Improved presence through screen sharing for a distributed pair programming tool

Preparation presentation for Bachelor thesis
by Stephan Lau

Tutored by Karl Beecher and Stephan Salinger
Overview

1. Motivation
2. Screensharing in general
3. Basic functionality
4. Short comparison of screensharing and collaborative awareness
5. How can screensharing extend collaborative awareness
6. Basic requirements
7. My implementation
   1. Requirements
   2. Transmission
   3. Controlling
   4. Bottlenecks in transmission
8. Integration in Saros
9. Approximate timeline
Motivation

There are parts of collaboration awareness which Saros can't satisfy.

- sharing applications outside Eclipse
  - webbrowser
  - build programs
  - other developer tools (MySQL, other editors, Terminals, …)
Motivation

- Share parts of Eclipse
- Wizards
- Configuration dialogues
- Other views than text editor and package browser, e.g., console, task list.
Screensharing in general

• passive
  • see the screen of a remote computer

• active
  • interact with remote screen
  • like remote desktop's (windows terminals)
Basic functionality

• client – server principle

• server
  • provides screen
    • sequence of screenshots, video (e.g. VNC)
    • native windows, like vectorgraphics (e.g. RDP, X11)
  • executes actions

• client
  • displays screen of server
  • sends activities (e.g. clicks, keystrokes)
## Short comparison of screensharing and collaborative awareness

<table>
<thead>
<tr>
<th>Collaborative awareness</th>
<th>Screensharing</th>
</tr>
</thead>
</table>
| • multiuser support in application  
  → implement support for every desired feature | • multiuser support on desktop  
  - everybody can see hosts environment  
  → missing privacy |
| • transmits changes in artifacts  
  → effects instead cause  
  - low network load | • changes visible on desktop through interaction  
  → cause can be understood  
  - high bandwidth needed |
How can screensharing extend collaborative awareness

Composition of CA and SS can balance these drawbacks.

+ unsupported (and visible) artifacts can be shared
+ can be used for demonstrations
+ improved learning effect
+ problems with external dependencies can be compensated
+ privacy is protected when screensharing can be limited on chosen windows

− high network load
  − can be reduced by compression
Basic requirements

Live screensharing over network

• functional
  • exact view (quality like at server)
  • exchange activities between client – server

• non-functional
  • realtime or as close as possible
  • not use more bandwidth than available
    • deal with temporary bottlenecks
    • don’t block other activities
  • efficient encoding
    • low bandwidth with low computing time at high quality
My implementation - Requirements

- **video**
  - resizing of input images to size of video stream (when needed)
  - resizing of output images to size of player

- **two modes**
  - fullscreen on primary display
    - zooming into areas
    - pointing
    - clicking
  - following hosts mouse pointer
    - zooming
  - (optional) focus one window

- (optional) change image source
  - e.g. switch between webcam and screen

- (optional) share screen to many participants
  - vice versa it is already possible
My implementation - Transmission

• host with one client (optional: many)
  • bidirectional bytestream between them
    • videostream for carrying remote screen
    • control and activity stream

• videostream
  • xuggler 3.3 for en-/decoding
    • videocoding library based on FFMPEG
      • native en-/decoding
      • many supported codecs
  • license: LGPL
  • easy to install
  • available for Windows, Unix & OSX
  • preferred codec is libx264 aka H.264
    • high quality encoding at low bitrates
      • but consumes a lot of computing time
My implementation - Controlling

- **controlstream** is a simple ObjectStream
  - client sends periodically statistics about decoding
    - frames per second, bitrate, delay of stream
    - allows server to detect bottlenecks
  - activities to server, e.g. clicks

- **adaptive bandwidth usage**
  - encoder uses constant bitrate mode
    - target bitrate is about 80% of available bandwidth
    - bitrate-tolerance is 20%
    - at lower bitrates VBV guarantees no exceeding of bw
  - average bandwidth needs to be known and configured
    - maybe later feature: detect bandwidth automatically
My implementation – Bottlenecks in transmission

• detecting temporary bottlenecks
  • analyze clients fps and delay
  • keep track of servers sendbuffer usage
    • growing → client can't receive that fast

• scale down needed bandwidth
  • drop frames to lower bitrate until delay has gone
    • other parameters (resolution, target bitrate) can't be adjusted while encoding
      → restart streaming if dropping is not sufficient
  • maybe later feature: detect bandwidth automatically
Integration in Saros

- integrate streaming into network layer
  - build abstract stream by using DataTransferManager.send(...)
    - transfer possible via
      - IBB (4kb/s) XEP-0047
      - SOCKS5 (80kb/s) XEP-0065
      - Jingle XEP-0166
  - use stream provided by Smack
    - Hennings job → DPPX

- UI
  - configuration
  - selecting client
  - view for video
Approximate timeline

Week 1-3
• architecture
• implementation of core features
  • two modes with zooming, clicking, pointing
  • adaptive bandwidth

Week 4-6
• integration in Saros
  • streams
  • UI

Week 7-9
• deal with scientific questions
• implement additional features
• optimize current implementation
Approximate timeline

Week 10-12
• final work on written thesis
Questions?

pepper me ;}
Thank you very much!

http://boredzo.org/codec-comparison/