Client-Server Kommunikation Twixt

Software-Challenge Germany 2016
Stand 19. August 2015

Inhaltsverzeichnis

1. Einleitung

Zum Inhalt

Wie in den letzten Jahren wird zur Client-Server Kommunikation ein XML-Protokoll genutzt[1]. In diesem Dokument wird die Kommunikationsschnittstelle definiert, sodass ein komplett eigener Client geschrieben werden kann. Es wird hier nicht die vollständige Kommunikation dokumentiert bzw. definiert, dennoch alles, womit ein Client umgehen können muss, um spielfähig zu sein.

An wen richtet sich dieses Dokument?

Die Kommunikation mit dem Spielserver ist für diejenigen, die aufbauend auf dem Simpleclient programmieren, unwichtig. Dort steht bereits ein funktionierender Client bereit und es muss nur die Spiellogik entworfen werden.
Nur wer einen komplett eigenen Client entwerfen will, beispielsweise um die Programmiersprache frei wählen zu können, benötigt die Definitionen.

Hinweise

Falls Sie beabsichtigen sollten, diese Kommunikationsschnittstelle zu realisieren, sei darauf hingewiesen, dass es im Verlauf des Wettbewerbes möglich ist, dass weitere, hier noch nicht aufgeführte Elemente zur Kommunikationsschnittstelle hinzugefügt werden. Um auch bei solchen Änderungen sicher zu sein, dass ihr Client fehlerfrei mit dem Server kommunizieren kann, empfehlen wir Ihnen, beim Auslesen des XML jegliche Daten zu verwerfen, die hier nicht weiter definiert sind.

Die vom Institut bereitgestellten Programme (Server, Simpleclient) nutzen eine Bibliothek um Java-Objekte direkt in XML zu konvertieren und umgekehrt. Dabei werden XML-Nachrichten nicht mit einem newline abgeschlossen.

Wie das Dokument zu lesen ist

Es finden sich für einzelne Arten von Nachrichten kurze Definitionen. Dabei ist ein allgemeiner XML-Code gegeben, bei dem Attributwerte durch Variablen ersetzt sind, die über dem Code kurz erläutert werden.
Eingebettete XML-Elemente werden hier allgemein als

`{TAGNAME}`

geschrieben, wenn an der Stelle eine beliebige Anzahl von *TAGNAME*-Tags eingebettet sein *kann* und als

`[TAGNAME]`

[1] siehe auch: SC Wiki: Die Schnittstelle zum Server

, wenn an der Stelle ein *TAGNAME*-Tag optional stehen kann.

1.1. Beispiel-Definition

Die Definitionen von a- und b-Tags unten lassen unter anderem folgende a-Tags zu:

```
- <a name="Sinnvoll">
  <\a>
- <a name="Sinnvoll">
      <b anzahl="2"/>
  <\a>
- <a name="Hans">
      <b anzahl="2"/>
      <b anzahl="2"/>
      <b anzahl="2"/>
  </a>
```

Definition 'b'

```
<b anzahl="2">
```

Definition 'a'

N Ein beliebiger Name

b ein b-Tag wie in obiger Definition

```
<a name="N" >
{b}
<\a>
```

Teil I.
Client → Server

2. Spiel betreten

2.1. Ohne Reservierung

Betritt ein beliebiges offenes Spiel:

```
<join gameType="swc_2016_twixt"/>
```

2.2. Mit Reservierungscode

Ist ein Reservierungscode gegeben, so kann man den durch den Code gegebenen Platz betreten.

RC Reservierungscode

```
<joinPrepared reservationCode="RC"/>
```

3. Züge senden

3.1. Der Move

Der Move ist der allgemeine Zug, der in verschiedenen Varianten gesendet werden kann.

RID ID des Raumes

ZUG Informationen über den Zug

```
<room roomId="RID">
  ZUG
</room>
```

3.2. Zug

X die x-Koordinate des Feldes

Y die y-Koordinate des Feldes

```
<data class="move" x="X" y="Y"/>
```

3.3. Debughints

Zügen können Debug-Informationen beigefügt werden:

S Informationen zum Zug als String

```
<hint content="S"/>
```

Damit sieht beispielsweise ein Laufzug so aus:

```
<room roomId="RID">
  <data class="move" x="3" y="7">
    <hint content="S"/>
    <hint content="noch ein Hint"/>
  </data>
</room>
```

Teil II.
Server → Client

RID ID des Raumes, in dem das Spiel stattfindet

4. Raum beigetreten

```
<joined roomId="RID"/>
```

5. Willkommensnachricht

Der Spieler erhält zu Spielbeginn eine Willkommensnachricht, die ihm seine Farbe mitteilt.

C Spielerfarbe (red/blue)

```
<room roomId="RID">
  <data class="welcomeMessage" color="C"/>
</room>
```

6. Spielstatus

Es folgt die Beschreibung des Spielstatus, der vor jeder Zugaufforderung an die Clients gesendet wird. Das Spielstatus-Tag ist dabei noch in einem *data*-Tag der Klasse *memento* gewrappt:

6.1. memento

status wie in 6.2 definiert

```
<room roomId="RID">
  <data class="memento">
   status
  </data>
</room>
```

6.2. Status

Z aktuelle Zugzahl

S Spieler, der das Spiel gestartet hat (RED/BLUE)

C Spieler, der an der Reihe ist (RED/BLUE)

red, blue wie in 6.3 definiert

board Das Spielbrett, wie in 6.4 definiert

lastMove Letzter getätigter Zug (nicht in der ersten Runde), wie in 6.6 definiert

condition Spielergebnis, wie in 11 definiert; nur zum Spielende

```
<state class="state" turn="Z" startPlayer="S" currentPlayer="C">
  red
  blue
  [board]
  [lastMove]
  [condition]
</state>
```

6.3. Spieler

C Farbe (RED/BLUE)

N Anzeigename

P Punktekonto

```
<C displayName="N" color="C" points="P"/>
```

6.4. Spielbrett

FIELD Ein Spielfeld wie in 6.5 definiert.

```
<board>
 <fields>
  {FIELD}

   ... (576 St\"uck davon)
 </fields>
 <connections>
  {CONNECTION}
 </connections>
</board>
```

6.5. Spielfeld

Owner Spielerfarbe des Besitzers (RED/BLUE). null falls kein Besitzer

Type Typ des Feldes (NORMAL/SWAMP/RED/BLUE).

X X-Koordinate des Feldes

Y Y-Koordinate des Feldes

```
<field owner="Owner" type="Type" x="X" y="Y"/>
```

6.6. Letzter Zug

Der letzte Zug ist ein Move (siehe hierzu 3.2).

ZUG Informationen über den Zug.

```
<lastMove class="T" ZUG/>
```

7. Zug-Anforderung

Eine simple Nachricht fordert zum Zug auf:

```
<room roomId="RID">
  <data class="sc.framework.plugins.protocol.MoveRequest"/>
</room>
```

8. Fehler

Ein „spielfähiger“ Client muss nicht mit Fehlern umgehen können. Fehlerhafte Züge beispielsweise resultieren in einem vorzeitigen Ende des Spieles, das im nächsten gesendeten Gamestate erkannt werden kann (siehe 11).

MSG Fehlermeldung

```
<room roomId="RID">
<error message="MSG">
<originalRequest>
Request, der den Fehler verursacht hat
</originalRequest>
</error>
</room>
```

9. Spiel pausiert

Ein „spielfähiger“ Client muss Pausierungsnachrichten nicht beachten, da er nur auf Aufforderungen (Zug-Aufforderung siehe 7) des Servers handelt.

N Spielername

C Spielerfarbe

P Punktekonto

F Eisschollenkonto

```
<room roomId="RID">
<data class="paused">
<nextPlayer class="player" displayName="N" color="C" points="P" fields="F"/>
</data>
</room>
```

10. Spiel verlassen

```
<left roomId="RID"/>
```

11. Spielergebnis

Zum Spielende enthält der Spielstatus eine *Condition*, der das Spielergebnis entnommen werden kann:

W Gewinner (RED/BLUE), bei Unentschieden wird kein winner mitgeschickt.

R Text, der den Grund für das Spielende erklärt

```
<condition winner="W" reason="R"/>
```

Teil III.
überblick

Hier nochmal ein kurzer überblick, der etwas genauer zeigt, wie die Kommunikation ablaufen kann.

1. Ein Spielserver startet ein Spiel und wartet auf den Client (die Clients).
2. Der Client stellt eine TCP Verbindung zum Spielserver her (er kennt dessen IP-Adresse und Port)
3. Die Verbindung ist aufgebaut und der Client sendet

```
<protocol>
    <join gameType="swc_2016_twixt"/>
```

4. Der Server sendet

```
<protocol>
    <joined roomId="65d3d704-87d9-42eb-8995-51c3e6324513"/>
```

5. der Client merkt sich die roomId.
6. der Server startet das Spiel und sendet

```
<room roomId="65d3d704-87d9-42eb-8995-51c3e6324513">
    <data class="welcomeMessage" color="blue"/>
</room>
```

7. der Client merkt sich seine Spielerfarbe.
8. der Server sendet das Ausgangsspielfeld:

```
<room roomId="65d3d704-87d9-42eb-8995-51c3e6324513">
  <data class="memento">
    <state class="state" turn="0" startPlayer="RED" currentPlayer="RED">
      <red displayName="Spieler 1" color="RED" points="0"/>
      <blue displayName="Spieler 2" color="BLUE" points="0"/>
      <board>
        <fields>
         <field type="SWAMP" x="0" y="0"/>
         <field type="BLUE" x="0" y="1"/>
```

```
          <field type="BLUE" x="0" y="2"/>
          <field type="BLUE" x="0" y="3"/>
          <field type="BLUE" x="0" y="4"/>
          ...
          <field type="BLUE" x="23" y="22"/>
          <field type="SWAMP" x="23" y="23"/>
        <fields>
        <connections/>
      </board>
    </state>
  </data>
</room>
```

9. da der andere Spieler anfängt folgt noch eine *memento*-Nachricht vom Server, die den Spielstatus nach dem ersten Zug von Spieler 1 beinhaltet.

10. es folgt die erste Zug-Aufforderung vom Server:

```
<room roomId="65d3d704-87d9-42eb-8995-51c3e6324513">
    <data class="sc.framework.plugins.protocol.MoveRequest"/>
</room>
```

11. Der Client antwortet mit einem Zug:

```
<room roomId="65d3d704-87d9-42eb-8995-51c3e6324513">
    <data class="move" x="1" y="2"/>
</room>
```

12. so geht es weiter...

13. die letzte Nachricht enthält ein

```
</protocol>
```

    Daraufhin wird die Verbindung geschlossen