Antrittsvortrag Masterarbeit

Evaluation einer gemeinsamen Oberfläche für Saros/E und Saros/I mit Testframework

Christian Cikryt
Freie Universität Berlin

15.01.2015

Überblick

Freie Universität Berlin

Startpunkt

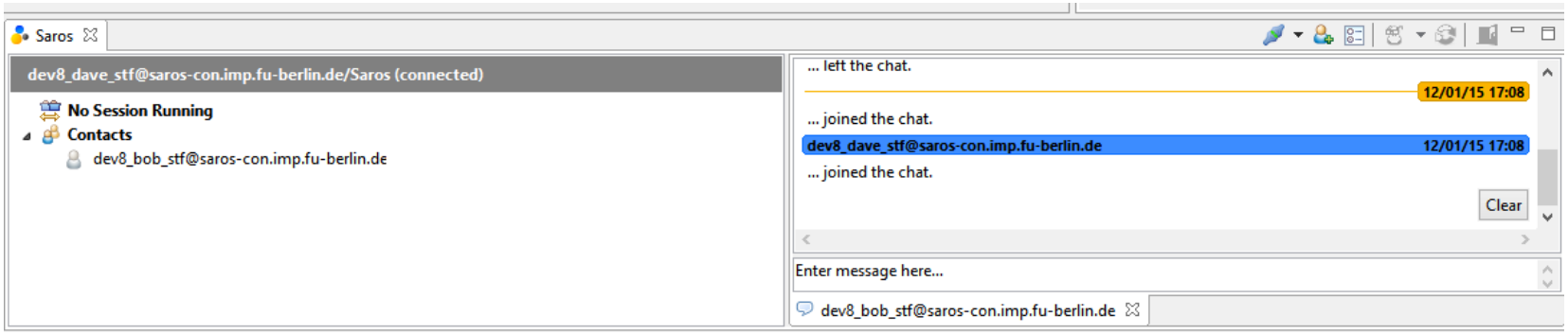
Bisheriger Wegabschnitt

Mein Fahrplan

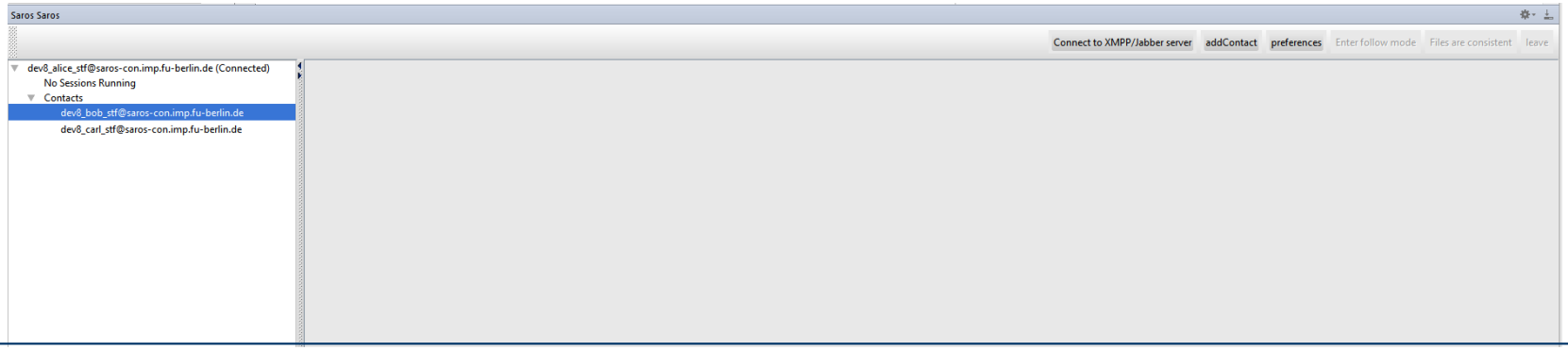
Baustellen

Portierung von Saros nach IntelliJ

SWT



Swing



Motivation einer HTML-basierten GUI

- Vermeidung von doppeltem GUI-Code
- Vereinheitlichung der Oberfläche
- Erfahrung mit SWT Browser in Eclipse
- modernere Anzeigetechnologien
- möglicherweise Performanzvorteile

Technische Hürden

- Zwei Java-GUI-Technologien: Swing und SWT
- IntelliJ verwendet Swing, Eclipse SWT
- Auswahl an Java-Browsern begrenzt:
  - SWT Browser
  - JxBrowser (Swing, proprietär)
  - JavaFx Webview (verlangt Java ≥ 7)
  - Native Swing (Fortentwicklung ungewiss)
- nahtlose Einbettung des Browsers in die IDE

Lösungsansätze

- Swing und SWT ineinander einbettbar
- SWT Browser funktioniert
- SWT Browser in Eclipse funktioniert
- SWT Browser in IntelliJ denkbar

Punkte der Evaluation

- technische Umsetzbarkeit
  - Darstellungsmöglichkeiten
  - Stabilität und Kompatibilität
  - Performanz
  - Testbarkeit
- versprochener Nutzen
  - auf Codeebene (Evolvierbarkeit)
  - in Bezug auf Aufwand
- Benutzbarkeit
  - aus Anwendersicht
  - Integration in IDEs

Fokus meiner Arbeit

- technische Umsetzbarkeit
  - Darstellungsmöglichkeiten
  - Stabilität und Kompatibilität
  - Performanz
  - Testbarkeit
- versprochener Nutzen
  - auf Codeebene (Evolvierbarkeit)
  - in Bezug auf Aufwand
- (Benutzbarkeit)
  - (aus Anwendersicht)
  - (Integration in IDEs)

Mögliche UI-Implementierungen

1. gleicher Browser in beiden IDEs
   - SWT Browser
   - JxBrowser
   - JavaFx Webview
2. SWT in Eclipse und JxBrowser in IntelliJ
3. separate IntelliJ und Eclipse GUI (Swing bzw. SWT)

1. Entscheidung: Browser oder nicht

Freie Universität Berlin

Browser

Aufwand

+ Moderne Technologien
- technisches Neuland
- Browserintegration

Quellcode

+ weniger Duplikation
+ Eclipse-Code refaktorisiert
- Technologiemix
- Boilerplate (Java-Javascript)

Konventionell

Aufwand

+ geradlinig
+ keine Browserschicht
- Neuimplementation Saros/I

Quellcode

- unnötig Logik in UI-Code
- Duplikationen

1. Entscheidung: Browser oder nicht

Freie Universität Berlin

Browser	**Konventionell**
Testbarkeit	**Testbarkeit**
+ einfacher - keine Frameworks	0 umständliche Frameworks + Eclipse-Seite vorhanden
Darstellungsmöglichkeiten	**Darstellungsmöglichkeiten**
+ grundsätzlich riesig - Browserbereich beschränkt	+ sehr gute IDE-Integration - beschränkt und umständlich

1. Entscheidung: Browser oder nicht

Freie Universität Berlin

Browser	Konventionell
Stabilität	**Stabilität**
0 vorsichtig optimistisch	+ unproblematisch
- Risiken	

1. Entscheidung: Pro Browser

- Aufwand vertretbar und interessantes Wissen
- Quellcode besser wartbar
- Stabilität: Herausforderung für den Prototyp

2. Entscheidung: Welcher Browser

- JavaFx
  - Beste Lösung (stabil)
  - Nicht kompatibel zu Java 6
- JxBrowser
  - proprietär, aber kostenlos
  - 200MB groß
  - Integration in Eclipse nicht unterstützt

⇒ SWT Browser
  - erprobt in Eclipse
  - Browser Erweiterung vorhanden
  - Native Swing

2. Entscheidung: Welcher Browser

- JavaFx
  - Beste Lösung (stabil)
  - Nicht kompatibel zu Java 6
- JxBrowser
  - proprietär, aber kostenlos
  - 200MB groß
  - Integration in Eclipse nicht unterstützt

⇒ SWT Browser

  - erprobt in Eclipse
  - Browser Erweiterung vorhanden
  - Native Swing

3. Entscheidung: Gleicher Browser

- Browsereinbindung umfangreich
  - ⇒ beachtliche Codeduplikation

⇒ SWT Browser für beide IDEs

Mein Fahrplan

Ziele

- Realisierbarkeit beantworten (Prototyp)
  - auf technischer Ebene
  - Stabilität und Kompatibilität
- Gerüst für letztendliche Realisierung vorgeben
  - Testframework und Implementierung
- Endergebnis: Prototyp + Wissen

Identifizierte Risiken

- Einbettung von SWT in IntelliJ Toolwindow
  - Zerstören und Neuerstellung des Browsers
  - bisher bereits viel Aufwand
  - Geht es besser?
  - keine Beispiele im Internet
- MAC OS
  - Cacoa Restriktion an UI

  ⇒ Work-around
- Zukunftssicherheit
  - SWT_AWT Brücke wenig benutzt
  - z. B. ungefixter Bug für Java 7 unter Mac OS
  - JavaFx als Nachfolger von Swing

Identifizierte Risiken

- Einbettung von SWT in IntelliJ Toolwindow
  - Zerstören und Neuerstellung des Browsers
  - bisher bereits viel Aufwand
  - Geht es besser?
  - keine Beispiele im Internet
- MAC OS
  - Cacoa Restriktion an UI

  ⇒ Work-around
- Zukunftssicherheit
  - SWT_AWT Brücke wenig benutzt
  - z. B. ungefixter Bug für Java 7 unter Mac OS
  - JavaFx als Nachfolger von Swing

Einschränkungen

- Browser funktioniert nicht unter Linux Eclipse 3.7 / 3.8
- Konfigurationsaufwand unter Linux:
  - zwei Browser (Webkit und Mozilla)
  - betrifft Eclipse und IntelliJ

  ⇒ ein Paket nachinstallieren

Baustellen

Realisierbarkeit beantworten: Herausforderungen

- Technologie funktioniert nicht vs. fehlendes Wissen in der Anwendung
- Unmenge an Kombinationen aus
  - Betriebsystemen (32- und 64-Bit)
  - JVM-Versionen
  - SWT-Versionen
  - Eclipse bzw. IntelliJ-Versionen
  - Betriebsystem-Browsern

Realisierbarkeit testen

- manuelle Tests
- Spezifikation und Kaputtklicken
  - Voraussetzung: funktionaler Prototyp
  - Äquivalenzklassen aus der Menge an Kombinationen
- Saros interner Test
  - Voraussetzung: Release
  - Build- und Release-Infrastruktur suboptimal
- Automatische Tests bedingt anwendbar

Integration Björns Browser

- Wrapper um SWT-Browser mit vielen nützlichen Funktionen
- Änderungen zurück ins ursprüngliche Projekt?
  - Relevante Klassen nach Saros kopieren
  - Alternative: Forken und als Bibliothek einbinden
- Schnittstellen abtesten und festigen
  - z. B. Laden von Resourcen noch vom Betriebssystem abhängig
- Was tun mit nicht-benötigter Funktionalität?

Programmierung der HTML GUI

- Java-Javascript Schnittstelle
  - Übergeben von Parametern und Fehlern
    - umständlich (Objektumwandlungen)
  - Javascript-Code in Java-String
    - fehlende IDE-Unterstützung
    - Debugging schwierig
  - Absprache zwischen 3 Leuten
  - flache, sehr lange Liste an Funktionen

Integration STF

- Saros-Testframework
- Klickroboter über RMI
- kein Selenium o. ä. benutzbar
  - Mapping zw. UI-Element und Java-Objekt fehlt
  - Björns Browser benutzen bzw. erweitern
- Schnittstellen
  - durch fehlendes Mapping Page Objects umständlich
  - Herausforderung: IDE-spezifische Teile

Weitere Fragen oder Anmerkungen?