Support for End User Participation using Replicated Versions and Group Communication

Gérôme Canals, Pascal Molli, Claude Godart

Support for End User Participation using Replicated Versions & Group Communication

G. Canals, P. Molli & C. Godart
LORIA
INRIA Lorraine & Université Henri Poincaré Nancy I
Campus Scientifique, BP 239, F54506 Vandoeuvre-lès-Nancy
{canals,molli,godart}@loria.fr

ABSTRACT
We present Tuamotu, a prototype software system for the support of engineering projects distributed over the Internet. This system offers basic services for sharing persistent data between distant activities cooperating in a design project. It is based on replicating versions at each of the participant sites with the use of an atomic multicast mechanism and includes concepts for structuring the cooperation space. The application of these concepts to end-user support in design projects is discussed.

Keywords
Cooperation, version management, replication, group communication, group structuring

INTRODUCTION
Engineering projects are now on the way of being distributed over geographically distant sites interconnected through a large-scale network (e.g. Internet). In addition, they often involve several distinct organizations temporarily glued together in order to work and cooperate as a virtual enterprise. The different partners of a given virtual enterprise generally play different roles corresponding to different organizational or professional competencies, including those of the intended end-users which can provide some feedback all along the development of a product.
Our goal is to provide effective support to such "*enterprise-project*" applications, i.e. applications that are settled around a given project and for the duration of the project. We particularly target telecooperative applications that can be modeled as a network of distributed cooperative activities carried out in parallel by several participants in order to reach a common objective. We assume also that coordination and the interactions between the different activities are (at least partly) governed by specific and explicit project policies that describe process models, integrity constraints or general organizational rules. Our approach is based on the definition and construction of a generic infrastructure offering an integrated package of services and facilities for the building of such applications. This infrastructure should be very light, easy to install and to deploy and must make very few assumptions about the underlying hardware and software system. The system should work typically on desktop or laptop PCs connected to the Internet through a modem and an Internet provider; efficient support to disconnected work in thus a strong requirement in this context. Finally, the common tools of any particular domain should be usable in the system. Every engineering project uses standard software (e.g. text processors, spreadsheets...) plus project specialized tools (e.g. CAD software, project management tools...).
This paper shortly presents the TuaMotu prototype, our first step towards a generic infrastructure for telecooperative engineering applications. This prototype is intended to provide the basic services to share data between distributed cooperative activities interconnected through the Internet. It includes persistent objects management services, object sharing and cooperation management services, and uses an original replication mechanism to render efficient the evaluation of global cooperation and coordination policies at the different sites.
After a quick overview of our system in the next section, we present its architecture and the concepts we use to organize the cooperation space. Then, we discuss how this can be used to provide support to end-user participation in different stages of a development project.

OVERVIEW
We focus in this paper on services for *indirect, asynchronous cooperation*, i.e. cooperation that occurs through the sharing by different activities of some products or artifacts of the project. In this model, cooperation is provided by a distributed copy / modify / merge paradigm. Different activities at different worksites share objects by owning *copies* of these objects. They can thus *modify* these copies in parallel without any interaction with the others. When needed, transferring from one site to the other and then merging the corresponding copies can reconciliate parallel modifications at different sites of a shared object. We think that this model is well suited in our context due to

the nature of the activities we want to support. Indeed, in engineering activities, participants need to have isolated periods of work followed, at their own initiative, by periods of work dedicated to sharing and exchanging data and information with the others. This particular form of cooperative work can be qualified as *insulated work*.

From an operational point of view, this model is supported through the use of versions: the different copies of a shared object correspond to different versions of that object. Each worksite owns a particular version branch for each of the objects it shares with other worksites. This branch is used to store the local modifications that need to be stabilized as a new version, while remote values of a shared object are locally stored as versions in separate branches. A merge operation is thus explicitly represented by the creation of a version having two predecessors in two distinct branches.

Finally, the system provides a replication service that is used to maintain up-to-date versions of the shared objects at the different sites. Thus, the global state of the cooperation space is locally available at each participant's site. This renders very easy and efficient the evaluation of some cooperation policies or coordination rules (e.g. [1,2]). Replication is done on a *per-branch* basis. For each shared object, a motu maintain two distinct sets of version branches: local branches that store the local modifications, and read-only branches that store remote modifications. Read-only branches are in fact *replicas* of the local branches of each remote motu. Of course, object contents are not automatically replicated, but are transferred on a *per-demand* basis. Only object descriptions (versioning information, attributes…) are replicated. Replica consistency is ensured by the use of an atomic multicast protocol for message dissemination.

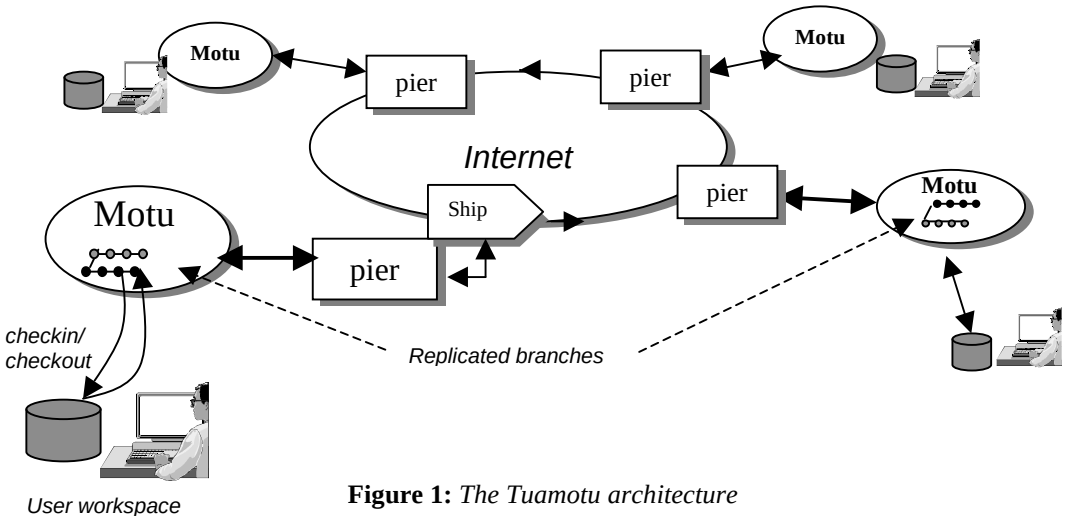

Figure 1: *The Tuamotu architecture*

ARCHITECTURE

The system is organized in a two-level architecture depicted in figure 1.The first level consists in a peer-to-peer set of interconnected data managers, called Motus[1]. Motus are in charge of data and version management services and communicate with the other motus to provide the services related to object sharing. A motu accesses the Internet through a peer. Peers provide reliable communication services including the atomic multicast protocol and can store ingoing messages to render unnecessary a permanent connection between motus and their peer. The second level allows activities and users to access the motu services in a client/server manner. Motus act as data and communication servers whose clients are the private workspaces of the different users. Users check out objects from the motu server into their private workspace before to modify them using their usual tools. Objects can then be checked in into the motu; this operation produces a new version. Communication services offered by the motu apply only on object versions residing inside the motu. Thanks to this approach, users keep a total control on what happens in their workspace and on the visibility of their own modifications. The atomic multicast protocol used by the replication service offers the guaranty of a reliable and totally ordered delivery of messages. It is implemented by organizing the set of motus in a virtual token ring on which circulates a mobile agent (a ship) that holds a FIFO queue to transport messages. When the ship reaches a pier, it downloads the messages it holds in the FIFO order and delivers them in that order to the local motu, and then uploads messages from the local motu by appending them to the FIFO queue. It then reaches the next pier in the ring. The total delivery order thus corresponds to the appending order of

[1] A *Motu* is a small island in Polynesia

messages in the ship queue [3]. Note that this delivery order maintain the causal dependencies between the messages.

STRUCTURING THE COOPERATION SPACE

We argue that concepts and mechanisms for structuring and organizing the cooperation space are necessary. Indeed, the sharing of objects and their visibility need to be controlled. We think that the fact that a given object is shared must be explicitly declared and it must be possible to specify the sites on which it is accessible. To that order, we introduce the notion of sharing group that consists of a set of motus with a set of objects shared by these motus. Sharing groups are explicitly declared and define the visibility scope of users/activities: a user connected to a motu can access remote versions of some objects provided that the local and remote motu and the objects belong to the same group. For example, in figure 2, a user connected to motu M1 can only access remote versions of object ■ from M2 and M3, but not from M5. Note that the same object may belong to multiple distinct groups and that different groups may share one or more members. For example, M3 belongs to Group1 and Group2 with the ■ object and to Group3 with the ⬭ object.
Interactions between groups are managed by motus on a per-branch basis, as depicted in figure 2 (right). When a motu creates or joins a new group for a given set of objects, it must specify a local version branch of that object dedicated to that group. Of course, the same

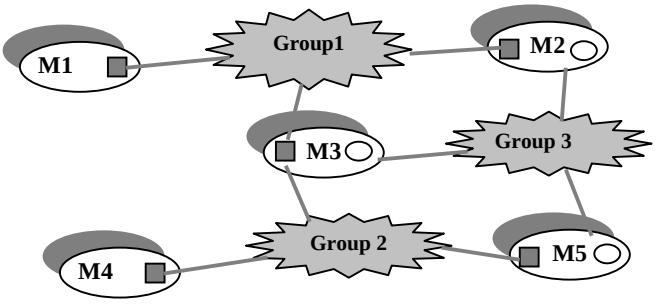

Figure 2: *Structuring the cooperation space*

branch may be used for different groups. Thus, when a new version is created, the user can select the groups in which it will be visible by specifying the branch where the version is created. For example, a version created in branch 1 by merging a local value with a remote one from Group1 will be visible in Group1 and Group2 but not in Group3. In the same way, users can control the propagation of object versions from groups to groups just by transferring versions from a branch to another.
This is depicted in figure 3 where a remote version in Group3 is transferred to the local branch2 and then transferred to local branch1. This last operation renders it visible in Group1 and Group2.
Of course, this notion of sharing group is also used to organize replication: a local version branch belonging to one or more groups is replicated only on the motus belonging to the same groups. This is done by simply providing group memberships to the multicast protocol.

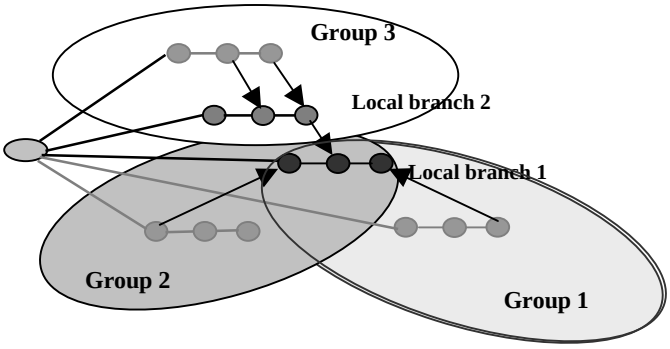

Figure 3: *Multiple groups in a Motu*

SUPPORTING END-USERS PARTICIPATION

Although it has not been designed for that explicit purpose, TuaMotu provides useful facilities for supporting user participation in design projects.
Among others, we particularly think about cooperation space structuring and replication.
The different mechanisms Tuamotu provides in order to organize the cooperation space allow the specification of different group organizations and an efficient control on the objects propagated from one group to another. These features can be used by a project to closely involve different classes of participants and potential end-users, while keeping a clear separation between the project development space and the user participation space (see figure 4).

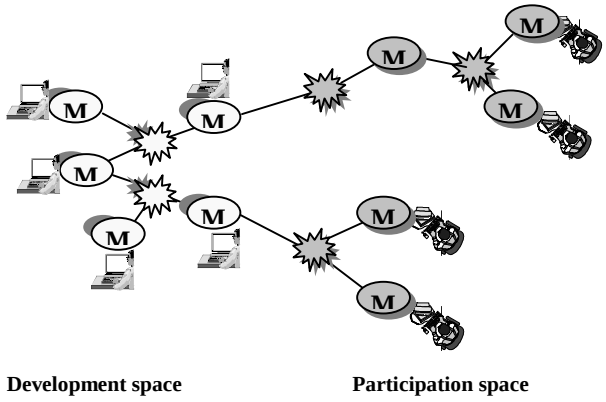

Figure 4: *Space structuring for user participation*

In such an organization, the control about what it is made visible to end-users and on how feedback from them is propagated to the development teams is devoted to the motus interfacing the development and participation spaces.
Replication and version management in this context is also very interesting. Version management services allow keeping track of product histories. Thus, they can be used to offer awareness information about product evolutions and may help users in understanding how they influence the actual development.
Replication facilitates document diffusion and access since it renders this automatic. Provided the creation of

sharing groups and the specification of the objects belonging to these groups, the successive versions of these objects are transparently transferred to the user environments. In addition, the multicast protocol Tuamotu uses maintain strong consistency properties about the replicas and ensure that the different sites sharing a common set of artifacts observe the same sequence of states of these artifacts.
Yet it is clear that these features are not sufficient by themselves to efficiently handle end-user involvement, we belief that they provide an interesting framework to build more advanced facilities.

RELATED WORK AND CONCLUSION

It actually exists very few effective solutions to efficiently support distributed engineering projects. The software engineering field seems to be the more advanced in this domain. Numerous projects have proposed process-centered software development environments. Some of them were particularly focused on cooperation support (e.g. [4]) and decentralized environments (e.g. [5]). In addition, some commercial software configuration management systems now offer facilities for multi-sites software developments. This kind of systems is dedicated to software development and seems difficult to adapt to other engineering domains. Research projects in the concurrent engineering field have also done works close to our (for example [6]), but the approaches used in these systems, although well suited for co-design, are not designed to be scalable to wide-area networks. Some CSCW systems have also studied the problem of collaborating over large-scale networks. Many of them however are synchronous systems that are inefficient the engineering context. DistView [7] is an example of such systems that uses object replication like TuaMotu.

Other CSCW systems are dedicated systems designed for a special use (e.g. text editors, group decision making...). Duplex [8] for example, uses an asynchronous approach and object replication but is dedicated to collaborative text edition. BSCW is a system close to Duplex based on extending a WEB server and on using WEB browsers to access it. Its client/server architecture makes difficult its use for supporting arbitrary organizations of cooperative activities.
TuaMotu, the system we presented in this paper, appears as a synthesis of the different solutions developed for software engineering, concurrent and cooperative design and CSCW. It will provide efficient support to distributed engineering projects. Many works still need to be done.
Actually, implementation of the motu servers, the atomic multicast and the client interface is finished. All the developments are done in Java. The ship is implemented using Voyager, a toolkit for programming mobile agents with Java. Persistence is provided by a persistent java machine [9]. We are currently working on a notify mechanism that will allow activities to specify events they want to be notified. The notion of cooperation group is being enhanced in order to add more meanings to specify group organizations.
Some points coming from our first experiments need to be underlined. Our asynchronous model of cooperation seems well suited to our purposes and goodly fit the actual needs and the expected usage in engineering projects. However, we think that this model is acceptable only if it is complemented with more direct and synchronous mechanisms to cover particular periods of very close collaboration like conflict resolution, decision taking, negotiation and so on.
This asynchronous model must also be supported by a very effective awareness mechanism. In particular, we guess that group perception plays a central role in such a system. It is necessary to develop concepts and metaphors that will allow a clear feeling and understanding of who are the actors of the cooperation and what they are doing. A possible way to that goal is to develop some divergence control mechanisms and metrics that will monitor actions undertaken by different users sharing common objects. Exploiting the replicated version graph seems very promising to design such mechanisms.
Another point concerns the use of the checkin/checkout paradigm. This model do not appears as very adequate in each situation. It is effective when explicit versioning is needed, but this is not always the case. The point is that we merged communication and cooperation concepts together with storage management concepts. We think that storage management should be as transparent as possible to let the user concentrate on his job and on cooperating. Our plan is to increase the capabilities of the client interface at the user workspace level to allow them to manage communication and cooperation concepts. This interface will still access the motu server to store and retrieve object versions, but this will not happen at the user initiative.
The last point concerns the use of an atomic multicast protocol. This kind of mechanisms is very costly and we have scalability problems when groups become too big. Our plan here is to complement it with a more lightweight mechanism based on email and to reserve the atomic broadcast to the case where a strong replica consistency is needed.

REFERENCES

[1] G. Canals, P. Molli, M. Munier, and C. Godart. *A criterion to enforce correctness of cooperative executions,* Information Sciences, 110, Sept. 1998

[2] H. Skaf, F. Charoy, and C. Godart, *A Hybrid Approach to Maintain Consistency of Cooperative Software Development Activities,* SEKE'97 , Madrid, 1997.

[3] D. Agarwal, P. Melliar-Smith, and L.E. Moser. *Totem: A protocol for message ordering in a wide area network*, 1st ISMM International Conference on Computer Communications and Networks, pages 1--5, June 1992.

[4] C. Godart, G. Canals, F. Charoy, P. Molli, and H. Skaf, *Designing and Implementing COO: Design Process, Architectural Style, Lessons Learned*, ICSE18 , 1996, IEEE Press.

[5] I. Ben Shaul and G. Kaiser, *Federating process centered environments: The oz experience*, Automated Software Engineering, 5(1), January 1998.

[6] Marina Zanella and Paolo Gubian, *A conceptual model for design management*, CAD, 28(1), January 1996.

[7] Atul Prakash and Hyong Sop Shim, *Distview: Support for building efficient collaborative applications using replicated objects*, ACM Conference on Computer Supported Cooperative Work (CSCW), Chapel Hill, October 1994.

[9] A. Schiper, F. Pacull, A. Sandoz, *Duplex: A distributed collaborative editing environment in large scale*, In ACM Conference on Computer Supported Cooperative Work (CSCW'94), Chapel Hill, October 1994.

[9] M.P Atkinson, M.J. Jordan, L. Daynes, and S. Spence. *Design Issues for Persistent Java: a type-safe, object-oriented, orthogonally persistent system*, POS-7, 1996.