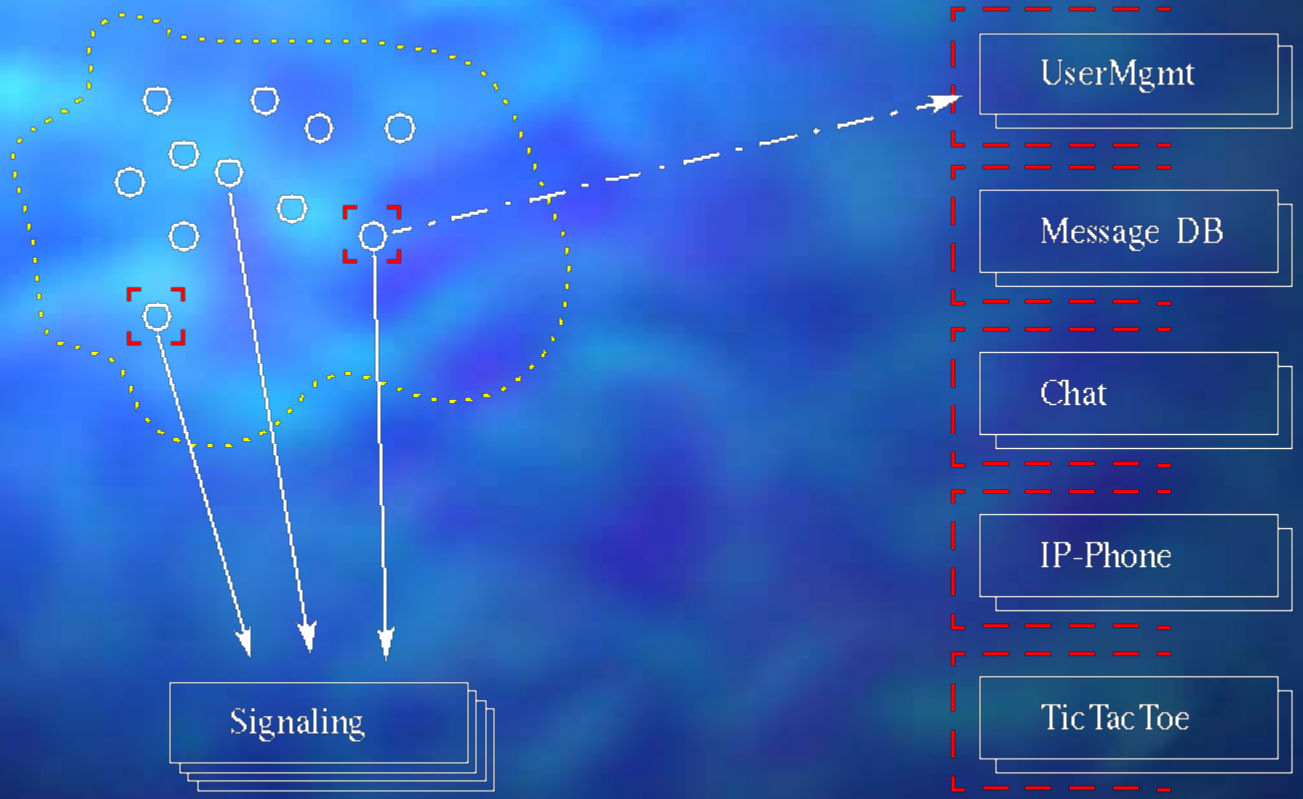
Instant Messaging

- Status
- Nachrichten
- Multicast Daten
- Chat ?
- IETF IMPP
- IMUnified

BlueMessenger Server



Designentscheidungen

- TCP-Verbindungen
- Kein Status an Kanälen
- Kein Verzeichnisdienst im Signalserver
- Limitierte Packetgrösse
- Unabhängige Diensteverbindungen

TCP statt Multicast

- Verschlüsselung der Verbindungen
- TCP/GSM ohne ACK möglich
- Firewalls
- Packetverlustgewitter
- IMPP Anforderung
- Multicast on the server

Statusfreie Kanäle

- Netztrennung -> Divergenz
- Statusabgleich ist teuer (IRC)
- Speicherbedarf

- Zugriffsrechte via Ticket
- Ticketentzug via Namensänderung

Externer Verzeichnisdienst

- Minimalistischer Server
- Kein externer Speicher nötig
- Einbindung anderer Suchmechanismen
- Niedrigere Schutzanforderungen

Limitierte Paketgrösse

- Einfache Pufferbehandlung
- Minimierung vom system calls
- Bessere Memory Management
- Geringe Paketverzögerung

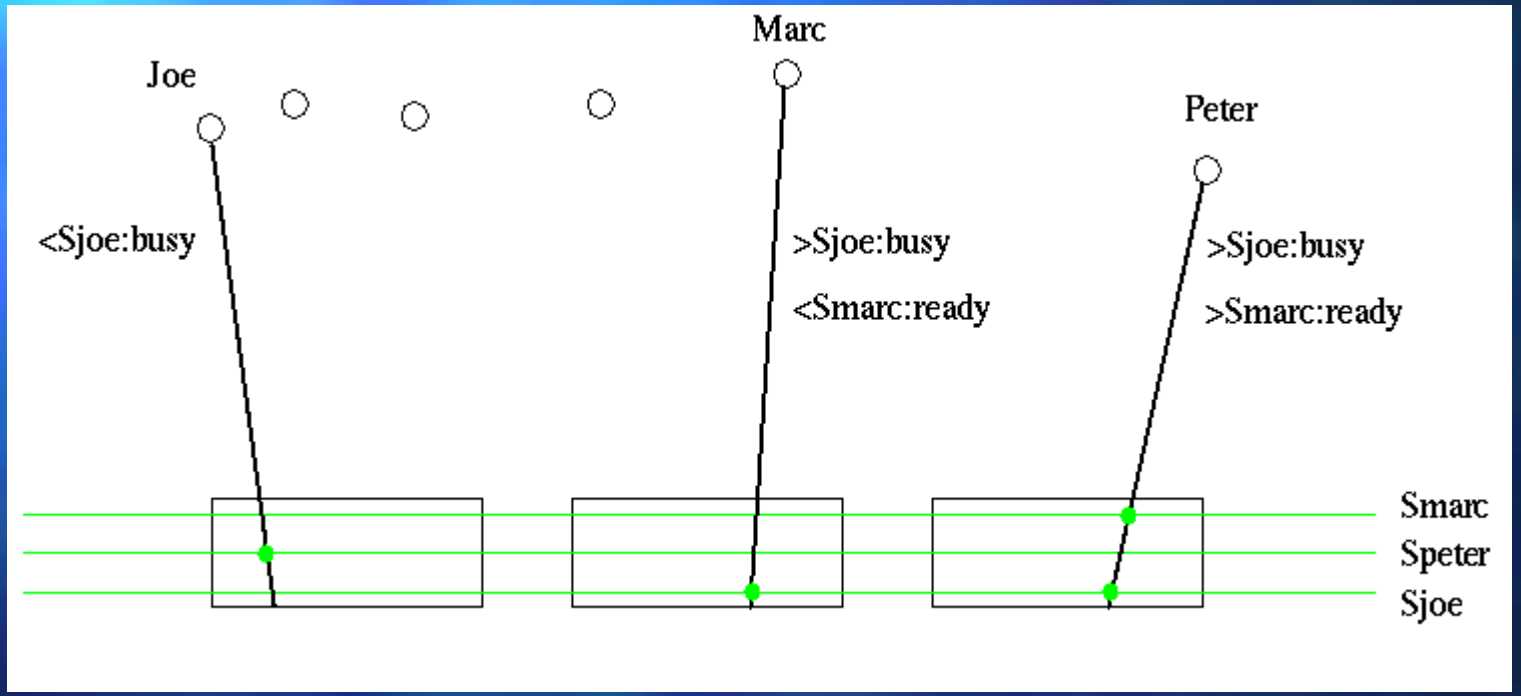
- Geordnete Auslieferung

Unabhängige Dienstverbindungen

- Skalierbarkeit
- Vereinfachte Administration
- Verschiedene Anforderungen
- Client2Client Dateitransfer

- Setup and Inform

Verteiltes Kanalverzeichnis



Verteiltes Kanalverzeichnis 2

- Flaschenhals der Skalierbarkeit
- Vermeiden von Lookups via IDs
- Speicherreduktion via Komprimierung
  - ideale Hashfunktion
  - Rezyklierung toter Kanal IDs
  - ausserhalb der eigenen Domain ?

Verteiltes Kanalverzeichnis 3

- Subscribe statt Publish
- Bootstrap <-> subscribe/unsubscribe
- Backbone unsubscribe via GC
  - keine Kollisionen mit Bootstrap
  - Lebensdauer von Kanälen ?
  - weniger Verzeichnislocks

Threads vs. /dev/poll

- Threads sind einfach
- Dispatch ist Kernkompetenz des Kernels
- Limit ca. 3000-4000 Threads
- OutputStream.write() blockiert!
- Neues Java IO API mit callbacks

JNI Poller

- jdk-1.2.2/demo/jni/Poller
- ca. 20% langsamer
- über 10.000 Verbindungen möglich

Java Synchronisation

- Uncontented synchronized ~12 Instr.
- Datenabgleich zwischen Threads teuer
  - einmal schreiben, n-mal lesen
- Java kennt keine Arrays
  - volatile nutzlos

Double check idiom

```
if (msg == null) {
   synchronized(lock) {
      if (msg == null) {
         msg = new Message();
      }
   }
}
msg.use();
```

Lazzy evaluation richtig

```
private synchronized getMsg () {
    if (msg == null) {
        msg = new Message();
    }
    return msg;
}

...
getMsg().use();
```

Java Server Performance

- IO Antwortzeiten vergleichbar C
- IO Durchsatz vergleichbar C
- SSL problematisch
- JDK 1.2.2 evtl. schneller als Hotspot

Antwortzeiten C / Java

Paketgrösse 1024 Byte:

total pingpong time =	2341 ms -> 0.4682 ms/call	1.00x (C Server)
total pingpong time =	3229 ms -> 0.6459 ms/call	1.38x (Java Server)

Paketgrösse 128 Byte:

total pingpong time =	1302 ms -> 0.2604 ms/call	1.00x (C Server)
total pingpong time =	2162 ms -> 0.4325 ms/call	1.66x (Java Server)

- Java Server mit JDK 1.2.2
- Zeitmessung auf Client-Seite mit gethrtime()
- Client sendet Ping, Server liest Ping, Server sendet Pong, Client liest Pong
- 5000 Ping/Pongs
- C Client auf anderem Rechner im gleichen Ethernet Segment
- Solaris 7, TCP_NODELAY=1

JNI native -> Java

- VM ~1.5 MB
- auxiliary Funktionen
- Server Core schützen
- Multithreaded Apache still in work
- Zeit aufwenden für gute Algorithmen nicht für Fehlersuche

Hotspot rules

- Object nursing area pro Thread
- almost copying garbage collector
- Optimierung mit Laufzeitinformation
  - Verwendung von Interfaces gratis
  - Accessor Funktionen gratis
- Medium Lifetime Objects sind schlecht
- Object Setup nicht ignorieren

Asortiertes Know-How

- Interrupted, InterruptedException
- finally
- final
- javac -O -g:none

Thread.interrupted()

- Löscht interrupted Flag!

```
if (Thread.interrupted()) {
    throw new InterruptedException();
}
```

InterruptedException

- Immer propagieren!

```
try {
   libObject.doYourThing();
}
catch (InterruptedException ie) {
   Thread.currentThread().interrupt();
}
```

finally

```
final InputStream is = ...
try {
    ... use it ...
    return result;
}
catch (IOException ioe) {
    ... handle errors, even throw Exception ...
}
finally {
    is.close();   // -- always excecuted --
}
```

final

```
final String name;
if (fullname.indeOf(´.´) != -1) {
    name = fullname.substring(fullname.indexOf(´.´));
}
else {
    name = fullname;
}
```

- Funktioniert auch für Instanzvariablen

javac -O ist ein no op

- -O wird ab JDK 1.2 ignoriert
- Javac -g:none verkleinert Klassen
- javac/jikes erkennt dead code

```
private final boolean DEBUG = false;
...
if (DEBUG) {
        ... this will be gone ...
}
```

gamelab.ch

- Plugin Mechanismus
- Uebertragbar auf beliebige IM
- Kundenspezifische Plugins
  - Calender
  - Ringruf
  - LDAP-Search
  - Games ...

Referenzen

Doug Lea - Concurrent Programming in Java

http://gee.cs.oswegu.edu/

IMUnified

http://www.imunified.org/

IMPP (IETF)

http://www.imppwg.org

BlueMessenger (BlueWin)

http://www2.bluewin.ch/services/bluemessenger/

Messenger Plugin (gamelab.ch)

http://gamelab.ch/

/dev/poll Beispiel

jdk-1.2.2/demo/jni/Poller/