certain degree of success in each collaborative process. The support must provide a direct collaborative-oriented support, where basic abstractions for collaboration can be described and managed. The basic group-communication system support and the related group-communication primitives and semantics are only low-level basic abstraction for this purpose.

Accordingly with these assumptions, we argue that in the structuring design model for synchronous CSCW support, the group communication subsystem must provide the basic object-group abstractions for group communication and membership group management. However, a complete object-model must include also fundamental object-collaboration abstractions, directly relevant to the applicational support.

3. Architectural model and topology organization of a generic cooperative platform

The flexibility of a generic support platform must be understood as the ability of the system to be configured and tuned for different requirements in the time/space scenario.

Large scale cooperation between users can be made more effective if it is easy and natural for them to alternate between multiparticipant synchronous sessions and asynchronous interactions (e.g.,a document outline can be created in a multi-participated synchronous session with the remaining work performed asynchronously by different users; finally users could join again in another session to consistently produce the final document version). Our architecture assumes that the two working paradigms should coexist in a complementary fashion.

3.1. Synchronous CSCW support

Synchronous cooperative applications are highly environment sensitive: network protocols and the group-oriented system support should provide active upcalls driving the application.

The programming can be facilitated by providing an object-group abstraction, which relies on a group communication layer specially conceived for synchronous interactions. Due to the interactive nature of such applications, some new requirements (not necessarily found in general settings) are raised, namely: short or immediate responsiveness, short event-notification times, and implementation of appropriated object consistency control criteria. On the other hand, group-communication protocols specifically oriented for synchronous interactions can make certain particular assumptions and strongly benefit from them, in a way that some well-known group-communication protocols can not. In this case we consider for example the ability to use the semantic knowledge of each application. We think that the presence of other parallel communication channels, (e.g., video, sound and other bilateral communication facilities) can be used as *ad-hoc* synchronization mechanisms. These considerations can be addressed by a special CSCW group-communication support that provides the appropriate basic group-oriented abstractions: the basic object-group communication service and a specialized group

membership management control. Later (§3.4) we will describe in more detail the relevant characteristics of this support.

3.2. Asynchronous CSCW support

Groupware for large scale scenarios can benefit from a data storage and coordination service (DSCS) incorporating some additional features:

- An appropriate data-object model for data-repositories;
- Binding service;
- A weak consistency control system;
- A support to establish coordination methodologies and session management policies;(2)
- A mechanism supporting conflict-detection and conflict-resolution.

In extension to the basic services offered by usual data-storage systems, CSCW requires specific consistency control mechanisms specially tailored for asynchronous and/or disconnected operations.

3.3. Topology organization

Groups can overlap and can be open or closed (in terms of participation). Each group can be composed by an unpredictable number of objects. Our architectural model support hierarchic groups, because we recognize that the communication semantics, the availability, the quality of service and the coordination rules can be different within a global hierarchy. Each multiparticipant activity will be organized accordingly with certain sociological or organizational structure. The way to achieve the best organization to fulfill the expectable results of a cooperative task, is a problem related with the application design. However, from a generic system support level viewpoint, the necessary mechanisms to realize certain policies must be offered.

The global cooperative space is sub-divided accordingly with basic connectivity criteria and quality of service parameters (QoS) established by different application domains. Figure 1 represents three different spaces of cooperation: the Permanent Core Membership Space is composed by a set of DSCS servers, collaborating in a special purpose coordination group service. The Stationary Membership Space is composed by closely-coupled participants operating in the context of a low latency cluster. The Variable Membership Space is an highly dynamic space of mobile users.

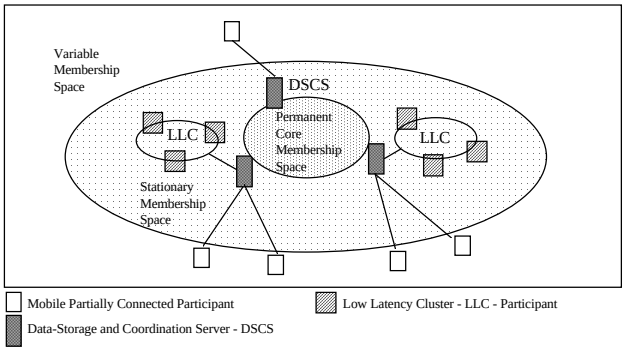

Figure 1. The global space structure

LLC - Low Latency Clusters - are typically implemented by LANs or high-speed interconnection infrastructures offering LAN-TO-LAN connectivity. This environment is a set of closely coupled machines where the "same time/different place" paradigm can be applied. LLC offers a realistic environment to support synchronous "soft-real-time" interactions among strongly-coupled groups of participants. The fundamental criteria to define an LLC is only its potential low-responsiveness characteristic due to low latency and a certain level of quality of service.

The **DSCS** - Data Storage and Coordination Service - is implemented by a group of servers managing a data-repository and providing coordination facilities. Each server provides the dynamic reintegration of mobile and partially connected participants. This service offers high availability and support for disconnected operations.

In the variable membership space, mobile participants can work and collaborate asynchronously together, doing disconnected work in the most part of the time. 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 in a permanent way to a well-coordinated group, where it is registered in a convenient way with certain coordination rules.

The DSCS consolidates an high-availability infrastructure providing inter-group communication (communication inside the group) and part-to-group communication (communication between an external participant and all the members of a group). The communication is provided at two different levels: in the first level, a group of DSCS servers (consolidating an LLC-Server Cluster) can provide strong consistency criteria using a transparent replication mechanism. In the second level, each one of the DCSC cluster (cooperating

(2) The term "coordination" is used as the functionality needed to manage dependencies between cooperative activities. A cooperative proccess does not involve necessarly coordination at all. Coordination can be also managed indirectly by other mechanisms, including *ad-hoc* decisions based on social roles. However, we believe that the coordination support is crucial.

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 different ways: using an extended virtual synchrony model, as in Isis [2] or Horus [9] and using a local replication service implementing an optimistic replication protocol based in a lazy replication strategy [10].

Whereas traditional group-oriented toolkits and other group based systems only define a single type of membership space [6], [1], [11], [12] [9], the architectural model we propose offers a structure where, in a given moment, a sub-set of the cooperation is represented by a process group, while in other moments they can be represented as a loosely-coupled or even completely disconnected set of machines.

3.4. Group-communication subsystem for peer-synchronous multiparticipant sessions

This is one important direction of our work, consisting in actively devising new fault-tolerant lightweight object-group communication protocols and view-membership management services specially tailored for Peer-Synchronous Multipart Sessions support. These sessions take place in an LLC scenario. Some of the protocols we are developing take an optimistic replication strategy since for many applications they better match user requirements. These protocols are supported "atop" of a lightweight stackable machine, providing facilities to allow dynamic downloading from the data-repository server, accordingly with an object-group abstraction, offering multi-semantics options.

Our approach and developments aim primarily to provide truly lightweight group-communication protocols (specifically tailored for synchronous environments).

To structure synchronous multiparticipant sessions a fully replicated model is used. The idea is to replicate the application state, the execution contexts and 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 support must offer 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 is offered as part of the object-group abstraction provided by the group communication sub-system in a transparent way.

The group communication service is specially tailored for the requirements recognized in the case of synchronous fully-replicated cooperative applications. Basically, this is a fault-tolerant group communication protocol, enforcing adequate consistency criteria added to the simple best-effort guarantees offered by the usual IP multicasting transport services at the operating system level.

The most important characteristic of peer-synchronous applications is the user interactivity. This imposes particular requirements in the group-communication support and consistency criteria. In table 2, the requirements discussed are mapped in the fundamental characteristics provided in a flexible way by the object-group communication subsystem.

Table 2. Mapping the requirements for peer-synchronous applications in the group-communication support

Requirement	User Expectation	Impact in the Group Communication Service
Low responsiveness	Short (fast soft-real-time) respond time, comparable to a single interactive user interface.	The protocol must grant low latencies and good global performance, with no impact on the user actions
Fast event notification	Users are driven by the occurrence of events in the overall system. Events mainly depend by user-actions and not only by other exceptions.	The protocol must be lightweight with an adequate performance and must provide upcalls associated with the relevant events
Lightweight dynamic membership	The participation criteria in each application must be simple and dynamic. The users can start or finish working in different moments, joining or leaving the collaborative sessions.	The design of the group support and membership service should allow dynamic process joining and leaving without much overhead. When a process joins to a group, the consistency can be obtained gradually
Fault-tolerance	Each user expects a certain degree of fault-tolerance in case of unexpected or uncontrolled process or processor crashes. In many situations it will be expectable to continue working even in presence of network partitions.	The group communication service must deal at least with the recognized faults besides message lost, processor crashes, fails or delays on communication links and network partitions.
Scale	It is difficult to define precisely this requirement "à priori", because this is strongly dependent on the number of objects and their grain size as well as concurrency control guarantees. But if the application configures a certain quality-of-service, the LLC scenario must be configured based in this figure	The group-communication sub-system must be prepared to scale in the scope of each application domain. The use of lightweight groups is very usefull to support multiple groups of replicated fine-grain objects within the same application
Security	This is related with the authentication mechanisms (to implement access-control policies) and privacy in the group-dissemination channels	Cryptography and eventually a wrapped architecture design can be useful.

3.5. Characterizing the data-storage and coordination service - DSCS

The data storage and coordination service integrates several facilities: naming and binding for objects, binding to synchronous sessions, data repository and support for coordinating asynchronous sessions. Several architectural design decisions have already been retained, while others are still open.

Data repository functionality. A client can manipulate a data-object by issuing a get/check-in operation in order to cache a copy of it. After having modified the data-object contents, it can reintegrate the new value in the system by issuing a put/check-out

operation. This style of interaction is well suited for disconnected or independent operation. This solution has been adopted in several file services (e.g., [13],[14], object systems (e.g., [15]), software development systems and other collaborative environments (e.g., [16],[17]).

Server organization. The data and coordination service is composed by a set of servers. Servers are well-trusted entities that replicate data-objects as object versions. As long as a data-object is being modified by a client it is considered as a cached version whose final value is not yet known to the system. At the moment of the put/check-in operation the new value becomes a tentative version. Eventually, due to the cooperation rules associated with the data-object, it will become a new and unique version, a new stable version among several others, a tentative version to be merged with others, or will be subsumed by some other version.

Replication. More than one server stores a data-object. Clients can get or put in any replica. Clients can even view several different versions of the same data-object. There is no preferred server notion in the system. Lazy replication, is used to propagate the updates in a tunable way. Clients are ware of this scheme and must be prepared to deal with inconsistencies. Clients can also accelerate the replica propagation process trying to put/check-in in several servers at the same time as in [13].

Weak consistency control system. A weakly consistent system maximizes availability of replicas instead of providing one copy serializability which is not acceptable in environments with partitioned networks and mobile users. Such a system will lead to conflicting updates.

Conflict resolution. We want the system to be quite effective in the support of conflict resolution. Conflicts should be automatically detected. However, we believe that automatic resolution of inconsistencies can be avoided. Inconsistencies are not necessarily a bad thing, they can represent different views of the same data-object. The system must not prevent its occurrence, nor stop the evolution in its presence. Cooperation among users should be used to help the coordination of the merge when needed. We are studying several approaches to this problem and will concentrate our efforts in providing several facilities to help applications in dealing with it, but the related development is beyond the scope of this paper and is the subject of ongoing research. Some preliminary assumptions and related-work on this subject are summarized in the related work section.

DSCS object structure and model. The data repository can be seen as a distributed, and replicated file service, enhanced with several facilities for the specific purpose of our research. A data-object is a file hierarchy containing values as well as classes (code). These object hierarchies are replicated lazily and maintained in several versions. Associated with each object, a set of replication and version conflict resolution rules can support the dissemination process, cooperation sharing data and code.

The DSCS system maintains the notion of asynchronous session as a special replicated object. This object has methods for listing and defining the members of each session, their roles, authentication and access control information, binding about the synchronous sessions (that are in fact sub-sessions of the asynchronous session) currently active and a list of data-objects belonging to the session.

We have retained a data centered object model for the service, while providing tools and foundations to use a more general object oriented model in the synchronous application scenario (established by a low latency cluster domain). In this case, replication and multi-semantics group communication is used to achieve it more easily (if the persistence criteria is dropped).However, this leads to a probably too high gap between the data-object model and the need to use some semantic knowledge about data-object contents to help programmers, or simply a too high gap between the synchronous and the asynchronous version of the application.

Our design must be able to provide an optimum mixed approach. The experience with the development of some collaborative applications (a collaborative text/graphic editor, a meeting room scheduler, a cooperative agenda and a shared whiteboard) will be crucial to validate different design options.

4. The framework object-oriented structuring model

The structuring model of the framework (represented in the figure 2) is focused on a layered system design offering distributed structuring classes for synchronous CSCW as well as asynchronous (and/or disconnected) CSCW applications.

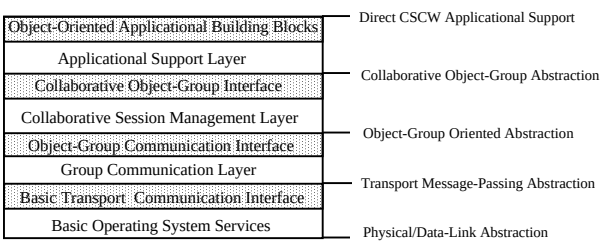

Figure 2. Conceptual reference model

Each layer of the framework provides an appropriate abstraction by means of a programming interface. There are important structural differences between the object-model for synchronous sessions and the object-model for asynchronous sessions. In the section 4.1 we will discuss in more detail the implementation of the object-model for the case of peer-to-peer synchronous sessions.

The layers of the generic reference model are:

The **Basic Operating System Services** layer is the low level abstraction of the model. The interface of this layer offers a generic standard transport service by means of a message-oriented programming interface provided at operating system level. This layer is composed by a set of transport classes (using a basic TCP or UDP socket interface and a UDP/IP multicast interface). The multicast service provided at this level is supposed to be unreliable and based only on a best-effort strategy.

The **Group-Communication Layer** is a sub-system offering a reliable multi-semantics group-communication stackable runtime system (as in [5], [6] and [9]), providing group membership services and group management classes. At this level, the system provides specific lightweight group-communication protocols and a basic membership service, particularly conceived for synchronous peer-to-peer groupware. On top of this level, a thin layer offers the object-group adaptation abstractions, mapping each application-dependent semantics on a specific protocol stack provided by the stackable runtime sub-system.

The **Collaborative Session Management Layer** provides the components offering collaboration-object orientation abstractions. This layer uses the basic group-oriented abstractions provided by the group-communication sub-system to implement the main components to support synchronous collaborative-sessions management. Essentially it is characterized by an object-oriented design offering a basic set of common abstractions used by different groupware applications. For synchronous applications, the set of classes provided by this layer consolidates a synchronous session management service, the binding service and a persistent data-repository service provided by the DSCS. In the synchronous case, a fully object replication in all the participants involved in an LLC scenario is supported. For asynchronous sessions, the collaborative session information is supported in the DSCS. Each participant will mainly interact only with an available server. This interaction is implemented by means of a remote object invocation interface.

The **Applicational Support Level** can be understood as a collection of classes composing a general purpose object library implementing several common building blocks used by different synchronous applications. This library implements usual artifacts and tools. Examples are: a voting tool, a calendar-tool, a WWW-browser, a simple-mail interface tool, a multitalk window and a simple shared whiteboard.

4.1. Implementation of the collaborative synchronous object model

In this section, the implementation of the model for the case of synchronous applications running in a LLC is discussed. The model is represented in the figure 4.

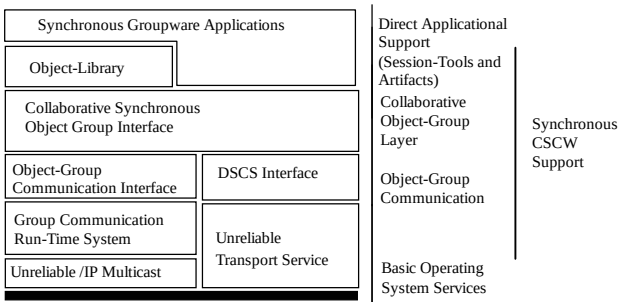

Figure 4. The implementation of the object-model for synchronous applications

The synchronous CSCW support level is implemented in a lightweight runtime system running in each participant machine. Each machine has a minimal set of basic classes. Each site can dynamically download from the DSCS server other classes with different semantics requirements. The group communication interface (including the group membership management control) as well as the DSCS interface are organized as basic native static classes in the basic runtime system.

The DSCS maintains a list of current sessions as special structured objects containing data relevant for coordination, access-control, conflict resolution and notification, cooperation-awareness and synchronous session binding.

The main concept at this level is a session-object. It maintains the following information: a session unique identifier, access-control rules, a data-container, the coordination rules, and the participation criteria. The coordination rules are described by a coordination-object containing coordination information such as the user with the coordinator role, the participation policy and the quorum. The participation rules are described in a participation-object which contains active information about the current group size view, the current members and the largest and lowest group-view during the session. Each participant in a DSCS registered session is described by an object containing the relevant user

information: the username, access-control keys, the real life name, the address of the site, and a pre-defined role.

The next layer (Object-Library) is a specialized layer composed by object-session-tools. These objects implement some common tools re-used in different applications. Object-session-tools are building blocks in the application programming and implement artifacts that can be dynamically loaded in certain circumstances. Examples are: a voting-tool, a simple mail interface or a generic multi-talk artifact. For instance, the voting-tool is an object managing additionally a specific set of attributes: a registration list (users must be registered to vote), the number of users registered for voting, the registration quorum, the deadline for registration, the deadline for voting, the state of the voting session, the final result, a voting rule (e.g., majority, qualified majority 2/3, at least N votes or unanimity).

Not all kind of object-session-tools are necessarily replicated. The information of a secret voting session, for example, can be centralized. In this case, the user only loads an interface to interact with the tool.

To develop an application, the programmer uses a meta-object session called a project. A project is essentially a session-object with some meta information: a project number, a logical project name and some DSCS management information. Additionally, a project-object can include a list of object-session-tools that will be used. The figure 5 summarizes and exemplifies the interaction model of a synchronous application.

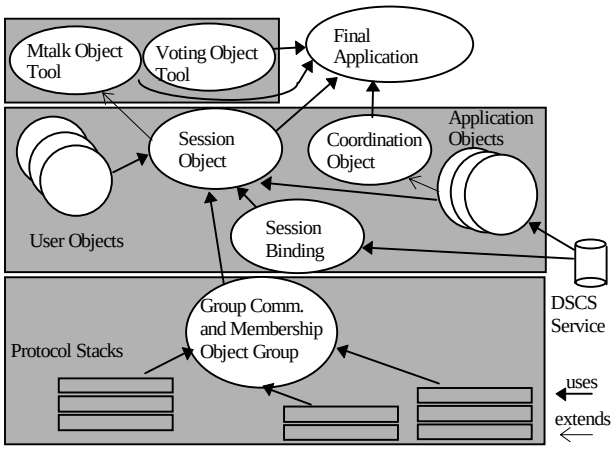

Figure 5. Example of the interaction model of a synchronous application

5. Related work

The design of generic platforms for scalable collaborative applications has been stressed in recent ongoing research.

Few authors examined the requirements which distributed, object-oriented groupware applications place on the application designer and only some contributions [19], [20], [21] show how objects can be adopted as a basic structural design for structuring modern group-oriented applications.

Object groups are a natural object-oriented abstraction for the process-group concept, which have been introduced and provided in some relevant and fundamental distributed systems research [22], [12], [11].

The need of well-structured group communication supports based on extensively layered systems with multi-semantics group communication [9], [5], [6] are very important references for our work.

However, we have recognized some interesting features to explore in the specific group communication design, based on some particular assumptions for the case of synchronous groupware. This motivates the development of special peer-synchronous oriented group communication protocols with some specific characteristics not achieved in the basically fault-tolerant group-oriented protocols provided by those systems.

We must refer some relevant recent contributions (e.g., [23]) with a general similar approach in the development of adequate system supports provided in terms of generic platforms dealing with the problems caused by the asynchronies of most interconnected large-scale environments and broadband networking technologies. In our case however, we are particularly interested in developing also a generic object-oriented programming framework, exploring collaborative-object orientation paradigms and a related linguistic support. In this approach, the fusion between object-oriented integration architectures (as in CORBA [25]) with object group-communication programming (as in the Electra system [24]) are interesting references for this purpose.

Systems offering generic group-communication transport services coexisting with traditional point-to-point communication services are also a pragmatic approach particularly important. The GTS system [4] is a reference to build effective asynchronous large scale infrastructures and services.

Research work in projects like [26], [27], [28], [16] are good examples of the replication approach we also propose. But our work aims to achieve a more generic group-oriented platform basis to support in a complementary fashion synchronous and asynchronous applications with different semantics criteria.

The code-shipping environment provided by the JAVA programming language and runtime [29] is also very useful in developing on-demand dynamic loading of specialized collaborative classes and to implement an extensively layered multi-protocol stackable runtime system (like in the HORUS system [9]).

The structuring model for the DSCS, in building a generic distributed and persistent object-based system and coordination service is an interesting future trend. The fundamental approach for the DSCS sub-system borrows from work in distributed and replicated file-systems like Locus [18] Coda [13] and Cedar [30] (check-in/check-out model of data sharing). The Bayou project also defines an architecture for sharing data in a weekly-consistent environment [17]. However, propagation updates and conflict resolution is our main departure from these other approaches.

6. Future work

The integration model between synchronous and asynchronous session management through the global DSCS infrastructure is an interesting challenge. The direction of our future work addresses an object model, where application objects are typed, supporting a general graph of object references and enhanced with semantic-ware update conflicts/detection resolution. To facilitate the initial implementation we use object models offering a fixed number of object types (e.g., file system and directory hierarchies). The use of relational databases is another possible approach simpler to implement. Moreover, there is a set of valuable experiences and successful results available [13], [18], [14], [16], [17].

The fundamental references of our preliminary analysis to this future work are the following:

- data-object partition dictated by its users will decrease the number of conflicts because users sharing the data-object know about sharing boundaries; this is special suited for cooperative editing (e.g., [16]);
- data-object semantics knowledge embedded in data-object conflict resolution procedures, supplied by users, can also be used (like in [13] and [17]);
- data-object version ownership knowledge in relation with roles can also provide a basis for conflict resolution;
- a global update order based on loosely synchronized clocks or on logical clocks can also be used to decide which version will be kept;
- asynchronous user notification of update conflicts is also a possible way to deal with this problem.

It is challenging to divise a framework allowing the exploitation of these different approaches in an integrated way. We think that the same application dealing with the same data-objects in different moments could take advantages of using different conflict resolution policies.

7. Conclusions

This paper proposes an integrated generic platform and framework to develop and support large scale CSCW. The main motivation is to explore a common support for applications characterized by peer-synchronous interactive multiparticipant sessions, alternated with periods of asynchronous and disconnected work.

We have proposed a basic topology organization and an architectural model implemented by a flexible interoperable platform and an object-group-oriented framework to support the development of complex groupware applications and services. We explained how an object-collaborative-group based approach can offer the abstractions to use, in an effective way, the technological basis provided by the recent research work in multicasting and reliable group communication. We describe the characteristics of peer-synchronous group-oriented protocols and membership control we are working on, using a stackable runtime architecture developed in JAVA.

In our approach to build synchronous CSCW, we explore the advantages of a full object-replication model to achieve: scalability, best performance, availability and fault-tolerance, fulfilling the requirements of synchronous highly interactive applications. These are important issues not present in many conventional CSCW products and research projects.

The paper also argues that to provide useful generic CSCW support infrastructures a basic group-oriented communication support isn't enough. We need also collaborative-object group abstractions to implement the coordination needs.

References

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

[2] Birman, K.P., and Van Renesse, R., Eds. Reliable Distributed Computing with the Isis Toolkit, IEEE Computer Society Press, 1994.

[3] Maffeis, S., "A Flexible System Design to Support Object-Groups and Object-Oriented Distributed Programming", ECOOP'93 Workshop on Object-Based Distributed Programming, Lecture Notes in Computer Science 791, Springer Verlag, 1994.

[4] Maffeis S., Bischofberger W., Matzel Kai-Uwe, "A Generic Multicast Transport Service to Support Disconnected Operation", in Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing, April 1995.

[5] Mishra, S. , Peterson, L., and Schlichting, R. D., "Consul: A Communication Substrate for Fault Tolerant Distributed Programs", Distributed Systems Engineering Journal 1,2, December 1993.

[6] Amir, Y., Dolev, D., Kramer,S., and Malki, D., "Transis: A Communication Sub-System for High Availability", in 22nd International Symposium on Fault-Tolerant Computing - IEEE, July 1992.

[7] Noble, Brian D., Price, Morgan and Satyanarayanan, "A Programming Interface for Application-Aware Adaptation in Mobile Computing", in Proc. of the Usenix - Mobile and Location-Independent Computing Symposium, pp. 57-66, 1995.

[8] Huston, L.B., Honeyman, P., "Partialy Connected Operation", in Proc. of the Usenix - Mobile and Location-Independent Computing Symposium, pp 91-97, 1995.

[9] Van Renesse, Robbert., Takako, M. Hickey and Birman, Kenneth P., "Design and Performance of Horus: A Lightweight Group Communication System", Technical Report 94-1442, Cornell University, Dep. of Computer Science, August 1994.

[10] Ladin, Rivka, Liskov, Barbara., Shrira, Liuba.,"Lazy Replication: Exploiting the Semantics of Distributed Services", Proc. 4th ACM-SIGOPS European Workshop - Bologna - published as 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. of the International Conference on Distributed Computing Systems, pp.-230, 1991.

[13] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., Steere, D.C., "Coda: A Highly Available File System for a Distributed Workstation Environment", IEEE Transactions on Computers, Vol.39, n.4, pp. 447-459, April 1990.

[14] Reiher, P., Heidemann,J., Ratner, D., Skinner,G., Popek, G., "Resolving File Conflicts in the Ficus File System", Proc. Summer USENIX Conference, pp 183-195, June 1994.

[15] Joseph, A.D., deLespinasse, A.F., Tauber, J.A., Gifford, D.K., Kaashoek, M.F.,"Rover: A Toolkit for Mobile Information Access", Proc. of the 15th Symposium on Operating Systems Principles, December 1995.

[16] Pacull,F., Sandoz, A., Schiper, A., "Duplex: A Distributed Collaborative Editing Environment in Large Scale", Proc. of the ACM Conference on Computer Supported Cooperative Work (CSCW), October 1994.

[17] Terry, D.B., Theimer, M.M., Petersen, Karin., Demers, A.J., Spreitzer, M. and Hauser, C.H., "Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System", Proc. of the 15th ACM Symposium on Operating Systems Principles, December 1995.

[18] Popek,G., Walker,B., Chow,J., Edwards,D., Kline, C., Rudisin, G., Thiel, G., "Locus: A Network Transparent, High Reliability Distributed System", Proc. of the 8th Symposium on Operating Systems Principles, pp. 169-177, ACM, December 1981.

[19] Shapiro, M., Gourhant, Y., Habert, S., Mosseri, L., Ruffin, M., Valot, C., "SOS: An Object-Oriented Operating System - Assessment and Perspectives", Computing Systems, Vol.2, N.4, 1989.

[20] Hagsand, O., Herzog, H., Birman, K., Cooper, R., "Object-Groups: An Approach to Reliable Distributed Systems", Technical Report - Cornell University, Ithaca, New York, 1991.

[21] ISIS RDO/C++ Reference Manual - "Isis Reliable Distributed Objects for C++", ISIS Distributed Systems, Inc., February 1994.

[22] Birman, K., "Exploiting Replication in Distributed Systems", Distributed Systems - 1st Edition - Sape Mullender, Ed. Addison-Wesley, 1989.

[23] Felber, P., Guerraoui, "Programming with Object Groups in PHOENIX", ESPRIT Basic Research Project 6360 - Basic Research on Advanced Distributed Computing: From Algorithms to Systems, 3rd Year Report - July 1995 VOL.3 - Systems Architecture - Chapter 2, 1995.

[24] Mafeis, S., "Adding Group Communication and Fault-Tolerance to CORBA" - Proc. of the 1995 USENIX Conference on Object Oriented Technologies, Monterrey,CA, June 1995.

[25] OBJECT MANAGEMENT GROUP - "Common Object Services Specification", Vol.I OMG - Doc. 94-1-1 and "The Common Object Request Broker: Architecture and Specification", Rev.2.0, 1995.

[26] Ellis, C.A., Gibbs, S.J., "Concurrency Control in Groupware Systems", Proceedings of ACM SIGMOD, 1989.

[27] Greenberg,S., Bohnet,Roseman, M., Webster,D., Bohnet,R., "Issues and Experiences Designing and Implementing Two Group Drawing Tools", Proceedings of Hawaii International Conference on System Sciences, Kuwaii,Hawaii, January 1992.

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

[29] Gosling, J., McGilton, H., "The JAVA(tm) Language Environment: A White Paper", SUN Microsystems (ref. http://www.sunsite.com/java), 1995.

[30] Gifford, D.K., Needham, R., Schroeder,D., "The CEDAR File System", CACM,31 (3):pp. 288-298, March 1988.