LaCOLLA: Middleware for Self-Sufficient Online Collaboration

The LaCOLLA middleware makes it possible for collaborators to interact using their own resources without depending on centralized regimes. By contributing their own resources, group members can organize and communicate using a federated peer-to-peer model. This lets the group function regardless of whether a member removes resources and despite network or node failures or disconnection. In turn, this capacity for self-organization, together with location transparency, lets application developers create self-sufficient applications for collaborative activity.

Joan Manuel Marquès, Xavier Vilajosana, and Thanasis Daradoumis
Open University of Catalonia

Leandro Navarro
Technical University of Catalonia

When a group of people wants to work together over the Internet, it must agree on a tool set, including general communication tools (such as email or instant messaging), and whether the group will require user registration or application installation. In any case, the tools will use resources or servers that the collaborating participants don't administer. Although it's appropriate for formal groups (possibly within companies) to use external resources, more spontaneous groups — which might include informal school associations, people with similar professional or personal interests, and campaigns for social and political activists — are unlikely to have support entities that automatically and transparently guarantee the necessary resources at all times. These group members must thus collaborate by using applications that only partially meet their needs (such as email), by having a few members manage resources for the whole group, or by paying for third-party resources or accepting advertising.

To promote such online collaborations, we designed the LaCOLLA middleware to let participants in collaborative activities self-organize using only the resources they themselves provide. (*Colla* is a Catalan word that means a group of people joined together freely with the aim to work or act on a certain task.) LaCOLLA

1089-7801/07/$25.00 © 2007 IEEE

is based on the components' decentralized and autonomous behavior and uses algorithms based on *epidemic propagation of information*, which lets any two sites that happen to communicate exchange their local information as well as information they received from a third site, optimistic object replication, or randomization.

LaCOLLA is a peer-to-peer grid optimized for collaborative interaction and resource sharing. Once the participants provide the computational resources (storage and processing) and applications that users need to carry out the group's activities, LaCOLLA manages them in such a way that any resource owner can disconnect them at any time without warning and without affecting the overall group's functionality. The proposed middleware then guarantees, despite intermittent node and network availability, that the system can self-organize without requiring the participants to manually intervene. In the end, collectivism exists if the participants ensure the group's self-sufficiency by providing sufficient resources. The middleware and some demo applications (with instant messaging and a basic desktop) are available at http://lacolla.uoc.edu/lacolla.

Collectivism and Online Collaboration

We can use different types of distributed architectures to implement the LaCOLLA middleware's functionality. A client-server architecture, as in a Web 2.0 or Asynchronous JavaScript and XML (Ajax) application, is the simplest to manage. However, this paradigm's centralization introduces asymmetry in the group (by giving a position of strength in the group to the resource owner who contributed the key resources), dependency on resources from third parties (given that the group uses external servers instead of sharing its own), and technical limitations (such as points of failure). LaCOLLA overcomes these limitations by supporting self-sufficiency, availability, and self-organization.

In the P2P model,[1] communication is carried out directly among participants who form a network in which they all act as servers and clients. This model has been successful in communities based on individual interests, centered on sharing based on competition, rather than collaboration. In collaboration, and in small groups in particular, it's important that shared information be available at all times.

Grid-based systems[2] present a third approach that focuses on virtualizing resources and providing them to the community. These solutions increase individual capabilities but don't optimize interaction patterns in group collaborations or help manage resources by applying group policies.

Instead, we propose a distributed systems paradigm based on the idea of *collectivism*, which emphasizes the participants' explicitly providing resources for the group's benefit.

We designed LaCOLLA's middleware to maintain group members' freedom; they can decide to disconnect resources from the group at any moment. While a resource is within the group, however, the group is in charge of its administration and the group policies apply. Resources provided to a group are used transparently — that is, users don't know which resources they're using to carry out an action. Thus, members carry out their actions with group resources in line with group policies.

Contributing resources "for the benefit of the group" makes sense in environments in which participants share more than just resources, values, or a common goal (for example, groups within a company, student projects, groups with researchers from different organizations, and nongovernmental organizations [NGOs]). A level of trust among this type of group's participants is required. Our collective model isn't designed to support groups of people who don't know one another or who don't share a spirit of mutual collaboration. Such cases would require additional mechanisms to control access and resource use.

Middleware Requirements

We designed LaCOLLA to support collaboration in collective environments because currently available systems can't sufficiently handle the requirements for self-sufficient communities. (See the "Related Work in Collaborative and Distributed Systems Architectures" sidebar for specifics.) The LaCOLLA middleware, on the other hand, can support asynchronous and synchronous-like collaborative scenarios and applications. The middleware specifically deals with

- different aspects of interaction, such as dispersed participants, many-to-many collaboration, participation at different times, and individuals participating from different locations at different times;
- group idiosyncrasies, such as flexibility,

Related Work in Collaborative and Distributed Systems Architectures

We can compare certain systems to LaCOLLA. Groove (www.groove.net) is a peer-to-peer system that lets users create and share workspaces on their local PCs, collaborating freely across corporate boundaries and firewalls, without the permission, assistance, or knowledge of any central authority or support group.[1] Groove is based on synchronizing these workspaces, but it doesn't manage the group's resources collectively as does LaCOLLA.

JXTA (www.jxta.org) is a generic platform for supporting the development of peer-to-peer applications.[2] More precisely, it is a lower-level middleware that offers generic mechanisms to discover and connect to peers. In contrast, LaCOLLA is a middleware designed to facilitate the construction of applications for collaborative activities based on optimistic propagation of events and weak-consistency, which is adequate in a dynamic and partially disconnected environment.

Referenes

1. M. Hurwicz, "Groove Networks: Think Globally, Store Locally," *Network Magazine*, May 2001.
2. I. Gong, "JXTA: A Network Programming Environment," *IEEE Internet Computing*, vol. 5, no. 3, May/June 2001, pp. 88–95.

dynamism, decentralization, and participants' autonomy; and
- technical and administrative issues, such as information availability, interoperability among applications, security issues, and participants from different organizations with different administrative authorities.

In addition, we designed our middleware to offer collective self-organization. Thus, we identified the following requirements:[3]

- *Decentralization*. No component is responsible for coordinating other components, and no information is associated with any single component. Centralization leads to simple solutions, but critical components restrict participants' autonomy.
- *Self-organization*. The system should be able to function automatically without external intervention. This requires the ability to reorganize its components spontaneously when faced with failures or dynamism (connection, disconnection, or mobility).
- *Group-oriented*. The group is the unit of organization.
- *Group availability*. The group should continue to operate if some components malfunction or become unavailable. Replicating objects, resources, or services can help improve availability and quality of service.
- *Individual autonomy*. The group's members should be free to decide which actions to carry out, what resources and services to provide, and when to connect or disconnect.
- *Self-sufficiency*. A group must be able to operate with only the resources its members provide.
- *Sharing*. Information generated in the group (such as events, objects, and presence information) can be available to multiple applications.
- *Security*. A group must guarantee the identity of its members and guarantee selective and limited access to shared information by protecting information and authentication.
- *Internet scale*. A group is formed by several components (distributed). Members and components can be at any location (dispersed).
- *Scalability*. The number of groups is unlimited because each group uses its own resources.
- *Universal and transparent access*. Participants should be able to connect from any computer or digital device, with a connection-independent view.
- *Transparency of object and member location*. Applications should use location-independent identifiers so the location of the group's objects or members isn't significant.
- *Disconnected operational mode*. A component should be able to operate without being connected to the group.

Middleware Design

Based on the literature and our experiences with collaborative environments, we decided that LaCOLLA's functionalities should focus on awareness,[4] group communication and coordination, storage, and computational resources. As such, we provided the following:[3]

- *Communication and coordination by disseminating events*. An event is spread to connected components when an action occurs or when provided by applications to LaCOLLA. Disconnected members receive events during the reconnection process. This dissemination of events helps applications provide awareness to members.

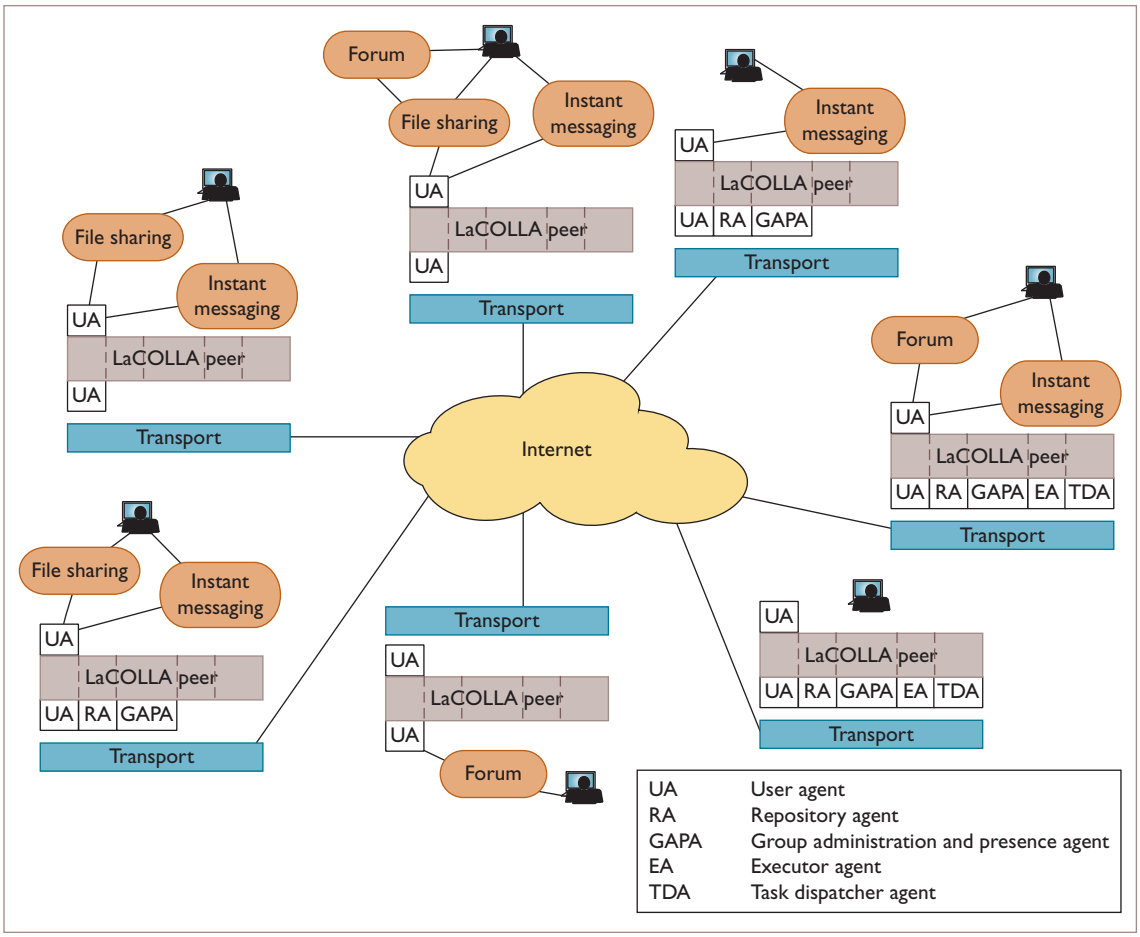

Figure 1. Snapshot of a collaborative group using applications connected to LaCOLLA. Every user decides autonomously which resources to contribute to the group. Collaboration will occur using only those components.

- *Object storage.* Objects represent any entity that can be stored in the resources that the members provide to the group. LaCOLLA guarantees that objects are replicated in a manner that ensures their availability, even if some members decide to disconnect the repositories they've provided the group.
- *Task execution.* Any application can submit tasks to be executed by computational resources provided to LaCOLLA.
- *Presence.* Presence information indicates which components and members are connected to the group.
- *Location transparency.* LaCOLLA resolves the location of objects and members internally so that applications don't have to know this information.
- *Instant messaging.* LaCOLLA lets applications send messages to subgroups of members.
- *Managing groups and members.* Users can add, delete, or modify information about members or groups.
- *Disconnected mode.* LaCOLLA lets applications operate offline. During reconnection, the middleware automatically propagates and synchronizes changes.

We implemented this functionality in a manner that satisfies the previous list of requirements. Security is the only requirement we weren't able to fully implement. We're exploring how to provide decentralized registration and authentication without depending on external entities.

Architecture

As Figure 1 illustrates, people collaborating using LaCOLLA must install a peer application on their computers. The participants decide which components to install based on their degree of involvement in the group as well as their computers'

capacity and availability. The LaCOLLA architecture[3,5] consists of five kinds of component:

- *user agents* (UAs) interact with applications and represent the participants connected;
- *repository agents* (RAs) persistently store objects and events that participants generate;
- *group administration and presence agents* (GAPAs) are in charge of administration, managing information about the group and its members and authenticating members;
- *executor agents* (EAs) execute tasks; and
- *task dispatcher agents* (TDAs) distribute tasks to executors.

(If all executors are busy, the TDAs queue tasks. They also ensure that tasks are executed even if the UA and member disconnect.)

Components interact with each other autonomously. Internal mechanisms coordinate the components connected to a group. We've grouped these internal mechanisms[5] together as events, objects, tasks, presence information, location information, groups, members, and instant messaging. We implemented them by combining multicast, weak-consistency optimistic protocols, and random-decision techniques.

Internal Mechanisms

LaCOLLA's three key mechanisms are *presence*, which lets components and applications know which components and members are connected to the group; *events*, which let the UAs receive events generated in the group (and pass them on to the connected applications); and *objects*, which store objects in the group.

We can assume that collaboration is taking place in small groups, and Internet users are increasingly connecting over higher-capacity connections such as DSL or cable. Therefore, it's also reasonable to assume that we can use a decentralized mechanism to manage component presence in this collective environment. Our implementation of this mechanism is based on *epidemic dissemination*[6] of information, a common technique in optimistic replication.[7] Any message a component sends includes the list of components the sender knows are connected to the group. Those receiving the message learn of the components connected to the group. This mechanism provides each component connected to the group with a fairly precise view — with statistical guarantees — of all the connected components (see the "Validation" section for specifics).

One risk with optimistic designs is that some components might have imprecise views, and some might take a long time to learn about changes. For this reason, we accelerated the epidemic dissemination with an application-layer multicast mechanism through which a component multicasts a message to inform the group it has (re)connected. Given that our work involves small groups, we should be able to use multicasts without overloading the network or components.

This optimistic view is perfectly acceptable from the end user's perspective because the applications involved allow for asynchronous and synchronous-like collaborative activities, for which the participant doesn't need an exact view of the presence information.

The components use a soft-state or leasing technique to find out which are no longer connected. Each time a component sends a message, the receivers update a timer to indicate that the component is connected. When the timer expires, the component assumes that the other component has failed and removes it from its list. Decision making based on the information available to the component itself simplifies the management of the presence mechanism. Once again, the design is optimistic. If component C1 deletes its information about component C2, which is connected, the epidemic propagation allows, after a short period of time, that C1 will find out that C2 is still connected.

Multicast and epidemic dissemination are also the basis for the *events* mechanism. An event can inform the application that a member has created or read a document, that a member has been added, or any other event that's worthy of being sent to the group. Applications can also use events to coordinate the different application instances. The components multicast new events to the UAs and RAs connected to the group. The UAs pass them on to the connected applications, whereas the RAs store them persistently. The UAs and RAs that haven't received a given event during the multicast will eventually receive it in some consistency session carried out with an RA. Consistency sessions take place at two times:

- when the component connects to the group, which lets it learn of the events that have taken place while it was disconnected, or
- every so often once the component is connect-

ed, so it receives events that it hadn't received when the originator multicast them.

In both cases, the component carries out the consistency session with an RA chosen randomly from among those that it knows are connected. The consistency sessions implement a variant of the Time Stamped Anti-Entropy protocol designed by Richard A. Golding.[8] This is an optimistic mechanism for disseminating information in which each site periodically contacts another and the two sites exchange information until both have the same information or events.

LaCOLLA currently provides a causal ordering on the generated events. If an application requires stronger ordering guarantees, it should implement them itself based on the event-dissemination API calls that LaCOLLA provides. In that sense, a general-purpose application-level module should implement those stronger ordering guarantees and provide them to applications.

The event's approach occurs in two ways. On the one hand, components multicast new events to all components the sender knows are connected. On the other, an epidemic mechanism acts in the background and ensures that all the components receive all the information. The multicast rapidly disseminates the group's coordination information. This helps avoid conflicts and gives applications a view that closely resembles the actual situation. The epidemic mechanism is slower, but it ensures convergence of all the components in terms of their group views.

That all the UAs and RAs in LaCOLLA receive all the events simplifies the object-management mechanism. A UA can directly obtain an object from any of its locations. Previously received events provided that information. The objects mechanism must simply ensure that the group has at least a replication-factor number of replicas of any object — that is, a minimum number of replicas of every object that LaCOLLA should have to ensure its availability. Thus, we reduce the objects mechanism to a series of repositories that ensure all the objects are replicated a minimum number of times.

Replication ensures availability of objects despite node departure and network failures. When an RA (RA1) detects that one of the objects it has stored locally is replicated less than replication-factor times, RA1 replicates it in any other RA.

Table 1. Probability of and duration range for dynamic behaviors (per step).

Agent	Failure		Disconnections		Mobility
	Probability	Interval (in steps)	Probability	Interval (in steps)	Probability
User agents	0.0005	[15, 120]	0.0025	[60, 540]	0.00035
Repository agents and GAPAs*	0.000125	[12, 60]	0.0005	[15, 120]	0.0001

*Group administration and presence agents

Validation

We implemented LaCOLLA middleware together with some collaborative demo applications to test its usefulness. To validate LaCOLLA's functionality, we created a module to simulate user behavior (generating events, instant messages, and operations) and have modified components to simulate dynamic behavior (connections, disconnections, failures, and mobility [change of location]).

We carried out the tests with 10 RAs, 10 GAPAs, and between 10 and 80 UAs. For the tests, we assumed that a member would connect from only one UA. We used simulation time (steps) for the tests rather than real time. To obtain significant results, we repeated each experiment 150 times.

We divided each experiment into two phases. The first phase involved user activity, component dynamism, and internal mechanisms. We then simulated the system's dynamic behavior. Table 1 shows the probability per step that a failure, disconnection, or mobility will occur. We also simulated the user's workload while collaborating using the following data:

- events were 0.025 likely to occur in each step;
- objects, 0.015; and
- instant messages, 0.025.

There was a 0.025 probability that no activity would occur (random with a [60, 540] interval). Each experiment in this phase involved approximately 3,000 steps, which corresponds to a rather long session resembling real collaborative scenarios.

The second phase involved only the internal mechanisms (repair); we evaluated the number of steps it took LaCOLLA to achieve consistent information in all the components. Figure 2a shows, in number of steps after entering the second phase, the

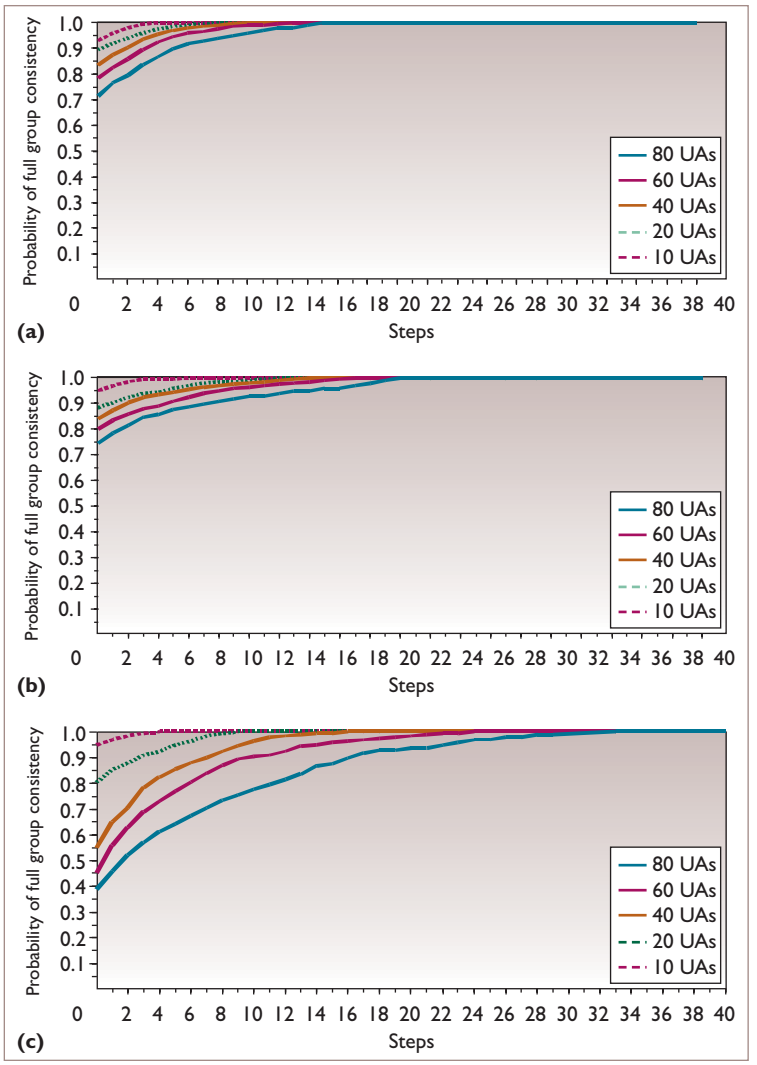

Figure 2. Cumulative probability that in N *seconds (a) information about events is consistent in all components, (b) information about objects in all components is consistent and objects are replicated at least replication-factor times, and (c) information about presence and location is consistent in all components for between 10 and 80 user agents (UAs).*

cumulative probability that all the components have received all the events. LaCOLLA is designed for groups of 10, 20, or 30 members. In groups with 10 UAs, more than 93 percent of the components had consistent event information when the first phase ended (0 steps in the second phase); 100 percent of components were consistent by the third step of the second phase. For 20 members, these values were 89 percent and the sixth step, respectively.

As the number of UAs increases, the performance decreases almost linearly. Despite this, LaCOLLA converges in a small number of steps (for 80 UAs, 95 percent in nine steps and 100 percent in 14 steps). Translating simulation time to real time, LaCOLLA provides users with a fairly precise view of the group's state with little overhead. Time per step is a trade-off between system load and *immediateness*. (In this context, immediateness refers to a time that users perceive as immediate enough for an asynchronous collaboration.) When actions occur, information is immediately multicast to the group. In the case of an incidence such as a network failure, the system also converges (consistency sessions) in a few steps.

Figure 2b shows how the components gain a consistent view of the objects after a small number of steps. Likewise, Figure 2c shows the cumulative probability for presence information. We should stress that the optimistic design we used to implement LaCOLLA implies that convergence occurs at different rates. Despite this, convergence appears adequate, and the consistency sessions rate can be adjusted by users to guarantee that members can perceive the effect of their actions according to each application's requirements.

Finally, LaCOLLA can achieve system convergence at a low cost in terms of the time and space resources used to attain synchronization and availability of information on various nodes. We can keep the cost low for small groups that require short messages for the presence information exchanged. In the current LaCOLLA release, consistency sessions are carried out periodically (approximately two or three per minute) and are distributed fairly uniformly among components — a component does a consistency session with another selected randomly from among all known components. The weak-consistent optimistic approach introduces a low overhead compared to pessimistic approaches and avoids the bottlenecks and points of failure of centralized solutions.

Implementation and API

LaCOLLA middleware has an open source license and is written in Java. Tables 2 and 3 show the API between the applications and LaCOLLA. Communication among LaCOLLA components is carried out through TCP sockets. When an application wants to log in, send an event, or get an object, it invokes the appropriate API function in the UA. Similarly, the UA can send notifications to the application.

The current LaCOLLA release implements the functionality we present in this article. A proto-

Table 2. Functions that LaCOLLA user agents offer to applications.

Category	Function	Description
Presence	login	Connects user to the group
	logout	Disconnects user from the group
	whoIsConnected	Indicates which members are connected to the group.
Events	disseminateEvent	Sends an event to all applications belonging to the group
	eventsRelatedTo	Indicates which events have occurred to a specific object
Objects	putObject	Stores an object in LaCOLLA
	getObject	Obtains an object stored in LaCOLLA
	removeObject	Removes an object stored in LaCOLLA
Tasks	submitTask	Submits a task to be executed by computational resources belonging to the group
	stopTask	Stops a task
	getTaskState	Gets the task's state
Instant messaging	sendInstantMessage	Sends a message to specified group members
Groups	addGroup	Creates a new group
	removeGroup	Removes a group
	modifyGroup	Modifies a group's properties
	getGroupInfo	Gets information about the properties of a group. (see groupInfo function)
	getGroupInfoSync	Gets information about the properties of a group synchronously (function doesn't return until the operation is completed and a result is available)
Members	addMember	Creates a new member
	removeMember	Removes a member
	modifyMember	Modifies a member's properties
	getMemberInfo	Gets information about the member's properties

Table 3. Functions that applications should offer to LaCOLLA user agents.

Category	Function	Description
Presence	newConnectedMember	Notifies that a new member has connected
	memberDisconnected	Notifies that a member has disconnected
Events	newEvent	Receives an event that occurred in the group
Tasks	taskStopped	Notifies that the task has been stopped correctly
	taskEnded	Notifies the end of a task
Instant messaging	newInstantMessage	Receives a new instant message
Groups	groupInfo	Receives the group information
Other functions	exception	Notifies that an internal exception or anomalous situation has occurred
	appIsAlive	User agent queries the application state to see if application is alive and connected to group

type application that uses LaCOLLA in a real situation showed that people can tolerate certain degrees of inconsistency, but extensive collaborative use cases are certainly needed to characterize this further. We're developing additional collaborative applications that use LaCOLLA and are looking to gain experience with real users at the online Open University of Catalonia.

Ongoing work is focused on deploying services using computational resources provided to the group and on exploring how to provide registration and authentication in a decentralized manner without depending on external entities. Finally, we're studying the feasibility of extending LaCOLLA to support radically decentralized and limited-capability devices, such as PDAs, mobile phones, and location and presence sensors.

Acknowledgments

The Spanish Ministry of Education partially supported this work under grant TSI2005-08225-C07-05.

References

1. R. Steinmetz and K. Wehrle, *Peer-to-Peer Systems and Applications*, Springer-Verlag, 2005.
2. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Source Int'l J. High Performance Computing Applications*, vol. 15, no. 3, 2001, pp. 200–222.
3. J.M. Marquès and L. Navarro, "Autonomous and Self-Sufficient Groups: Ad Hoc Collaborative Environments," *Proc. CRIWG 2005, Groupware: Design, Implementation, and Use*, LNCS 3706, H. Fucks, S. Lukosch, and A.C. Slagado, eds., Springer-Verlag, 2005, pp. 57–72.
4. C. Gutwin and S. Greenberg, "The Importance of Awareness for Team Cognition in Distributed Collaboration," *Team Cognition: Understanding the Factors that Drive Process and Performance*, E. Salas and S.M. Fiore, eds., APA Press, 2004, pp. 177–201.
5. J.M. Marquès, *LaCOLLA: Una Infraestructura Autònoma i Autoorganitzada per Facilitar la Col·Laboració*, [LaCOLLA: An Autonomous and Self-Organized Infrastructure to Facilitate Collaboration], doctoral thesis, Dept. of Computer Architecture, Tech. Univ. of Catalonia, 2003; http://people.ac.upc.es/marques/LaCOLLA-tesiJM.pdf.
6. A.J. Demers et al., "Epidemic Algorithms for Replicated Database Maintenance," *Proc. 6th Symp. Principles of Distributed Computing* (PODC), ACM Press, 1987, pp. 1–12.
7. Y. Saito and M. Shapiro, "Optimistic Replication," *ACM Computing Surveys*, vol. 37, no. 1, Mar. 2005, pp. 42–81.
8. R.A. Golding, *Weak-Consistency Group Communication and Membership*, doctoral thesis, Univ. of Calif., Santa Cruz, 1992.

Joan Manuel Marquès is an assistant professor at the Open University of Catalonia. His research interests include design of scalable and cooperative Internet services and applications, peer-to-peer systems, and distributed computing. Marquès has a PhD in computer science from the Technical University of Catalonia. His is a member of the ACM and Kaleidoscope Network of Excellence. Contact him at jmarquesp@uoc.edu.

Leandro Navarro is an assistant professor at the Technical University of Catalonia. His research interests include the organization and coordination of peer-to-peer grids and economics-inspired resource-allocation mechanisms. Navarro has a PhD in telecom engineering from the Technical University of Catalonia. His is a member of the IEEE, the ACM, IFIP, the Association for Progressive Communications, and Computer Professionals for Social Responsibility (CPSR). Contact him at leandro@ac.upc.edu.

Xavier Vilajosana is a PhD student at the Open University of Catalonia. His research interests include communication protocols, replication, and autonomous and distributed mechanisms for peer-to-peer systems. He is also interested in economical models for resource allocation in distributed systems. Vilajosana has an MS in computer science and artificial intelligence from the Open University of Catalonia. His is a member of the Kaleidoscope Network of Excellence. Contact him at xvilajosana@uoc.edu.

Thanasis Daradoumis is an assistant professor at the Open University of Catalonia. His research interests include e-learning and learning technologies, ontologies and semantic Web, distributed learning, computer-supported cooperative work, interaction analysis, and grid technologies. Daradoumis has a PhD in computer science from the Technical University of Catalonia. His is a member of the Kaleidoscope Network of Excellence, the International Society of the Learning Sciences, and the International Society of Artificial Intelligence in Education. Contact him at adaradoumis@uoc.edu.