A Collaborative Graphic Editor Based on Transactions

Oz/Mozart Workshop
June 5, 1998

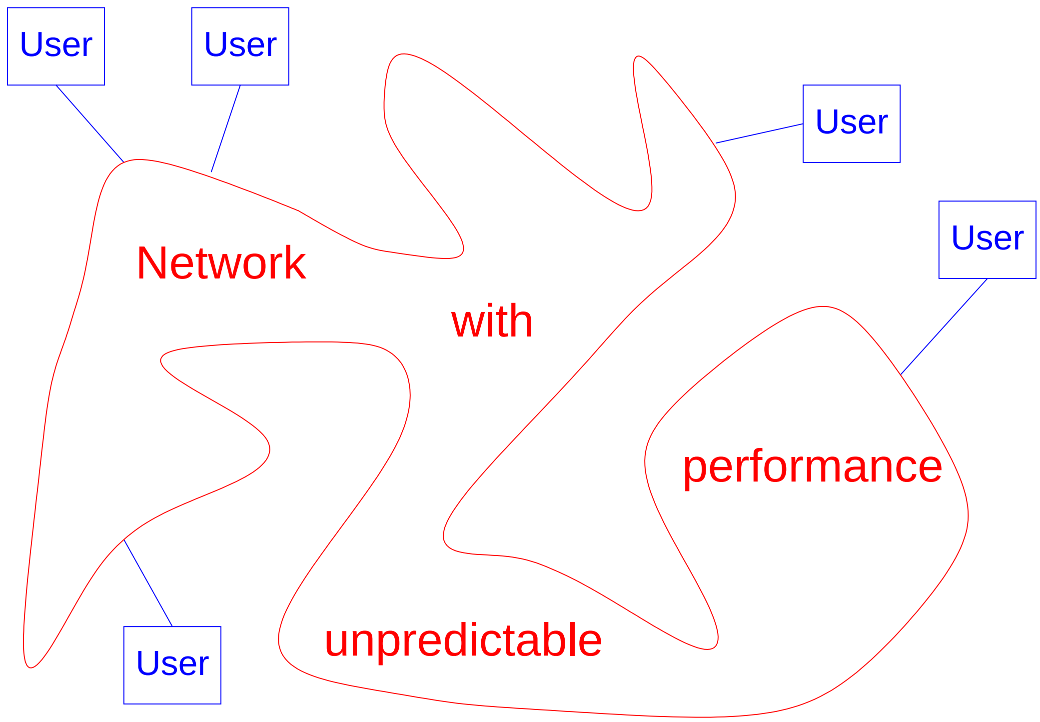
Donatien Grolaux
Peter Van Roy

Université catholique de Louvain

Overview

- Problem: Usable Editor over the Net
- Solution: Speculative Edits with Transactions
- Logical Architecture
- Scenario with Two Clients
- User Interface
- Full Transaction Protocol
- Physical Architecture and Initialization
- Conclusions

Problem:
Graphic Editor over the Net



Collaborative design over Internet:

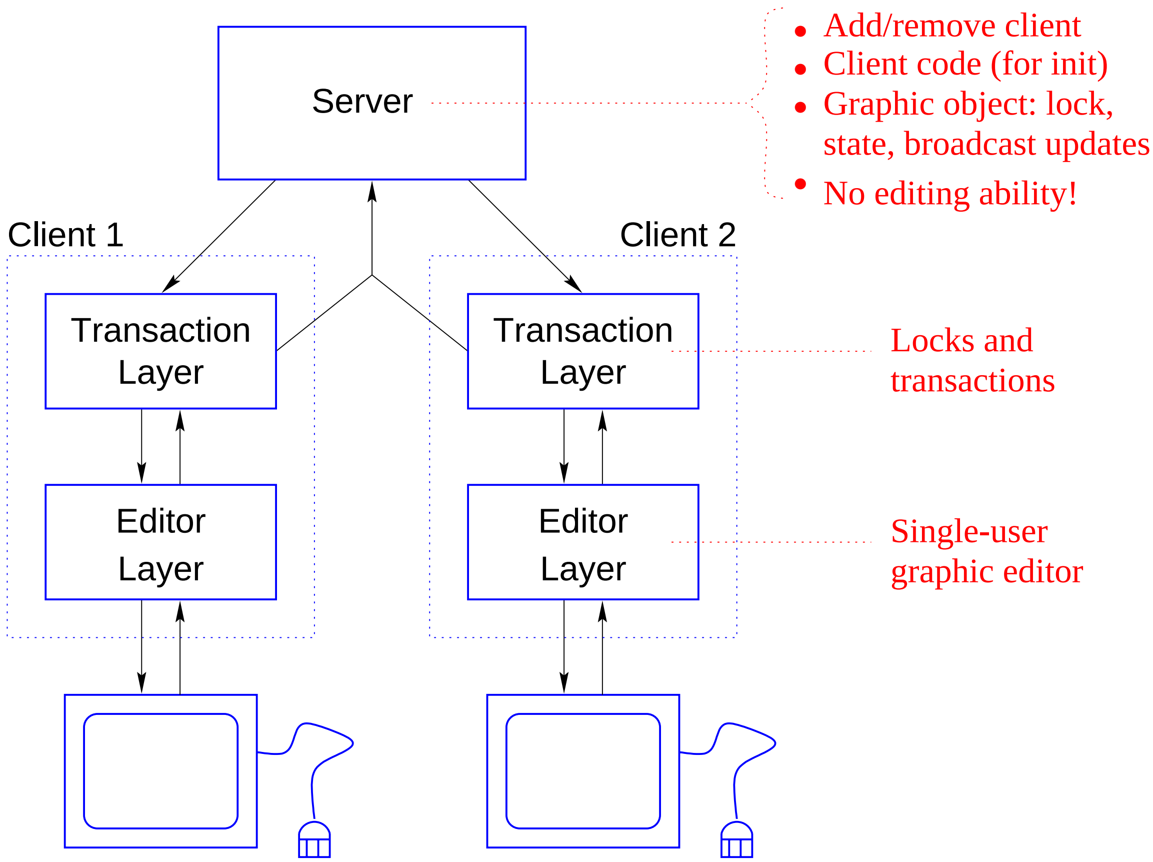
Specification $\left\{ \begin{array}{l} \text{• All users manipulate the same drawing} \\ \text{• All users have instantaneous response time} \end{array} \right.$

Solution:

Speculative Edits with Transactions

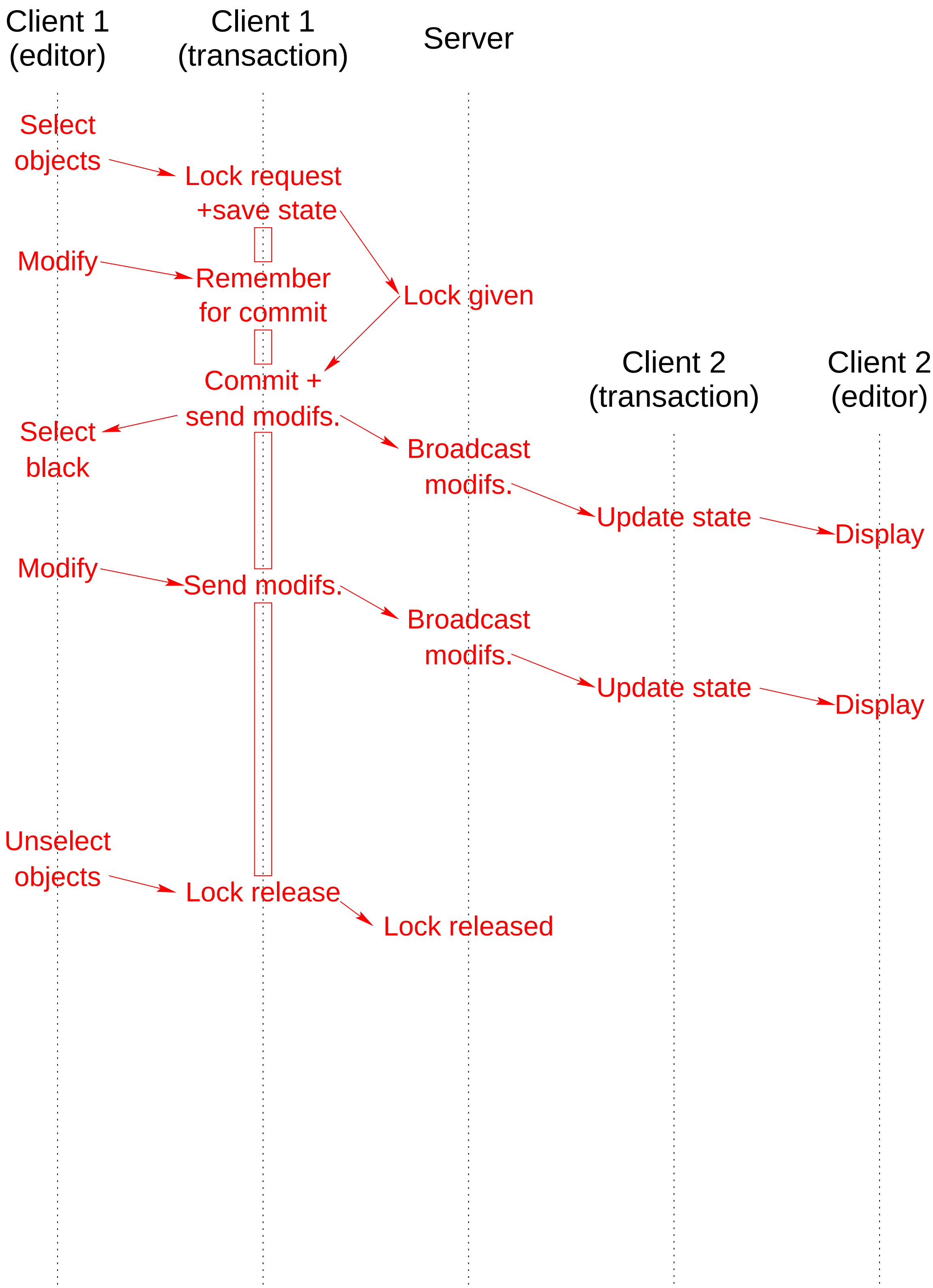
- Transactions are a concept from databases used to maintain consistency during multiple concurrent updates
- Transactions can also be used to bridge the delay time of a network:
  - Each user instantly makes local modifications to part of the drawing. These modifications are not seen by the other users.
  - Concurrently, the editor requests global locks on all the graphic objects modified.
  - If the locks are obtained, the modifications are made global.
  - If the locks are refused, the modifications are cancelled.
- How can we design an editor that is based on this principle with a user interface that minimizes interference from other users and from the network?

Logical Architecture

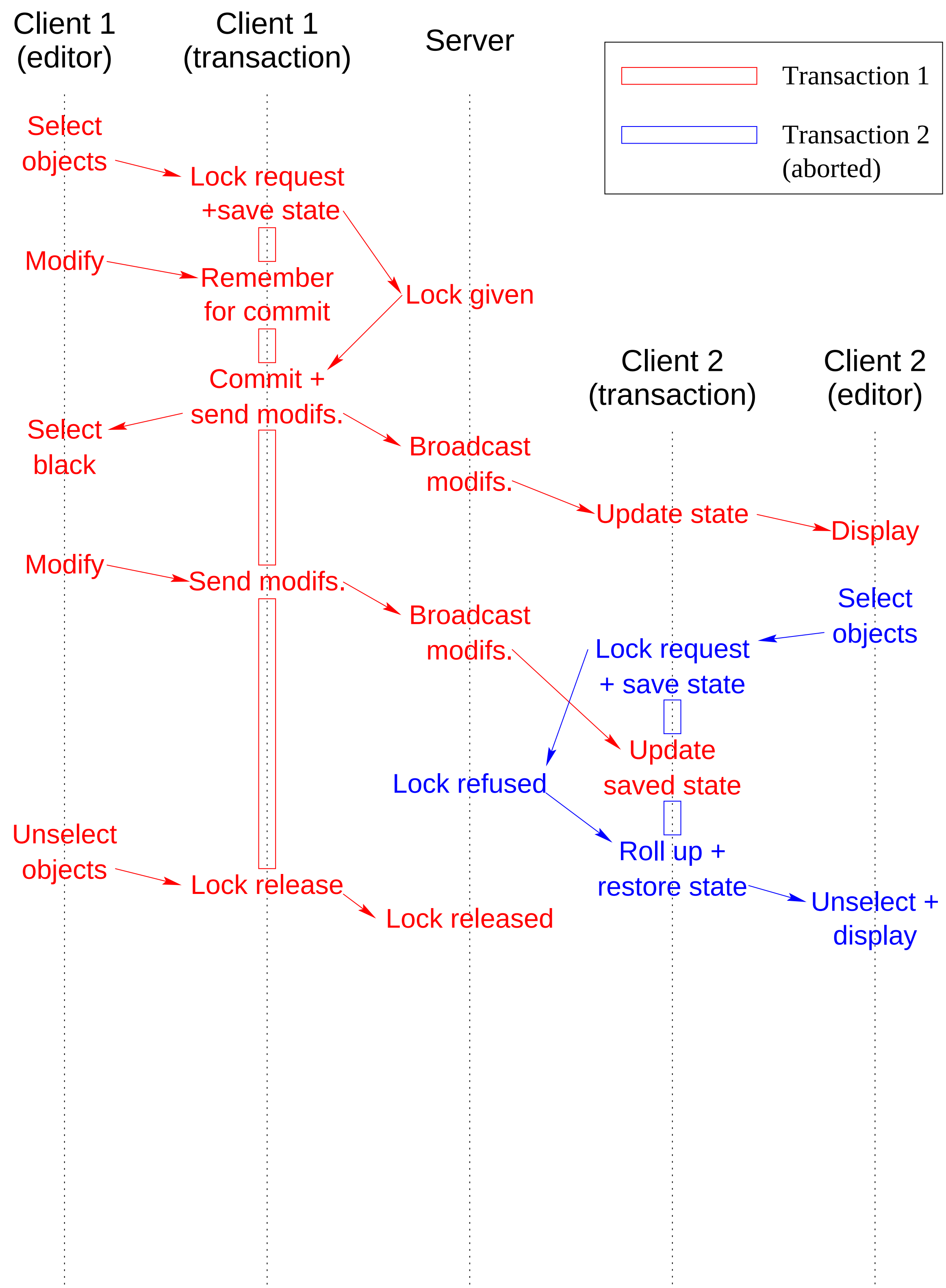


- Built as layers of (almost) independent functionality
- Messages from client to server: lock request/release, create/modify/delete graphic object
- Messages from server to client: lock given/refused, broadcast create/modify/delete graphic object

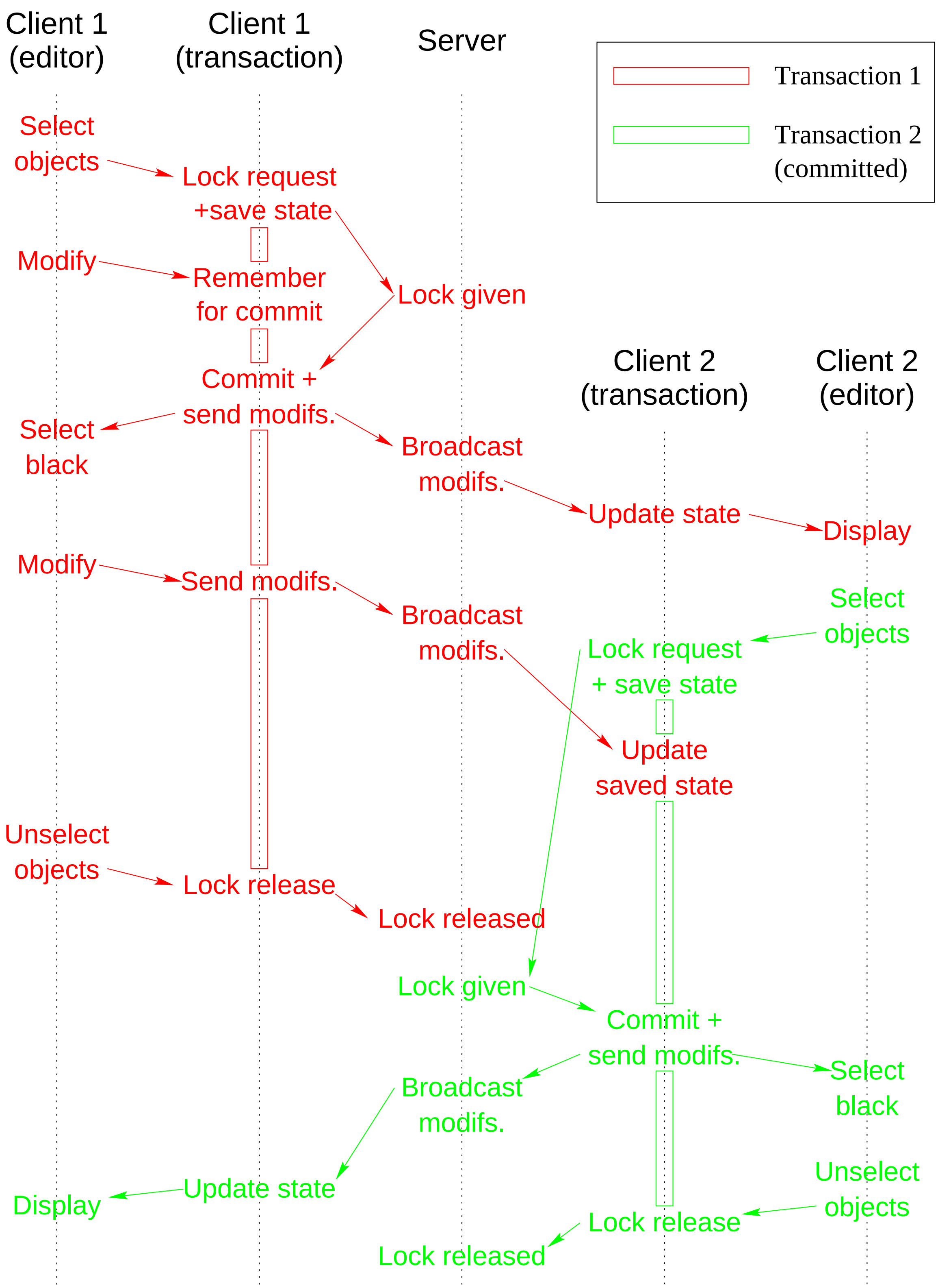
Scenario with two Clients (1)



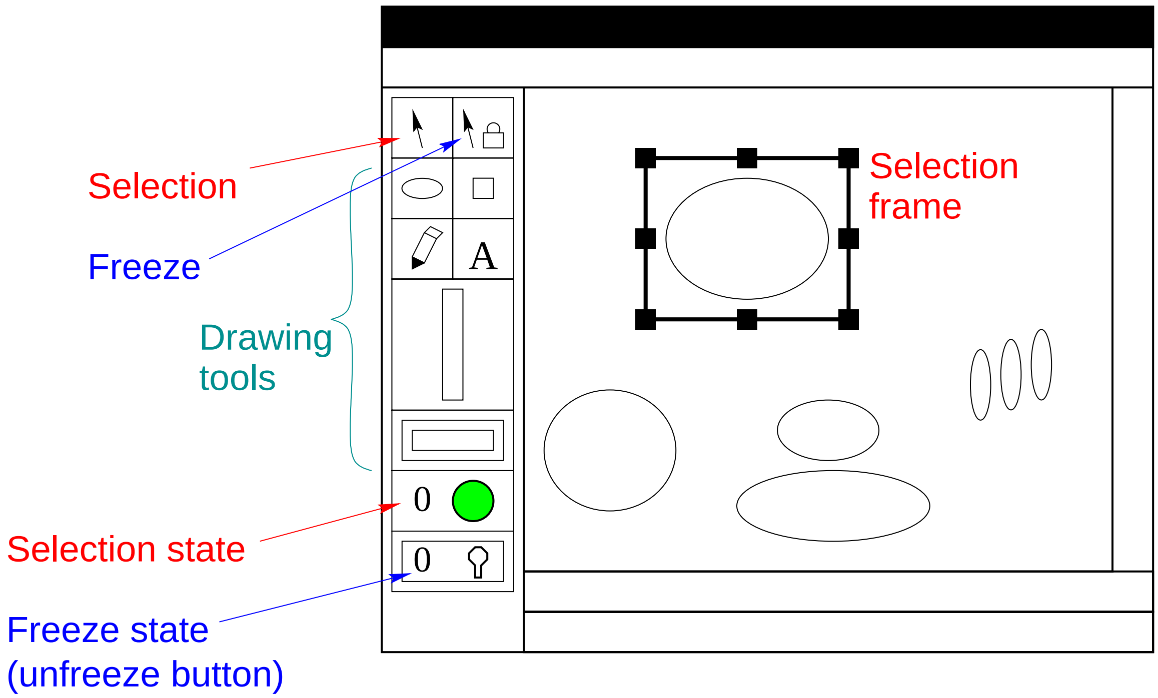
Scenario with two Clients (2)



Scenario with two Clients (3)



User Interface

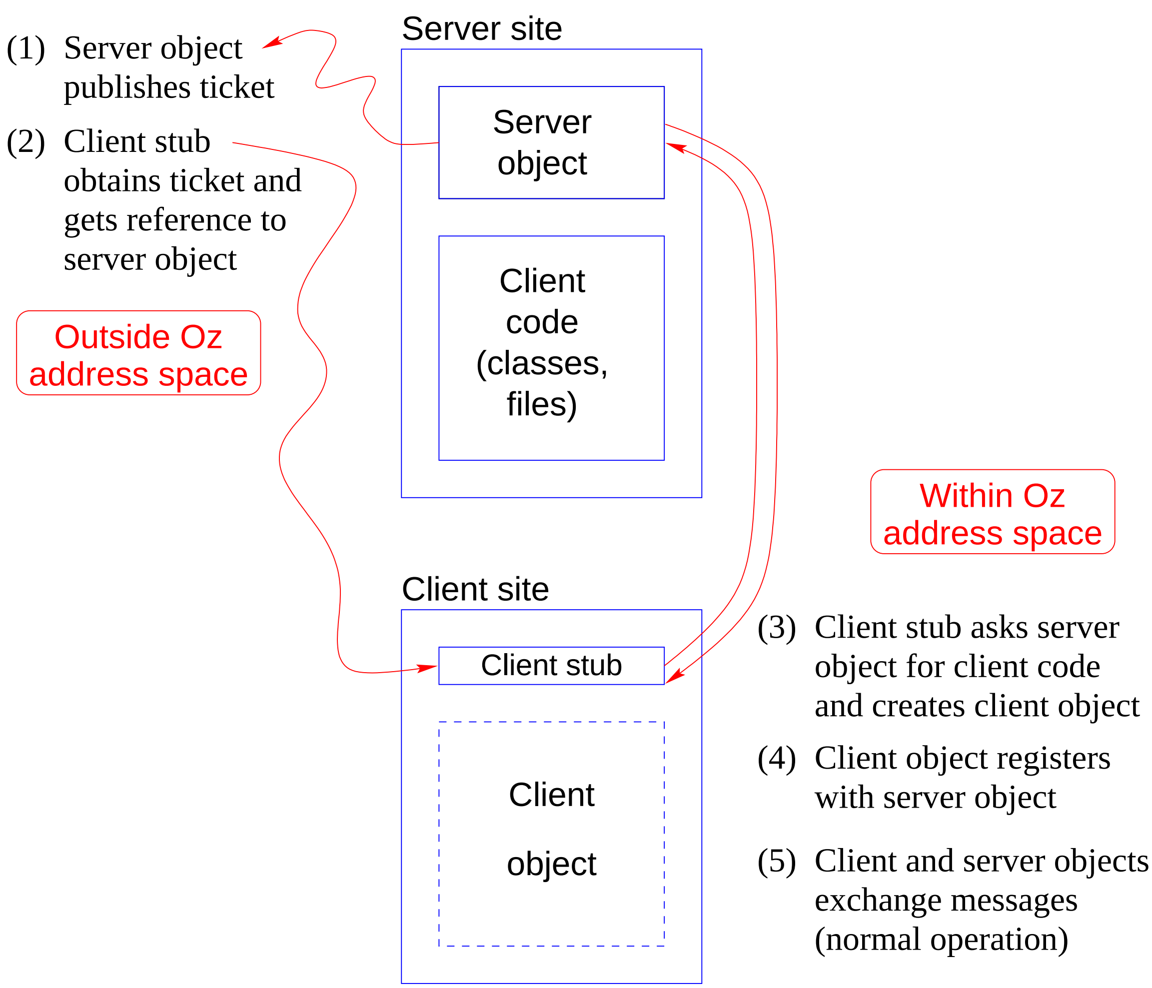


- Drawing tools: standard set (circle, rectangle, text, freehand, polyline, fill, thickness, color)
- Selection tool: standard (click, shift-click, drag, handles) with extensions:
    - Selection frame: black (committed) / red (not committed)
    - Selection state: green (consistent view) / red (otherwise)
- Freeze tool: like selection, but locks only (keeps other users from modifying)
    - Unfreeze button: click to unfreeze everything

Full Transaction Protocol

- Multiple transactions can be active at one client. Oldest is committed first, abort rolls up all newer ones.
- Undo is local to each client. The undo transaction is possible if no other client has modified any relevant object. Undo actions are logged for each modification.
- Delete initially hides the object, and removes it at commit. Undo recreates the object from scratch.
- Grouping/ungrouping through a group object that plays the role of client for its components.
- Display order can be changed. Displayed order is local order modified by active order-changing commands. When these commit they become part of local order.

Physical Architecture and Initialization



Two phases:

1. On startup, client obtains its functionality from the server
2. During operation, client and server exchange messages

Conclusions

Evaluation of application:

- Proof of concept: prototype exists and works well
- Extend to make a usable collaborative tool:
  - User feedback, "steal" tool, functionality, fault tolerance
- Basis for a generic ’transactional application’ module
  - Allow to plug in any single-user application!
- Prototype is publicly available on the Web

Use of Distributed Oz:

- High-level language requires learning period

Release Fall 98

- Prerelease system: small quirks, lack of documentation
- Debugging of concurrent dataflow language not easy
- Raw Tcl/Tk not completely hidden (need interface builder)

+ After learning period, development is rapid
+ Large functionality with small amount of code
+ Fully transparent distribution is major advantage
+ Graphic interface much better than raw Tcl/Tk
+ Failure model allows building robust application