Chapter 2
Improving DUIs with a Decentralized Approach with Transactions and Feedbacks

Jérémie Melchior, Boris Mejías, Yves Jaradin, Peter Van Roy, and Jean Vanderdonckt

Abstract When multiple users work collaboratively, coherence is not an easy feature to guarantee. It requires an exclusive access to some part of the User Interface (UI) and needs to give some feedbacks to other users. This synchronization needs a true concurrency control algorithm. One of the most common solution is to use a server as a transactional manager. Unfortunately, a central point of control is also a single point of failure. This paper proposes a decentralized architecture based on a peer-to-peer network providing decentralized transactional support with replicated storage. As a consequence, there is a gain in fault-tolerance and the transactional protocol eliminates the problem of network delay improving the overall usability. The addition of a feedback mechanism allow the users to understand better the behavior of the system.

2.1 Introduction

There are many software applications supporting collaborative work, such as drawing, text editing or software development. Collaborative work can be done synchronously or asynchronously. In the latter case, the participants make their modifications on their local copy without direct interaction with the other participants. Once the changes are made, they are committed to the global state. In the former case, which is the focus of this paper, all participants are concurrently working on a shared working space. Such scenario requires continuous synchronization of the

J. Melchior (✉) • J. Vanderdonckt
Louvain School of Management, Université catholique de Louvain, Louvain-la-Neuve, Belgium
e-mail: Jeremie.Melchior@uclouvain.be; Jean.Vanderdonckt@uclouvain.be

B. Mejías • Y. Jaradin • P. Van Roy
Computing Science and Engineering Pole, Université catholique de Louvain,
Louvain-la-Neuve, Belgium
e-mail: Boris.Mejias@uclouvain.be; Yves.Jaradin@uclouvain.be; Peter.VanRoy@uclouvain.be

M.D. Lozano et al. (eds.), *Distributed User Interfaces: Usability and Collaboration*,
Human–Computer Interaction Series, DOI 10.1007/978-1-4471-5499-0_2,

participants in order to avoid conflicts. One way of achieving such synchronization is by letting the participants lock the part of the shared space they want to modify, granting exclusive access to that part. Since all participants can take any lock, having a single point of control make sense, resulting in the classical client–server architecture. Unfortunately, it is well known that having a single point of control also means having a single point of failure, because the whole application relies on the stability of the server.

The case study we present in the paper is based on TransDraw [1]; a distributed collaborative vector-based graphical editor with a shared drawing area. Each user runs the application and joins a server to get access to the shared area. When someone is drawing in this area, feedback is sent to other users reflecting the action. In addition, TransDraw uses a transactional protocol to allow users to make optimistic changes on the drawing with immediate conflict resolution. This feature eliminates the problem of performance degradation caused by network latency and it is a crucial property of TransDraw. The synchronization and storage of the global state is done on a server which centralizes the control of the work flow. When users modify an object on the drawing, they request exclusive access for it, which may succeed or fail depending on the behavior of the other users. All this is reflected graphically in the shared drawing space.

A problem of TransDraw, due to its centralized architecture, is its dependency on the server. If the server crashes the work is lost, and the application will not run until the server is rebooted. Peer-to-peer networks have the nice property of being self-organized, fault-tolerant and fully decentralized. We propose in this paper to redesign the transactional protocol of TransDraw to overcome the problem of the single point of failure. In order to do that, we use Beernet, a structured peer-to-peer overlay network providing a fault-tolerant distributed transaction layer with replicated storage. Every time a user attempts to modify a graphical object, this modification will be done inside a transaction with a different transaction manager, which is replicated to allow the transaction to finish in case of failure of the manager. Unfortunately, this fault-tolerance mechanism is not free. Replication requires a higher usage of network resources increasing latency of transactions, but the optimistic approach for starting the modification of an object counteracts the latency. We consider this a small drawback because the functionality of TransDraw is fully respected and there is an important gain in fault-tolerance.

For the management of DUIs another problem comes from the needs of feedback. The initiator of the distribution must know when the distribution is over and if everything went well. The destination platform should notice the distribution and not be affected negatively by it. For this, we use a feedback mechanism in order to notify both the source and the destination for any kind of distribution. If the result of the distribution is invisible to one of them, a feedback needs to ensure the action went well. If the result is creating information or modifying the destination remotely, the destination should understand this addition or change

What follows is a more detailed description of TransDraw and related works in Sects. 2.2 and 2.3. Beernet is described in Sect. 2.4. The core of the proposal is explained in Sect. 2.5, being followed by the conclusions.

2.2 Transdraw

2.2.1 Description

Transdraw is a collaborative vector drawing tool created by Donatien Grolaux using transactions [2]. The toolbar provides, not only the traditional tools of vector editing (e.g. lines, ellipse, rectangles), but also a pair of tools supporting collaboration. As soon as a user selects an object, a request is sent to the server for the corresponding lock. However, the user is permitted to edit the object optimistically before the server can answer the request. The optimistic nature of the operation is visually presented to the user by feedback in the form of a red selection frame. When the server grants the lock, the transaction on the object is committed and the user can continue to edit the object in exclusive mode, indicated by black selection handles until he deselects it at which time the lock will be returned. If the lock was already held by another user, the server has to refuse it to the user and the transaction is aborted. The user sees the modifications he did optimistically undo themselves and the object is deselected.

A user can also manage explicitly his locks by using the *take lock* tool, for example to make a complex reorganization of the drawing, involving several individual objects. He then has to release the locks manually using the flashing *release-locks* button. In order to prevent starvation which could happen as simply as by a user inadvertently selecting every object before taking a rest, a lock stealing mechanism is provided. The *steal lock* tool make a request to steal a lock to the server which forwards it to the current owner of the lock. This user then has a few seconds to accept or reject the stealing of her locks. On timeout, the stealing is considered accepted. Once accepted, the previous owner notifies the server to forward the lock to the stealer.

2.2.2 Example Scenario

Figure 2.1 presents the view of two users working on the same drawing, each in his own window. Bob, on the right, had the top ellipse selected long enough for the server to grant him the lock as can be seen by the black selection handles around it. Alice, on the left has just tried to select this ellipse. After a, normally brief, period during which she was able to do optimistic changes to this ellipse, her transaction is aborted, and she is notified of it by the disappearance of her selection and the red dot on the ellipse which will blink a few times to explain that Bob is a currently editing this object.

The diagram in Fig. 2.2 describes a possible continuation of the scenario in which Alice steals the lock from Bob to perform the update she wants. Alice ask to steal the lock to the server. Since Bob currently has the lock, the server ask Bob whether he

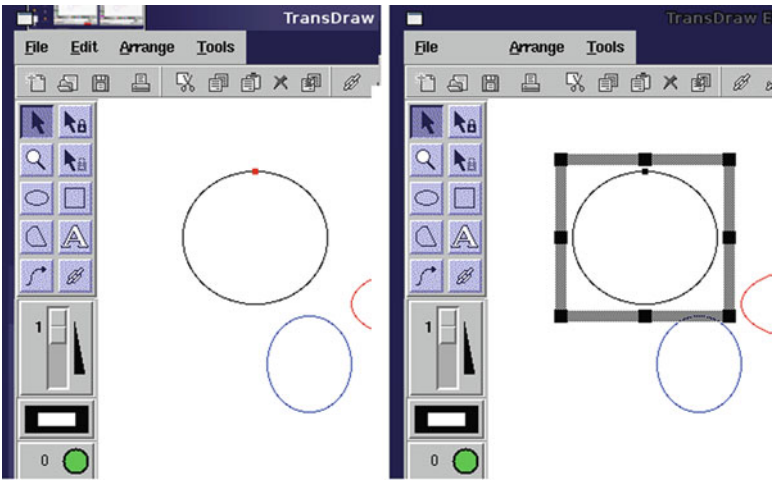

Fig. 2.1 Alice, on the *left*, see a locked and non-editable ellipse while Bob has it is selected and editable

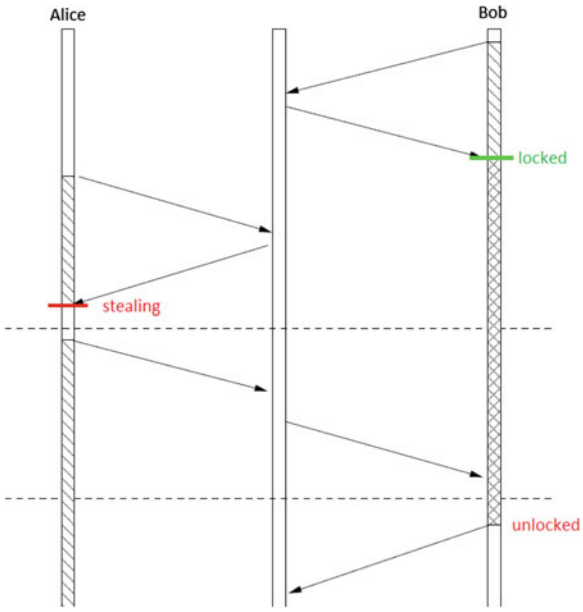

Fig. 2.2 Scenario of complex interaction

allows his lock to be stolen or not. This is shown to Bob as two blinking buttons at the bottom of his edition window as we can see in Fig. 2.3. If Bob allows his lock to be stolen, either explicitly or by ignoring the request long enough, he loose selection of the object and possession of the lock and the server transfer them to Alice.

All of this assumes that the server does not crash.

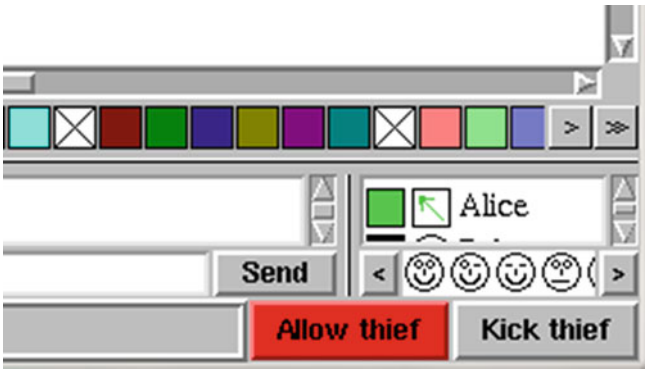

Fig. 2.3 Bob is asked whether he allows his lock to be stolen

2.3 Related Works

There are some applications that already support collaboration in different ways. We describe and comment some of them briefly.

2.3.1 BOUML

BOUML [3] is a free UML tool that allows drawing diagrams and generating code in multiple languages. The tool has been developed as a multiuser application in a sequential way. Each user of the application must choose an identifier which allows working on some diagrams. The work may be done in parallel but there is not any feedback on other users' work as there is no support for concurrent work. There are many problems with the tool. The lack of feedback prevents user to know what others are doing and to see their changes. It is also impossible to know which files are currently being modified or that have been modified and saved. There can be conflicts when saving the project. When users are working collaboratively, the work of a user will be saved but not all the modification of other users. This leads to irreversible lost work without any warning. Another problem is the impossibility to lock part of the work to prevent modification from another user.

2.3.2 Gobby

Gobby is a free text-editor that allows collaborative work [4]. It supports multiuser parallel edition on multiple documents and a multiuser chat. A user has to start a session and create the documents, he will host the server needed to centralize

the information. Other users must choose a name and a color and connect to the server host. The collaboration between all the users is simple thanks to the feedback brought to users with colors. As the BOUML application, Gobby does not support any lock of some part of the text and all the users can edit what they want. There is a problem when the server crashes. The unsaved modifications can be saved but the whole process of creating a server and joining the server must be restarted.

2.3.3 Google Docs

Google Docs [5] is an online office suite that allows multiple users to modify the same file at the same time. One particular feature, similar to TransDraw, can be seen on spreadsheets. Once a user is modifying a cell, this one is colored differently as in any single user spreadsheet application. When other users connect to Google servers to edit the same file, then, the cells they select will appear with a different color on the view of the other users, and with a tag identifying the user. Instead of locking the cell, changes are save incrementally using versioning. Google Docs uses also a centralized architecture because everything is controlled at Google side. But, there is a very important difference. There is not only one server to rely on, but a set of servers with replicated information, so if a server crashes, another one takes over. Of course, these are only conjectures about Google's back-end.

2.4 Decentralized Transactional DHT

Beernet [6] is a structured overlay network providing a distributed hash table (DHT) with symmetric replication. Peers are self-organized using the relaxed-ring topology [7], which is derived from Chord [8], with cost-efficient ring maintenance and self-healing properties. Data replication is guaranteed with a decentralized transactional protocol allowing the modification of different items within a single transaction. The transactional protocol implements a Paxos-consensus algorithm [2, 9], which requires the agreement of the majority of peers holding the replicas of the items. We will focus on the transactional layer of Beernet because it will be our mean to decentralize TransDraw.

Figure 2.4 describes how the Paxos-consensus protocol works. The client, which is connected to a peer that is part of the network, triggers a transaction in order to read/write some items from the global store. When the transaction begins, the peer becomes the transaction manager (TM) for that particular transaction. The whole transaction is divided in two phases: *read phase* and *commit phase*. During the *read phase*, the TM contact all transaction participants (TPs) for all the items involved in the transaction. TPs are chosen from the peers holding a replica of the items. The modification to the data is done optimistically without requesting any lock yet. Once all the read/write operations are done, and the client decides to commit the transaction, the *commit phase* is started.

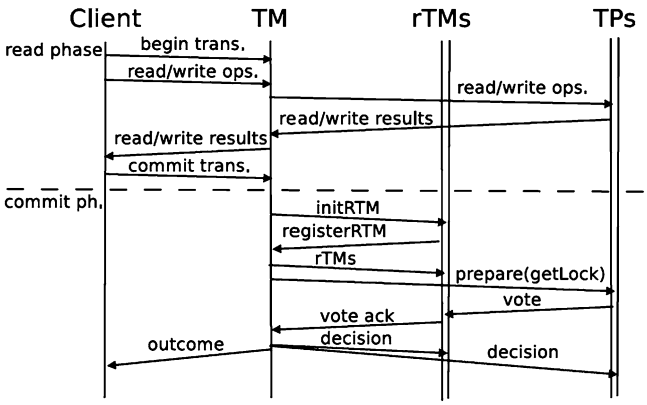

Fig. 2.4 Paxos consensus protocol for distributed transactions

In order to commit the changes on the replicas, it is necessary to get the lock of the majority of TPs for all items. But, before requesting the locks, it is necessary to register a set of replicated transaction managers (rTMs) that are able to carry on the transaction in case that the TM crashes. The idea is to avoid locking TPs forever. Once the rTMs are registered, the TM sends a *prepare* message to all participants. This is equivalent to request the lock of the item. The TPs answer back with a $vote$ to all TMs (arrow to TM removed for legibility). The vote is acknowledged by all rTMs to the leader TM. Like that, the TM will be able to take a decision if the majority of rTMs have enough information to take exactly the same decision. If the TM crashes at this point, another rTM can take over the transaction. The decision will be *commit* if the majority of TPs voted for commit. It will be *abort* otherwise. Once the decision is received by the TPs, locks are released.

The protocol provides atomic commit on all replicas with fault tolerance on the transaction manager and the participants. As long as the majority of TMs and TPs survives the process, the transaction will correctly finish. These are very strong properties that will allows us to run TransDraw on a decentralized system without depending on a server.

2.5 Decentralized TransDraw

Instead of using a big infrastructure, we can achieve replication and fault-tolerance by building TransDraw on top of a peer-to-peer network, and by decentralizing the synchronization of locks and data storage. Our proposal is to build TransDraw on top of Beernet.

The Paxos-consensus protocol as described in Sect. 2.4 is not sufficient to provide exactly the same functionality of TransDraw as it was described in Sect. 2.2.

The main difference lies on the moment where the locks are granted. As it is currently, locks are granted too late for TransDraw, because it is not possible to inform users about the intention of the others.

The first modification we have to do to the transactional protocol is to allow eager locking request. One idea is to request the locks when read/write operations are sent to the transaction participants during the *read-phase*. If locks are not granted, the transaction is immediately aborted. The problem introduced by this modification is that if leader TM crashes after requesting the locks, there is no rTM yet to take over the transaction, and items would be locked forever. Considering this, the registration of rTMs must also be moved up to the read-phase. After this two modifications we realized that in fact it is better to avoid the read-phase and start immediately with an extended commit phase that first needs to gather the participants.

The second modification is an eager notification mechanism. Currently, out transactional layer is meant for asynchronous access to the share state. When a peer writes a new value for item, other users are not notified unless they read the item. In the case of TransDraw, other users not only need to be notified of every modification on the value of items, but also on the intention of other users when they lock items. To achieve this, the leader must broadcast its decision to the network once it get enough locks, and once the final decision is taken. Note that eager locking and the notification mechanism are only needed on synchronous collaborative work.

2.6 Classification of the Case Study

In Chap. 1, Villanueva et al. have introduced a classification for several case studies. According to their Table 1.1 TransDraw is really closed to the WallShare case study. The interface is not “Divisible” or “Distributable” because there are no interaction objects (IO) in two different interaction surfaces (IS). It can reach a unified state and has a divided state. Unlike WallShare, it is here possible to have all the ISs on the same platform. It has at least one distributed state because each IO can be in a different platform.

2.7 Conclusion and Future Work

We have seen that several synchronous collaborative applications are currently We have seen that several synchronous collaborative applications are currently based on centralized synchronization. This strategy is efficient but not fault-tolerant because it strongly relies on the stability of the server. Some applications achieve fault tolerance by replicating the state of the server, but this requires a more sophisticated infrastructure and it is still inherently centralized. Single point of control is a single point of failure.

We propose to implement these kind of applications on top of structured overlay networks with symmetric replication, and a transactional layer based on consensus.

This strategy provides synchronization and fault-tolerance by decentralizing the control of the work flow. We present our approach by taking the TransDraw application and the Beernet peer-to-peer network.

Beernet as is, can help to decentralize asynchronous collaborative applications. In order to achieve the functionality of TransDraw, which is synchronous, eager locking and a notification mechanism need to be added to the current transactional protocol.

We still need to study in detail the new transactional protocol, implement it and compare the performance with the centralized approach. We expect to have a small degradation in performance at the level of the transactional protocol due to replication cost, but with a huge gain in fault-tolerance. There is no degradation in performance for the user in case of no conflicts, because its changes are done optimistically, eliminating the problem of network latency.

Acknowledgments This work has been funded by the European Commission FP6 IST Project SELFMAN (Contract 034084), with support of the ITEA2-Call3-2008026 UsiXML European project.

References

1. Mejías, B. (2009). Beernet – the relaxed peer-to-peer network. http://beernet.info.ucl.ac.be.
2. Grolaux, D. (1998). Editeur graphique réparti basé sur un modèle transactionnel, 1998. Mémoire de Licence.
3. Pages, B. (2009). The bouml tool box. http://bouml.sourceforge.net.
4. 0x539 dev group. (2009). The gobby collaborative editor. http://gobby.0x539.de.
5. Gray, J., & Lamport, L. (2006). Consensus on transaction commit. *ACM Transactions on Database Systems, 31*(1), 133–160.
6. Google. (2009). Google docs. http://docs.google.com.
7. Mejías, B., & Van Roy, P. (2007). A relaxed-ring for self-organising and fault-tolerant peer-to-peer networks. *XXVI International Conference of the Chilean Computer Science Society, IEEE Computer Society* (November 2007).
8. Stoica, I., Morris, R., Karger, D., Kaashoek, F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *Proceedings of the 2001 ACM SIGCOMM Conference,* pp. 149–160.
9. Moser, M., & Haridi, S. (2007). Atomic commitment in transactional dhts. *Proceedings of the CoreGRID Symposium, CoreGRID series*, Springer.