Freie Universität Berlin

Abschlussvortrag
zur Diplomarbeit

„Stabilitäts- und Testbarkeitsverbesserungen der Netzwerkschicht in Saros“

von Björn Gustavs

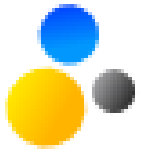
Saros

Eclipse Communication Framework

Stabilitätsverbesserungen

SAROS UND DIE NETZWERKSCHICHT

Saros



- Was ist „Saros“
  - Eclipse-Plugin für Programmierung in verteilter Gruppe (*Distributed-Party Programming*)
  - entstand als Diplomarbeit 2006 an der FU-Berlin
  - stetige Weiterentwicklung in der Arbeitsgruppe Software Engineering

Saros – DPP

- Programmierung mit verteilten Beteiligten
  - mehrere Entwickler arbeiten an verschiedenen Orten am gleichen Projekt
  - Hauptziele
    - Teilnehmer arbeiten zusammen an identischem Stand eines Projektes
    - Teilnehmer sehen wo andere im Code arbeiten
  - Wie wird dies erreicht
    - Änderungen und Awarenessinformationen eines Entwicklers werden an alle anderen übertragen und reproduziert/angezeigt

Saros – Netzwerkschicht

- Saros nutzt zur Kommunikation zwischen Teilnehmern einer Sitzung das XMPP (Jabber)
- Bibliothek „Smack“ (nach 2 Jahre wieder aktiv)

- alle Teilnehmer sind am XMPP-Netzwerk angemeldet
- Änderungsinformationen und weitere Statusinformationen werden als Nachrichten an andere Teilnehmer gesendet
- Datenübertragung als Bytestream
  - Socks5 Bytestream (dedizierte Datenverbindung)
  - In-Band Bytestream (Daten innerhalb von Nachrichten)

Saros – Netzwerkschicht verbessern

- Probleme aus Erfahrungen, Umfragen, Tests
  - Verbindungsaufbau
  - Verbindungsabbruch
  - Datendurchsatz
- verlässlicher Datenaustausch essentiell
- Verbesserungsmöglichkeit: Kommunikationsframework von Eclipse?

Eclipse Communication Framework

- Eclipse Communication Framework (ECF), was ist das?
  - stellt APIs für Interprozesskommunikation und auch Benutzerkommunikation bereit
    - APIs für Messaging, Presence, File Transfer, …
  - zielt darauf ab, Kommunikationsfunktionalitäten für Entwickler von Eclipse-Anwendungen bereitzustellen
  - Kommunikationsprotokolle werden gekapselt und als Provider zur Verfügung gestellt (XMPP, MSN, IRC, Yahoo, Bittorrent, …)

ECF Prüfung – Aufbau des ECF

- ECF sinnvoll, geeignet?
- Analyse des ECF, Aufbau, Funktionsweise
  - ECF bietet Provider, nur XMPP-Provider unterstützt wünschenswerte APIs
  - XMPP Provider nutzt Smack, ohne Mehrwert
  - Neben Nachrichten, Datenaustausch nur via File Transfer API

ECFConnection	**Saros**

`Chat localChat = connection.getChatManager().`
`  createChat(receiverName, new MessageListen-`
`er()…);`
 	
`localChat.sendMessage(aMsg);`
	`Chat chat = getChat(jid);`
	`chat.sendMessage(message);`

ECF Prüfung - Prototyp

- Entwicklung eines experimentellen Prototyps
  - Ziel: praktische Erfahrungen mit Integration des ECF
  - Implementierung eines Eclipse-Plugins zum Nachrichtenaustausch zwischen Instanzen
  - Erkenntnis: Einsatz des ECF führt zu ähnlicher Einbindung der Netzwerkschicht → gleicher Implementierungsaufwand

ECF Fazit

- Fazit: Einsatz von ECF nicht förderlich
  - selbe Basis (Smack)
  - kein Zusatznutzen (Absicherung, Abstraktion)
  - strukturell ähnliche Integration

- neues Ziel: aktuelle Netzwerkschicht stabilisieren
- Quellen für Versagen
  - Fragebögen
  - Bug Reports im Team
  - Tests

Bekanntes Versagen

- Bemängelung in Fragebögen (bzgl. Netzwerk):
  - Aufbau der Datenverbindung
  - geringe Geschwindigkeit
- Erfahrungen
  - langsame Datenverbindung
  - mangelnde Stabilität der Datenübertragung
- Tests
  - langsame Datenverbindung
  - Probleme beim Aufbau der Datenverbindung

→ Debugging

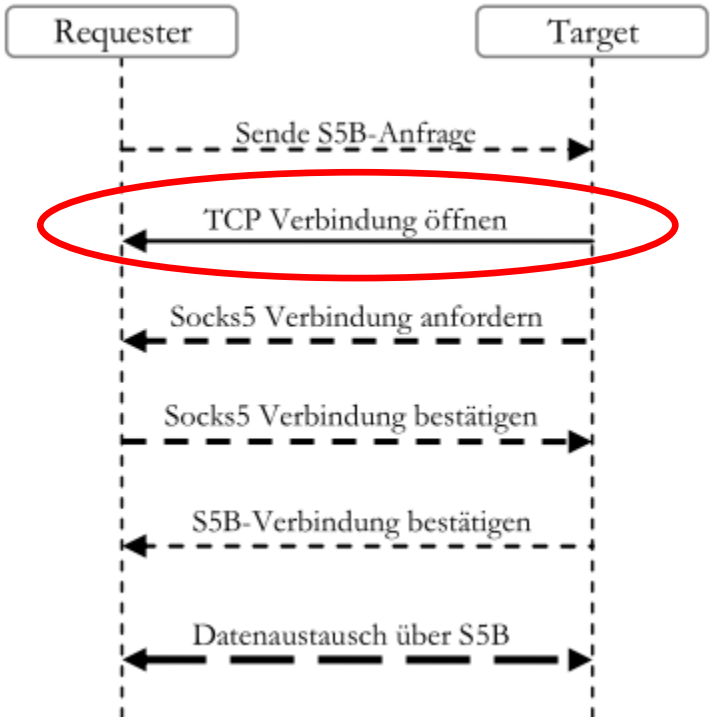
  - Verstehen der Netzwerkschicht
  - Nachverfolgung der Netzwerkkommunikation

Realisierte Verbesserungen in Netzwerkschicht

- Probleme beim Einladungsprozess behoben
  - Abbrüche und Hängenbleiben
- Bugfix in Smack bzgl. Umbenennung im Roster
  - Patch in Bug Tracker aufgenommen, Issue geöffnet
- In-Band Bytestream stabilisiert
  - Timeout von Paketbestätigungen
- Verfügbarkeit des Socks5 Bytestream erhöht
  - Was waren die Probleme?

Socks5 Bytestream, Probleme

- Requester ermittelte und übermittelte falsche Adressen
  - nur eine oder eine ungültige IP → lokale IPs aller Netzwerkadapter bestimmen
- Rechner im LAN nicht von außen adressierbar
  - globale IP mittels STUN bestimmen
  - Gateway kann Erreichbarkeit verhindern → UPnP



UPnP

- Universal Plug and Play
  - Protokollsammlung, selbstständige Konfiguration von Netzwerkgeräten zwecks Zusammenarbeit
  - Geräteunabhängig, baut auf IP, UDP, TCP, HTTP, SOAP auf
- Warum für Saros?
  - Saros hinter Router nicht von außen erreichbar
  - Ziele
    - Portweiterleitung konfigurieren
    - Socks5 Bytestream ermöglichen

UPnP Integration

- Nutzung der Bibliothek weupnp
  - Open Source, LGPL
  - klein, auf Portweiterleitung spezialisiert (IGD Protokoll)
  - http://code.google.com/p/weupnp/
- Verbesserungen
  - Unterstützung für mehrere Netzwerkadapter
  - GUI für Testanwendung
  - Unterstützung für Lease Duration
  - Refactoring
- Kontakt mit Projektgründern, Verbesserungen werden integriert

UPnP Integration II

- Ziele der UPnP Integration
  - UPnP-Unterstützung ist optional
  - Portweiterleitung nur wenn aktiviert, automatisch
  - Portweiterleitungen wieder aufheben wenn nicht mehr benötigt
- Saros hinter Router kann somit von außen erreicht werden
  - Test zwischen Saros-Heim und Saros-FU

Faktoren

Bewertung von Testbarkeitsfaktoren Saros

TESTBARKEIT

Testbarkeit

- Testbarkeit: Grad, wie sehr eine Software Tests ermöglicht und unterstützt.
- Faktoren
  - Beobachtbarkeit (interne Daten, Zustände beobachten)
    - Testschnittstellen, Ereignisprotokolle, Assertionen
  - Kontrollierbarkeit (Eingaben injizieren, Zustände auslösen)
    - Exception seeding
    - Fault injection
    - Test points (interne Daten einsehen und ändern)
  - Komplexität
  - Innere Abhängigkeiten
  - Trennung der Belange

Testbarkeitsfaktoren in NWS Saros I

- Testbarkeitsfaktoren in Netzwerkschicht von Saros untersucht
  - Asserts: 12 in 848 Methoden
  - Testschnittstellen: Saros nein, Smack ja
  - Ereignisprotokollierung: ausgiebig (300 Ausgaben)
  - Kontrollierbarkeit (Exception seeding, Fault injection, Test points): ungenutzt

Testbarkeitsfaktoren in NWS Saros II

- Komplexität, beispielsweise
  - Umfang: knapp 10.000 LOC
  - Kaskadierung von Methodenaufrufen: ø4 Ebenen (max. 6)
  - Anzahl von Methodenparameter: ø2-3 (max. 9)
- Testgetriebene Entwicklung: gefördert
- Singletons: in Saros eingesetzt, schlecht für Unittests
- Trennung der Belange: Plugin & Netzwerkschicht vermischt

Testbarkeit, SarosNet

- Ein Beispiel für mangelnde Trennung von Belangen:
  - Plugin-Klasse `Saros` und Netzwerkschicht vermischt
  - Ziel: Trennung dieser Belange, Netzwerkkomponenten aus `Saros` in `SarosNet`

SarosNet

- `SarosNet` gut, weil
  - Klasse `Saros` aufgeräumt
    - Verständlicher, kein Netzwerkcode
    - übersichtlicher (auf 60% LOC geschrumpft)
  - Netzwerk-Tests möglich ohne `Saros` Instanz
    - zwei `SarosNet` Instanzen verbinden sich ohne `Saros`

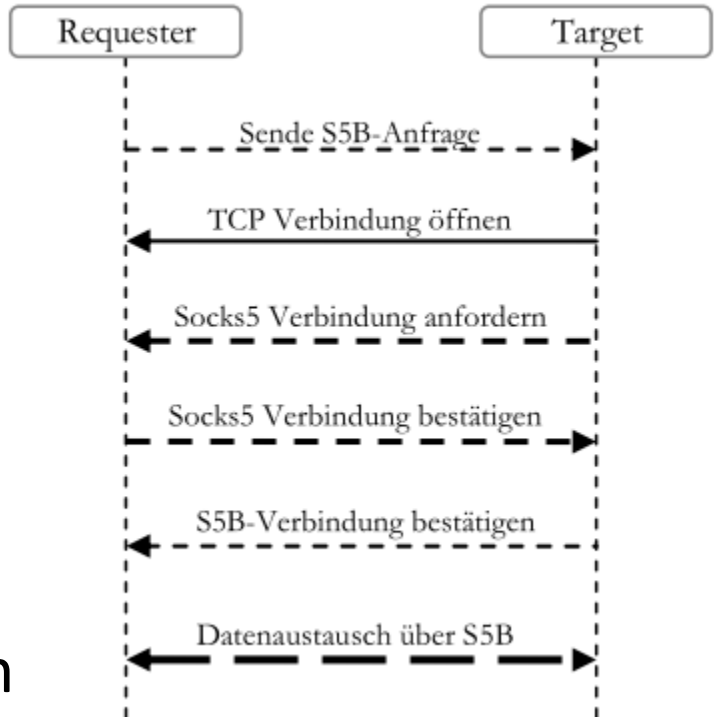
Zusammenfassung

- ECF evaluiert und abgewiesen
- Versagen in Netzwerkschicht zusammengetragen und korrigiert
  - Stabilität beim Einladungsprozess verbessert
  - Stabilität von Bytestreams verbessert
  - Verfügbarkeit von Socks5 Bytestreams verbessert
    - Lokale und globale Adressermittlung
    - Integration von Universal Plug and Play
- Testbarkeit der Netzwerkschicht analysiert und an einem Punkt verbessert

Danke

Socks5 Bytestream, Funktionsweise

1. Requester sendet Service-Discovery an Server, u.a. Proxyermittlung
2. Aushandeln der Verbindung mit Target
   - Unterstützte Features (Bytestream?)
3. Requester sendet S5B Anfrage an Target (inkl. StreamHost-Adressen)
4. Target versucht TCP Verbindung aufzubauen
5. Socks5 Verbindung und Bytestream aufbauen



Socks5 Bytestream, Service Discovery

- Client sendet Service Discovery Anfrage an Server:

```
<iq id="vYFP6-23" to="saros-con.imp.fu-berlin.de"
type="get">
  <query xmlns="http://jabber.org/protocol/disco#items"/>
</iq>
```

- Server antwortet auf Anfrage:

```
<iq id="vYFP6-23" to="bg2@saros-con.imp.fu-berlin.de/Saros"
   from="saros-con.imp.fu-berlin.de" type="result">
  <query xmlns="http://jabber.org/protocol/disco#items">
    <item jid="conference.saros-con.imp.fu-berlin.de"/>
    <item jid="irc.saros-con.imp.fu-berlin.de"/>
    <item jid="proxy.saros-con.imp.fu-berlin.de"/>
    <item jid="vjud.saros-con.imp.fu-berlin.de"/>
  </query>
</iq>
```

Socks5 Bytestream – Verbindung aushandeln

- Requester sendet StreamHost-Konfiguration an Target

```
<iq id="vYFP6-29" to="bg1@saros-con.imp.fu-berlin.de/Saros" type="set">
  <query xmlns="http://jabber.org/protocol/bytestreams"
sid="js5_17501995492966118 69" mode="tcp">
    <streamhost jid="bg2@saros-con.imp.fu-berlin.de/Saros"
host="87.77.220.24" port="7777"/>
    <streamhost jid="bg2@saros-con.imp.fu-berlin.de/Saros"
host="169.254.132.138" port="7777"/>
    <streamhost jid="bg2@saros-con.imp.fu-berlin.de/Saros"
host="192.168.110.1" port="7777"/>
        <streamhost jid="proxy.saros-con.imp.fu-berlin.de"
host="160.45.111.17" port="7777"/>
  </query>
</iq>
```

- Target versucht nacheinander Verbindung aufzubauen, informiert Requester über erfolgte Socks5 Verbindung

```
<iq id="vYFP6-29" to="bg2@saros-con.imp.fu-berlin.de/Saros" from="bg1@saros-
    con.imp.fu-berlin.de/Saros" type="result">
  <query xmlns="http://jabber.org/protocol/bytestreams">
    <streamhost-used jid="bg2@saros-con.imp.fu-berlin.de/Saros"/>
  </query>
</iq>
```

Socks5 Bytestream, Mediated

- Requester sendet Anfrage an Target, inkl. seiner Adresse
- Target führt Verbindungsaufbau mit Proxy durch
- Requester führt Verbindungsaufbau mit Proxy durch
- Vermittlung wird aktiviert