Bullshit-Bingo
Gruppe 4

Faraz Ahmed
Felix Bruckner
Dennis Cisowski
Michael Stapelberg
Thorsten Töpper

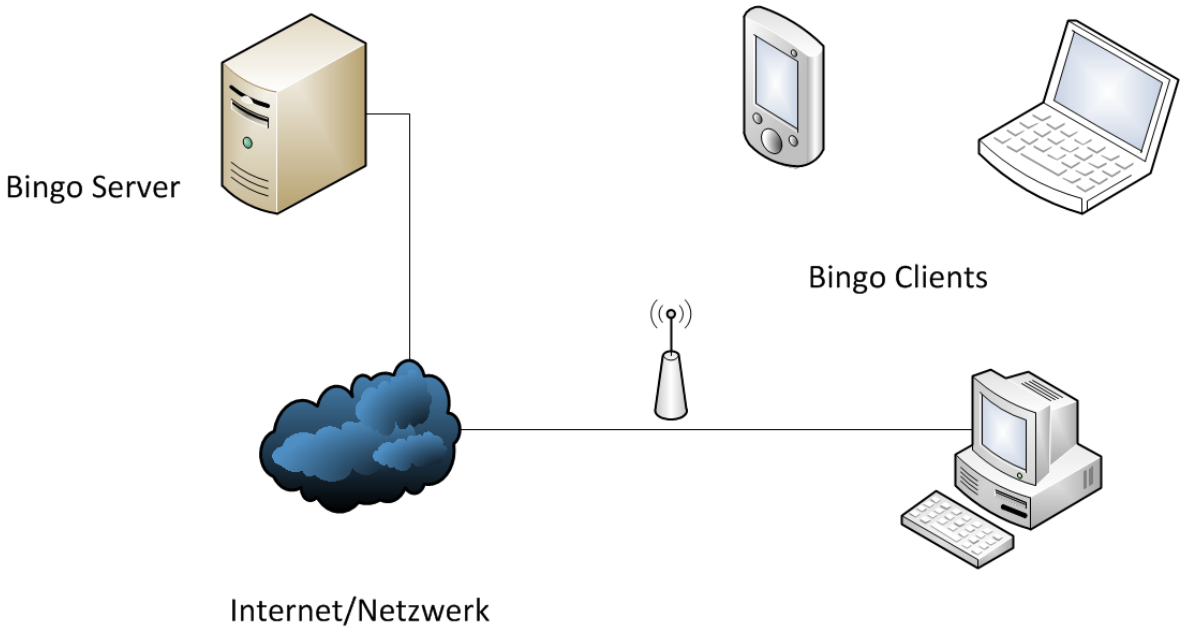
Fakultät für Informatik
Hochschule Mannheim

8.4.2011

Agenda

- Architektur & Kommunikationskonzept
- Server
- Qt-Client
- Java-Client
- Android-Client
- AJAX-basierte Web-Anwendung

Architektur



- Klassisches Client/Server-Modell
- Leicht zu implementieren
- Im Bingo-Kontext sinniger als z.B. Peer-2-Peer

Spiel-Architektur, Persistenz

- Jeder Spieler erhält ein „Token“ (24h gültig), jedes Spiel eine „ID“
- Wortlisten auf dem Server, werden pro Spiel und pro Spieler gemischt
- Spielzüge werden gespeichert und ausgewertet
- Clients holen sich wiederholt die Spielerliste und fragen nach Gewinner

Kommunikation: HTTP & JSON

- Protokoll: HTTP
  - ASCII - leicht zu parsen
  - Implementierungen für (fast) jede Programmiersprache
  - Load-Balancing/Proxies möglich
- Serialisierung: JSON
  - JavaScript Object Notation
  - Leichtgewichtiger, textbasierter Datenaustausch
  - Elementare Datenstrukturen: Arrays, Hashes/Objects, Strings, ...
  - Selbstdefiniertes Bingo-Protokoll:

```
Request: POST /RegisterPlayer
Body: {"nickname":"sECuRE"}
Response:
{"success":true,
 "error":"",
 "token":"939393"}
```

Server

- Geschrieben in modernem Perl, objekt-orientiert
- Test-Driven Development
- Verwendete Module:
  - Moose (Objektsystem für OOP)
  - Tatsumaki (Asynchrones Web-Framework)
  - JSON::XS (JSON En-/Decode)
  - Test::More (Test-Framework)

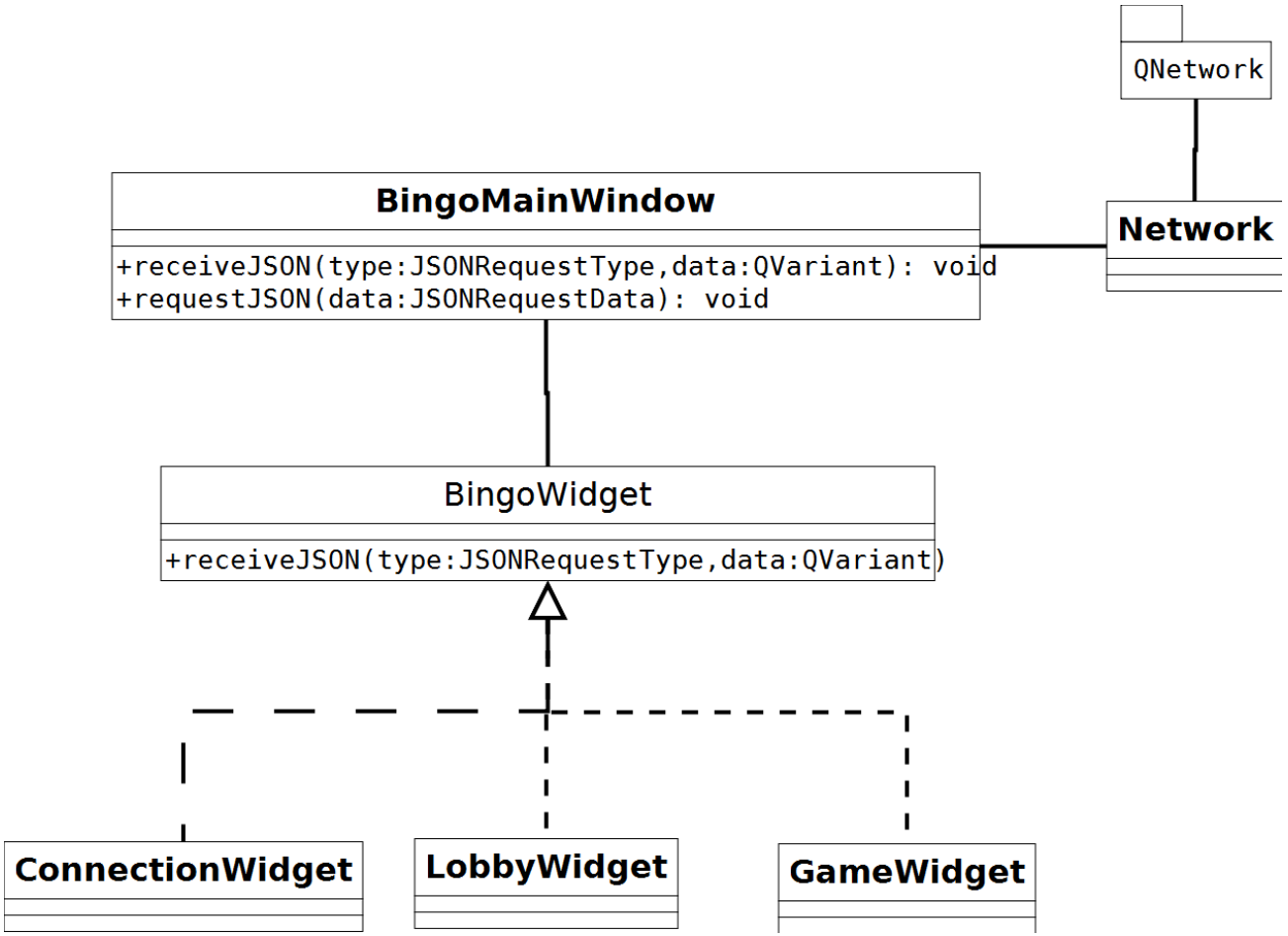
Code-Beispiel (Server)

```
package Player;
use Moose;
use Digest::SHA1 qw(sha1_hex);

has 'nickname' => (is => 'ro', isa => 'Str',
                   required => 1);
has 'token' => (is => 'ro', isa => 'Str',
                builder => 'build_token');

sub build_token {
    my @random_values;
    push @random_values, rand(255) for (1..10);
    return sha1_hex(join('', @random_values));
}
1
```

Qt-Client - Eingesetzte Software

- Qt
  - GUI-Bibliothek für C++
  - Für alle gängigen Plattformen verfügbar
  - Viele Zusatzbibliotheken (z.B. QNetwork)
- CMake
  - = "Cross-Platform Make"
  - Eine Art Meta-Makefile
  - Ein Makefile für alle Plattformen
- QJson
  - JSON-Erweiterung für Qt
  - Serialisiert Klassen in Request-Daten
  - Liefert Antworten als abstrakten `QVariant` Datentyp

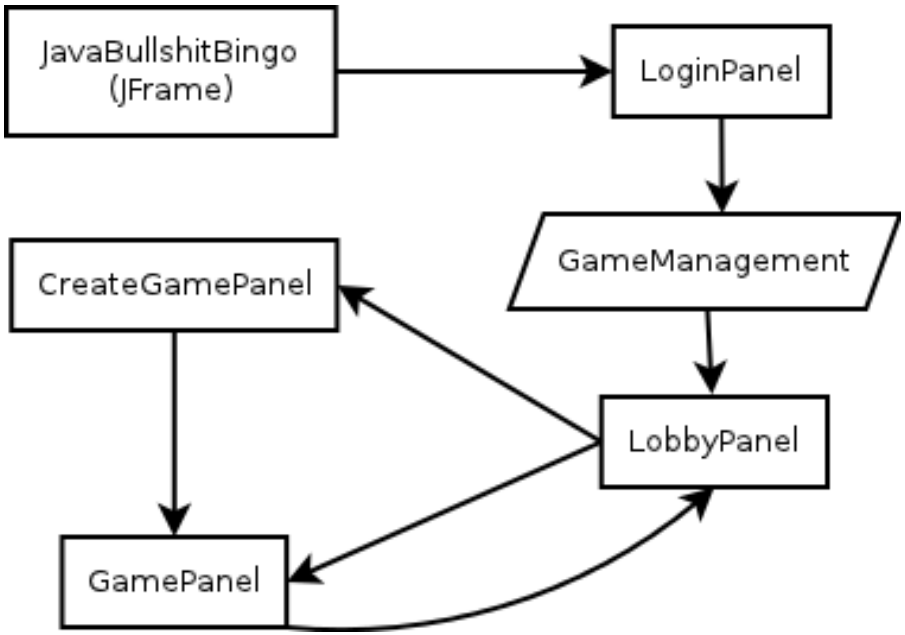
Qt-Client - Architektur



Java-Client

- GUI in Swing
- JSON.org Java Implementierung
- POST/GET via HTTPClient-Bibliothek der Apache Foundation

Java-Client - Architektur



Android-Client

- Geschrieben in Mirah
- Ruby-ähnliche Sprache für die JVM

Code-Beispiel

```
import android.preference.PreferenceActivity

class Preferences < PreferenceActivity
  def onCreate(saved:Bundle)
    super(saved)

    addPreferencesFromResource R.xml.preferences
  end
end
```

Android-Client - Großes Geheimnis

- Lüften des großen Geheimnisses

AJAX-basierte Web-Anwendung

- Variante Nummer 4 :)
- Basierend auf jQuery
- JSON-Support durch jQuery gegeben
- bbq-Plugin für Zustände (Token/Spiel-ID)
- Quick & Dirty, mal schauen was so geht (in < 2 Stunden implementiert)