CluedoCommunication

Version 0.1
6. April 2001

Praktikum Programmiermethodik SS 2001
Lehrstuhl für Praktische Informatik IV, Universität Mannheim

Inhaltsverzeichnis

0 Modifikationen

Im Verlauf des Praktikums werden Veränderungen am Kommunikationsprotokoll und somit an diesem Dokument erforderlich werden. Die Modifikationen werden in diesem Abschnitt zusammengefaßt.

Version 0.1 Initialversion

1 Einführung

Das vorliegende Dokument spezifiziert die Kommunikation zwischen CluedoServer (im folgenden als Server bezeichnet) und CluedoClient bzw. CluedoRobot (im folgenden als Client bezeichnet). Kommunikation ist immer dann erforderlich, wenn Client und Server interagieren, z.B. wenn ein neuer Spieler sich bei einem Server anmeldet, ein Server ein Cluedo-Spiel startet etc.

Am Beispiel der Anmeldung eines neuen Spielers bei einem Server illustrieren wir einen Kommunikationsablauf:

1. Der Spieler startet seinen Client und wählt einen Server aus, bei dem er sich anmelden will. Er gibt seinen Namen und etwaige weitere Informationen an.
2. Der Client nimmt Kontakt mit dem Server auf und teilt ihm den *Anmeldungswunsch* des Spielers mit.
3. Der Server wertet den Anmeldungswunsch aus und vermerkt den Spieler in seiner Spielerliste.
4. Der Server erstellt nun seinerseits eine *Anmeldebestätigung* und verschickt diese an den Client.
5. Der Client wertet die Bestätigung aus und kann – da er nun beim Server angemeldet ist – weitere Aktionen ausführen.

Die Kommunikation zwischen Client und Server erfolgt also durch den Austausch bestimmter Informationen – in obigen Beispiel werden die Informationen „Anmeldungswunsch“ und „Anmeldebestätigung“ verwendet.

In dem von uns gewählten Ansatz werden diese Informationen in Datenpaketen oder so genannten Protocol Data Units (PDUs) versendet. Darin wird zusätzlich zur eigentlichen Information noch die serverweit eindeutige Identifikationsnummer des Senders vermerkt.

Entsprechend dieses Ansatzes stellen wir unser obiges Beispiel genauer dar:

1. (wie oben)
2. Der Client erzeugt eine PDU, d.h. ein Objekt der Klasse `CluedoPDU`. Er trägt `UNKNOWN` als Sender-ID in die PDU ein, da er vom Server noch keine ID erhalten hat. Als Information wird ein Objekt der Klasse `PlayerAddRequestInfo`, das den Namen des Spielers sowie sein Paßwort enthält, in der PDU vermerkt. Die PDU wird an den Server gesendet.
3. Der Server empfängt die PDU und ermittelt, daß ein Objekt der Klasse `PlayerAddRequestInfo` in der PDU enthalten ist. Er vermerkt den neuen Spieler in seiner Spielerliste und erzeugt für ihn eine serverweit eindeutige ID.
4. Der Server erzeugt eine PDU und trägt als Sender-ID 0 ein.[1] Zusätzlich wird ein Objekt der Klasse `PlayerAddConfirmInfo`, das die dem Spieler zugewiesene Identifikationsnummer enthält, in die PDU eingetragen. Diese PDU wird an den Client geschickt.
5. (wie oben)

Bevor wir die Kommunikationsabläufe im Rahmen von Cluedo sowie die dabei beteiligten Kommunikationsklassen (`CluedoPDU`, `PlayerAddRequestInfo` etc.) besprechen, gehen wir zunächst kurz auf die Implementierung von Client-Server-Kommunikation in Java ein. Danach erweitern wir unser Kommunikationskonzept um eine prioritätenbasierte Komponente.

[1] Wir verwenden für einen CluedoServer als Sender-ID immer die 0.

2 Client-Server-Kommunikation mit Java

Der Austausch von Informationen bzw. PDUs erfolgt in unserem Ansatz über das Transportprotokoll TCP/IP. Java stellt im Paket `java.net` die Klassen `Socket` und `ServerSocket` zur Verfügung über die eine zuverlässige und geordnete Datenübertragung möglich ist.

Mittels sogenannter Objektströme (`ObjectInputStream`, `ObjectOutputStream`) aus dem Paket `java.io` ist es möglich, Objekte (in unserem Fall Objekte der Klasse `CluedoPDU`) zwischen Client und Server auszutauschen.

Im folgenden geben wir ein rudimentäres Server- bzw. Client-Programm an, mit dem eine einfache Kommunikation möglich ist. Im Wesentlichen implementieren wir den Kommunikationsablauf aus obigem Beispiel „Anmeldung eines Spielers“.

Das Server-Programm erzeugt einen Socket über den Verbindungswünsche von Clients abgehandelt werden können.[2] Danach wartet der Server auf die Anfrage eines Clients, die mit einem `PlayerAddConfirmInfo`-Objekt beantwortet wird.

```
import java.net.*;
import java.io.*;

import Cluedo.*;       // die Packages Cluedo (enthaelt CluedoPDU) und
import Cluedo.Info.*;  // Cluedo.Info (enthaelt die Informationsklassen)
                       // sind auf den Praktikumswebseiten erhaeltlich

public class Server
{
    // simpler Server, Aufruf mit "java Server [port]"
    // z.B. java Server 10000

    public static void main(String[] args)
    {
        /* Socket fuer eingehende Verbindungen erzeugen */
        ServerSocket serverSocket = null;
        try {
            int port = Integer.parseInt(args[0]);
            serverSocket = new ServerSocket(port); // Socket fuer Port "port" erzeugen
            System.out.println("server listening on port " + port);
        } catch (IOException e) {
            System.err.println("IO exception while creating server socket");
        }

        /* Auf Verbindungswunsch warten */
        Socket socket = null;
        try {
            socket = serverSocket.accept(); // Socket, ueber den die Verbindung zum
                                            // Client abgewickelt wird
        } catch (IOException e) {
            System.err.println("IO exception while waiting for connections");
        }

        /* Eingabe-/Ausgabestroeme an den Socket anbinden */
        ObjectOutputStream oos = null; // fuer ausgehende Objekte
        ObjectInputStream ois = null;  // fuer eingehende Objekte

        try {
            oos = new ObjectOutputStream(socket.getOutputStream());
            ois = new ObjectInputStream(socket.getInputStream());
        } catch (IOException e) {
```

[2] In unserer Beispielimplementierung kann nur ein Verbindungswunsch bearbeitet werden.

```
            System.err.println("IO exception while creating streams");
        }

        /* Warten auf eine Anfrage vom Client */
        try {
            CluedoPDU cpdu = (CluedoPDU) ois.readObject(); // PDU auslesen

            if (cpdu.getCluedoInfo() instanceof PlayerAddRequestInfo) {
                // es ist ein Anmeldungswunsch
                System.out.println("PlayerAddRequestInfo received.");
            } else {
                // es ist irgendetwas anderes
                System.err.println("Unknown info object received");
            }
        } catch (IOException e) {
            System.err.println("IO exception while reading");
            } catch (ClassNotFoundException e) {
            System.err.println("Class not found while reading from input stream");
        }

        /* Anmeldebestaetigung senden */
        try {
            // Informationsobjekt erzeugen (ID = 1)
            PlayerAddConfirmInfo info = new PlayerAddConfirmInfo(1);
            // PDU erzeugen ...
            CluedoPDU cpdu = new CluedoPDU(0, info);
            // ... und abschicken
            oos.writeObject(cpdu);
        } catch (IOException e) {
            System.err.println("IO exception while writing");
        }
    }
}
```

Der Client baut eine Verbindung zum Server auf und sendet eine PDU mit einem `PlayerAddRequestInfo`-Objekt. Daraufhin wartet er auf den Eingang einer Antwort.

```
import java.net.*;
import java.io.*;

import Cluedo.*;       // die Packages Cluedo (enthaelt CluedoPDU) und
import Cluedo.Info.*;  // Cluedo.Info (enthaelt die Informationsklassen)
                       // sind auf den Praktikumswebseiten erhaeltlich

public class Client
{
    // simpler Client, Aufruf mit "java Client [servername] [serverport]"
    // z.B. java Client localhost 10000

    public static void main(String[] args)
    {
        /* Socket fuer die Kommunikation mit dem Server */
        Socket socket = null;
        try {
            // Name des Servers
            String hostname = args[0];
            // Port ueber den der Dienst angeboten wird
```

```
            int port        = Integer.parseInt(args[1]);
            socket = new Socket(hostname, port);
        } catch (IOException e) {
            System.err.println("IO exception while creating server socket");
        }

        /* Eingabe-/Ausgabestroeme an den Socket anbinden */
        ObjectOutputStream oos = null; // fuer ausgehende Objekte
        ObjectInputStream ois = null;  // fuer eingehende Objekte

        try {
            oos = new ObjectOutputStream(socket.getOutputStream());
            ois = new ObjectInputStream(socket.getInputStream());
        } catch (IOException e) {
            System.err.println("IO exception while creating streams");
        }

        /* Anmeldungswunsch an den Server senden */
        try {
            // Informationsobjekt erzeugen
            PlayerAddRequestInfo info = new PlayerAddRequestInfo("name", "passed");
            // PDU erzeugen ...
            CluedoPDU cpdu = new CluedoPDU(CluedoInfo.UNKNOWN, info);
            // ... und abschicken
            oos.writeObject(cpdu);
        } catch (IOException e) {
            System.err.println("IO exception while writing");
        }

        /* Warten auf Antwort des Servers */
        try {
            // warten auf Objekt (blockierendes read)
            CluedoPDU cpdu = (CluedoPDU) ois.readObject();
            if (cpdu.getCluedoInfo() instanceof PlayerAddConfirmInfo) {
                // es ist eine Anmeldungsbestaetigung
                System.out.println("PlayerAddConfirmInfo received.");
            } else {
                // es ist irgendetwas anderes
                System.err.println("Unknown info object received");
            }
        } catch (IOException e) {
            System.err.println("IO exception while reading");
            } catch (ClassNotFoundException e) {
            System.err.println("Class not found while reading from input stream");
        }
    }
}
```

3 Prioritätenbasierte Kommunikation

Bei bestimmten Informationen ist es notwendig, daß sie den Empfänger erreichen. Dieses gilt z.B. für alle Informationen, die innerhalb eines Spiels entstehen (Züge eines Spielers, Verdächtigungen, Anklagen etc.). Bei anderen Informationen hingegen ist es nicht unbedingt erforderlich, daß der Client sie erhält. Befindet sich z.B. ein Client in einer Spielrunde, so ist es für ihn nur bedingt interessant, wenn auf dem Server eine weitere Session eröffnet wird oder sich der Status einer anderen Session ändert.

Aus diesem Grund setzen wir auf der Seite des Servers ein Konzept ein, das auf Prioritäten basiert. Wir unterscheiden zwischen zwei Priotitäten 0 und 1. Informationen der Priorität 0 müssen einen Client unbedingt erreichen, während Informationen mit Priorität 1 nur dann versendet werden, wenn die Bandbreite der Verbindung es zuläßt.

Der Server unterhält für jeden Client eine Warteschlange in die die ausgehenden PDUs eingereiht werden. Erreicht die Warteschlange nun eine gewisse Länge[3] so werden PDUs der Priorität 1 verworfen.

Entsprechend erweitern wir das oben angeführte Beispiel:

1. (wie oben)
2. (wie oben)
3. (wie oben)
4. Der Server erzeugt eine PDU und trägt als Sender-ID 0 ein. Zusätzlich wird ein Objekt der Klasse `PlayerAddConfirmInfo`, das die dem Spieler zugewiesene Identifikationsnummer enthält, in die PDU eingetragen. Diese PDU wird *mit Priorität 0* an den Client geschickt. *Zusätzlich erzeugt der Server eine weitere PDU in die ein Objekt der Klasse* `PlayerAddedInfo` *eingetragen wird. Diese PDU wird mit Priorität 1 an alle beim Server angemeldeten Spieler versendet, d.h. sie wird nur an einen Client verschickt, wenn die aktuelle Länge der Ausgabewarteschlange unterhalb einer gewissen Grenze liegt.*
5. (wie oben)

Im folgenden Abschnitt spezifizieren wir die verschiedenen Kommunikationsabläufe im Rahmen des verteilten Cluedo-Spiels. Zusätzlich zu den Interaktionen, die das Spiel erfordert, geben wir Informationsabläufe bezüglich einer Chat-Komponente an.

4 Kommunikationsabläufe

Bei der Darstellung der einzelnen Abläufe verwenden wir folgende Notation:

- C bezeichnet die Menge aller Clients, die an einem Server angemeldet sind.
- C_U bezeichnet die Menge aller Clients, die noch in keiner Session angemeldet sind.
- $C_{S(i)}$ bezeichnet die Menge aller Clients, die zu der Session gehören, der auch der Client i angehört. Wir verwenden die verkürzte Notation C_S, wenn aus dem Kontext zu ersehen ist, um welche Session es sich handelt,
- Der Server wird mit s gekennzeichnet. Für die Clients verwenden wir c_i oder c_j, wobei in der Regel $i \neq j$ gilt. Den Spielleiter einer Session bezeichnen wir mit c_d (um welche Session es sich dabei handelt, ist aus dem Kontext erkennbar).

Die Kommunikationsabläufe sind tabellarisch dargestellt. Am Anfang einer Zeile wird der Sender angegeben. Danach folgen der bzw. die Empfänger. Im Anschluß befindet sich – sofern es sich um eine Information des Servers handelt – die Priorität der Information. Am Ende einer Zeile wird die Informationsklasse angegeben (Die Informationsklassen werden in Abschnitt 5 beschrieben).

Zum Beispiel bedeutet die Zeile

$c_i \rightarrow s$ - `PlayerAddRequestInfo`,

daß der Client c_i eine PDU an den Server s sendet. Die PDU enthält ein Informationsobjekt der Klasse `PlayerAddRequestInfo`.

Im folgenden Beispiel

[3] Dieser Parameter kann beim Server eingestellt werden.

$s \rightarrow c_i$	0	`PlayerAddedInfo`
$s \rightarrow C \backslash \{c_i \cup C_U\}$	0	`PlayerAddedInfo`
$s \rightarrow C \backslash C_{S(c_i)}$	1	`PlayerAddedInfo`

sendet der Server eine `PlayerAddedInfo` (1) an den Client c_i, (2) an alle angemeldeten Clients (ohne den Client c_i und die Clients, die keiner Session angehören) und (3) an alle Clients, die beim Server angemeldet sind (ohne die Clients der Session, der c_i angehört).

Zusätzlich geben wir bei einem Kommunikationsablauf mögliche Fehlerfälle an. Tritt ein Fehler auf, so wird ein Objekt der Klasse `ErrorInfo`, das eine Fehler-ID enthält, versendet. Wir spezifizieren die Fehlerfälle durch die Angabe der Fehler-ID und des Empfängers (z.B. `INVALID_PLAYER_ID`(c_i)). Wenn kein Empfänger angegeben ist, so ist dieser aus dem Kontext ersichtlich.

Die Fehlermeldung `INVALID_SENDER_ID` tritt bei allen Client-Anfragen auf, wenn die ID des anfragenden Clients nicht in der Spielerliste des Servers gefunden werden kann (eine Ausnahme ist natürlich `PlayerAddRequestInfo`). Diese Fehlermeldung ist daher nicht explizit bei den einzelnen Abläufen aufgeführt.

4.1 Anmeldung und Abmeldung eines Spielers

4.1.1 Anmeldung eines neuen Spielers

Beschreibung: Ein Spieler sendet einen Anmeldungswunsch an den Server. Der Server bestätigt dem Spieler die Anmeldung oder sendet ihm eine Fehlermeldung (z.B. wenn die Kapazitäten erschöpft sind). Desweiteren teilt der Server allen bei ihm angemeldeten Spielern mit, daß ein neuer Spieler hinzugekommen ist.

$c_i \rightarrow s$	-	`PlayerAddRequestInfo`
$s \rightarrow c_i$	0	`PlayerAddConfirmInfo`
$s \rightarrow C$	1	`PlayerAddedInfo`

Vorbedingungen: keine

Fehlerfälle: `CONNECTION_REFUSED` – Der Server hat einen Anmeldungswunsch abgelehnt. Dieses kann durch ein ungültiges Paßwort oder eine erreichte Obergrenze an Benutzern ausgelöst werden.

`PLAYERNAME_EXISTS` – Es gibt bereits einen Spieler mit dem angegebenen Namen.

4.1.2 Wiederherstellung einer Verbindung

Beschreibung: Die Verbindung eines Spielers zum Server ist mitten im Spiel abgebrochen. Der Spieler stellt die Verbindung unter Angabe der bereits zugewiesenen ID wieder her.

$c_i \rightarrow s$	-	`PlayerReAddRequestInfo`
$s \rightarrow c_i$	0	`PlayerAddConfirmInfo`
$s \rightarrow C_{S(c_i)}$	0	`PlayerAddedInfo`
$s \rightarrow C \backslash C_{S(c_i)}$	1	`PlayerAddedInfo`

Vorbedingungen: Der Spieler muß vor Abbruch der Verbindung beim Server angemeldet gewesen sein.

Fehlerfälle: keine

4.1.3 Abmeldung eines Spielers

Beschreibung: Ein Spieler meldet sich beim Server ab. Der Server teilt allen Spielern mit, daß ein Spieler sich abgemeldet hat.

$c_i \rightarrow s$	-	`PlayerRemoveRequestInfo`
$s \rightarrow C_{S(c_i)}$	0	`PlayerRemovedInfo`
$s \rightarrow C \backslash C_{S(c_i)}$	1	`PlayerRemovedInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein.

Fehlerfälle: keine

4.1.4 Abbruch einer Verbindung

Beschreibung: Der Server stellt fest, daß die Verbindung zu einem Spieler abgebrochen ist. Er teilt dieses allen Spielern mit.

$s \rightarrow C_{S(c_i)}$	0	`PlayerLostInfo`
$s \rightarrow C \backslash C_{S(c_i)}$	1	`PlayerLostInfo`

Vorbedingungen: keine

Fehlerfälle: keine

4.2 Sessionverwaltung

4.2.1 Anmeldung einer neuen Session

Beschreibung: Ein Spieler meldet eine neue Session beim Server an. Der Spieler ist automatisch der Spielleiter dieser Session. Der Server teilt allen angemeldeten Clients mit, daß eine Session hinzugekommen ist.

$c_i \rightarrow s$	-	`SessionAddRequestInfo`
$s \rightarrow c_i \cup C_U$	0	`SessionAddedInfo`
$s \rightarrow C \backslash \{c_i \cup C_U\}$	1	`SessionAddedInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Desweiteren darf er noch nicht einer anderen Session beigetreten sein bzw. eine andere Session angemeldet haben.

Fehlerfälle: `SESSIONNAME_EXISTS` – Es gibt bereits eine Session mit dem angegebenen Namen.
`ALREADY_IN_SESSION` – Der Spieler nimmt bereits an einer anderen Session teil.

4.2.2 Abmeldung einer Session

Beschreibung: Der Spielleiter meldet eine Session ab. Der Server teilt dieses allen angemeldeten Spieler mit.

$c_i \rightarrow s$	-	`SessionRemoveRequestInfo`
$s \rightarrow C_{S(c_i)} \cup C_U$	0	`SessionRemovedInfo`
$s \rightarrow C \backslash \{C_{S(c_i)} \cup C_U\}$	1	`SessionRemovedInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Die abzumeldende Session muß existieren. Der Spieler muß Spielleiter dieser Session sein.

Fehlerfälle: `INVALID_SESSION_ID` – Der Server hat die Session nicht in der Sessionliste gefunden.
`NOT_DIRECTOR` – Der Spieler ist nicht der Spielleiter der Session und kann sie daher nicht löschen.

4.2.3 Schliessen einer Session

Beschreibung: Der Spielleiter schließt eine Session, d.h. es werden keine weiteren Mitspieler aufgenommen. Der Server teilt dieses allen angemeldeten Spieler mit. (Anmerkung: Im Anschluß startet der Spieler das Spiel innerhalb dieser Session, siehe Abschnitt 4.4.)

c_d	$\rightarrow$	s	-	`SessionCloseRequestInfo`
s	$\rightarrow$	$C_{S(c_d)} \cup C_U$	0	`SessionClosedInfo`
s	$\rightarrow$	$C \backslash \{C_{S(c_d)} \cup C_U\}$	1	`SessionClosedInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Die Session muß existieren. Der Spieler muß Spielleiter dieser Session sein.

Fehlerfälle: `INVALID_SESSION_ID` – Der Server hat die Session nicht in der Sessionliste gefunden.
`NOT_DIRECTOR` – Der Spieler ist nicht der Spielleiter der Session und kann sie daher nicht schließen.

4.2.4 Beitritt zu einer existierenden Session

Beschreibung: Der Spieler möchte an einer bestimmten Session teilnehmen. Der Server leitet die Anfrage des Spielers an den Spielleiter der Session weiter. Dieser teilt dem Server mit, ob er den Spieler akzeptiert oder nicht. Der Server gibt die Rückmeldung des Spielleiters an den anfragenden Spieler und die Spieler der Session weiter.

c_i	$\rightarrow$	s	-	`SessionJoinRequestInfo`
s	$\rightarrow$	c_d	0	`SessionJoinInfo`
c_d	$\rightarrow$	s	-	`PlayerAcceptInfo`
s	$\rightarrow$	$c_i \cup C_{S(c_d)}$	0	`PlayerAcceptInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Er darf an keiner anderen Session teilnehmen. Die Session muß existieren und darf noch nicht geschlossen sein.

Fehlerfälle: `ALREADY_IN_SESSION` – Der Spieler nimmt bereits an einer anderen Session teil.
`INVALID_SESSION_ID` – Der Server hat die Session nicht in der Sessionliste gefunden.
`SESSION_ALREADY_CLOSED` – Die Session ist bereits geschlossen.

4.3 Informationsabfragen

4.3.1 Anforderung von Informationen zu einem speziellen Spieler

Beschreibung: Ein Spieler fordert eine Beschreibung eines anderen Spielers an. Der Server liefert dem Spieler die Beschreibung.

c_i	$\rightarrow$	s	-	`PlayerRequestInfo`
s	$\rightarrow$	c_i	0	`PlayerInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Der Spieler, nach dem gefragt wird, muß existieren.

Fehlerfälle: `INVALID_PLAYER_ID` – Der nachgefragte Spieler existiert nicht.

4.3.2 Anforderung von Informationen zu allen Spieler

Beschreibung: Ein Spieler fordert Beschreibungen zu allen am Server angemeldeten Spielern an. Der Server liefert dem Spieler die Beschreibungen

c_i	$\rightarrow$	s	-	`PlayerListRequestInfo`
s	$\rightarrow$	c_i	0	`PlayerListInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein.

Fehlerfälle: keine

4.3.3 Anforderung von Informationen zu einer speziellen Session

Beschreibung: Ein Spieler fordert eine Beschreibung einer Session an. Der Server liefert dem Spieler die Beschreibung.

$c_i \rightarrow s$ - `SessionRequestInfo`
$s \rightarrow c_i$ 0 `SessionInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Die Session, nach der gefragt wird, muß existieren.

Fehlerfälle: `INVALID_SESSION_ID` – Die nachgefragte Session existiert nicht.

4.3.4 Anforderung von Informationen zu allen Sessions

Beschreibung: Ein Spieler fordert eine Beschreibung zu allen beim Server angemeldeten Sessions an. Der Server liefert dem Spieler die Beschreibungen.

$c_i \rightarrow s$ - `SessionListRequestInfo`
$s \rightarrow c_i$ 0 `SessionListInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein.

Fehlerfälle: keine

4.4 Spielvorbereitung

4.4.1 Verteilen der Spielkarten

Beschreibung: Nachdem eine Session durch den Spielleiter geschlossen wurde, verteilt der Server die Spielkarten an die teilnehmenden Spieler, d.h. jeder Spieler erhält eine bestimmte Anzahl an Karten.

$s \rightarrow C_S$ 0 `CardListInfo`

Vorbedingungen: Die Session muß geschlossen sein.

Fehlerfälle: keine

4.5 Spielzug

4.5.1 Aufforderung zum Zug

Beschreibung: Der Server teilt den Spielern einer Session mit, wer am Zug ist.

$s \rightarrow C_S$ 0 `PlayerDrawInfo`

Vorbedingungen: Die Session muß geschlossen und die Spielkarten verteilt sein.

Fehlerfälle: keine

4.5.2 Setzen der Spielfigur

Beschreibung: Ein Spieler ist am Zug. Er nimmt entweder einen Geheimgang (in diesem Fall entfallen die ersten beiden Zeilen des Kommunikationsablaufs) oder er fordert beim Server ein Würfelergebnis an. Der Spieler ermittelt seine neue Position und gibt diese an den Server weiter. Der Server gibt die Veränderung an alle Spieler der Session weiter.

c_i	$\rightarrow$	s	-	`DiceRequestInfo`
s	$\rightarrow$	$C_{S(c_i)}$	0	`DiceInfo`
c_i	$\rightarrow$	s	-	`PlayerMoveRequestInfo`
s	$\rightarrow$	$C_{S(c_i)}$	0	`PlayerMovedInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Er muß desweiteren am Zug sein.

Fehlerfälle: `NOT_ACTIVE_PLAYER` – Der Spieler ist nicht am Zug.
`INVALID_MOVE` – Der Spieler hat eine ungültige neue Position gesendet.

4.5.3 Äußern eines Verdachts

Beschreibung: Ein Spieler äußert einen Verdacht. Der Server teilt diesen Verdacht allen Teilnehmern der Session mit wobei zusätzlich vermerkt wird, wer diesen Verdacht widerlegen kann (möglicherweise kann niemand den Verdacht widerlegen). Der betreffende Mitspieler widerlegt ggf. den Verdacht. Der Server teilt dem Spieler, der am Zug ist, mit, wer durch welche Karte den Verdacht widerlegt hat.

c_i	$\rightarrow$	s	-	`SuspicionRequestInfo`
s	$\rightarrow$	$C_{S(c_i)}$	0	`SuspicionInfo`
c_j	$\rightarrow$	s	-	`SuspicionCardInfo`
s	$\rightarrow$	c_i	0	`SuspicionCardInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Er muß desweiteren am Zug sein.

Fehlerfälle: `NOT_ACTIVE_PLAYER`(c_i) – Der Spieler ist nicht am Zug.
`INVALID_CARD`(c_j) – Der Spieler, der den Verdacht widerlegt, hat eine ungültige Karte gezeigt.

4.5.4 Erheben einer Anklage

Beschreibung: Ein Spieler erhebt Anklage. Der Server teilt diese Anklage allen Teilnehmern der Session mit. Der Server übermittelt dem Spieler die tatsächlichen Tatbestände. Desweiteren teilt er allen Teilnehmern der Session mit, ob die Anklage zutrifft und damit das Spiel beendet ist. Ist das Spiel beendet so kann der Spielleiter eine weitere Spielrunde starten, in dem er den Status der Session von „beendet“ auf „geschlossen“ setzt (siehe Abschnitt 4.4).

c_i	$\rightarrow$	s	-	`AccusationRequestInfo`
s	$\rightarrow$	$C_{S(c_i)}$	0	`AccusationInfo`
s	$\rightarrow$	c_i	0	`CrimeFactInfo`
s	$\rightarrow$	$c_{S(c_i)}$	0	`WinnerInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Er muß desweiteren am Zug sein.

Fehlerfälle: `NOT_ACTIVE_PLAYER` – Der Spieler ist nicht am Zug.

4.5.5 Beenden eines Spielzugs

Beschreibung: Ein Spieler beendet seinen Zug. Der Server teilt dieses allen Teilnehmern der Session mit.

c_i	$\rightarrow$	s	0	`DrawFinishedInfo`
s	$\rightarrow$	$C_{S(c_i)}$	0	`DrawFinishedInfo`

Vorbedingungen: Der Spieler muß beim Server angemeldet sein. Er muß desweiteren am Zug sein.

Fehlerfälle: `NOT_ACTIVE_PLAYER` – Der Spieler ist nicht am Zug.

4.6 Chat-Komponente

Beschreibung: Ein Spieler möchte eine Textnachricht an einen anderen Spieler senden.

c_i	$\rightarrow$	s	0	`ChatRequestInfo`
s	$\rightarrow$	c_j	0	`ChatInfo`

Vorbedingungen: keine

Fehlerfälle: `INVALID_PLAYER_ID` – Der Spieler existiert nicht.

Beschreibung: Ein Spieler möchte eine Textnachricht an alle Spieler einer Session senden.

c_i	$\rightarrow$	s	0	`SessionChatRequestInfo`
s	$\rightarrow$	$C_{S(c_i)}$	0	`SessionChatInfo`

Vorbedingungen: keine

Fehlerfälle: `INVALID_SESSION_ID` – Die Session existiert nicht.

4.7 Ein Beispiel

In diesem Abschnitt geben wir zur Verdeutlichung einen möglichen Spielablauf an. Wir beschränken uns der Einfachheit halber auf zwei Spieler c_1, c_2.

1. Spieler 1 meldet sich beim Server an. Nach Erhalt seiner Anmeldebestätigung legt er eine neue Session mit dem Namen „Kluedo“ an. Er ist automatisch Spielleiter dieser Session.

c_1	$\rightarrow$	s	-	`PlayerAddRequestInfo`
s	$\rightarrow$	c_1	0	`PlayerAddConfirmInfo`
s	$\rightarrow$	C	1	`PlayerAddedInfo`
c_1	$\rightarrow$	s	-	`SessionAddRequestInfo`
s	$\rightarrow$	$c_1 \cup C_U$	0	`SessionAddedInfo`
s	$\rightarrow$	$C \backslash \{c_1 \cup C_U\}$	1	`SessionAddedInfo`

2. Spieler 2 meldet sich beim Server an. Nach Erhalt der Anmeldebestätigung fordert er eine Sessionliste beim Server an. Er möchte an der Session „Kluedo“ teilnehmen. Spieler 1 akzeptiert Spieler 2 als Mitspieler.

c_2	$\rightarrow$	s	-	`PlayerAddRequestInfo`
s	$\rightarrow$	c_2	0	`PlayerAddConfirmInfo`
s	$\rightarrow$	C	1	`PlayerAddedInfo`
c_2	$\rightarrow$	s	-	`SessionListRequestInfo`
s	$\rightarrow$	c_2	0	`SessionListInfo`
c_2	$\rightarrow$	s	-	`SessionJoinRequestInfo`
s	$\rightarrow$	c_1	0	`SessionJoinInfo`
c_1	$\rightarrow$	s	-	`PlayerAcceptInfo`
s	$\rightarrow$	$c_2 \cup C_S$	0	`PlayerAcceptInfo`

3. Spieler 1 schließt die Session. Die Karten werden verteilt und das Spiel startet mit Spieler 1.

c_1	$\rightarrow$	s	-	`SessionCloseRequestInfo`
s	$\rightarrow$	$C_S \cup C_U$	0	`SessionClosedInfo`
s	$\rightarrow$	$C \backslash \{C_S \cup C_U\}$	1	`SessionClosedInfo`
s	$\rightarrow$	c_1	0	`CardListInfo`
s	$\rightarrow$	c_2	0	`CardListInfo`
s	$\rightarrow$	C_S	0	`PlayerDrawInfo`

4. Spieler 1 würfelt und setzt seine Spielfigur. Er erreicht keinen Raum und gibt daher an den nächsten Spieler weiter.

c_1	$\rightarrow$	s	-	`DiceRequestInfo`
s	$\rightarrow$	C_S	0	`DiceInfo`
c_1	$\rightarrow$	s	-	`PlayerMoveRequestInfo`
s	$\rightarrow$	C_S	0	`PlayerMovedInfo`
c_1	$\rightarrow$	s	-	`DrawFinishedInfo`
s	$\rightarrow$	C_S	0	`PlayerDrawInfo`

5. Spieler 2 würfelt und setzt seine Spielfigur. Er erreicht einen Raum und spricht einen Verdacht aus. Spieler 1 kann den Verdacht widerlegen. Danach erhebt der Spieler 2 Anklage.

c_2	$\rightarrow$	s	-	`DiceRequestInfo`
s	$\rightarrow$	C_S	0	`DiceInfo`
c_2	$\rightarrow$	s	-	`PlayerMoveRequestInfo`
s	$\rightarrow$	C_S	0	`PlayerMovedInfo`
c_2	$\rightarrow$	s	-	`SuspicionRequestInfo`
s	$\rightarrow$	C_S	0	`SuspicionInfo`
c_1	$\rightarrow$	s	-	`SuspicionCardInfo`
s	$\rightarrow$	c_2	0	`SuspicionCardInfo`
c_1	$\rightarrow$	s	-	`AccusationRequestInfo`
s	$\rightarrow$	C_S	0	`AccusationInfo`
s	$\rightarrow$	c_2	0	`CrimeFactInfo`
s	$\rightarrow$	C_S	0	`WinnerInfo`

6. Es stellt sich heraus, das die Anklage falsch ist. Spieler 2 meldet sich enttäuscht ab. Spieler 1 löscht die Session und meldet sich ebenfalls ab.

c_2	$\rightarrow$	s	-	`PlayerRemoveRequestInfo`
s	$\rightarrow$	C_S	0	`PlayerRemovedInfo`
s	$\rightarrow$	$C \backslash C_S$	1	`PlayerRemovedInfo`
c_1	$\rightarrow$	s	-	`SessionRemoveRequestInfo`
s	$\rightarrow$	$C_S \cup C_U$	0	`SessionRemovedInfo`
s	$\rightarrow$	$C \backslash \{C_S \cup C_U\}$	1	`SessionRemovedInfo`
c_1	$\rightarrow$	s	-	`PlayerRemoveRequestInfo`
s	$\rightarrow$	C_S	0	`PlayerRemovedInfo`
s	$\rightarrow$	$C \backslash C_S$	1	`PlayerRemovedInfo`

5 Kommunikationsklassen

Bei der Kommunikation zwischen Client und Server werden Objekte der Klasse `CluedoPDU` versendet. Diese Klasse enthält die Datenelemente:

`version`	Versionsnummer des Kommunikationsprotokolls
`senderId`	ID des sendenden Spielers
`cluedoInfo`	Informationsobjekt

Von besonderer Bedeutung sind dabei die Informationsobjekte, die verschiedenen Klassen, die alle von `CluedoInfo` abgeleitet sind, angehören können. Im folgenden stellen wir die verwendeten Informationsklassen mit ihren Datenelementen zusammen. Die Kommunikationsklassen sind auf den Praktikumswebseiten zum Download erhältlich.

An den Klassen dürfen keine Veränderungen vorgenommen werden, da sonst die Kommunikation mit Modulen anderer Gruppen nicht mehr funktioniert. Änderungs- und Verbesserungsvorschläge bitte über die Praktikumsmailingliste bekanntgeben. Wir werden dann eine überarbeitete, zentrale Version der Kommunikationsklassen bereitstellen.

Mit dem folgendem Codefragment läßt sich herausfinden welcher Klasse ein Informationsobjekt angehört (wir gehen davon aus, daß ein Objekt `cpdu` der Klasse `CluedoPDU` verfügbar ist):

```
CluedoInfo ci = cpdu.getCluedoInfo();  // Informationsobjekt auslesen

if (ci instanceof CardInfo) {
   // ci ist ein CardInfo-Objekt
} else if (ci instanceof PlayerInfo) {
   // ci ist ein PlayerInfo-Objekt
}
...
```

5.1 Basisklasse `CluedoInfo`

Die Basisklasse aller Informationsklassen enthält eine Reihe von Konstantendefinitionen.

5.2 `AccusationInfo`

`accuserId`	ID des Spielers, der anklagt
siehe `WhoWhereHowInfo`	

5.3 `AccusationRequestInfo`

siehe `WhoWhereHowInfo`

5.4 `CardInfo`

`ownerId`	ID des Spielers, der die Karte besitzt
`type`	Kartentyp (Person, Tatwerkzeug, Raum)
`item`	abhängig vom Kartentyp, die ID der Person/des Tatwerkzeugs/des Raums

5.5 `CardListInfo`

`cardList`	Liste der ausgeteilten Karten

5.6 `ChatInfo`

`senderId`	ID des sendenden Spielers
`message`	Nachricht

5.7 ChatRequestInfo

receiverId	Spieler, der die Nachricht erhalten soll
message	Nachricht

5.8 CrimeFactInfo

siehe WhoWhereHowInfo

5.9 DiceInfo

playerId	ID des Spielers, der gewürfelt hat
dice	Würfelergebnis

5.10 DiceRequestInfo

keine Datenelemente

5.11 DrawFinishedInfo

keine Datenelemente

5.12 ErrorInfo

errNo	ID des aufgetretenen Fehlers
errMsg	Fehlertext mit Zusatzinformationen

5.13 PlayerAcceptInfo

playerId	ID des Spielers, der angenommen/abgelehnt wurde
sessionId	ID der Session für die angefragt wurde
teamId	Identität (d.h. Spielfigur) des Spielers
state	Status (angenommen, abgelehnt)

5.14 PlayerAddConfirmInfo

playerId	serverweit eindeutige Identifikationsnummer des Spielers

5.15 PlayerAddedInfo

siehe PlayerInfo

5.16 PlayerAddRequestInfo

name	Name des Spielers
passwd	Passwort des Spielers

5.17 PlayerDrawInfo

drawingPlayerId	ID des Spielers, der am Zug ist

5.18 PlayerInfo

playerId	serverweit eindeutige Identifikationsnummer des Spielers
name	Name des Spielers
state	Status des Spielers (Spielleiter, Mitspieler)
sessionId	Session an der der Spieler teilnimmt
teamId	Identität (d.h. Spielfigur) des Spielers
x, y	Position des Spielers
currentDice	letztes Würfelergebnis

5.19 PlayerListInfo

playerList	Liste der beim Server angemeldeten Spieler

5.20 PlayerListRequestInfo

keine Datenelemente

5.21 PlayerLostInfo

lostPlayerId	ID des Spielers zu dem die Verbindung abgebrochen ist

5.22 PlayerMovedInfo

playerId	ID des Spielers, der gezogen hat
x, y	neue Position des Spielers

5.23 PlayerMoveRequestInfo

x, y	Position des Spielers

5.24 PlayerReAddRequestInfo

playerId	ID des Spielers, der die Verbindung wieder herstellen will

5.25 PlayerRemovedInfo

playerId	ID des gelöschten Spielers

5.26 PlayerRemoveRequestInfo

keine Datenelemente

5.27 PlayerRequestInfo

requestedPlayerId	ID des Spielers nach dem gefragt wird

5.28 SessionAddedInfo

siehe SessionInfo

5.29 SessionAddRequestInfo

sessionName	serverweit eindeutiger Name für die Session

5.30 SessionChatInfo

`senderId`	ID des sendenden Spielers
`message`	Nachricht

5.31 SessionChatRequestInfo

`sessionId`	ID der Session, an deren Spieler die Nachricht gesendet werden soll
`message`	Nachricht

5.32 SessionClosedInfo

`sessionId`	ID der geschlossenen Session

5.33 SessionCloseRequestInfo

`sessionId`	ID der zu schließenden Session

5.34 SessionInfo

`sessionId`	serverweit eindeutige ID der Session
`sessionName`	Name der Session
`directorId`	ID des Spielleiters
`teamMates`	IDs der Mitspieler
`activePlayerIndex`	aktiver Spieler (Index in das `teamMates`-Feld)
`state`	Status der Session (offen, geschlossen)

5.35 SessionJoinInfo

`sessionId`	ID der Session an der der Spieler teilnehmen möchte
`requesterId`	ID des Spielers, der an der Session teilnehmen möchte

5.36 SessionJoinRequestInfo

`sessionId`	ID der Session an der der Spieler teilnehmen möchte

5.37 SessionListInfo

`sessionList`	Liste der beim Server angemeldeten Sessions

5.38 SessionListRequestInfo

keine Datenelemente

5.39 SessionRemovedInfo

`sessionId`	ID der gelöschten Session

5.40 SessionRemoveRequestInfo

`sessionId`	ID der zu löschenden Session

5.41 SessionRequestInfo

`sessionId`	ID der Session nach der gefragt wird

5.42 SuspicionCardInfo

siehe CardInfo

5.43 SuspicionInfo

suspectorId	ID des Spielers, der verdächtigt
answererId	ID des Spielers, der den Verdacht widerlegen konnte

siehe WhoWhereHowInfo

5.44 SuspicionRequestInfo

siehe WhoWhereHowInfo

5.45 WinnerInfo

playerId	ID des Spielers, der Anklage erhoben hat
sessionId	ID der Session
state	Status (Anklage richtig, Anklage falsch)

5.46 WhoWhereHowInfo

who	welche Person
where	in welchem Raum
how	mit welchem Tatwerkzeug