Building a Flexible Object-Group-Oriented Framework to support Large Scale Cooperative and Adaptive Applications

Henrique J. Domingos, José Legatheaux Martins, Jorge Simão
{ hj, jalm, jsimao }@di.fct.unl.pt

ADVANCED TOPICS WORKSHOP - COOTS 96
2nd USENIX Conference on Object Oriented Technologies and Systems
Toronto, Canadá - June 1996

Abstract

In this paper we examine the requirements of large scale cooperative applications, namely those alternating multi-part synchronous work sessions with variable periods of asynchronous interactions and disconnected work. We discuss the topological organization of the large scale cooperative space and the materialization of an integrated generic platform to support those requirements. Our approach emphasizes the need of an adequate programming support. We will describe and characterize an object-group-oriented layered and wrapped infrastructure as a contribution to build generic programming frameworks offering high degree of orthogonality and, consequently, high flexibility.

Key words: *Large Scale Distributed Computing Systems (LSDCS), Flexibility, Computer Support for Collaborative Work (CSCW), Group Communication Group-Dissemination, Distributed Programming,, Object Group-Orientation, Collaborative Object-Group Orientation*

1. Introduction

Distributed systems and infrastructures providing efficient information dissemination channels for collaborative applications and services 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 in many cases 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* recent criteria involve new concepts of variable geometry systems with the participation of mobile users and disconnected work.

Relevant research results in group-communication protocols, group-oriented platforms and new objet-group-oriented programming paradigms are being introduced in several application fields. Modern collaborative applications and services (CSCW), provided by large scale distributed computing systems (LSDCS) using broadband infrastructures (WANs, MANs and LAN-Intranets) are emerging. Projects (in areas like multimedia and teleconferencing systems or wide-area collaborative editing or authoring systems) are examples of on-going research in the integration of

stationary as well as mobile computers by means of group-oriented and multicasting system supports.

Although the support for multipart large scale group 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 distributed systems has not been applied in this research field, as quickly as one could initially expect. This is particularly visible in large scale scenarios. Possible reasons for this situation are: (1) the absence of generic integration platforms and frameworks implementing and structuring well-accepted flexible collaborative-oriented models with real scalable properties, (2) the lack of realistic programming environments offering the necessary expressiveness to use the potential of group-oriented communication subsystems and (3), the gap originated by the insufficient validation of complex social and organizational models and the mapping of their related requirements in useful and pragmatic usable multi-part group coordination and collaboration electronic supports.

The difficulties described above emphasizes the flexibility as a key-criterion inherent to the CSCW distributed system support. Considering the recognized difficulties, several projects have studied some common and well-accepted requirements for collaborative applications. We summarize briefly all them in the following three considerations: (i) fast and reliable information dissemination group channels granting good responsiveness times and feedback for synchronous cooperation, (ii) efficient event notification and membership control among all the participants, accordingly with different application-awareness and semantics needs, and (iii) replicated and effectively scalable data-storage infrastructures and coordination services providing the basis for an interoperable middleware infrastructure supporting the active participation of mobile users and disconnected collaborative work support.

The above considerations have motivated our interest in providing, in a generic way, the necessary support for cooperation and coordination control for applications involving multi-part synchronous sessions (like breakpoint electronic meetings or synchronous brainstorming) alternated with asynchronous sessions with a possible unpredictable duration. During these asynchronous sessions, disconnected work can take place.

2. CSCW, scale and the distributed system support

CSCW is defined generally by many authors as a computer-based system that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment. Large scale cooperation requires that the system support, accordingly with the above definition, must have additionally scalable properties. In this simple definition, we recognize however many hidden obstacles and complex problems, from the sociological and organizational aspects to the technical ones directly related with the technological basis. Moreover, people can collaborate in a common task at the same time (synchronously) or at different time (asynchronously) and may be at the same place or at different places. The two fundamental dimensions of groupware: the time and

the distance, must be considered independently of specific requirements and criteria inherent to different application domains, which can be essentially different as in the Table 1.

In many cooperative applications, the nature and domain of the cooperative scenario was used accordingly with two essential approaches: (1) some cooperative applications are single user applications extended with some specific and limited collaborative tools. (2) more recently many projects focused in full group-oriented application environments of shareable objects and functions (see the related-work section).

In our opinion, the later approach, based on a common platform provided by a distributed system support using object-replication has several advantages in large scale scenarios. Additionally, there are interesting advantages in crucial aspects like: standardization, integration, re-usage of components, openness, resource sharing, global reliability, fault tolerance, flexibility and scalability. These are fundamental advantages, in despite of some difficulties related with certain guarantees in consistency control (that are obviously more easy to implement by means of simple multiuser extensions to single user monolithic based in a centralized client/server coordination model and classical windowing systems).

	Same Time	**Different Time**
Same Place	Synchronous classroom; Face-to-Face meetings;	Shared file-server;
Different Place	Teleconferencing; Large scale shareable whiteboard; Group meetings for decision support; Synchronous Internetworking Cooperative Editing;	Electronic mail; Newsgroups; Asynchronous Cooperative Editing;

Table 1 Taxonomy and examples of CSCW applications

To take the advantages 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 [2] group-oriented systems [3] and object-group programming paradigms [4], 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 [34[,[32],[29,[28],[27],26],[25]. This seems today an obvious general direction. However, we still recognize some limitations considering flexibility and scale as fundamental requirements to build global synchronous and asynchronous CSCW infrastructures:

- 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 in the same application. This flexibility it is not easy to achieve.

• Different approaches for group-oriented synchronous CSCW use well established group-communication technologies as group communication sub-systems. However, state-of-the-art toolkits offering process-group and reliable order-preserving multicast like [6], [7], [8], and [9] aim, primarily, the support for fault-tolerance in a LAN environment. Moreover, these systems offer too low-level primitives and abstractions for collaborative purposes. This originates a considerable gap between the abstractions needed to describe collaborative processes and the strict group-communication abstraction provided by those low level primitives.

• 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 fact, in most collaborative operations, user actions are reactive requiring system feedback. Interactive cooperation operations are driven by appropriate control mechanisms of event-notification and membership control. Systems do not support in general the idea that the feedback must be closed to the application semantics and that the system must provide "knobs and dials" to the application. In addition, the well known group-communication protocols do not have properties capable of distinguish different group membership roles. However, this is very important to infer the group membership at application level.

• 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 is motivating the research on new lightweight group communication and membership protocols.

The need to integrate mobility and support for disconnected operations in asynchronous environments has been stressed more recently in several research projects [11],[12]. 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 [5],[10]. In addition, most operating systems available today, fail in providing basic primitives and abstractions needed by CSCW applications for mobile environments [13]..

In the distributed systems approach, group-communication sub-systems implementing efficient multicasting channels to disseminate messages are a fundamental technological basis for large scale CSCW. Group membership control provided by the group-communication sub-system is also fundamental to achieve the necessary collaboration-aware basic support. But the discussion and characterization of the support for collaborative purposes in terms of reliable semantics effectively needed is only one part of the problem.

Groups of people may work with different (or common) social roles using different organizational methodologies and coordination models. In a cooperative scenario, people must agree on a certain group methodology which implies in a subjacent coordination agreement. This seems to be a necessary condition to grant initially a certain degree of success in each collaborative process. The system 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 a low-level basic abstraction for this purpose.

Following, we will discuss an architectural model that can be applied in a scenario for large scale cooperative environments. In the section 4 we will describe and purpose a structuring object reference model and its materialization in an integrated framework specially tailored for a complementary support of synchronous and asynchronous cooperative applications.

3. Architectural model of a generic cooperative platform

3.1 Cooperative scenario

As we described above, CSCW application domains are characterized by two fundamental dimensions: distance (a geographical dimension effectively based on time-distance criteria) and time (related with the "timing", in which events take place during a cooperative process). The flexibility of a generic support platform must be understood as the ability of the system to be configured and tuned for certain requirements, independently of the above dimensions.

Large scale cooperation between users can be made more effective if it is easy and natural for them to alternate between multi-part 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 can coexist in a complementary fashion.

3.2 Multi-Participant Synchronous 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 multipart interactions. Due to the interactive nature of such applications, some new requirements (not necessarily found in general settings) are raised to group-communication protocols, 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 generic group-communication protocols can not. In this case we consider for example the

necessary ability to use the semantic knowledge of each application domain. We consider that the presence of several parallel communication channels, (e.g.: video, sound and bilateral communication mechanisms) can be used providing ad-hoc synchronization mechanisms. The mapping of these general requirements with the group-communication support is done by means of a special group-communication support that gives to the programmer the fundamental group-oriented abstractions: the basic object-group oriented multi-semantics communication service and the basic object-group oriented group-view membership control.
Later (see 3.5) we will describe the more relevant characteristics of this support.

3.3 Asynchronous CSCW Support

Groupware for large scale scenarios can benefit from a data storage service incorporating some additional features and the following facilities and components, specifically aimed for multipart asynchronous support:

- 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.

3.4 Large Scale Scenario for the Architectural Model

We will consider that groups can overlap. Groups can be open or closed and can be composed by an unpredictable number of members. However, we consider that will be realistic that the architectural model must consider the possibility to configure hierarchic groups, because we recognize that the communication semantics, the quality of service and the coordination rules must be different within a global hierarchy. We assume that each multiparticipant activity will be organized accordingly with certain sociological or organizational hierarchic 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. One fundamental aspect to do this is to understand and structure the global space where cooperation will take place.

---

(2) We use the term “coordination” as a direct support to manage dependencies between cooperative activities at application level. It is acceptable that in certain circunstances a cooperative proccess do not involve coordination support at all. It is also acceptable that coordination can be managed indirectly by other mechanisms, including ad-hoc decisions based on pre-defined social roles. In any case, we believe that the coordination support will be very closed to each application context and its organizational structure.

The global cooperative space is sub-divided in different membership scenarios accordingly with basic connectivity criteria and quality of service (QoS) established by different application domains.

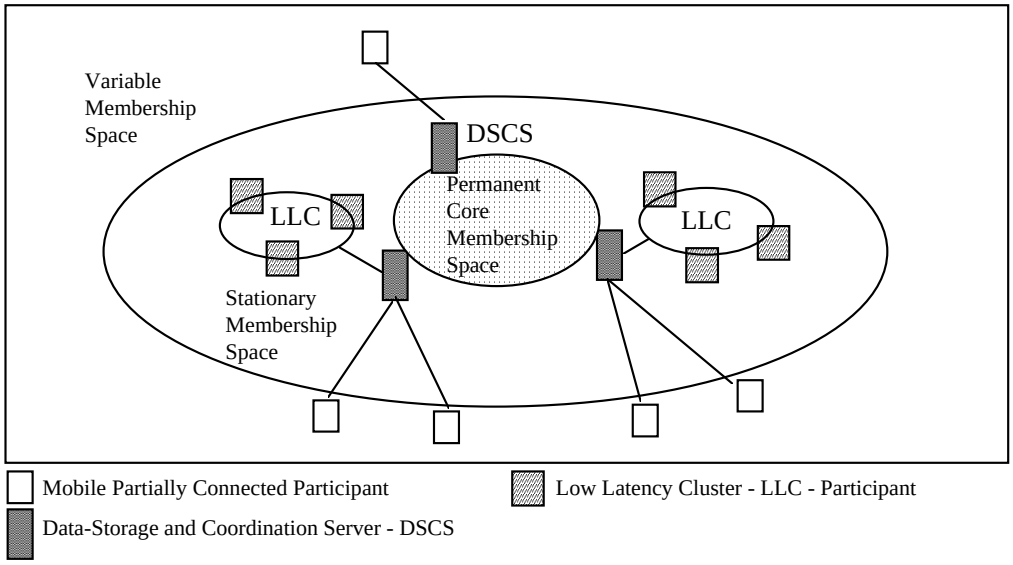

Fig.1 The global space structure

In the figure 1, we recognize three membership spaces. 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 configuration of variable geometry.

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 effectively applied. LLC offers a realistic environment to support synchronous "quasi-real-time" interactions. 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 (QoS).

The **DSCS -** Data Storage and Coordination Service - is implemented by a group of servers managing a data-repository and providing coordination facilities is the basis to achieve the scalability criteria and to provide the dynamic reintegration of mobile and partially connected participants. This service offers high availability and provides the 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 infrastructure. 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 is registered in a convenient way with certain established coordination rules.

The DSCS consolidates an high-availability infrastructure providing inter-group and part-to-group communication 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 2nd 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). To achieve the requirements recognized for asynchronous cooperation, we must guarantee that the DSCS service offers a persistent data-storage service. To increment the availability and to provide fault-tolerance, each DCSC server is replicated. Fault tolerance is achieved in different ways: using an extended virtual synchrony model, as defined in Isis [2][8] or in Horus [14] or using a local replication service implementing an optimistic replication protocol based in a lazy replication strategy [15][16].

As mentioned above, each LLC environment can be understood as a particular client from the DSCS point of view. Each LLC is treated by each DSCS as a Multi-Part Group Client.

Whereas traditional group-oriented toolkits and other group based systems only define a single type of membership cooperating space [9], [1], [17], [18] [14], 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.

3.5 Group-Communication Subsystem for Peer-Synchronous Multiparticipant Sessions

This is one of the important directions of our work, consisting in actively devising new fault-tolerant lightweight object-group communication protocols and view-membership services specially tailored for Peer-Synchronous Multipart Sessions support. These sessions take place in an LLC scenario. The implementation accomplishes the above mentioned requirements for synchronous cooperative applications. Some of the protocols we propose take an optimistic replication strategy since for many applications they better match user requirements. The development of those protocols and the adequate object-group layer in miscellaneous programming languages (C++ and JAVA), gives the opportunity to use them in an effective way to express object-oriented models of synchronous cooperation, providing programmers with several options to realize specific cooperation and coordination policies. Our approach and developments aim primarily to provide truly lightweight group-communication protocols (specifically tailored for synchronous environments). With this characteristic, these protocols can be supported “atop” of a lightweight applet-java stackable machine, providing facilities to allow dynamic downloading from the data-repository server, accordingly with an object-group abstraction offering multi-semantics options.

To structure synchronous multiparticipant sessions a fully-replicated architecture is used. The state of each application is replicated in each user workstation. Replication promotes fault-tolerance and load-sharing in overall system. However, using a replication strategy 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 management and the group binding is offered as part of the object-group abstraction provided by the group communication sub-system in a transparent way. Applications can use an object-oriented design based on some assumptions provided by means of two fundamental abstractions: an integrated group-multicast service and a group-view membership service.

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

Object-Group Communication and Consistency Requirements for Peer-Synchronous Cooperation

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 the table 2 the requirements discussed before are mapped in the fundamental characteristics provided by the object-group communication support.

Requirement	Description	Group Communication Support
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.

Table 2 Mapping the requirements for Peer-Synchronous applications in the Group-Communication Support

3.6 Characterizing the Data-Storage and Coordination Service - DSCS

The data storage and coordination service integrates several facilities often provided as different services: 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 [19, 20], object systems (e.g., [21]), several software development environments and other collaborative environments (e. g. [22], [23]).

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, will be 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 [19].

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 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 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), where full 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. Experience with the development of some well-known collaborative applications (e.g. collaborative editor, meeting room scheduler, shared agenda, shared bibliographic database, ...) will be crucial to clarify different design options we are considering.

4 The Framework Object-Oriented Structuring Model

The structuring model of the generic platform is focused on a layered system design offering distributed structuring classes for synchronous CSCW as well as asynchronous (and/or disconnected) CSCW applications. The programming support is provided by means of an integrated framework using the different abstraction levels accordingly with this structuring model. In the top of each layer the framework provides an appropriate abstraction by means of a programming interface.

There are important structural differences between the object-model supporting synchronous sessions and the object-model for asynchronous sessions. However, the conceptual reference model presents the basic layers and abstractions of the generic programming framework. We will discuss in more detail (§4.1) the implementation of the object-model for the case of peer-to-peer synchronous sessions.

The generic layers of the conceptual reference model are the following:

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 is the basic abstraction to adopt a multi-protocol network support. To clarify, the basic transport communication abstraction can be a message-passing transport programming interface (e.g.: a TCP or UDP socket interface), or a UDP multicast interface (e.g.: Deering UDP/IP socket 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 group-communication stackable runtime system (as in [14],[6],[9]), providing group-view membership services and group management. At this level, the system provides specific lightweight group-communication protocols and a basic view-membership service particularly conceived for the recognized requirements of synchronous peer-to-peer groupware. On top of this layer, a thin layer offers the object-group adaptation abstractions, mapping each application-dependent semantics on a specific protocol provided by the stackable run-time sub-system. It is possible to bypass this layer when needed, for example to establish bilateral point-to-point communication channels between two participants of a given LLC group session. This is also valid to implement the interactions between the applications and the DSCS.

The **Collaborative Session Management Layer** provides the necessary 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 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 fact, the abstractions provided by the collaborative object-group interface are essentially different for synchronous and asynchronous applications. Although the collaborative session concept can be shared, its implementation will differ. In the synchronous case, a fully object replication in all the participants involved in an LLC scenario is supported.

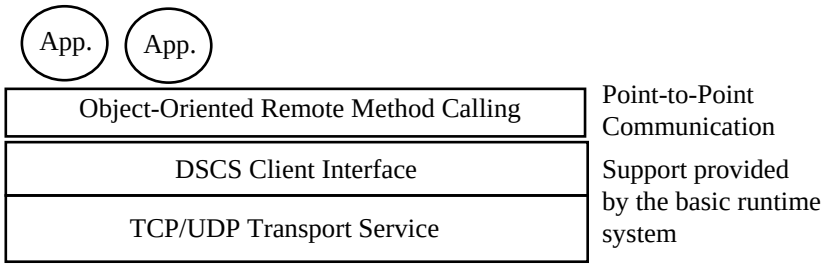

Fig. 2 The structuring model for asynchronous interaction

For asynchronous sessions, the collaborative session information is supported in the DSCS. Each participant will mainly interact only with it preferred (or available) server. This interaction is implemented by means of traditional remote object invocations using a remote method calling as in [4][41], with a very simple model represented in the figure 2.

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

Implementation of the Collaborative Synchronous Object Model

In this section, the implementation of the conceptual reference model for the case of synchronous applications taking place in a LLC will be discussed. The implementation is represented in the figure 3.

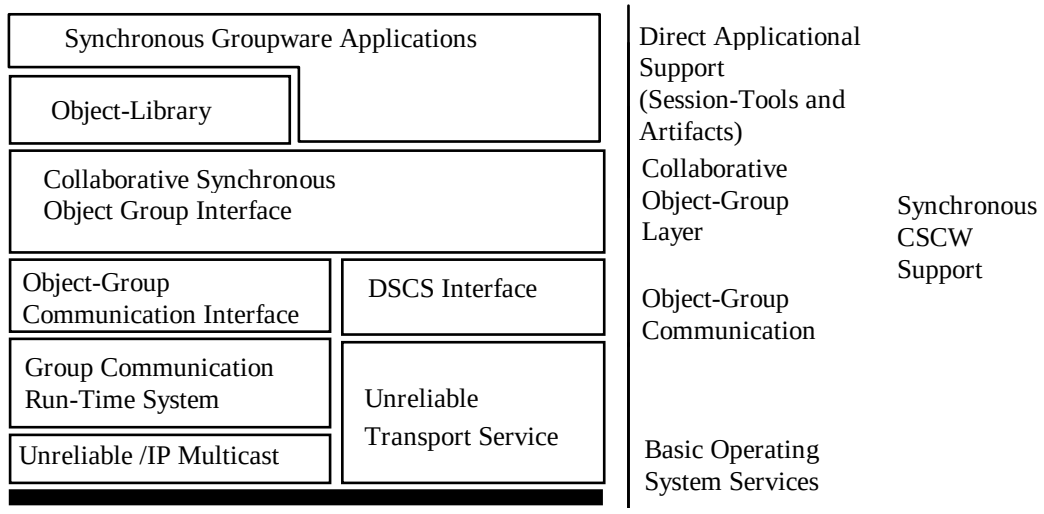

Fig. 3 The implementation of the object-model for synchronous applications

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 class. Depending on each application and its evolution, each site can dynamically download from the DSCS server extended

classes with different semantics requirements. Classes are implemented in two levels: the group communication interface (including the group-view membership and the group communication control) as well as the DSCS interface are organized as basic pre-installed static classes in the runtime system running in each user workstation. The collaborative session management classes can be dynamically loaded.

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. This support provides a common applicational-support for cooperation. This notion provides the basis for a group of cooperating users, associated with specific cooperation attributes and rules.

The above main concept at this level is described in a session-object. Each session object 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 (which is available from the object-group membership class objects), 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. There are different defined roles defined.

In the object-model, there are special session-objects: the object-session-tools. These objects implement some recognized common tools that can be re-used by different synchronous sessions. An object-session-tool is basically an object-session with a specific information and special classes. These objects can be used as building blocks in the application programming. Each building block implements an artifact that can be dynamically loaded in different moments or in certain circumstances.
Examples of object-session-tools are: a voting-tool, a simple smtp-mail interface, a generic multi-talk interface and a simple-management interface for the DSCS.
For instance, the voting-tool is an object-session-tool managing additionally a specific set of attributes: a registration list (users must 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. For instance, the information of a secret voting session is centralized, and the user only loads an interface to interact with the tool. But some of them are fully replicated.

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 4 summarizes and exemplifies the interaction model and the implementation of an application accordingly with the object model described.

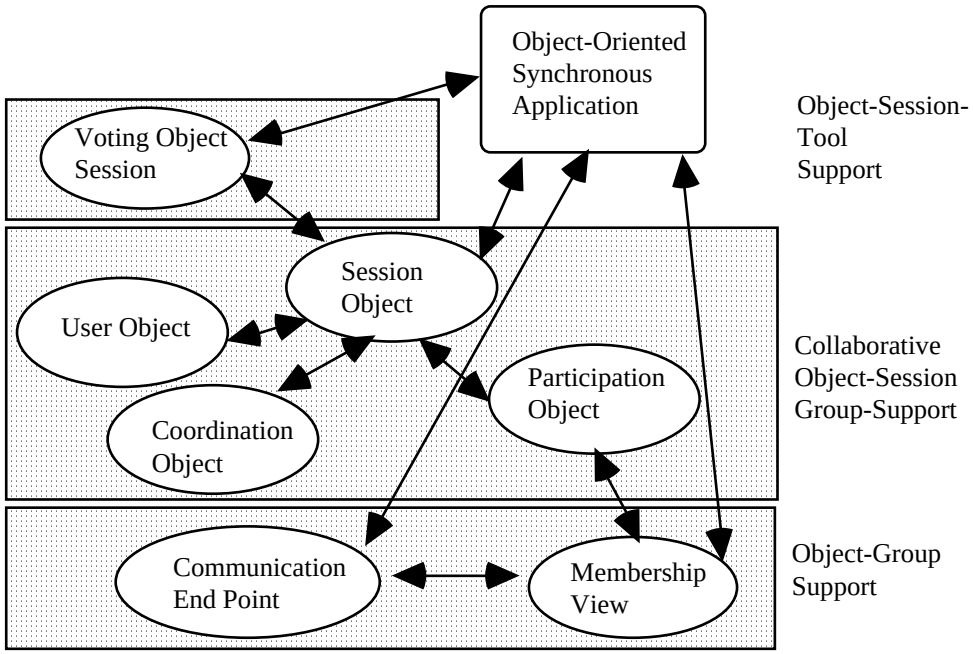

Fig. 4. Example of the class hierarchy of a synchronous application

5. Related and Future Work

Related Work

Few authors examined the requirements which distributed, object-oriented groupware applications place on the application designer and how object groups [25], [26], [27], [28] and [29] can be adopted as a basic structural design for structuring modern collaborative work supports.

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 [30],[31], [18], [17].

The need of well-structured group communication supports based on extensively layered systems with multi-semantics group communication [14],[6],[9] are very important references for our work.

However we have recognized some interesting features to explore in the 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 oriented group-oriented protocols provided by those systems.

We must refer some relevant recent contributions with a general similar approach in the development of more 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 [33],[32],[3]. In our case however, we are particularly interested in developing also a generic object-oriented programming framework, exploring collaborative-object orientation programming paradigms and the related linguistic support. In this approach, the fusion between object-oriented integration architecture (as in CORBA [35]) with object group-communication programming (as in the Electra system[34]) are interesting initial references for this purpose.

Systems offering generic group-communication transport services coexisting with traditional point to point communication services are a pragmatic approach particularly interesting. The GTS system [5] is a reference for us in the way to build effective useful cooperative large scale infrastructures.

Research provided by some projects [36],[37][38],[22] are examples of the replication approach we argue for synchronous groupware. But our work wish to achieve a more generic group-oriented platform basis to support in a complementary fashion synchronous as well as asynchronous applications with different semantics criteria.

The current ideas involved in programming with [39] are also very useful for us in developing on-demand dynamic loading of specialized collaborative classes "atop" of object-group layers provided by an extensively layered multi-protocol stackable runtime system (like in the HORUS system [14]).

Future Work

Concerning with the structuring model for the DSCS we consider that the problem of building a generic distributed and persistent object-based system for the purpose of the replicated data storage 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 [24] Coda [19] and Cedar [40] (check-in/check-out model of data sharing). The Bayou project also defines an architecture for sharing data in a weekly-consistent environment [23]. However, propagation updates and conflict resolution is our main departure from these other approaches.

The integration model between synchronous and asynchronous session management through the global DSCS infrastructure is a complex problem in our opinion. The direction of our future work addresses an object model where objects are typed supporting a general graph of object references, enhanced with semantic-ware update conflicts/detection resolution. However, 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 can be another possible approach simpler to implement. Moreover, there is a set of valuable experiences and successful results already available [19],[24],[20,[22],[23].

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.: [22]);
- data-object semantics knowledge embedded in data-object conflict resolution procedures, supplied by users, can also be used (like in: [19], [23]);
- 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 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 try to divise a general 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 in different moments.

6. Conclusions

The paper proposes an integrated generic platform and framework to develop and support the operation of large scale cooperative and coordinated applications supporting synchronous and asynchronous interactions in a complementary way.

We have proposed a basic architectural model where we can recognize some strategic components and its implementation strategy. The architectural model discussed motivates a flexible interoperable platform and an object-group-oriented framework to build modern infrastructures for groupware applications and services. We explained how an object-group based approach can offer the necessary basic abstractions to use, in an effective way, the technological basis provided by the recent research work in multicasting and reliable group communication. Recognizing some particular requirements for peer-synchronous groupware applications, we referred the fundamental characteristics of peer-synchronous group-oriented protocols and membership services we are working on.

In our approach, we explore the advantages of object-replication strategies to achieve: scalability, best performance and fault-tolerance for the case 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. Then, we referred the need to provide additionally

collaborative-object group abstractions. Based on this, we proposed a layering object model to be applied in the architectural scenario we defined for large scale cooperative environments. The implementation of this model emphasizes the adoption of fully replication strategies using a distributed systems approach.

Finally, we argued that programmers must be isolated from the details of the group-oriented collaborative supports. The well-known client/server programming model isn't enough. Alternatively we proposed new collaborative object-group abstractions based on programming paradigms to help in the development process of collaborative applications.

The results of our research are being validated and tested building a prototype called "Ágora: a generic platform and integrated framework to support large scale multi-participant cooperation and coordination".

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] Veríssimo, P., and Rodrigues L., "Group Orientation: A Paradigm for Distributed Systems of the Nineties", in Proceedings of the Third Workshop on Future Trends of Distributed Computing Systems, IEEE Computer Science, (Apr. 1992).

[4] 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).

[5] 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, Ann Arbor M1, (April 1995).

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

[7] Van Renesse, R., Birman, K. "Fault Tolerant Programming Using Process Groups", in Distributed Open Systems, F. Brazier and D. Johansen, Eds IEEE Computer Society Press (1994).

[8] Birman, K., Van Renesse, R. Eds, "Reliable Distributed Computing with the Isis System", IEEE Computer Society Press, (1994).

[9] 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).

[10] Cho, Kenjiro, Birman, Kennetg P., "A Group Communication Approach for Mobile Computing - MobileChannel: an ISIS tool for mobile services", Technical Report TR94-1424, Cornell University, Dep. of Computer Science, (May 1994)

[11] 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)

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

[13] Davies, Nigel., Blair, Gordon.S., Cheverst Keith., and Friday, Adrian., "Supporting Collaborative Applications in a Heterogeneous Mobile Environment", Internal Report MPG-94-18, Computing Dep., Lencaster University, (January 1994)

[14] 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)

[15] 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, 25(1):49-55 (January 1991)

[16] Liskov,Barbara., "High Available Distributed Services", Programming Methodology Group Memo 52 - Laboratory for Computer Science, MIT, Cambridge, M.A: (February 1987)

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

[18] 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)

[19] 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)

[20] 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)

[21] 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)

[22] 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)

[23] 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)

[24] 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)

[25] 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)

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

[27] Maffeis, S., "Distributed Programming Using Object Groups", IFI TR 93.38 - Dep of Computer Science - University of Zurich, Switzerland (1993)

[28] Oskiewicz, E., Edwards, N., "A Model for Interface Groups", (Ref. to the ANSA Architecture) - Architecture Report 002.01, Architecture Projects Management Ltd, February 1993

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

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

[31] Birman, K., "The Process Group Approach to Reliable Distributed Computing", Communications of the ACM, Vol.36, N.12 (December 1993)

[32] 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)

[33] Antunes,P., Guimar'aes N., "Structuring Elements for Group Interaction", ESPRIT Basic Research Project 6360 - Basic Research on Advanced Distributed Computing: From Algorithms to Systems, 3rd Year Report - July 1995 VOL.4 - Systems Engineering - Chapter 3 (1995)

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

[35] 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)

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

[37] 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)

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

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

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

[41] Maffeis,S., "Remote Method Calling and Object Group Communication", Proc. ECOOP'93 Workshop on Object Based Distributed Programming (1993)