Freie Universität Berlin

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

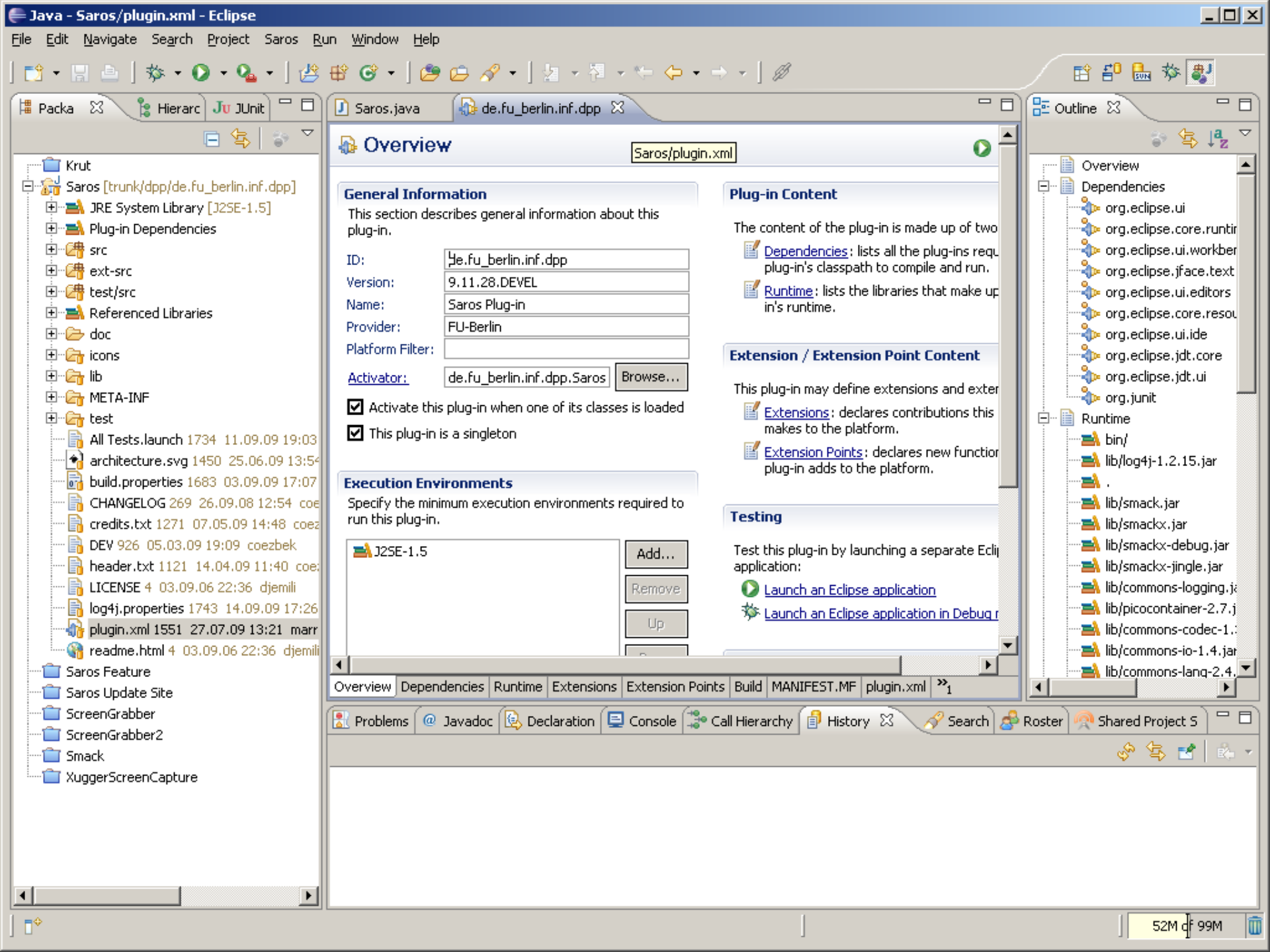
Overview

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 programms
  - other developer tools (MySQL, other editors, Terminals, ...)


Java - Saros/plugin.xml - Eclipse
File Edit Navigate Search Project Saros Run Window Help
Packa Hierarc JUnit
Krut
Saros [trunk/dpp/de.fu_berlin.inf.dpp]
JRE System Library [J2SE-1.5]
Plug-in Dependencies
src
ext-src
test/src
Referenced Libraries
doc
icons
lib
META-INF
test
All Tests.launch 1734 11.09.09 19:03
architecture.svg 1450 25.06.09 13:5
build.properties 1683 03.09.09 17:07
CHANGELOG 269 26.09.08 12:54 coe
credits.txt 1271 07.05.09 14:48 coez
DEV 926 05.03.09 19:09 coezbek
header.txt 1121 14.04.09 11:40 coe
LICENSE 4 03.09.06 22:36 djemili
log4j.properties 1743 14.09.09 17:26
plugin.xml 1551 27.07.09 13:21 marr
readme.html 4 03.09.06 22:36 djemili
Saros Feature
Saros Update Site
ScreenGrabber
ScreenGrabber2
Smack
XuggerScreenCapture
Saros.java
de.fu_berlin.inf.dpp
Overview
Saros/plugin.xml
General Information
This section describes general information about this plug-in.
ID: de.fu_berlin.inf.dpp
Version: 9.11.28.DEVEL
Name: Saros Plug-in
Provider: FU-Berlin
Platform Filter:
Activator: de.fu_berlin.inf.dpp.Saros
Browse...
Activate this plug-in when one of its classes is loaded
This plug-in is a singleton
Execution Environments
Specify the minimum execution environments required to run this plug-in.
J2SE-1.5
Add...
Remove
Up
Plug-in Content
The content of the plug-in is made up of two
Dependencies: lists all the plug-ins requ
plug-in's classpath to compile and run.
Runtime: lists the libraries that make up
in's runtime.
Extension / Extension Point Content
This plug-in may define extensions and exter
Extensions: declares contributions this
makes to the platform.
Extension Points: declares new functior
plug-in adds to the platform.
Testing
Test this plug-in by launching a separate Eclip
application:
Launch an Eclipse application
Launch an Eclipse application in Debug
Overview Dependencies Runtime Extensions Extension Points Build MANIFEST.MF plugin.xml
Outline
Overview
Dependencies
org.eclipse.ui
org.eclipse.core.runtir
org.eclipse.ui.workber
org.eclipse.jface.text
org.eclipse.ui.editors
org.eclipse.core.resou
org.eclipse.ui.ide
org.eclipse.jdt.core
org.eclipse.jdt.ui
org.junit
Runtime
bin/
lib/log4j-1.2.15.jar
.
lib/smack.jar
lib/smackx.jar
lib/smackx-debug.jar
lib/smackx-jingle.jar
lib/commons-logging.ja
lib/picocontainer-2.7.j
lib/commons-codec-1.
lib/commons-io-1.4.jar
lib/commons-lang-2.4.
Problems Javadoc Declaration Console Call Hierarchy History Search Roster Shared Project S
52M of 99M

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

Collaborative awareness	Screensharing
• 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 inputimages to size of videostream (when needed)
  - resizing of outputimages to size of player
- two modes
  - fullscreen on primary display
    - zooming into areas
    - pointing
    - clicking
  - following hosts mousepointer
    - zooming
  - (optional) focus one window

- (optional) change imagesource
  - e.g. switch between webcam and screen
- (optional) share screen to many participants
  - viceversa 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!

Sources

Djemili: **Entwicklung einer Eclipse-Erweiterung zur Realisierung und Protokollierung verteilter Paarprogrammierung**.
Diplomarbeit, FU Berlin, Institut für Informatik, 2006.

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