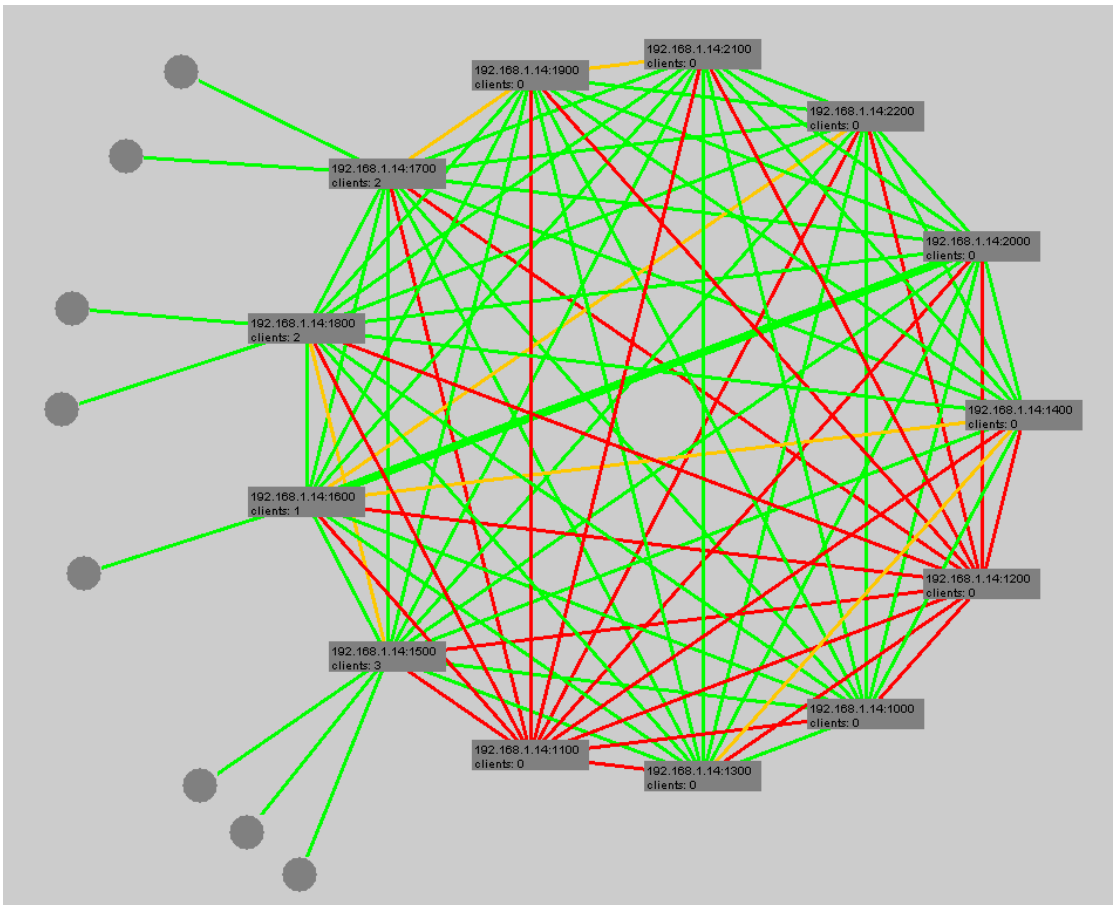
Clippee's Monitoring & Testing Tool



Semesterarbeit von Guido Hager
Sommersemester 2003
Distributed Computing Group
ETH Zürich

Betreut durch:
Prof. R. Wattenhofer
R. Arnold, K.Albrecht

Inhaltsverzeichnis

1 Einleitung

Das vorliegende Dokument beschreibt die Arbeitsschritte meiner Semesterarbeit und ist wie folgt gegliedert. In Kapitel 2 wird die eigentliche Aufgabenstellung beschrieben. Die Überlegungen zum System und das daraus resultierende Design wird in Kapitel 3 behandelt. Die Umsetzung sowie daraus resultierende Probleme und Änderungen werden in Kapitel 4 erläutert. Im Kapitel 5 werden Einschränkungen und Probleme aufgezählt die bis Abgabe der Arbeit nicht gelöst werden konnten. Im Kapitel 6 werden Details aus der Implementation besprochen, und im letzten Kapitel wird versucht persönliche Erkenntnisse aus dieser Semesterarbeit aufzuzeigen.

1.1 Einleitung Clippee

Clippee ist ein Client-Peer Forschungsprototyp im Bereich verteilte Systeme. Das Hauptziel dieses weiterlaufenden Projekts besteht darin, die Vorteile der beiden bekannten Paradigmen Client-Server sowie P2P zu vereinen.
In der aktuellen Implementation wird das Netzwerk aus zwei Komponenten gebildet: Das eigentliche Netzwerk, der logische Server, besteht aus einer beliebigen Anzahl von Peers welche vollständig miteinander verbunden sind und die eigentliche Funktionalität des Systems anbieten.
Clients melden sich dann bei einem beliebigen Peer an und können dadurch innerhalb des Systems arbeiten. Siehe Fig. 1.
Das ganze Netz ist sehr flexibel und robust geplant. So bieten die vollständig miteinander verbundenen Peers eine hohe Ausfallsicherheit und die Clients können sich an beliebigen Punkten neu in das Netz verbinden sollte ihr ursprünglicher Einwahlpeer nicht mehr erreichbar sein. Wenn eine direkte Verbindung zwischen Peers unterbrochen wird, dann können diese weiterhin via 2-Hop-Routes miteinander kommunizieren.

Für Details zu Clippee siehe Forschungsbericht [1].

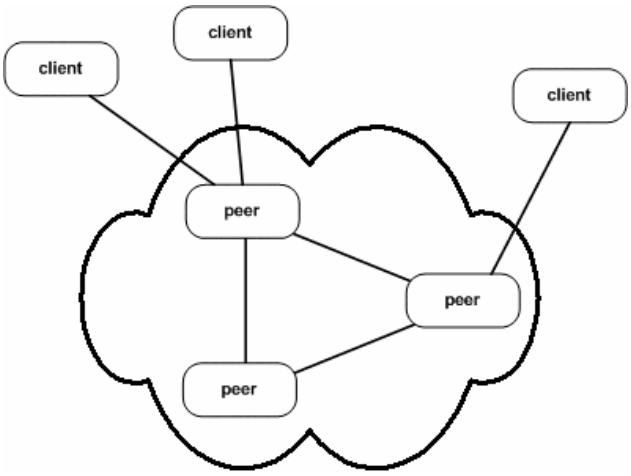


Fig. 1 – Netzwerkstruktur Clippe: Peernetz als logischer Server, Clients melden sich bei einem beliebigen Peer an.

2 Aufgabenstellung

Die aktuelle Version von Clippee ist in Java implementiert und beinhaltet neben dem eigentlichen System ebenfalls eine zentrale Steuereinheit um alle Komponenten des Netzwerkes fernzusteuern – ein wichtiger Punkt um spezifische Aspekte des Projektes testen zu können.

Dieses bereits vorhandene System sollte nun um eine grafische Darstellung erweitert, sowie die Struktur der Fernsteuerung angepasst werden.

Die Aufgabenstellung der vorliegenden Arbeit war zweigeteilt: Zuerst sollte die Darstellung und Steuerung von einem individuellen Peer erstellt werden (2.1). Als zweiter Teil galt es für das gesamte Netzwerk eine zentrale Darstellung sowie Steuerung zu implementieren resp. anzupassen (2.2).

Für die Aufgabenstellung siehe Anhang A.1.

2.1 Individuelles Peer GUI

Das GUI für einen einzelnen Peer soll jeweils alle lokal verfügbaren Informationen von diesem Peer anzeigen und dem Benutzer entsprechende Kontrollfunktionen anbieten:

Informationen	**Kontrollfunktionen**
Verbundene Peers (direkt oder indirekt)	Verbindung unterbrechen / erneuern, Peer stoppen / neu verbinden
Verbundene Clients	Verbindung abbrechen
Datenobjekte	Erzeugen,…. (bereits vollständig vorhanden)

2.2 Globales GUI

Das globale GUI soll alle Informationen des gesamten Netzwerks anzeigen und Fernsteuerungsfunktionen für alle Komponenten anbieten.

Der Benutzer kann somit von einem zentralen Rechner das gesamte Netzwerk überwachen und steuern.

Informationen	**Fernsteuerungsfunktionen**
Alle bekannten Peers sowie deren Status	Erzeugen / Starten / Anhalten
Alle bekannten Clients sowie deren Status	Zum Netzwerk verbinden
Verbindungen zwischen den Peers, Verbindungen Peer-Client	Verbindung unterbrechen / zulassen
Datenobjekte	Abfragen,..

2.3 Automatisierter Testlauf

Manuelle Eingriffe in das Netzwerk sind im kleineren Rahmen und für Demonstrationen angebracht, allerdings kann damit kaum ein grösseres Netz getestet werden. Zu diesem Zweck sollte der vorhandene automatisierte Testlauf portiert werden. Diese Simulation benutzt einstellbare Wahrscheinlichkeitswerte um sämtliche Komponenten fernzusteuern; der Benutzer ist während dieser Zeit nur Zuschauer.

2.4 Zusatzbedingungen

Die Entwicklung von Clippe lief im DCG parallel zur Semesterarbeit weiter. Um die erstellten Funktionen einfach in zukünftige Versionen migrieren zu können, sollte der Sourcecode so wenig wie möglich abgeändert werden und neu erstellter Code nur an spezifischen Punkten in das System einklinken.

3 Design

3.1 Vorgaben

Wie bereits erwähnt war die Fernsteuerungsoption zum Teil bereits implementiert:

Alle Komponenten, dh. Peers und Clients, müssen sich zu Beginn bei einer Central Control anmelden. Diese speichert daraufhin die Verbindungen und benutzt sie für jegliche Überwachung und Steuerung indem sie entsprechende Kontrollmeldungen verschickt. Die Steuerverbindungen gemäss Fig. 2 sind allerdings kein Bestandteil des eigentlichen Netzwerkes!

Für eine Diskussion zu diesem Ansatz siehe Kapitel 6.1.

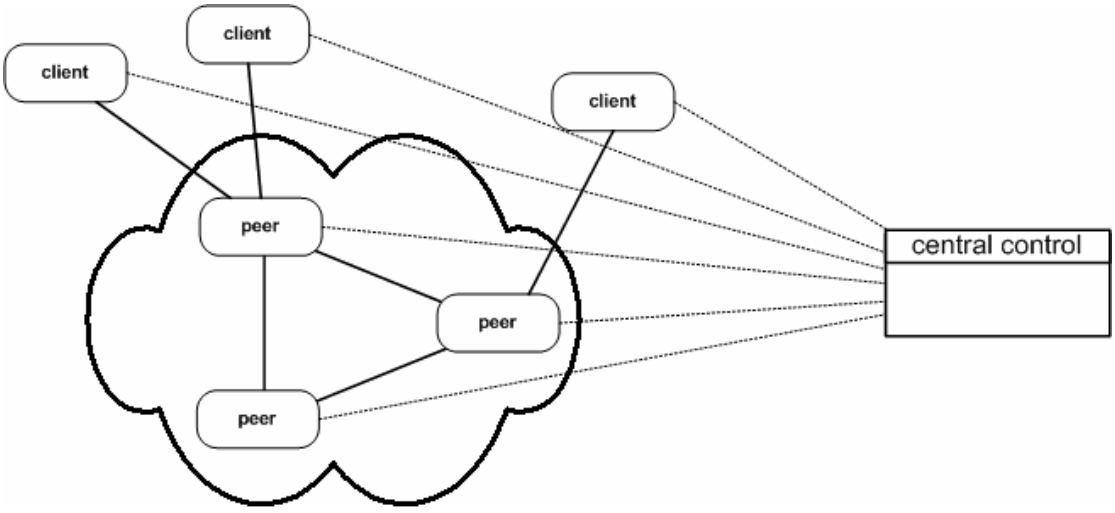


Fig. 2 – Das eigentliche Netzwerk sowie die Steuerverbindungen zur Central Control

3.2 Aufbau

Durch Analyse der vorhandenen Implementation sollte nun zuerst eine Grobaufteilung der Komponenten geschehen.

Das GUI für einen individuellen Peer kann mittels Model-View-Controller Paradigma direkt umgesetzt werden, wie der Aufbau in Fig. 3 zeigt.

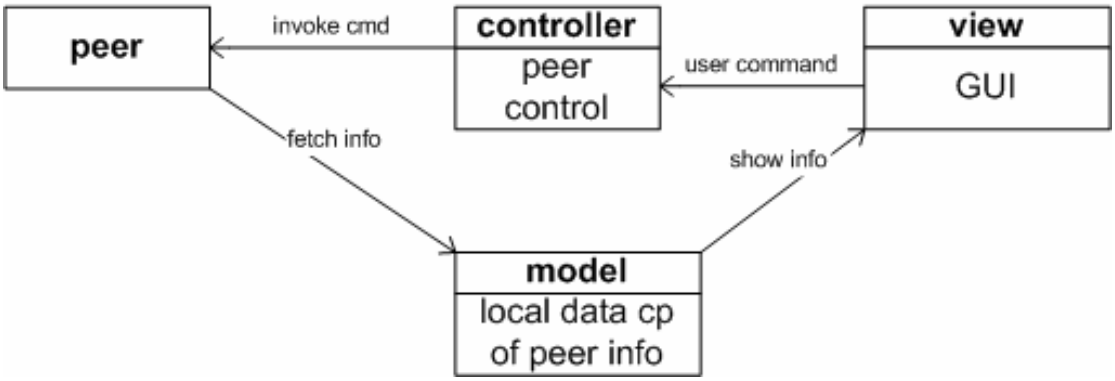


Fig. 3 – Lokales Peer GUI

Jeder Peer wird durch ein Model repräsentiert, welches intern die gewünschten Informationen zum Peer speichert und der View zur Verfügung stellt. Benutzerkommandos im GUI werden via Peer Controller an den eigentlichen Peer weitergereicht.

Die Aufteilung für das globale GUI ist komplexer da für die Darstellung zuerst alle einzelnen lokalen Informationen zusammengetragen werden müssen.
Die logische Trennung von Netzwerk und Central Control gibt zwei Komponenten vor. Dazu kommt noch die eigentliche Darstellung der Daten, womit das Design gemäss Fig. 4 feststeht.

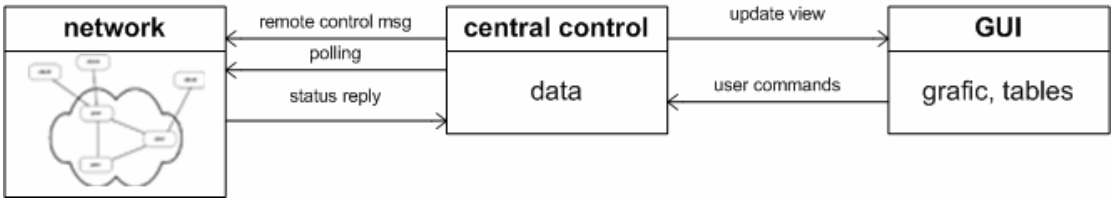


Fig. 4 – Globales GUI

Eine Zuteilung in das Model-View-Controller Paradigma ist auf dieser Ebene nicht möglich, da diese drei Komponenten zuerst weiter verfeinert werden müssen. Die Überlegungen und Details dazu werden in den folgenden Unterkapiteln (3.2.1 .. 3.2.3) aufgezeigt. Diese Werte dienen dann als Basis für die Implementation im nächsten Kapitel.

3.2.1 Netzwerk

Das Netzwerk bildet die unterste Schicht des Systems, wobei die Bestandteile Peer und Client bereits in der Einleitung erklärt wurden. Das Zusammenspiel dieser

Bestandteile ist durch die Implementation von Clippee ebenfalls vorgegeben. In Tabelle 1 werden nun alle möglichen Zustände aufgelistet als Grundlage, was überhaupt dargestellt werden kann.

Element / Zustand	**Erklärung**
Peer	(1..m) Elemente, Verbindung zu allen anderen Peers, beliebige Anzahl verbundener Clients
Client	(0..n) Elemente, Verbindung zu je einem beliebigen Peer möglich
inaktiv	Entität hat sich bei der Central Control registriert, wurde aber noch nicht gestartet und ist somit noch kein Bestandteil des eigentlichen Netzwerkes. (Bsp. Computer mit ausgestecktem Netzwerkkabel)
aktiv	Element ist bei der Central Control registriert und wurde durch diese gestartet. Element ist somit funktionsfähig im Netzwerk. (Bsp. Computer mit eingestecktem Netzwerkkabel)
Verbindung Peer-Peer	Sobald ein Peer von der Central Control gestartet wird, wird zu allen anderen gestarteten Peers eine Verbindung möglich. (Maximal n(n-1)/2 Verbindungen im Netzwerk)
direkt	Die direkte Verbindung von Peer A zu Peer B ist funktionsfähig und wird benutzt.
indirekt (2-Hop)	Die direkte Verbindung von Peer A zu Peer B ist unterbrochen, aber die beiden Elemente können via einem Nachbar-Peer trotzdem miteinander kommunizieren. Der Umweg wird von beiden Peers selbständig gewählt und muss somit nicht identisch sein! In der aktuellen Implementation unterstützen die Peers maximal 2-Hop-Routes.
keine	Die direkte Verbindung von Peer A zu Peer B ist unterbrochen, ebenso besteht keine Verbindung über einen 2-Hop-Route. Die beiden Peers können nicht miteinander kommunizieren.
Verbindung Client-Peer	Jeder Client kann maximal mit einem Login-Peer zum Netzwerk verbunden sein.
verbunden	Der Client ist mit einem aktiven Peer verbunden und kann somit innerhalb des Netzwerkes arbeiten.
nicht verbunden	Der Client ist nicht mit einem aktiven Peer …

	verbunden und somit nicht Bestandteil des Netzwerkes. Das Element versucht selbständig via Central Control einen neuen aktiven Peer zu erhalten.

Tab. 1 - Zustände im Netzwerk

3.2.2 Central Control

Die Central Control ist als Zwischenschicht zur Koordination zwischen Netzwerk und GUI zuständig.
In der vorhandenen Fassung wurde die Central Control bereits unterteilt, um entsprechend auf die Unterschiede von Peers und Clients zu reagieren:
Der Central Peer Coordinator (cpc) und der Central Client Coordinator (ccc) sind jeweils für sämtliche Belange der Peers resp Clients zuständig, wie in Fig. 5 gezeigt.

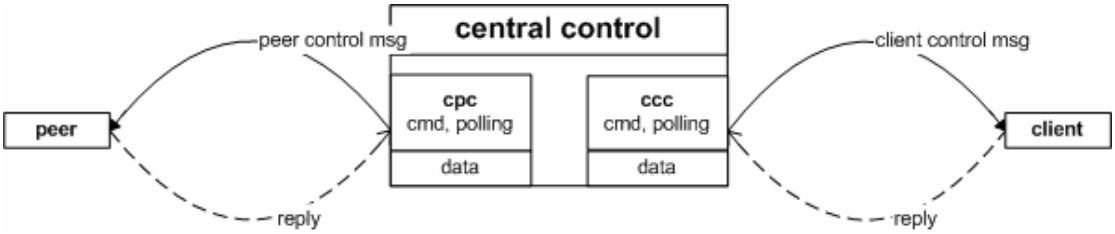


Fig. 5 – CPC und CCC

Aufgaben der Zwischenschicht zum Netzwerk:
- Fernsteuern der Peers und Clients via Kontrollnachrichten (control msg)
- Auswerten von etwaigen Rückmeldungen
- Kontinuierliche Aktualisierung der Netzwerkstruktur (polling)

Aufgaben der Zwischenschicht zum GUI:
- Daten bereitstellen zur Darstellung im GUI (data)
- Schnittstelle für sämtliche Benutzerkommandos im GUI (cmd)

3.2.2.1 Aktualisieren der Netzwerkstruktur

Wegen der unumgänglichen Registrierung kennt die Central Control alle Peers und Clients die überhaupt im Netz aktiv sein können.
Da die einzelnen Elemente allerdings auch unabhängig von der Central Control ausfallen können, darf die Anzeige nicht nur auf den ausgeführten Benutzerkommandos beruhen, sondern das System muss regelmässig alle bekannten Teilnehmer nach ihrem Status befragen. Andernfalls wird das System zB. nie bemerken, wenn eine Putzfrau das Netzwerkkabel zu einem Computer herausziehen sollte.

Diese Aktualisierung geschieht mit einem Pollingmechanismus wie in Fig. 6 dargestellt:
Erhält man vor Ablauf des Timeout eine Antwort, dann ist das entsprechende Element aktiv und man kann die Rückmeldung parsen. Wurde bis zum Timeout keine Antwort empfangen, dann muss man annehmen, dass das Element inaktiv ist. Falls eine

Exception abgefangen wurde, dann ist die Kontrollverbindung zu diesem Element unterbrochen und kann nicht mehr hergestellt werden.
Um alle gewünschten Informationen zusammenzutragen werden im System drei Threads benötigt, welche die folgenden Teildaten sammeln:

- direkte Nachbarn der Peers
- indirekte Nachbarn der Peers (2-Hop)
- Login-Peers der Clients

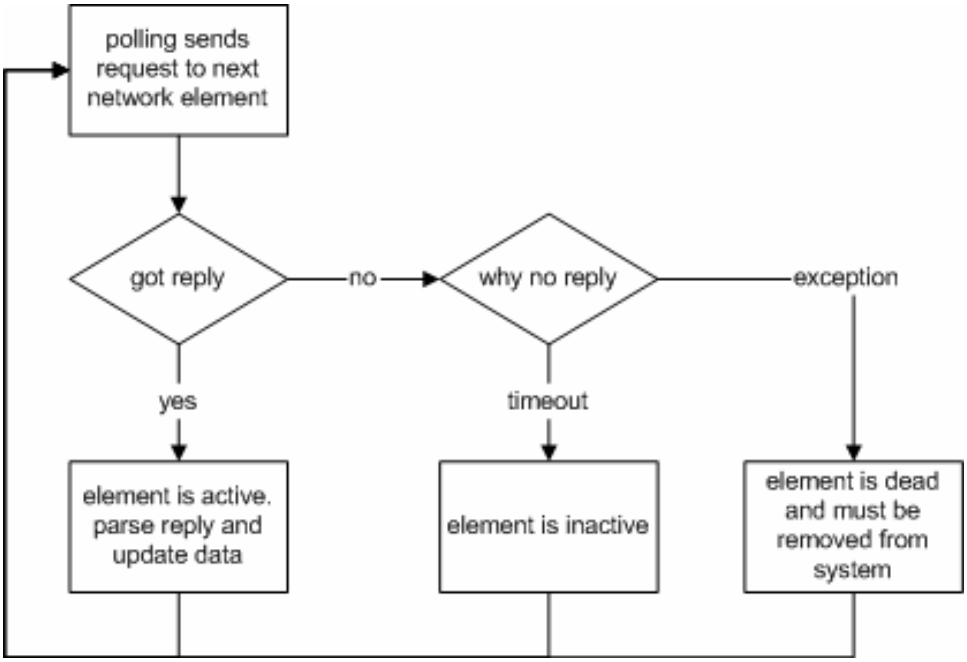


Fig. 6 – Entscheidungsablauf beim Pollingmechanismus

3.2.3 GUI

Das GUI bildet die oberste Schicht, mit direkter Interaktion durch den Benutzer. Ursprünglich war nur ein einziges GUI zur Darstellung sämtlicher Informationen vorgesehen. Da diese Ansicht allerdings kein effektives Arbeiten mit den Elementen erlauben würde und der Bildschirm schlicht überladen wäre, wurden insgesamt drei verschiedene Views eingeführt:

- die grafische Ansicht mit allen Peers, Clients sowie sämtlichen Verbindungen
- eine Tabelle mit allen Peers, deren Status und Verbindungen
- eine Tabelle mit allen Clients, deren Status und Login-Peers

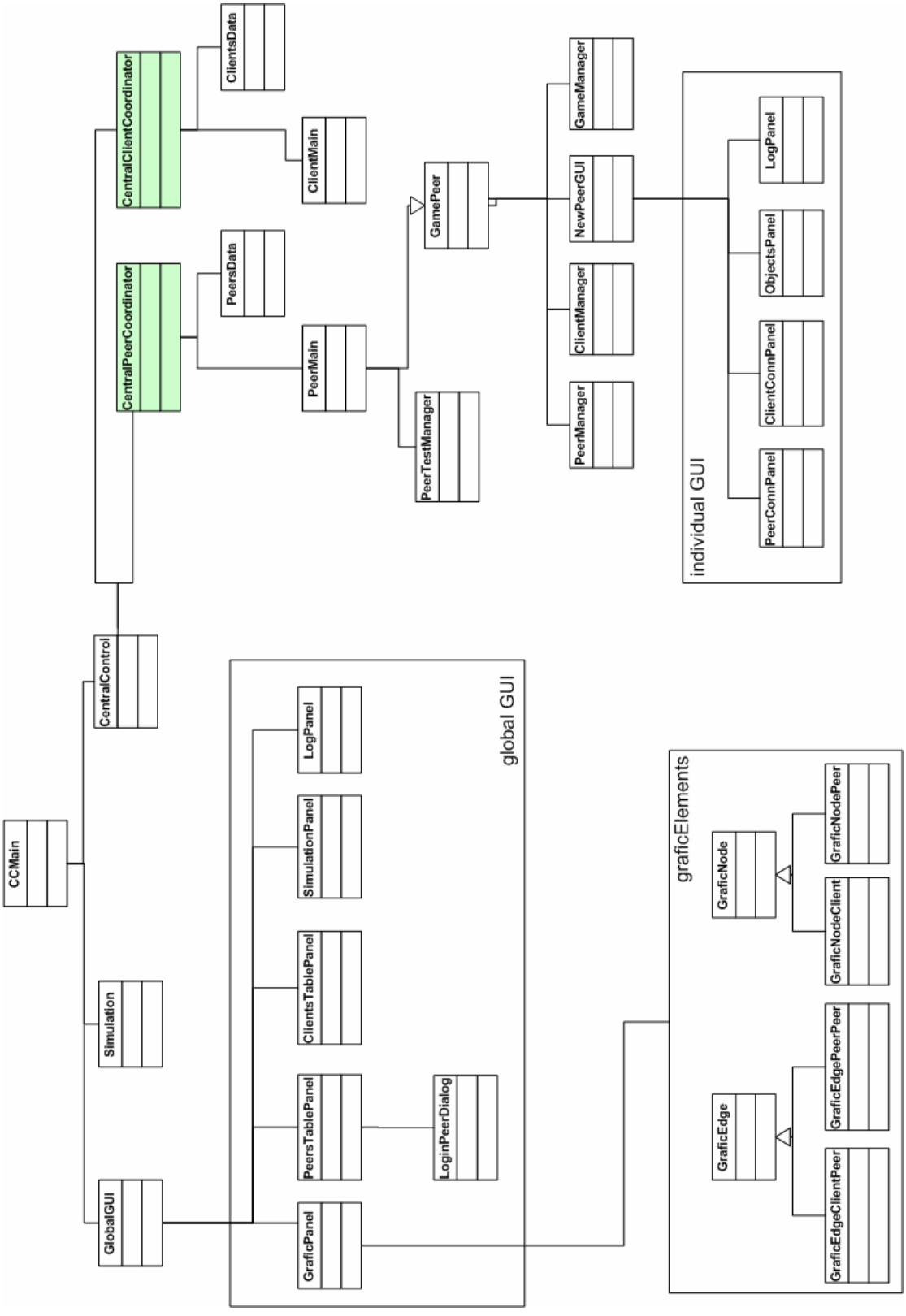
Die Kontrollfunktionen werden dem Benutzer entsprechend seiner Auswahl angeboten, um etwaige sinnlose Kommandos zu verhindern. Wegen der Vielzahl der möglichen Kombinationen kann dies allerdings nicht als Sicherheitsgarantie gesehen werden.

4 Implementation

Für die Implementation wurde Java 1.4. verwendet. Zur Umsetzung der grafischen Elemente in Swing wurde [3], [4] und [5] benutzt.
An dieser Stelle möchte ich Michael Gähwiler meinen Dank aussprechen, dass ich Teile seines Sourcecodes verwenden durfte [2].

4.1 UML Diagramm

Im folgenden Diagramm wird die Klassenstruktur der Implementation dargestellt. Die Variablen und Methoden wurden ausgeblendet, ebenso sind Hilfsklassen nicht im Diagramm enthalten um die Übersicht nicht zu verlieren. Die beiden eingefärbten Klassen bilden die Schnittstelle zum Originalsystem.



4.2 Screenshots

Bei den folgenden Screenshots sind sämtliche implementierten Befehle in den Buttons am oberen Bildschirmrand ersichtlich. Die Namen sind sprechend gewählt, und so wird nur in einem Ausnahmefall eine Erklärung dazu gegeben.

Um Platz zu sparen sind Teile der Screenshots abgeschnitten, wobei allerdings nirgends relevante Informationen verloren gehen.

Fig. 7 zeigt die grafische Ansicht vom ganzen Netzwerk. Die inaktiven Peers und Clients werden weiss hinterlegt am oberen resp. linken Rand dargestellt (1). Die aktiven Peers und Clients werden grau hinterlegt, Peer (2) hat allerdings noch keine Verbindung zu einem anderen Peer. Indirekte Verbindungen zwischen den Peers werden orange dargestellt (3). Client (4) ist bei einem Peer angemeldet.

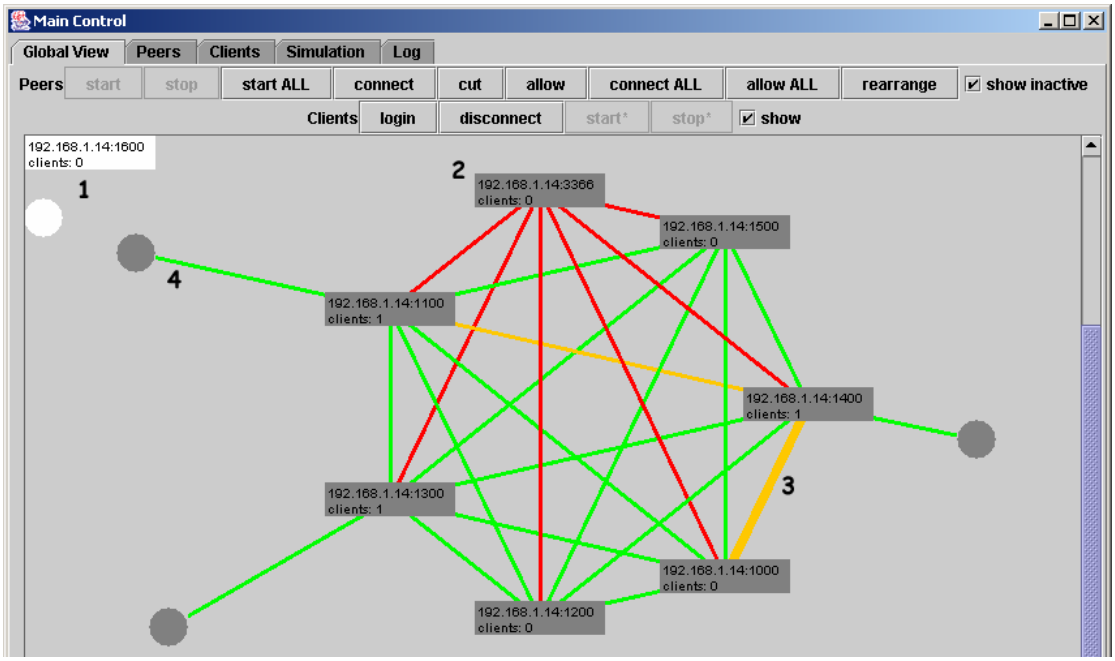


Fig. 7 - Global View Panel

Fig. 8 zeigt nur Informationen zu den Peers. Die Peers selbst können durch Selektion in der ersten Spalte manipuliert werden (1), während die Verbindungen zwischen den Peers in der restlichen Matrix dargestellt werden. Verbindungsunterbrüche können nur hier gezielt simuliert werden (2). Die Peers können keine Verbindung zu sich selbst erzeugen, deshalb ist die Diagonale leer (3). Peer (4) ist analog zu Fig.7 zu keinem Nachbarn verbunden.

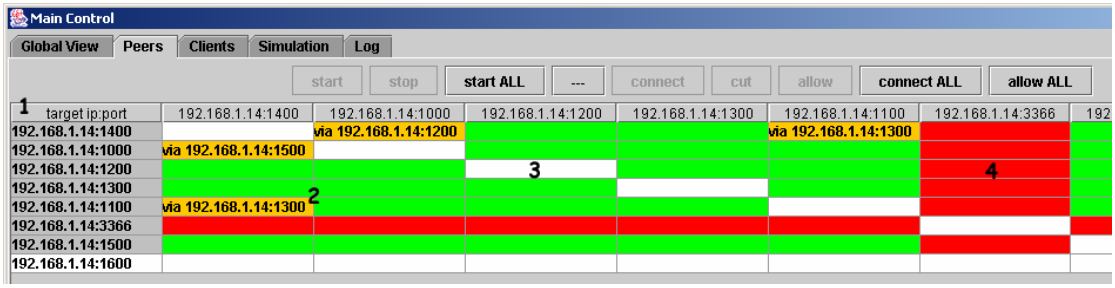


Fig. 8 - Peers Panel

Fig. 9 zeigt nur Informationen zu den Clients und ob diese mit einem Login Peer verbunden sind. Soll sich ein Client bei einem Peer einloggen, dann erscheint ein neues Fenster mit allen bekannten aktiven Peers zur Auswahl. Siehe Fig. 9b.

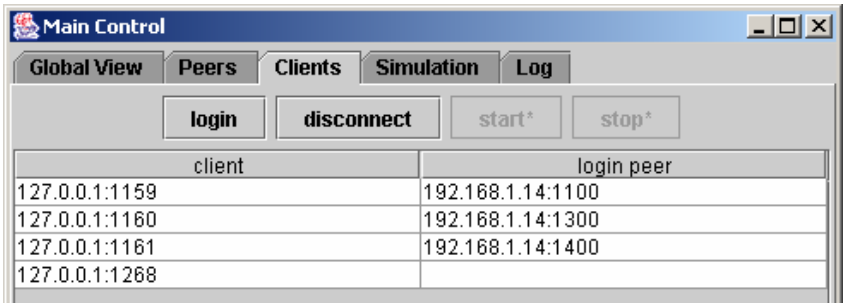


Fig. 9 - Clients Panel

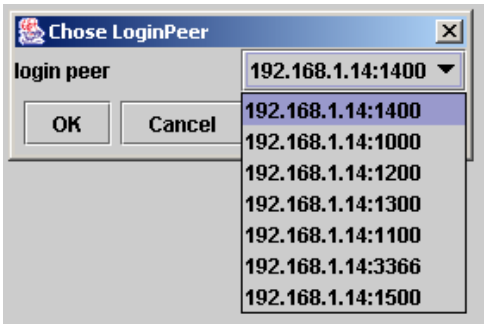


Fig. 9b - Client Dialog

Fig. 10 zeigt das Simulationsfenster. Bei (1) können neue lokale Entitäten erzeugt werden, (2) zeigt die Anzahl der Peers und Clients die zur Zeit bei der Central Control registriert sind. Im unteren Teil (3) kann der Benutzer die gewünschten Wahrscheinlichkeitswerte für den automatischen Testlauf eingeben. Der Testlauf selbst ist zur Zeit aber noch nicht lauffähig.

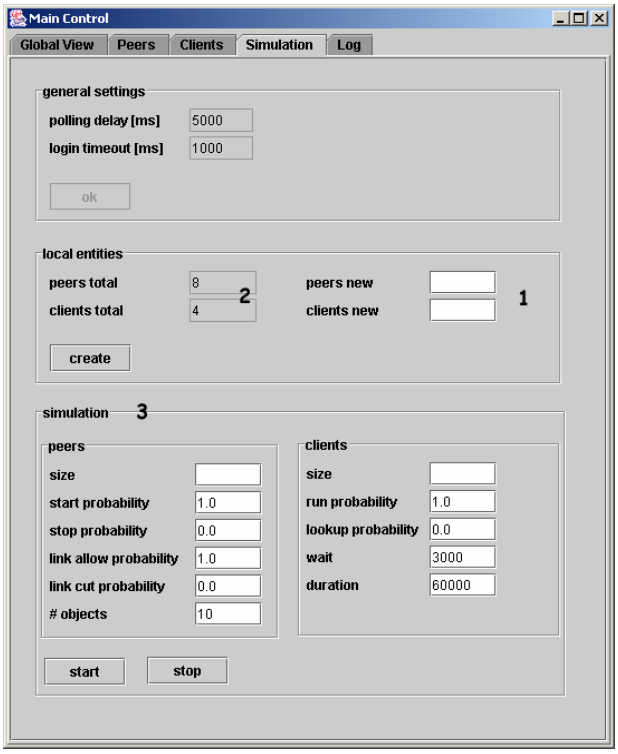


Fig. 10 - Simulation Panel

Fig. 11 zeigt das LogPanel in welchem sämtliche Benutzerinteraktionen aufgelistet werden.

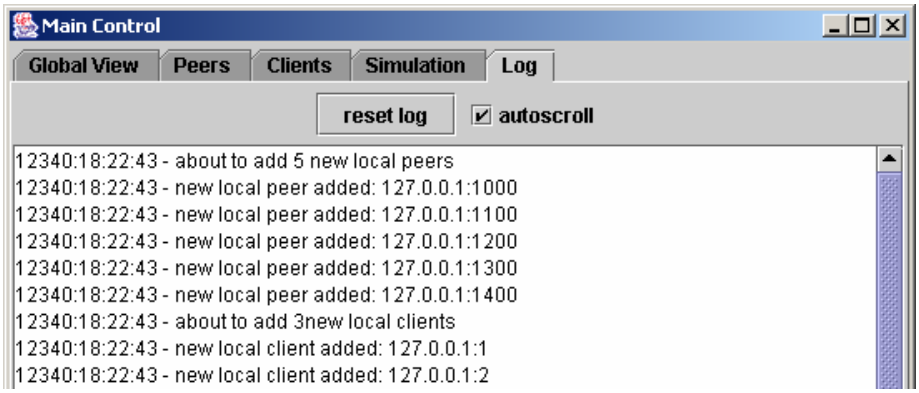


Fig. 11 - Log Panel

Fig. 12 zeigt das individuelle Peer GUI, welches gemäss Parameter bei allen nicht lokal erzeugten Peers angezeigt werden kann. In der Originalimplementation behalten Peers veraltete Routingeinträge und erneuern diese erst wenn sie den Pfad benutzen. Aus diesem Grund kann der Benutzer mit (1) Testnachrichten verschicken um die Anzeige zu aktualisieren. Ein individueller Peer kann keine anderen Entitäten fernsteuern, mit (2) kann er sich lediglich selbst stoppen. Mit (3) kann man den Peer entweder manuell verbinden oder aus einer Liste früherer Peernachbarn auswählen.

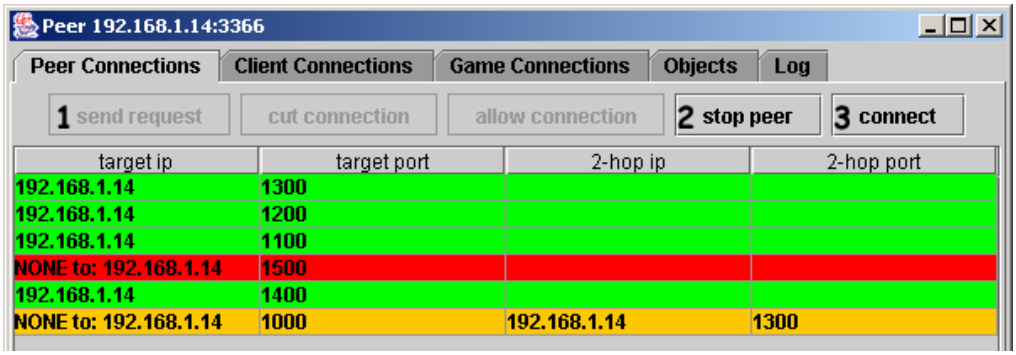


Fig. 12 - Individuelles Peer GUI, Peer Connections Panel

4.3 Veränderte Dateien

Die notwendigen Änderungen im bestehenden Sourcecode wurden jeweils speziell markiert (‚//°°°') um einfacher migriert werden zu können.

Änderungen wurden in folgenden Dateien vorgenommen:
peerTest.CentralControl
peerTest.CentralPeerCoordinator (umfangreiche Änderungen)
peerTest.PeerControlHandler
peerTest.PeerMain
peerTest.PeerTestManager
test.CentralClientCoordinator (umfangreiche Änderungen)
test.ClientMain

Sämtliche neu erzeugten Dateien wurden in den beiden folgenden Packages gespeichert und mit javadoc dokumentiert. Für eine Übersicht aller Funktionen sei darauf verwiesen.

Neu erzeugte Dateien in den Packages:
gamePeer.newPeerGUI
newTest

5 Einschränkungen

- Puffer in Java Prozessen

  Peers und Clients welche lokal auf der Central Control erzeugt werden, sind mit Java Prozessen implementiert.
  System.out.println innerhalb von diesen Prozessen müssen bei etwaigen Erweiterungen unbedingt verhindert, oder die Ausgabe muss umgeleitet werden. Anderenfalls wird der Puffer zu unbestimmter Zeit überlaufen und somit für den Prozess einen Deadlock erzeugen!

- Unterschiedliche Darstellungen

  Der Benutzer kann das Netzwerk wie bereits gezeigt in verschiedenen Ansichten manipulieren.
  In der grafischen Darstellung werden Peer-Verbindungen für beide Richtungen mit nur einer Kante dargestellt. Peer A - Peer B kann somit nicht unterschieden werden von Peer B - Peer A.
  Soll spezifisch eine der beiden Verbindungen manipuliert werden, so muss dies in der PeersTable Ansicht erfolgen.

- Speicherbedarf

  Die lokal erzeugten Peers und Clients sind mit Java Prozessen implementiert und benötigen pro Element ~7MB Speicher. Der Hauptspeicher gibt somit eine obere Grenze für die Anzahl Elemente vor, wenn die Central Control nur für einen lokalen Test benutzt werden soll.
  Zudem hat sich während der Tests gezeigt, dass bereits ab 15 lokalen Elementen der Timeout von 1s beim Polling nicht mehr ausreicht. Eine mögliche Erklärung wären Verzögerungen durch das GUI sowie unfaires Scheduling oder frühzeitiges Auslagern der Prozesse.

- Fernsteuerung

  Simulierte Verbindungsunterbrüche haben keinerlei Einfluss auf die Steuerverbindungen der Elemente zur Central Control. Sie beziehen sich lediglich auf die eigentlichen Verbindungen innerhalb des Netzwerkes.
  Wenn allerdings eine Steuerverbindung verloren geht, dann kann das entsprechende Element nicht mehr ferngesteuert werden und muss sich zuerst neu bei der Central Control anmelden.
  Dies ist kein logischer Fehler vom Design, sondern verdeutlicht nur die Wichtigkeit der Kontrollverbindungen.
  Wenn in Analogie bei einem ferngesteuerten Rechner die Putzfrau das

Netzwerkkabel heraus zieht, dann kann dieser Rechner ja ebenfalls nicht mehr ferngesteuert werden bis jemand das Kabel wieder einsteckt.

5.1 Fehlende Elemente

Bis zur Abgabe der Arbeit konnten folgende Aufgabenteile nicht vollständig zum Laufen gebracht werden:

- automatisierter Testlauf: Probleme mit den benötigten Wartezeiten bis sich der Zustand des Netzwerkes nach der Initialisierung stabilisiert hat. Der Simulationsthread hat keine Kenntnis über diese Werte und unterbricht das ganze System. Konflikte zwischen dem Aktualisieren des Netzwerkes und den Kommandos falls diese zu schnell abgeschickt werden.
- lokales Peer GUI: Anzeige der Verbundenen Clients funktioniert trotz Listeners und Analogie zu den Peers nicht. Möglicherweise wurde irgendwo ein Detail übersehen.

6 Diskussion

6.1 Steuerverbindung

Vor- und Nachteile für die gewählte Variante mit den starren Kontrollverbindungen von der Central Control zu den Peers und Clients:

Vorteile: die Kontrollverbindung ist kein echter Bestandteil des Netzwerkes und somit müssen im bestehenden System keine Änderungen bei den Entitäten selbst vorgenommen werden. Das verwendete Einbinden eines Handlers zur Kontrolle ist sehr flexibel. Wird eine Verbindung innerhalb des Netzwerkes manipuliert, dann kann die Central Control weiterhin ohne Einschränkungen mit ihren Steuerverbindungen arbeiten.

Nachteil: sobald die Kontrollverbindung aus irgendeinem Grund ausfällt, kann der Peer oder Client nicht mehr erreicht werden. Die Kontrollverbindung kann nur vom Teilnehmer zur Central Control hin aufgebaut werden, nicht aber in der Gegenrichtung.
Mögliche Verbesserung, wenn die Central Control diese Gegenrichtung unterstützen, und selbständig Kontrollverbindungen aufbauen könnte.

6.2 Datenkonsistenz

Das Ziel sollte möglichst sein, dem Benutzer die aktuellsten Informationen vom Netzwerk präsentieren zu können.

Die Zustände innerhalb des Netzwerkes können auf zwei Arten ändern:

- die Central Control hat ein Kommando abgeschickt (Benutzerkommando, Testlauf)
- eigenständige Änderung innerhalb des Netzwerk (Crash, Kabel herausgezogen,…)

Meistens wird der Benutzer eine Änderung durch ein Kommando hervorrufen. Diese Änderung wird dann allerdings auch einen Einfluss auf das restliche Netzwerk haben, und so wird der Status des Netzwerkes auf zwei Arten angepasst:

- jedes Benutzerkommando wird direkt auf die Anzeige übertragen, auch wenn das Kommando an sich noch gar nicht ausgeführt wurde.
- der Pollingmechanismus meldet regelmässig seine aktualisierten Daten

Durch diese beiden Aktualisierungsmechanismen kann die Anzeige kurzzeitig ändern und während einer kurzen Zeit ($<t_{polling}$) kann ein falscher Wert angezeigt werden wie in Fig. 13 dargestellt.

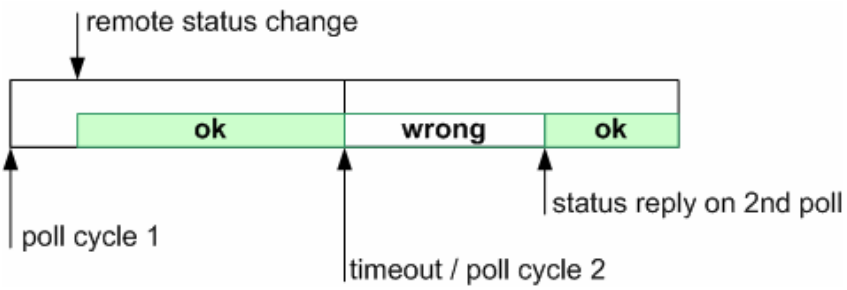


Fig. 13 – Inkonsistenz wegen Polling

Während dem automatisierten Testlauf darf der Zustand des Netzwerkes nur anhand dem Polling angepasst werden, sonst können Fehler auftreten!

Bsp. Peer X soll gestartet werden und gleichzeitig wird der Status neu gesetzt. Die Ausführung innerhalb des Prozesses X wird unerklärlicherweise verzögert, und bevor der Peer tatsächlich gestartet ist, möchte ein weiterer Peer Y zu diesem fiktiv gestarteten Peer X verbinden. Resultat: Socket Exception und die Kontrollverbindungen zu Peer X und Y sind verloren.

Bei einem automatisierten Testlauf werden die Daten spätestens nach $2*t_{polling}$ korrekt dargestellt: Wird der Status während dem laufenden Pollingzyklus 1 durch den Benutzer geändert, dann erkennt das der aktuelle Zyklus nicht, weil die Anfragen jeweils nur am Anfang verschickt werden. Nach Ablauf des ersten Timeout startet der zweite Pollingzyklus. Die manipulierte Entität ist nun zur Zeit der Anfragen vorhanden und wird ihren Zustand vor Timeout2 zurückmelden.

7 Persönliche Erkenntnisse

- Die vorliegende Semesterarbeit wurde zweimal durch Einschub von Prüfungen unterbrochen und hinausgezogen. Sofern irgend möglich sollte eine Arbeit am Stück bearbeitet werden.
- In einer Diskussion oder Präsentation lernt man mehr als wenn man nur Dokumentationen liest!
- GUIs mit Java Swing kann man zwar relativ leicht erstellen, in der Anwendung sind diese aber unangenehm langsam; sobald mehrere JComponents gleichzeitig darzustellen sind, wird das ganze System langsam.

A. Anhang
A.1 Arbeitsplan

Semesterarbeit “Implementation of a monitoring and testing tool for a Peer-to-Peer System”

Dieses Dokument gibt den Rahmen der Semesterarbeit von Guido Hager im SS03 vor. Abweichungen oder Änderungen sind in gegenseitiger Absprache möglich. Die vorgegebenen Zeiten sind als ungefähre Richtlinien gedacht.

Arbeitsplan:

1) Kennen lernen des vorhandenen Systems CLIPPEE und Einarbeitung in den Java-Code. [10h]

2) Analyse und Design der zu erstellenden graphischen Überwachungs- und Testumgebung in Rücksprache mit Ruedi (und Keno). [20h]

3) Implementierung des graphischen Überwachungs- und Testsystems für einen Peer. [40h]

4) Implementierung einer zentralen graphischen Umgebung für das gesamte System. [40h]

5) Zum Abschluss wird ein kompakter Bericht (rund 10 Seiten, Sprache wählbar) erstellt, der über die Arbeit und die Resultate Auskunft gibt. Dieser Bericht soll unter anderem auch eine kritische Beurteilung der eigenen Arbeit enthalten. [40h]

Allgemeines:

- Selbstständiges Arbeiten ist Voraussetzung.
- Im HRS steht ein Arbeitsplatz zur Verfügung. Es besteht jedoch auch die Möglichkeit, zu Hause zu arbeiten.
- Präsentation der Resultate zum Abschluss der Arbeit.
- Pro Woche mindestens ein Treffen mit Ruedi.

Kontaktpersonen:

1. Ruedi Arnold rarnold@inf.ethz.ch
2. Keno Albrecht albrecht@inf.ethz.ch
3. Roger Wattenhofer wattenhofer@inf.ethz.ch

A.2 Starten des Systems

Start der Central Control:
`java peerTest.CCMain`

Erzeugen eines Peers und Registrierung bei der Central Control:
`java peerTest.PeerMain <peerPort> <cpcIP> <cpcPort> <showGui>`
Bsp. `java peerTest.PeerMain 1000 127.0.0.1 7777 true`

Erzeugen eines Client und Registrierung bei der Central Control:
`java test.ClientMain <cccIP> <cccPort> <id>`
Bsp. `java test.ClientMain 127.0.0.1 9999 123`

A.3 Beispielablauf

Im Folgenden soll anhand eines Beispielablaufes die Interaktion des Benutzers mit dem System gezeigt werden. Behandelt wird nur eine grobe Funktionsübersicht um den Einstieg und die Bedienung zu erleichtern.

Für sämtliche Anzeigen kann der Benutzer durch Drücken der ctrl-Taste mehrere Komponenten markieren. Die Buttons werden anhand der Benutzerauswahl automatisch aktiviert resp. deaktiviert um jeweils nur sinnvolle Kommandos zu ermöglichen.

1) Starten der Central Control

Neues Terminal öffnen, `java peerTest.CCMain` ausführen

2) Elemente zum Arbeiten erzeugen (zuerst lokal)

Im GUI zum SimulationPanel wechseln, im mittleren Feld die gewünschte Anzahl lokaler Peers (5) und Clients (3) eingeben und mit <create> bestätigen.

Diese lokal neu erzeugten Entitäten registrieren sich nach kurzer Zeit bei der Central Control, sind aber inaktiv und nicht Bestandteil des eigentlichen Netzwerkes solange sie nicht gestartet werden.

3) Starten der Entitäten

Zum Global View Panel wechseln, dort die drei Peers 1000, 1100, 1200 markieren und mit <start> starten.

Die aktiven Peers werden nun grau hinterlegt und die möglichen Verbindungen untereinander werden eingezeichnet. Da bis jetzt noch keine Peers miteinander verbunden sind erscheinen alle Linien zuerst rot.

Die Darstellung vom Netzwerk ist nun ziemlich unübersichtlich. Die Knoten können entweder manuell nach Belieben plaziert, oder automatisch mit dem Kommando <rearrange> ausgerichtet werden. Dies wird sämtliche aktive Peers zirkulär anordnen, mit etwaigen verbundenen Clients in einem äusseren Kreis.

Die Anordnung der Entitäten wird nicht automatisch aktualisiert, da sonst bei Timeouts alles ständig hin und her verschoben wird und der Benutzer nicht wirklich etwas erkennen kann. Zudem würden manuelle Einstellungen gleich wieder überschrieben.

Zum Peers Panel wechseln und die restlichen inaktiven Peers mit <start ALL> starten.

Im Peers Panel werden in den Zeilen immer zuerst die aktiven Peers dargestellt und erst anschliessend die inaktiven (neu oder gestoppt).

In der Matrix zeigt das Feld (Zeile, Spalte) den Verbindungszustand von $Peer_{Zeile}$ zu $Peer_{Spalte}$. Achtung: Symmetrie besteht nicht in allen Fällen.

4) Peers verbinden (Aufbauen des Netzes)

Das rote Feld (1000, 1300) markieren und mit <connect> verbinden.

Der $Peer_{1000}$ versucht nun, sich zum $Peer_{1300}$ zu verbinden. Falls dies geklappt hat, dann erscheint das Feld neu grün (direkte Verbindung), ebenso wird aus Symmetriegründen die Verbindung (1300, 1000) grün.

Wechsel zum GlobalView Panel wo nun eine grüne Verbindung angezeigt werden sollte. Auswahl der roten Verbindung $Peer_{1300}$ - $Peer_{1200}$ (Markieren der Kante) und Verbinden mit <connect>. Die Peers tauschen untereinander selbständig ihre Nachbarn aus und somit sollte automatisch auch die Verbindung $Peer_{1200}$ - $Peer_{1000}$ nach grün wechseln.

Achtung: das Kommando <connect ALL> versucht alle aktiven Peers miteinander zu verbinden, auch wenn einige davon bereits miteinander verbunden sein sollten. Diese Funktion ist also nur ganz am Anfang zu benutzen, sonst erzeugt das System Mehrfachverbindungen was zu Socket Exceptions in der unteren Schicht führt.

Verbinden der restlichen Peers, bis alle Kanten grün angezeigt werden.

5) Verbindungsunterbrüche im Netz simulieren

Markieren der Verbindung $Peer_{1000}$ - $Peer_{1300}$ und mit <cut> unterbrechen. Die Kante ändert nach rot (nicht verbunden) und sollte dann orange werden, weil die Peers merken, dass sie zwar die direkte Verbindung verloren haben, aber über Nachbarn immer noch Kontakt zueinander haben (2 Hop). Die beiden Umwege werden als Popup angezeigt, wenn man mit der Maus kurze Zeit über der Kante verweilt.

Wechsel zum Peers Panel. In der Matrix können die beiden Richtungen einer Verbindung gezielt manipuliert werden, was in der grafischen Ansicht nicht möglich ist.

6) Clients in das Peernetz einloggen

Wechsel zum Clients Panel. Zwei Clients markieren und <login> wählen. In einem neuen Fenster kann man nun aus allen aktiven Peers den Login-Peer auswählen.

Wechsel zum GlobalView Panel.

7) Elemente zum Arbeiten erzeugen (nicht lokal)

Die Idee von Clippee besteht ja darin, dass Peers und Clients auf entfernten Computern laufen und über das reale Netz miteinander kommunizieren. Die lokalen

Entitäten waren lediglich für die Simulation implementiert worden. Das System selbst kann lokale und entfernte Entitäten allerdings nicht unterscheiden.

Für entfernte Peers wurde zusätzlich das individuelle GUI implementiert (siehe Fig.12), da die einzelnen Peers kein Wissen über das globale Netz haben.

Neues Terminal auf Computer2 öffnen und `java peerTest.PeerMain 3366 computer1 7777 true` ausführen.

Die Central Control zeigt den neuen Peer an, aber erst wenn der $Peer_{3366}$ gestartet wird, erscheint auf dessen Computer das individuelle GUI.

Im Global View Panel den $Peer_{3366}$ markieren und <start> drücken.

Im Global View Panel die Verbindung $Peer_{3366}$ - $Peer_{1000}$ auswählen und <connect> drücken. Im individuellen GUI sollten nun alle bekannten Peernachbarn angezeigt werden.

Im individuellen GUI mit <stop> den $Peer_{3366}$ stoppen. Das GUI verschwindet und auch im Global View Panel wird dieser Peer inaktiv angezeigt.

$Peer_{3366}$ erneut markieren und mit <start> starten. Wechsel zum individuellen Peer. Auswahl von <connect> bringt ein neues Fenster hervor und der Benutzer kann entweder manuell einen Peernachbarn eingeben, oder aus allen früher bekannten Peers auswählen.

Um die Central Control real zu testen, kann man nun den individuellen $Peer_{3366}$ in dessen Terminal abschiessen. Die Central Control sollte diesen Peer zuerst inaktiv darstellen und nach einem weiteren Pollingzyklus ganz vom System entfernen, da die entsprechende Kontrollverbindung nicht mehr vorhanden ist.

8) Beenden

Sobald das Fenster der Central Control geschlossen wird, werden sämtliche lokal erzeugten Entitäten automatisch terminiert und das System beendet. Sollte das GUI aus irgendeinem Grund abstürzen, so bleiben die lokal erzeugten Entitäten als Prozesse weiterhin bestehen und müssen manuell beendet werden.

A.4 Literaturliste

[1] Keno Albrecht, Ruedi Arnold, and Roger Wattenhofer. Clippee: A Large-Scale Client/Peer System**.** In the Proceedings of the International Workshop on Large-Scale Group Communication, held in conjunction with the 22nd Symposium on Reliable Distributed Systems (SRDS), Florence, Italy, October 2003.

[2] Michael Gähwiler. VisualRemos: Visualisierungstool für topologie-sensitive Applikationen. Semesterarbeit am Institut für Computersysteme, ETH, 2002.

[3] Matthew Robinson, Pavel Vorobiev. Manning, Swing

[4] Robert Eckstein, Marc Loy, Dave Wood. O'Reilly, Java Swing

[5] Suns Java Swing Tutorial