

Autonomous and Self-sufficient Groups: Ad Hoc Collaborative Environments

Joan Manuel Marquès^{1,2} and Leandro Navarro²

¹ Universitat Oberta de Catalunya, Departament of Computer Sciences,
Av. Tibidabo, 39-43, 08035 Barcelona, Catalonia, Spain
jmarquesp@uoc.edu

² Universitat Politècnica de Catalunya, Department of Computer Architecture,
Jordi Girona, 1-3, D5-105, Campus Nord, Barcelona, Catalonia, Spain
{marques, leandro}@ac.upc.edu

Abstract. Asynchronous collaborative applications and systems have to deal with complexities associated with interaction nature, idiosyncrasy of groups and technical and administrative issues of real settings. Existing solutions address asynchronous collaboration via simplified and centralized models. In this paper we present LaCOLLA, a fully decentralized middleware for building collaborative applications that provides general purpose collaborative functionality without requiring anyone to provide resources for the whole group. This helps applications to incorporate collaboration support and deal with most complexities derived from groups and its members. The implementation of LaCOLLA follows the peer-to-peer paradigm and pays special attention to the autonomy of its members and to the self-organization of its components. Resources (e.g. storage, task execution) and services (e.g. authorization) are provided by its members, avoiding dependency from third party agents or servers. This work was first validated by simulation. Then we built the middleware and adapted some collaborative applications.

1 Introduction

One of the most significant benefits of the Internet has been the improvement on people's interactions and communication. E-mail, Usenet News, Web and Instant Messaging are four of the most well-known and successful examples of this. Internet has allowed the creation of asynchronous virtual communities where members interact in a many-to-many basis. Many-to-many interaction, uncommon in the physical world, has transformed the way people learn, do work together, find others with common interests and share information among them, etc. After a decade of great excitement, the pace of this transformation is slowing down because collaboration is much more than these tools, because the Internet is designed for one-to-one interaction (the Internet transport is designed for the communication between two hosts) and that applications with collaborative necessities have to deal with complexities derived from:

- *Interaction nature:* participants are dispersed, many-to-many collaboration, people participate in the collaboration at different times, the same person connecting from different locations at different times of the day (home, work, mobile).

- *Idiosyncrasy of groups*: variety of issues such as flexibility, dynamism, decentralization, autonomy of its participants, different kinds of groups (task oriented, long-term, weak commitment groups, etc), groups exist while its members participate in group activities and provide necessary resources, etc.
- *Technical and administrative issues*: guarantees for the availability of information generated in the group, interoperability among applications, security aspects (authorization, access rights, firewalls), participants belonging to different organizations or departments with different authorities that impose rules and limits to facilitate administration, internal work and individual use, etc. [1]

Development of applications that take into account all those requirements are too complex and costly, therefore collaborative applications focus only in a few key aspects while neglecting others. In that way, most of the solutions resort to simpler client/server centralized models using resources administrated by a third party (a service provider). Client/server solutions –or more generally speaking, all solutions that require some sort of centralization– impose technical, administrative and economic restrictions that interfere with the interaction nature and idiosyncrasy of groups.

In contrast, Peer-to-Peer (P2P) systems or networks are distributed systems formed only by the networked PCs of the participants. All machines share their resources: computation, storage and communication. They all act both as servers and as clients. P2P systems are self-sufficient and self-organizing, applying protocols in a decentralized way to perform search and location, and sharing the burden of object transfers. As resource provision and coordination is not assigned to a central authority, all participants have similar functionalities and there is no strict dependency to any single participant. P2P networks may be robust and attain tolerance to failures, disconnections and attacks. [2]

In this paper we present LaCOLLA, a fully decentralized P2P middleware for building collaborative applications that provides general purpose collaborative functionalities based on the resources provided by group participants only. The provision of these functionalities will avoid applications deal with most of complexities derived from groups, members working across organizational boundaries and requiring additional resources. This simplification (transparency) will help include collaborative aspects into applications in an ad-hoc manner.

LaCOLLA began as a middleware implemented following the peer-to-peer paradigm paying special attention to the autonomy of its members and to self-organization of its components. Another key aspect was that resources (e.g. storage) and services (e.g. authorization) were provided by its members (avoiding dependency from third party agents). At this first stage, it provided support to: storage, awareness, groups, members, instant messaging and location transparency. Now we are incorporating the ability to execute tasks using computational resources provided to group. With that ability, groups will definitely evolve to become entities per se, not only gatherings or collections of members.

Having groups as units of organization and use of resources would help to change to a view of the Internet as a collection of communities: groups of individuals sharing resources among them (an individual may belong to different groups and a resource may belong to different groups). As an example, in virtual learning environments

students may need to do activities in groups using some kind of software. It will be useful that any member of the group could install the software by deploying it using the computational resources available to the group. After that, any member of the group could use that software and the results will be stored on storage resources belonging and available to group.

This ability of pooling resources belonging to groups has been strongly influenced by grid systems. Grids are large-scale geographically distributed hardware and software infrastructures composed by heterogeneous networked resources owned and shared by multiple administrative organizations which are coordinated to provide transparent, dependable, pervasive and consistent computing support to a wide range of applications [3]. In contrast, our focus is on the ad-hoc creation of groups based solely on the resources provided by the participants, independently of underlying administrative organizations or external service providers.

Groove (<http://www.groove.net>) is a platform that partially covers some of the ideas behind LaCOLLA approach. In [4] Groove is defined as a system that lets users create shared workspaces on their local PCs, collaborating freely across corporate boundaries and firewalls, without the permission, assistance, or knowledge of any central authority or support groups. Groove allows transparent synchronization among workspaces, but depends on relay servers to provide offline queuing, awareness, fan-out and transparency (to overcome firewall and NAT problems). Those relay servers are provided by third parties. The main differences between Groove and LaCOLLA are that groove emphasizes transparent synchronization of collaborating PCs, along with direct communication among them. Also the fact that provides third party relay servers. Whereas LaCOLLA emphasizes on self-organization as a group and uses only resources provided by its participants (no dependency on third parties). Participants are not obliged to provide resources to group, but group works only with resources provided by its members. All resources connected to group are synchronized transparently and are used to articulate collaboration.

The rest of the paper is organized as follows: Section 2 presents the requirements that should satisfy an asynchronous collaborative middleware. Section 3 describes the functionalities and architectural aspects of LaCOLLA, with emphasis on what we call virtual synchronism: virtually immediate access to changes and latest versions of objects, along with the API offered to applications and an overview of internal mechanisms behavior. Section 4 presents experimental results from a simulator, concluding in Section 5.

2 Requirements for an Asynchronous Collaborative Middleware

As mentioned previously, asynchronous collaborative applications have to deal with many aspects to support collaboration. The basic requirements a middleware should satisfy to facilitate the development of this kind of applications are [5]:

- *Decentralization*: no component is responsible of coordinating other components. No information is associated to a single component. Centralization leads to simple solutions, but with critical components conditioning the autonomy of participants.
- *Self-organization of the system*: the system should have the capability to function in an automatic manner without requiring external intervention. This requires the

ability of reorganizing its components in a spontaneous manner in presence of failures or dynamism (connection, disconnection, or mobility).

- *Oriented to groups*: group is the unit of organization.
 - *Group availability*: capability of a group to continue operating with some malfunctioning or not available components. Replication (of objects, resources or services) can be used to improve availability and quality of service.
 - *Individual autonomy*: members of a group freely decide which actions perform, which resources and services provide, and when connect or disconnect.
 - *Group's self-sufficiency*: a group must be able to operate with resources provided by its members (ideally) or with resources obtained externally (public, rent, interchange with other groups, ...)
 - *Allow sharing*: information belonging to a group (e.g. events, objects, presence information, etc.) can be used by several applications.
 - *Security of group*: guarantee the identity and the selective and limited access to shared information (protection of information, authentication).
 - *Availability of resources*: provide mechanisms to use resources (storage, computational, etc.) belonging to other groups (public, rented, interchange between groups to improve availability, etc.)
- *Internet-scale system*: formed by several components (distributed). Members and components can be at any location (dispersion).
 - *Scalability*: in number of groups, guaranteed because each group uses its own resources.
- *Universal and transparent access*: participants can connect from any computer or digital device, with a connection independent view (e.g. as a web browser).
 - *Transparency of location of objects and members*: applications don't have to worry about where are the objects or members of the group. Applications use a location independent identifier and may access to different instances as people move, peers join and leave, or any other conditions change.
 - *Support disconnected operational mode*: work without being connected to the group. Very useful for portable devices.

3 LaCOLLA

LaCOLLA is a middleware that follows the requirements presented in the previous section. Four main abstractions have inspired the design process of LaCOLLA: oriented to groups, all members know what is happening in the group, all members have access to latest versions of objects, and tasks can be executed using the computational resources belonging to the group. These abstractions take shape in the following functionality.

3.1 Functionality

LaCOLLA provides to applications the following general purpose functionality [5]:

- *Communication by "immediate" and consistent dissemination of events*: information about what is occurring in the group is spread among members of the

group as events. All connected members receive this information right after it occurs. Disconnected members receive it during the re-connection process. This immediate and consistent dissemination of events helps applications to provide awareness to members.

- *Virtually strong consistency in the storage of objects*: components connected to a group obtain access to latest version of any object. Objects are replicated in a weak-consistent optimistic manner. Therefore, when an object is modified, different replicas of the object will be inconsistent for a while. However, LaCOLLA guarantees that, when an object is accessed, the last version will always be provided (given that events are disseminated immediately).
- *Execution of tasks*: members of a group (or the applications these members use) can submit tasks to be executed using computational resources belonging (or available) to the group. In the present version, tasks are Java classes executed locally that perform computational activities. In future versions we want to be able to deploy services that would provide services at group level. Examples of this kind of services could be a service to coordinate some dynamic and volatile aspect in a synchronous collaborative activity, a session group-level awareness service, or any other service that can provide an added value to groups and that the fact of being deployed in a centralized manner (using only computational and storage resources belonging to group) doesn't affect the decentralization, autonomy and self-sufficiency of the group.
- *Presence*: know which components and members are connected to the group.
- *Location transparency*: applications don't have to know the location (IP address) of objects or members. LaCOLLA resolves them internally (similar to domain name services like DNS).
- *Instant messaging*: send a message to a subgroup of members of the group.
- *Management of groups and members*: administrate groups and members: add, delete or modify information about members or groups.
- *Disconnected mode*: allow applications operate offline. During re-connection, the middleware automatically propagates the changes and synchronizes them.

3.2 Architecture

The architecture of LaCOLLA [5] is organized in five kinds of components (figure 1). Each component behaves autonomously. Each member decides to instantiate any number of the following components in the peer is using:

- *User Agent (UA)*: interacts with applications (see section 3.4 for a more detailed explanation). Through this interaction, it represents users (members of the group) in LaCOLLA.
- *Repository Agent (RA)*: stores objects and events generated inside the group in a persistent manner.
- *Group Administration and Presence Agent (GAPA)*: in charge of the administration and management of information about groups and their members. It is also in charge of the authentication of members.

- *Task Dispatcher Agent (TDA)*: distributes tasks to executors. In case that all executors were busy, the TDAs would queue tasks. Also guarantees that tasks will be executed even though the UA and the member disconnects.
- *Executor Agent (EA)*: Executes tasks.

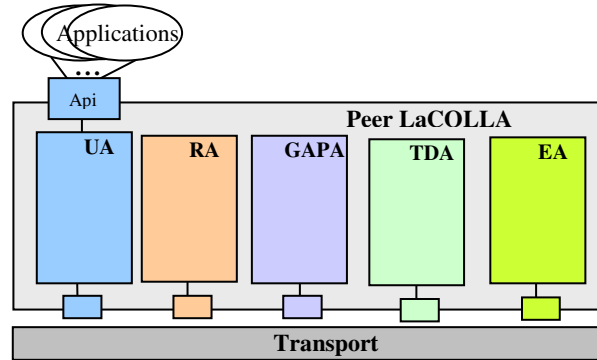


Fig. 1. Peer LaCOLLA

Components interact one to each other in an autonomous manner. The coordination among the components connected to a group is achieved through internal mechanisms. Internal mechanisms [5] have been grouped in: events, objects, tasks, presence, location, groups, members and instant messaging. They are implemented using weak-consistency optimistic protocols [6, 7] and random decision techniques [8]. Table 1 describes which components are involved in each category of mechanisms. More details about presence, events and objects mechanisms are provided in section 3.4.

Table 1. Categories of mechanisms implemented by each kind of component

Categories of Mechanisms	UA	RA	GAPA	TDA	EA
Events	X	X	-	-	-
Objects	X	X	-	-	X
Tasks	X	-	-	X	X
Presence	X	X	X	X	X
Location	X	X	X	X	X
Instant Messaging	X	-	X	-	-
Groups	X	X	X	X	X
Members	X	-	X	-	-
Security	X	X	X	X	X
Disconnected operational mode	X	-	-	-	-

Components and mechanisms related to tasks are based on the ideas used to design JNGI [9], a decentralized and dynamic framework for large-scale computations for problems that feature coarse-grained parallelization. While the components of JNGI communicate using JXTA [10], we use the communication facilities of LaCOLLA. Among the aspects that characterize LaCOLLA one that deserves special attention is what we have named virtual synchronism.

3.2.1 Virtual Synchronism

LaCOLLA guarantees to applications that all events delivered to LaCOLLA will be received almost immediately (i.e. immediately or just after reconnection) by the rest of connected members. This guarantee provides the feeling of knowing what is happening in the group while it is occurring. Disconnected members will receive the events during the re-connection process.

LaCOLLA also guarantees that the last version of all objects (based on the previous guarantee) belonging to group will be available immediately for all members.

The sum of both guarantees is what we have named virtual synchronism. Apart from the up-to-date perception that members of the group have at any moment, virtual synchronism has an interesting side effect. This side effect is very useful in an autonomous, decentralized and dynamic storage system: since all components know the location of all objects (and their replicas), components access them directly (without a resolver that informs about location of last version of objects). This allows LaCOLLA to have an autonomous and decentralized policy to handle objects and their replicas at the same time that guarantees immediate access to last versions.

3.3 Example of LaCOLLA Group

Figure 2 is a snapshot of a collaborative group that uses applications connected to LaCOLLA. Each member belonging to group provides to it the resources that she/he wants. As we have said, that decision depends on the capacity and connectivity of the computer the member is using and on the degree of involvement that she/he has in the group. In this example, two members (C and D) provide all possible components (RA, GAPA, EA and TDA). Other two members (B and F) provide all components except execution components (provide RA and GAPA). Three of the members (A, E and G) provide no resources to group.

The members of the group use several applications to perform the collaborative tasks. At the moment the picture was taken, they were using an asynchronous forum, a file sharing tool and an instant messaging application. Not all members use all applications at same time.

Those applications share presence, members and group information. On the one hand, this prevents users to register to each application and also provides presence information even though they are using different applications. On the other hand, application developers don't have to worry about where the necessary information is located. LaCOLLA middleware also facilitates the sharing of information among applications (if compatible formats are used) due the fact that information, events and objects are stored in LaCOLLA storing resources (RA).

Member D (represented by discontinuous lines) is not connected to group at this moment. Even though, her/his peer is connected to group, providing all its resources to it. That means that all generated events and some of the objects would be stored in her/his peer LaCOLLA (RA), tasks would be executed or planned using its resources (EA, TDA), or that users would be authenticated by her/his peer (GAPA), information of members and groups would be also stored in it.

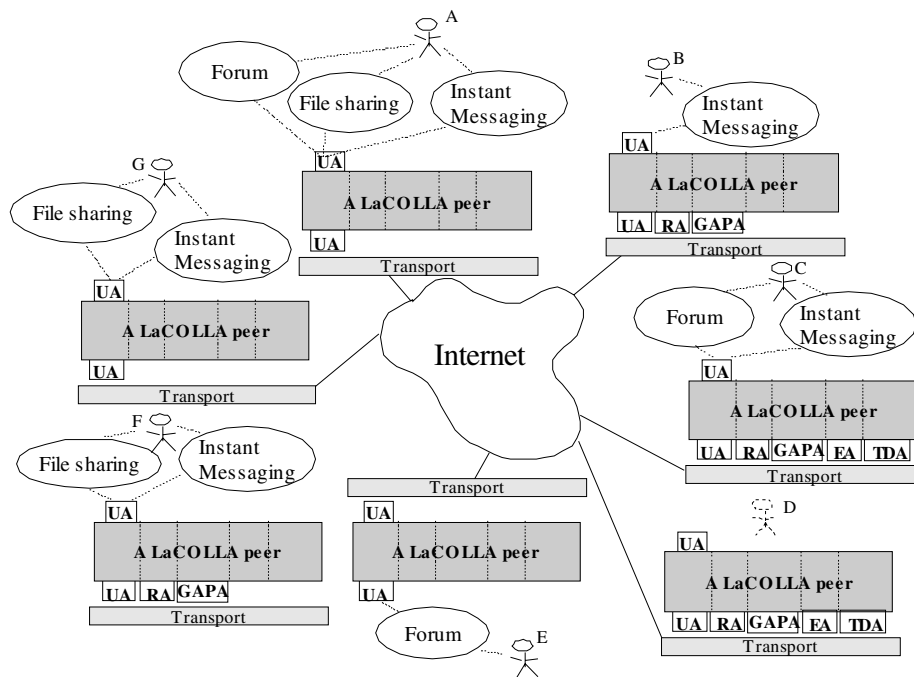


Fig. 2. Snapshot of a collaborative group that uses applications connected to LaCOLLA

An example of a group like the one presented in figure 2 could be a collaborative group doing a collaborative learning practice in a virtual university (as is UOC - Universitat Oberta de Catalunya). The learning practice could be a software development project or a case study. In those cases, a member of the group initiates the group (providing at least one RA and one GAPA components) and invites other members (who contribute with more resources and components to the group). From that point on, the group operates using the resources provided by its members. Although any member disconnects its resources or is removed as member of the group, the group will be operative. And most important, nothing will happen if the initiator of the group disconnects its resources or is removed from the group. As long as members provide resources to the group, it will exist. Whenever no member provides resources, the group would extinguish.

LaCOLLA is independent of the applications that use its functionalities. Many applications (not only the kind of applications presented in the figure) involved in a

collaborative task could benefit from the general purpose collaborative functionalities that LaCOLLA provides. These applications could range from applications that only share generated information to sophisticated collaborative applications exploiting awareness information and coordinating actions (as events) of participants in the collaboration.

3.4 LaCOLLA Middleware

At the moment of writing this paper, we had the first beta version of LaCOLLA middleware. This version can be found at: <http://lacolla.uoc.edu/lacolla/>. It includes the source code, some basic instructions on how to use LaCOLLA, and installation procedures. LaCOLLA middleware has an open source license and is written in the Java language, what makes it independent from the underlying platform.

From both building collaborative applications that use LaCOLLA and from using the applications we developed, we obtained valuable ideas and improvements to introduce in the second beta version. The new version will pay special attention to security issues, which are at its minimum expression in the first version. We are also planning to introduce new components and mechanisms that will allow mobile devices (PDA, mobile phone, sensors, etc.) become LaCOLLA peers.

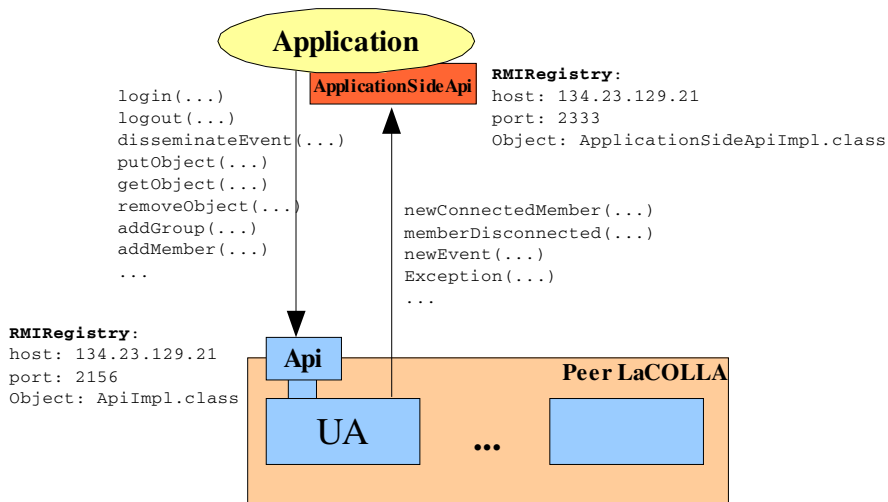


Fig. 3. LaCOLLA API. It has two parts. Applications use UA’s API to ask LaCOLLA to perform some action. The other API is provided by the applications to the UA were they are connected, that API is used by LaCOLLA to notify events or information to applications.

3.4.1 LaCOLLA API

LaCOLLA provides a powerful API that can be easily used by any application. As can be seen in figure 3, the API of LaCOLLA is divided in two parts. The first part is the API provided by LaCOLLA (through its UA) to applications. The detail of functions that an UA provides to applications is listed on table 2.

The second part of the API is used by an UA to notify events or information coming from LaCOLLA to applications connected to the UA. Table 3 lists the functions. As can be seen in figure 3, UAs invoke functions at `ApplicationSideApi` class. This class is provided with LaCOLLA middleware and must be extended by any application that wants to use LaCOLLA.

Java RMI is used to publish and invoke each part of the API. In the example of figure 3, UA's API is published at host 134.23.129.21 and port 2156. When an

Table 2. API functions that User Agents offer to applications

Category	Function	Description
Presence	login	Connects user to group.
	logout	Disconnects user from group.
	whosConnected	Which members are connected to the group?
Events	disseminateEvent	Sends an event to all applications belonging to group.
	eventsRelatedTo	Which events have occurred to a specific object?
Objects	putObject	Stores an object in LaCOLLA.
	getObject	Obtains an object stored into LaCOLLA.
	removeObject	Removes an object stored in LaCOLLA.
Tasks	submitTask	Submits a task to be executed by computational resources belonging to group.
	stopTask	Stops a task.
	getTaskState	In which state is the task?
Instant Messaging	sendInstantMessage	Sends a message to specified members of the group.
Groups	addGroup	Creates a new group.
	removeGroup	Removes a group.
	modifyGroup	Modifies the properties of a group.
	getGroupInfo	Gets information about the properties of a group. (Look at <code>groupInfo</code> function)
	getGroupInfoSync	Gets information about the properties of a group in a synchronous manner. This function does not return until the operation is completed and a result is available.
Members	addMember	Creates a new member.
	removeMember	Removes a member.
	modifyMember	Modifies the properties of a member.
	getMemberInfo	Gets information about the properties of a member.

Table 3. API functions that UA invokes on applications

Category	Function	Description
Presence	newConnectedMember	Notifies that a new member has been connected.
	memberDisconnected	Notifies that a member has been disconnected.
Events	newEvent	Reception of an event occurred in the group.
Tasks	taskStopped	Notifies that the task has been stopped nicely.
	taskEnded	Notifies the ending of a task.
Instant Messaging	newInstantMessage	Reception of a new instant message.
Groups	groupInfo	Reception of the group information.
Other functions	exception	Notifies that an internal exception or anomalous situation has occurred.
	appsAlive	UA queries the state of the application. Used to know if application is alive and connected to group.

application wants to login, logout, send an event, put an object, get an object, etc. it has to invoke the API function at this location.

The same thing happens with the API provided by each application to LaCOLLA. In that case, each application extends `ApplicationSideApi` and publishes it. In the example, application is at host 134.23.129.21 and at port 2333. This API allows UAs notify to applications that a member has connected, that a member has disconnected, that there is a new event, an exception, etc.

It is also interesting to notice that all applications connected to a LaCOLLA peer will use the API provided by its UA, but that each application will have its own API and that the UA will notify to each application individually.

If the application is written in Java the integration with LaCOLLA is very easy. It has to use Java rmi to invoke the API of LaCOLLA at UA. The application also has to extend `ApplicationSideApi` class provided along with the LaCOLLA middleware. This makes very easy to adapt applications done in Java to benefit from LaCOLLA.

If the application is written in other programming languages, the developer has to build a module to be able to use the API of LaCOLLA. This module is very easy to build because it only has to translate parameters and results to and from Java. For instance, if the application is written in C/C++, JNI (Java Native Interface) can be used. The module will encapsulate both sides of the API: the invocations from application to UA and the notifications from UA to application.

3.4.2 Components and Internal Mechanisms

Components are implemented in Java and behave according to its local information (autonomy). Coordination among components connected to a group is achieved by internal mechanisms, which allow components learn new information and synchronize its local information with other components. Internal mechanisms behave in a decentralized and autonomous manner. Components communicate by message passing. Messages are serialized Java objects sent using TCP sockets.

There are 10 categories of internal mechanisms divided in several sub-mechanisms. Each sub-mechanism performs different actions depending on the kind of component. Is out of the scope of this paper to detail how the decentralized and self-organized behavior of LaCOLLA is achieved. A fully and detailed description can be found at [5, 11]. Alternately we are going to explain the general behavior of some LaCOLLA's internal mechanisms and the key aspects to understand its philosophy.

LaCOLLA middleware is based on the presence mechanism. To guarantee the consistency and a good performance of LaCOLLA it is required that each component connected to group knows which other components are connected to the group. The other key mechanism is event dissemination. Presence is the basis for the peer-to-peer behavior (decentralization, autonomy, self-organization and self-sufficiency). Event mechanism provides immediateness and consistency of view. In the next paragraphs more detail of both categories of mechanisms will be provided. Prior to that, is important to understand that LaCOLLA is a middleware intended to support asynchronous group collaboration and some sorts of synchronous collaboration. Therefore, groups are considered to have a small number of members and components (as is stated in validation section, LaCOLLA can deal with groups formed by 100 or

more components, but groups, to be realistic collaborative groups, should typically have 5, 10 or 20 members, not more). LaCOLLA was not created to support communities of members sharing or performing some weak-collaborative task.

Presence sub-mechanisms are: connection of a component, disconnection of a component, consistency of connected information, and detection that a component is no more connected. When a component wants to connect a group, sends its authentication information to a GAPA. If authentication is ok, GAPA answers with information about which components the GAPA knows that are connected to group. Then the new component sends a message to all components he knows that are connected to the group (those that GAPA has informed him) informing about its connection to the group. Prior to an ordered disconnection, the disconnecting component informs other components about its disconnection. To synchronize information about connected components, two techniques are used: a) every time a component sends a message to another component it includes the information about the components the sender knows that are connected. This allows the receiver to learn about connected components he didn't know that were connected. b) Time to time, a component randomly selects N^1 components and performs a consistency session with them. During a consistency session between A and B, A tells B which components A knows that are connected to group; B tells A which components knows that are connected to group. The last sub-mechanism related to presence refers to detection of components that are no longer connected to group: when a component (A) hasn't received any message from B for a long period of time², A tries to contact B. If A can't reach B, A removes B from its connected components list.

When an action occurs, an event is generated to inform about the action. Actions can be: new document, new member, document read, or any action that an application wants to disseminate to all members. As was explained in virtual synchronism part of the section 3.2, events are used to provide awareness information to members, but are also used to guarantee the internal consistency of the system. The dissemination of events mechanism guarantees that all connected components have all generated events in a time that users perceive as immediate.

When a new event is created, the component where the event was created sends it to all components the component knows that are connected. As can be seen, the performance of this mechanism is strongly related with presence mechanism. All RA store in a persistent manner all received events. Components not connected to the group or components that the sender of the event doesn't know that are connected to the group will not receive the event. To overcome this, a consistency sub-mechanism is implemented. Event's consistency mechanism is based on an adaptation of Golding's Time-Stamped Anti-Entropy algorithm [6] and is performed between an UA or RA and an RA. Consistency sessions among RA are as follows: time to time, an RA (RA1) randomly selects another RA (RA2) among the RA that knows are connected to group. Then, RA1 sends to RA2 a summary of all events that RA1 has received. RA2 sends to RA1 all events that RA2 has and that RA1 doesn't have along

¹ $\text{Max}(2, \log_2(\text{numberConnectedComponents})+1)$. This number was adjusted by simulation.

² This period of time is a parameter that can be adjusted. Component A can also know about component B through some other component (C). In that case, either by presence sub-mechanism a) or b) C has informed A that B was still connected.

with the summary of all events that RA2 has. Finally, RA1 sends to RA2 all events that RA1 has that RA2 doesn't have. Similarly, in the case of consistency sessions between UAs and RAs, the UA asks an RA for events that the UA doesn't have but UA never provides new information to RAs.

Events mechanism doesn't provide any order guarantee. Events mechanism provides immediateness (events are sent by originator to all other connected components right after the event is created) and consistency (consistency sessions guarantee that not connected components or components that haven't received the event will eventually receive it³). In the case some ordering guarantees were required, applications should provide them. As future work we are considering to implement some event's ordering policies in top of LaCOLLA and provide them to applications.

Events and presence mechanisms are in the basis of all other mechanisms. For example, objects mechanism is in charge of storing, retrieving, removing and guaranteeing the availability of objects stored in LaCOLLA. When an object is stored in LaCOLLA, the object is send to any RA and an event is disseminated to all components to inform about the new object and its location. The event will be used by any component to know where is located the object. When an UA wants to retrieve an object, the UA knows where the object is located (some moment in the past, the UA received and event informing about the location of the object). Consequently, it obtains the object from any of its locations. To guarantee the availability of objects, they are automatically replicated in a decentralized manner. Every time a new replica is created, an event is disseminated to inform about the availability of new replica and its location.

Other mechanisms also combine push, pull and autonomous decision behaviors as it has been explained for presence and events mechanisms. Even though the push behavior is frequently used, neither components nor the network are saturated because groups are usually small.

This combination of autonomy of components and direct communication among them (in a peer-to-peer manner) along with the common ownership of resources provides a flexibility that suits the idiosyncrasy of our groups.

4 Validation

As said in section 3.4, LaCOLLA middleware implements the functionalities presented in this paper. We also adapted and implemented some collaborative applications (an instant messaging tool, an asynchronous forum, and a document sharing tool) that benefit from LaCOLLA. These realistic applications helped us to improve the architecture and implementation of LaCOLLA. We have done limited tests with a number of ad-hoc users. All these tests confirm the usefulness of LaCOLLA. The next step is going to be to extend the functionalities of the applications we developed and use them in regular university courses at UOC.

Before implementing LaCOLLA middleware, a simulator was implemented to validate the proposed architecture under several realistic scenarios. The simulator

³ Implemented variant of TSAE algorithm used in events' consistency sessions [5, 6] ensures that.

used J-Sim [12] as network simulator and implemented the UA, RA and GAPA components, virtual synchronism and the internal mechanisms necessary to prove that LaCOLLA behaves, as expected, in an autonomous, decentralized and self-sufficient manner.

Several experiments were done with synthetic workloads with different degrees of dynamism (failures, connections, disconnections or mobility), with different sizes of groups (from 5 to 100 members) and with different degrees of replication (number of RA and GAPA). All components were affected by dynamism.

Simulations had two phases. The first phase simulated a realistic situation. In that phase all internal mechanisms were operative. During this phase members' activity was simulated and components connected, disconnected, moved or failed. The second phase was called repair phase and only internal mechanisms were active. This second phase was used to evaluate how long LaCOLLA required achieving: a) self-organization: all connected components have consistent the information about all internal mechanisms, b) virtual synchronism: all connected components have all events and have consistent the information about available objects, c) presence and location: all connected components have consistent the information about presence and location.

Experiments showed that, in spite of the dynamism and the autonomous and decentralized behavior of components, LaCOLLA required short amount of time (with respect to the rate of changes) to update the information referring to internal mechanisms in all components. Experiments also showed that members knew what was happening in the group and that they had access to the latest versions of objects in a time they perceived as immediate [5].

Figure 4 shows the time required by LaCOLLA (depending on group size) to be self-organized, to provide virtual synchronism, and to have consistent information about presence and location. Note that, for groups of typical size (10 members),

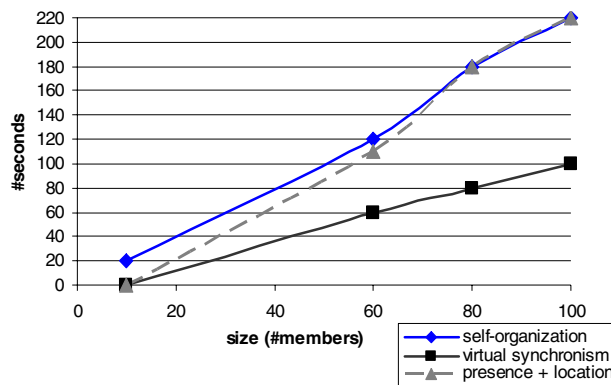


Fig. 4. Simulation results. The figure shows the time required a) to be self-organized, b) to have consistent all information related to virtual synchronism (events and objects) and c) to have consistent the information related to presence (presence and location).

LaCOLLA has good performance: it requires 20 seconds to self-organize, and less than 10 seconds to provide virtual synchronism. It deserves special attention the fact that, even though all components don't have all consistent information about internal mechanisms (self-organization), connected members know all what is happening in the group and have access to the last version of objects (virtual synchronism) in a time that they perceive as immediate. This is due to the decentralized implementation of internal mechanisms and to the fact that non-key mechanisms have long-term consistency policies. In this figure it is also plotted the time required to have consistent presence and location mechanisms because they have a great influence in the achievement of self-organization.

When the size of groups increases, the required time grows, but still maintains low enough values for asynchronous collaboration (e.g. with 60 members: self-organization takes 2 minutes, providing virtual synchronism in 1 minute). This also proves that LaCOLLA can be used in situations where quite large groups require asynchronous sharing capabilities. These values will be further adjusted based on experience with real users using the current middleware with specific applications.

5 Conclusions and Future Work

Asynchronous collaborative applications have to adapt to group idiosyncrasy and interaction style and support the formation of ad hoc collaborative environments for people willing to cooperate using only their own computers, without any additional computing resources (i.e. servers). This requires the autonomy and self-sufficiency that peer-to-peer networks can only offer. We have identified groups as units of resource sharing, by which several individuals dispersed through Internet may spontaneously start to collaborate by just sharing their own computers to form an independent ad hoc community.

In this paper we have described the general characteristics and properties of LaCOLLA, a decentralized, autonomous and self-organized middleware for building collaborative applications that operates with resources provided by their members, that adapts to the idiosyncrasy and to the interaction nature of human groups, and that allows execution of tasks using resources belonging to the group. We also presented the details of current LaCOLLA middleware implementation, paying special attention to its API.

From both building collaborative applications that use LaCOLLA and from using the developed applications we obtained valuable ideas and improvements to introduce in the next versions of LaCOLLA. These new versions will pay special attention to security issues, which are at its minimum expression in the first version; and to introduce new components and mechanisms that will allow mobile devices (PDA, mobile phone, sensors, etc.) become LaCOLLA peers.

We are also planning to use LaCOLLA in real collaborative settings. In that sense, we are planning to use collaborative applications that use LaCOLLA middleware in some collaborative learning practices at UOC. UOC is a virtual university that mediates all relations between students and lecturers through Internet. We think that this kind of collaborative environments where participants never physically meet one to each other will benefit from approaches like the one provided by LaCOLLA,

specially for the degrees of autonomy and self-sufficiency that can be achieved. These real experiences will be of great value for us to further refine the architecture and adjust the implementation of the middleware.

Acknowledgements

Work partially supported by MCYT-TIC2002-04258-C03-03.

References

1. Foster, I.; Kesselman, C.; Tuecke, S. (2001). The Anatomy of the Grid Enabling Scalable Virtual Organizations. Lecture Notes in Computer Science.
2. Navarro L., Marquès J.M., Freitag F. (2004). On distributed Systems and CSCL. The First International Workshop on Collaborative Learning Applications of Grid Technology (CLAG 2004). Held in conjunction with the IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2004). April 19 - 22, 2004, Chicago, Illinois, USA. <http://www.ccgrid.org/ccgrid2004>.
3. Bote, M., Dimitriadis, Y., Gómez-Sánchez, E. (2003) Grid uses and characteristics: a grid definition. In Proceedings of First European Across Grids Conference, 2003.
4. Hurwicz, M. (2001). Groove Networks: Think Globally, Store Locally. Network Magazine. May 2001.
5. Marquès, J.M. (2003). LaCOLLA: una infraestructura autònoma i autoorganitzada per facilitar la col•laboració. Ph.D. thesis, <<http://people.ac.upc.es/marquès/LaCOLLA-tesiJM.pdf>>
6. Golding, R.A. (1992). Weak-consistency group communication and membership. Doctoral Thesis, University of California, Santa Cruz.
7. Saito, Y.; Shapiro, M. (2002). Replication: Optimistic Approaches. Technical Report HPL-2002-33, HP Laboratories, 2002. <<http://www.hpl.hp.com/techreports/2002/HPL-2002-33.html/>>.
8. Carter R. L. (1995). Dynamic server selection in the Internet. In Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95).
9. Verbeke, J.; Nadgir, N.; Ruetsch, G.; Sharapov, I. (2002) Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. Manish Parashar (Ed.): Grid Computing, Third International Workshop, Baltimore, USA. LNCS 2536 Springer 2002, ISBN 3-540-00133-6. <<http://jngi.jxta.org/>>
10. JXTA: <http://www.jxta.org/>. An overview paper: L. Gong. Project JXTA: A Technology Overview, 2001. < <http://www.jxta.org/project/www/docs/TechOverview.pdf>>.
11. LaCOLLA: <http://lacolla.uoc.edu/lacolla>
12. J-Sim: <http://www.j-sim.org>