Client-Server Kommunikation Cartagena

Software-Challenge Germany 2013
Stand 26. Oktober 2012

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_2013_cartagena"/>
```

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 MoveContainer

Der MoveContainer kann bis zu 3 Teilzüge beherbergen. Sollen weniger als 3 Teilzüge mitgeschickt werden, können third-, second- oder auch firstMove weggelassen werden.

RID ID des Raumes

ZUG je nach Art des Zuges Informationen über diesen. (siehe Abschnitt 3.2 und 3.3)

```
<room roomId="RID">
  <data class="moveContainer">
    <firstMove ZUG/>
    <secondMove ZUG/>
    <thirdMove ZUG/>
  </data>
</room>
```

3.2. Vorwärtszug

ZUGNUMMER firstMove, secondMove oder thirdMove

FID der Index des Startfeldes

SYMBOL das Symbol der Karte, welche abgelegt werden soll.(siehe Abschnitt 3.4)

```
<ZUGNUMMER class="forwardMove" fieldIndex="FID" symbol="SYMBOL"/>
```

3.3. Rückwärtszug

ZUGNUMMER firstMove, secondMove oder thirdMove

FID der Index des Startfeldes

```
<ZUGNUMMER class="backwardMove" fieldIndex="FID"/>
```

3.4. Symbole

Die Strings der Symbolnamen sind:

1. SKULL
2. HAT
3. DAGGER
4. BOTTLE
5. KEY
6. PISTOL

Die Erkennung der Strings ist case-sensitive, alle Buchstaben müssen groß sein.

3.5. Debughints

Zügen können Debug-Informationen beigefügt werden:

S Informationen zum Zug als String

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

Damit sieht beispielsweise ein MoveContainer mit zwei Teilzügen so aus:

```
<room roomId="RID">
  <data class="moveContainer">
    <firstMove class="backwardMove" fieldIndex="FID">
      <hint content="2 Karten ziehen...."/>
      <hint content="Noch ein Hint"/>
    </firstMove>
    <secondMove class="forwardMove" fieldIndex="FID" symbol="SYMBOL">
     <hint content="Piraten ins Zielfeld bewegen..."/>
```

```
    </secondMove>
  </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="welcome" 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

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

red, blue wie in 6.3 definiert

card wie in 6.4 definiert. Die Karte, welche als Nächste gezogen werden kann, kommt zuerst.

lastMove Letzter getätigter Zug (nicht in der ersten Runde), wie in 6.8 definiert

board Das Spielbrett, wie in 6.5 definiert

condition Spielergebnis, wie in 11 definiert; nur zum Spielende

```
<state class="state" turn="Z" currentPlayer="C">
  red
  blue
  <openCards>
    {card}
  </openCards>
  [lastMove]
  [board]
  [condition]
</state>
```

6.3. Spieler

C Farbe (red/blue)

N Anzeigename

COL Farbe “RED”/”BLUE”

P Punktekonto

card Spielkarte, wie in 6.4 definiert

```
<C displayName="N" color="COL" points="P">
  <cards>
    {card}
  </cards>
</C>
```

6.4. Karte

S Symbol der Karte wie in 3.4 definiert.

```
<card symbol="S"/>
```

6.5. Spielbrett

FIELD Ein Spielfeld wie in 6.6 definiert.

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

6.6. Spielfeld

6.6.1. Start- und Zielfeld

T Typ des Feldes START/FINISH

PIRATE Pirat, wie in 6.7 definiert.

```
<field type="T">
  <pirates>
   {PIRATE}
  </pirates>
</field>
```

6.6.2. Symbolfeld

S Symbol, welches das Feld trägt, siehe 3.4

PIRATE Pirat, wie in 6.7 definiert.

```
<field type="SYMBOL" symbol="S">
  <pirates>
   {PIRATE}
  </pirates>
</field>
```

6.7. Pirat

O Besitzer RED/BLUE

```
<pirate owner="O"/>
```

6.8. Letzter Zug

Der letzte Zug ist ein MoveContainer (siehe hierzu 3.1).

ZUG Informationen über den Zug. (siehe 3.2 und 3.3)

```
<lastMove>
  <firstMove ZUG>
  <secondMove ZUG>
  <thirdMove ZUG>
</lastMove>
```

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

P Punktekonto

card wie in 6.4 definiert

```
<room roomId="RID">
<data class="paused">
<nextPlayer class="player" displayName="N" color="C" points="P">
<cards>
  {card}
</cards>
</nextPlayer>
</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_2013_cartagena"/>
```

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="welcome" 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" currentPlayer="RED">
      <red displayName="Spieler 1" color="RED" points="0">
        <cards>
          <card symbol="BOTTLE"/>
          <card symbol="DAGGER"/>
          <card symbol="DAGGER"/>
          <card symbol="HAT"/>
```

```
    <card symbol="KEY"/>
    <card symbol="PISTOL"/>
  </cards>
</red>
<blue displayName="Spieler 2" color="BLUE" points="0">
  <cards>
    <card symbol="PISTOL"/>
    <card symbol="SKULL"/>
    <card symbol="KEY"/>
    <card symbol="KEY"/>
    <card symbol="KEY"/>
    <card symbol="DAGGER"/>
  </cards>
</blue>
<openCards>
  <card symbol="BOTTLE"/>
  <card symbol="BOTTLE"/>
  <card symbol="HAT"/>
  <card symbol="KEY"/>
  <card symbol="PISTOL"/>
  <card symbol="PISTOL"/>
  <card symbol="PISTOL"/>
  <card symbol="HAT"/>
  <card symbol="PISTOL"/>
  <card symbol="BOTTLE"/>
  <card symbol="KEY"/>
  <card symbol="BOTTLE"/>
</openCards>
<board>
  <fields>
    <field type="START">
      <pirates>
        <pirate owner="RED"/>
        <pirate owner="BLUE"/>
        <pirate owner="RED"/>
        <pirate owner="BLUE"/>
        <pirate owner="RED"/>
        <pirate owner="BLUE"/>
        <pirate owner="RED"/>
        <pirate owner="BLUE"/>
        <pirate owner="RED"/>
        <pirate owner="BLUE"/>
        <pirate owner="RED"/>
        <pirate owner="BLUE"/>
```

```
  </pirates>
</field>
<field type="SYMBOL" symbol="SKULL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="DAGGER">
  <pirates/>
</field>
<field type="SYMBOL" symbol="KEY">
  <pirates/>
</field>
<field type="SYMBOL" symbol="HAT">
  <pirates/>
</field>
<field type="SYMBOL" symbol="BOTTLE">
  <pirates/>
</field>
<field type="SYMBOL" symbol="PISTOL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="HAT">
  <pirates/>
</field>
<field type="SYMBOL" symbol="BOTTLE">
  <pirates/>
</field>
<field type="SYMBOL" symbol="DAGGER">
  <pirates/>
</field>
<field type="SYMBOL" symbol="SKULL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="PISTOL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="KEY">
  <pirates/>
</field>
<field type="SYMBOL" symbol="KEY">
  <pirates/>
</field>
<field type="SYMBOL" symbol="HAT">
  <pirates/>
</field>
```

```
<field type="SYMBOL" symbol="PISTOL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="DAGGER">
  <pirates/>
</field>
<field type="SYMBOL" symbol="BOTTLE">
  <pirates/>
</field>
<field type="SYMBOL" symbol="SKULL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="SKULL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="HAT">
  <pirates/>
</field>
<field type="SYMBOL" symbol="KEY">
  <pirates/>
</field>
<field type="SYMBOL" symbol="DAGGER">
  <pirates/>
</field>
<field type="SYMBOL" symbol="BOTTLE">
  <pirates/>
</field>
<field type="SYMBOL" symbol="PISTOL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="BOTTLE">
  <pirates/>
</field>
<field type="SYMBOL" symbol="HAT">
  <pirates/>
</field>
<field type="SYMBOL" symbol="PISTOL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="SKULL">
  <pirates/>
</field>
<field type="SYMBOL" symbol="DAGGER">
  <pirates/>
```

```
          </field>
          <field type="SYMBOL" symbol="KEY">
            <pirates/>
          </field>
          <field type="FINISH">
            <pirates/>
          </field>
        </fields>
      </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="moveContainer">
      <firstMove class="forwardMove" fieldIndex="0" symbol="DAGGER"/>
      <secondMove class="forwardMove" fieldIndex="2" symbol="KEY"/>
      <thirdMove class="forwardMove" fieldIndex="0" symbol="HAT"/>
  </data>
</room>
```

12. so geht es weiter...

13. die letzte Nachricht enthält ein

```
</protocol>
```

    Daraufhin wird die Verbindung geschlossen