Hochschule
Ravensburg-Weingarten
Technik | Wirtschaft | Sozialwesen

Studiengang Angewandte Informatik
Fakultät Elektrotechnik und Informatik

Dokumentation Gruppe 10

Systemprogrammierung

vorgelegt von:

Frank Klameth
20688
frank.klameth@hs-weingarten.de

Simon Westphahl
20146
simon.westphahl@hs-weingarten.de

Michael Wydler
20168
michael.wydler@hs-weingarten.de

2. Dezember 2010

Inhaltsverzeichnis

1 Aufgabenverteilung

Client	Michael Wydler
Server	Simon Westphahl
Logger, Archivierer (und Sekretärin)	Frank Klameth

2 Entwurf der Softwarekomponenten

2.1 Funktionalität Client

Das Programm wird aus der Konsole gestartet. Es müssen folgende Parameter angegeben werden:

- -s - Servername oder IP (Servername wird in IP umgewandelt)
- -p - Port[1]
- -b - Benutzername
- -r - Rolle

Dabei können Benutzername und Rolle frei gewählt werden. Ist die angegebene Rolle schon belegt, wird der Benutzer automatisch als Student eingetragen.
Die Darstellung der Tafel, die Eingabe von Text auf der Tafel und die Anwahl von Kommandos durch Auswahl der Buttons wird durch unterschiedliche Threads realisiert.

Beispiele:

```
> ./client 127.0.0.1 50100 michael student
> ./client 127.0.0.1 michael dozent
```

2.1.1 Command-Thread

Der Command-Thread ist für das Senden von Aufträgen an den Client-Thread im Server zuständig. Aufträge werden durch Anwahl eines Buttons getätigt und durch den Command-Thread an den Server geschickt.

2.1.2 Live-Agent

Der Live-Agent wird aktiv, wenn sich der Tafelinhalt ändert, d.h. wenn der Eingabefokus auf der Tafel steht und der Anwender Text eingibt oder verändert. Der Live Agent schickt die Änderungen in der lokalen Tafel an den Client-Thread im Server. Der Client-Thread übernimmt die Änderungen in die virtuelle Tafel im Server und sorgt dafür, dass alle angemeldeten Clients die geänderte Tafel erhalten.

2.1.3 Listener-Thread

Der Listener-Thread hört auf Nachrichten von seinem Client-Thread im Server, wertet die Nachrichten aus und übernimmt die Verarbeitung der Nachricht im Client. Er ist daher auch für die Aktualisierung der lokalen Tafel und der Statusinformationen in der GUI zuständig.

[1] optional, wird sonst auf Default-Port (50000) gesetzt

2.1.4 GUI

- Die virtuelle Tafel wird im mittleren Bereich in der Farbe Grün mit weißer Schrift dargestellt.
- Status- und Fehlermeldungen für den Client werden in einem Bereich unter der Tafel angezeigt.
- Die Aufträge des Clients an den Server können über Buttons angewählt werden. Die Buttons werden in einem Bereich links neben der Tafel angezeigt. Es sind immer nur die Buttons aktiviert, die vom Client in seiner jeweiligen Rolle auch angewählt werden können.

2.1.5 Tafel-Trigger

Wenn auf die Tafel geschrieben wird, dann wird ein Timeout-Signal gestartet. Wenn dieses abgelaufen ist, wird die Tafel an den Server gesendet und somit an alle Clients verteilt. Bei jeder Änderung wird der Timeout zurückgesetzt. Wenn der Timeout 5x zurückgesetzt wurde, dann wird die Tafel dennoch zum Server gesendet und der Timeout-Counter zurückgesetzt.

2.2 Funktionalität Server

Der Server darf nur genau einmal gestartet werden. Ein wiederholtes Starten des Servers soll erkannt und zu einer Fehlermeldung führen.
Der Server erzeugt zwei Sohnprozesse, den Logger und den Archivierer. Falls beim Archivierer der Testmodus eingeschaltet werden soll, muss der Server dem Archivierer die Periodendauer übergeben. Diese Periodendauer erhält der Server beim Starten als Kommandoparameter und gibt den eingegeben Wert an den Archivierer weiter.
Für jeden Auftrag, der im Server abgearbeitet wird, erzeugen die beteiligten Threads eine oder mehrere Protokollnachricht(en) und senden diese über eine Message-Queue an den Logger.

Das Programm wird aus der Konsole gestartet. Es können folgende Parameter angegeben werden:

- -p - Port[2]
- -d - Debugging-Mode (ohne Argument)

Beispiele:

```
> ./server -p 8080 -d
> ./server
```

[2]optional, sonst wird Default-Port (50000) benutzt.

2.2.1 Login-Thread

Zur Verwaltung der angemeldeten Clients existiert im Server ein Login-Thread. Jeder Client meldet sich über diesen Login-Thread beim Server an und wird bei Erfolg in einer globalen Datenstruktur „Clientliste“ eingetragen. Danach wird für jeden Client ein Client-Thread im Server gestartet. Alle Clients-Threads im Server haben Zugriff auf die Clientliste.

2.2.2 Client-Thread

Der Server erstellt für jeden Client einen eigenen Client-Thread, der auf Aufträge von seinem Client wartet und diese Aufträge ausführt. Der Client-Thread hat Zugriff auf die virtuelle Tafel. Er kennt folgende Kommandos:

- login
- quit
- request
- shutdown
- release
- acquire
- modify
- clear

2.2.3 Broadcasting-Agent

Der Broadcasting-Agent sendet die Statusnachrichten an alle Clients bei einer Statusänderung. Außerdem ist er für das Versenden der Tafel bei einer Änderung an alle angemeldeten Clients zuständig.

2.3 Funktionalität Logger

Der Logger protokolliert alle Protokoll-Meldungen in einer Logdatei.

2.4 Funktionalität Archivierer

Der Archivierer ist ein Prozess zum Speichern von Tafelinhalten und ist als Sohnprozess des Servers zu realisieren.
Der Archivierer hat 2 Funktionen:

- Archivieren des Tafelinhaltes in eine Archivdatei, wenn die virtuelle Tafel gelöscht werden soll.

- Speichern der Tafel periodisch in eine Debugdatei, wenn der Testmodus eingestellt ist. Die Zeitdauer soll einstellbar sein. Der Testmodus und die Zeitdauer wird beim Starten des Archivierers eingestellt.

Der Archivierer wird vom Server einmal erzeugt und wartet dann auf einen Trigger. Dieser Trigger wird vom Client-Thread gesetzt, wenn die Tafel gelöscht werden soll oder aber von einem Timer, wenn der Testmodus eigestellt ist.

2.5 Synchronisationsprotokoll

Siehe Anhang A.

2.6 IPC-Schnittstellen

2.6.1 Shared-Memory

Shared-Memory wird an folgenden Stellen benutzt:

- Tafelinhalt

2.6.2 Message-Queues

Message-Queues werden an folgenden Stellen benutzt:

- Logger

Loglevel:
1 Fehler
2 Info
3 Debug

2.6.3 Semaphore

Semaphore werden an folgenden Stellen benutzt:

- Tafel (binär)
- Client-Liste (binär)
- Trigger Archivierer

2.6.4 Condition-Variablen

Condition-Variablen werden für den Trigger des Broadcasting-Agent verwendet.

2.7 Netzwerkschnittstellen

2.7.1 Typenübersicht

Nr.	Name	Beschreibung
0	Login	Login-Nachricht
1	Status-Msg	Status-Nachricht
2	Board-Content	Tafel-Inhalt
3	Clear-Board	Tafel löschen
4	Shutdown	System herunterfahren
5	Write-Request	Schreibrecht-Anfrage (Client -> Server)
6	Write-Requested	Schreibrecht-Weiterleitung (Server -> Dozenten)
7	Request-Reply	Schreibrecht-Antwort (Dozenten Client -> Server)
255	Error-Msg	Fehler-Nachricht

2.7.2 Header

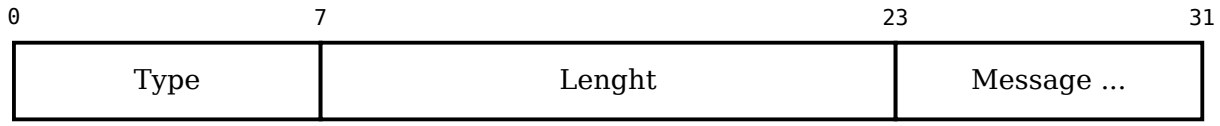

Abbildung 1: Header

Type:	uint8_t	beschreibt den Typ der folgenden Nachricht
Lenght:	uint16_t	gibt die Anzahl der folgenden Bytes an

2.7.3 Login

Der Client sendet eine Login Nachricht.
Der Server antwortet mit einer Status Nachricht oder Fehler Nachricht –> Auf beides reagieren!

- Kommt eine ungültige Rolle zurück, dann muss sich der Client beenden.
- Die Rolle die zurück kommt ist verpflichtend.
- Der Server schickt die Tafel dem Client.

Nach jedem Login wird eine Status Message an jeden Client gesendet, da sich die Zahl der Studenten oder Dozenten geändert hat (das Selbe gilt für einen Logout).

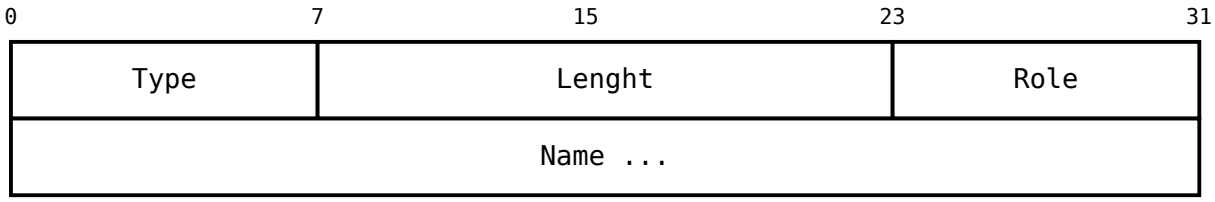

Abbildung 2: Login

Role:	uint8_t	Egal = 0; Student = 1; Dozent = 2; Alles andere Ungültig!
Name:	Char-Array	Nicht null-terminiert (Length - 1) Bytes lang

2.7.4 Tafelinhalt

Der Client tippt, und schickt die ganze neue Tafel, der Server nimmt die neue Tafel auf und verteilt die Tafel mit einer Board-Content Nachricht an alle, die keine Schreibrechte haben.

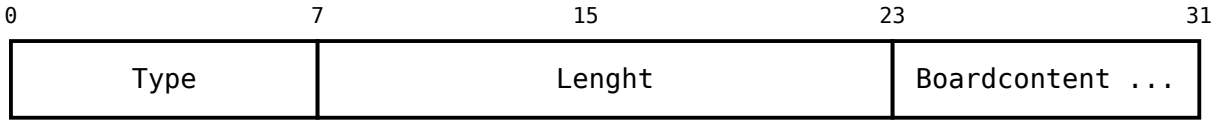

Abbildung 3: Tafelinhalt

Board-Content:	Char-Array	nicht null-terminiert Length Bytes lang

2.7.5 Tafel löschen

Der Client mit Schreibrechte schickt eine Clear-Board Nachricht an den Server. Der Server triggert den Archivierer, löscht den Tafelinhalt und schickt eine leere Tafel an alle Clients.

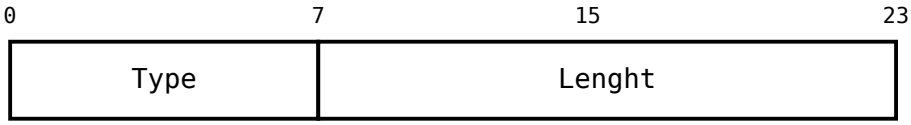

Abbildung 4: Tafel löschen

2.7.6 Schreibrechtanfrage

Der Student wünscht sich Schreibrecht:
Es wird eine Write Request Nachricht an den Server geschickt, dieser leitet eine Write-Requested Nachricht an den Dozenten Client weiter. Der Dozent beantwortet die Anfrage und es wird vom Dozenten Client eine Request-Reply Nachricht an den Server geschickt, dieser verarbeitet diese dann.

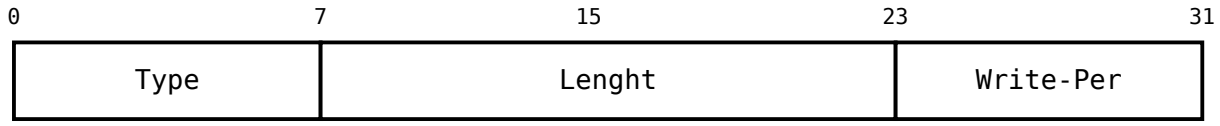

Abbildung 5: Schreibrechtanfrage C -> S

Write-Per: uint8_t | Gewünschtes Schreibrecht

Der Dozent entzieht einem Client das Schreibrecht:
Vom Dozenten Client wird eine Write-Request an den Server geschickt. Der Server entzieht dem Client mit Schreibrecht das Schreibrecht.

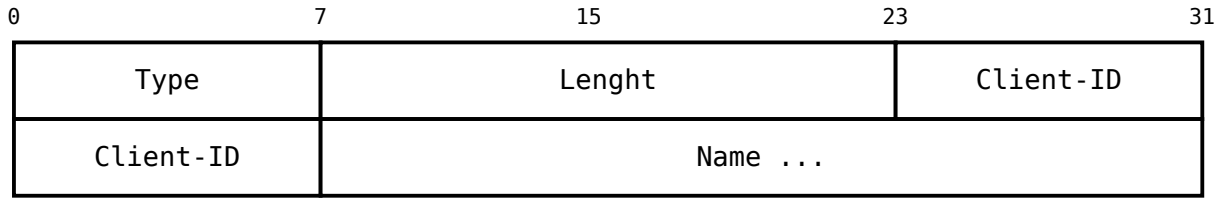

Abbildung 6: Schreibrechtanfrage S -> D

Client-ID: uint16_t | Die ID des Clients der Schreibrecht angefordert hat
Name: Char-Array | nicht null-terminiert, (Lendth - 2) Bytes lang

Der Dozent wünscht sich Schreibrecht:
Vom Dozenten Client wird eine Write-Request an den Server geschickt. Der Server entzieht dem Client mit Schreibrecht das Schreibrecht und setzt dem Dozenten das Schreibrecht.

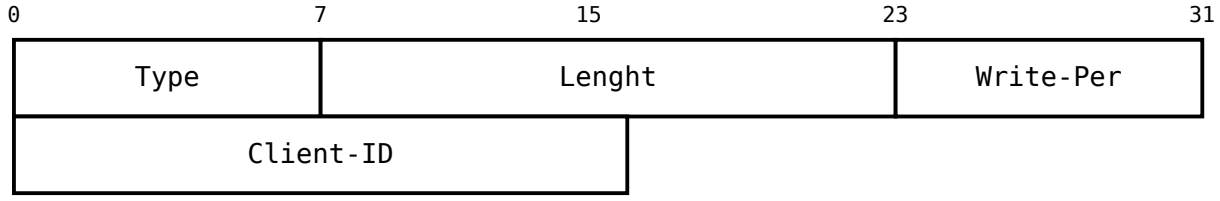

Abbildung 7: Schreibrechtanfrage D -> S

Write-Per: uint8_t | Das neue Schreibrecht
Client-ID: uint16_t | Die Identifizierungsnummer

2.7.7 Beenden

Der Dozenten Client sendet den Befehl das System herunter zu fahren. Der Server sendet eine Error-Msg mit Error-Code 0 (Servernachricht) und einem Shutdown Text an alle Clients

über den Broadcast Agent. Der Client zeigt diese Nachricht an. Der Server fährt sich herunter und schließt dabei alle Verbindungen. Der Client wartet bis die Verbindung schlussendlich getrennt wird.

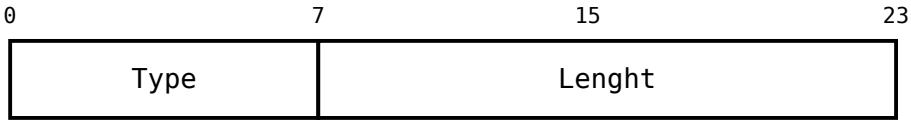

Abbildung 8: Beenden

2.7.8 Status-Nachricht

Nach jeder Rechteänderung wird eine Statusmeldung an alle Clients gesendet.

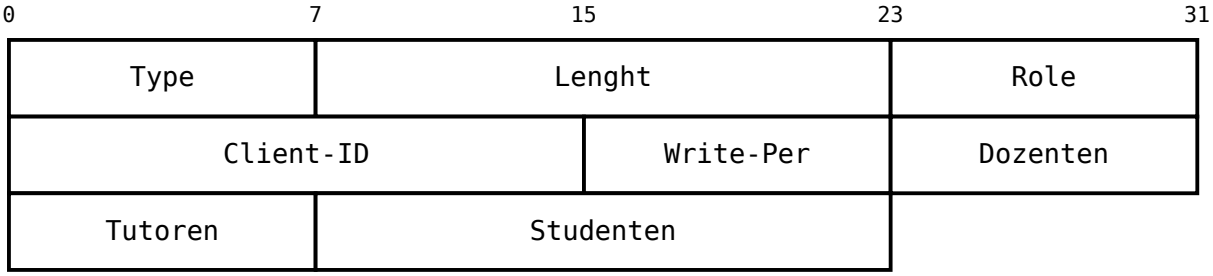

Abbildung 9: Status-Nachricht

Role:	uint8_t	Student = 1; Dozent = 2
Client-ID:	uint16_t	Die Identifizierungsnummer
Write-Per:	uint8_t	ohne Schreibrecht = 0; mit Schreibrecht = 1 –> alles andere ungültig
Dozenten:	uint8_t	Anzahl der Dozenten
Tutoren:	uint8_t	Anzahl der Tutoren
Studenten:	uint16_t	Anzahl der Studenten

2.7.9 Fehlernachricht

Fehlermeldungen können über Error-Msg gesendet werden, sowie Benachrichtigungen an den Client.

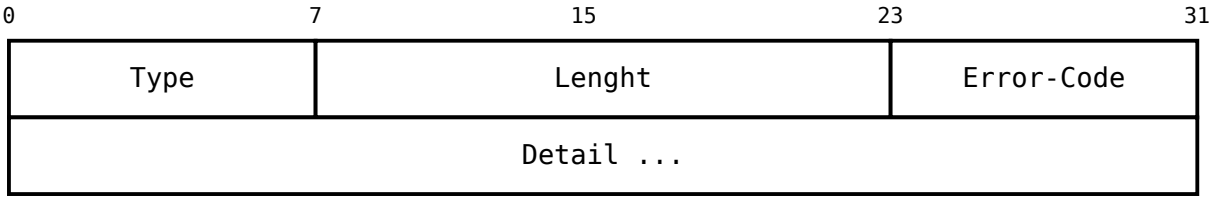

Abbildung 10: Fehlernachticht

Error-Code:	uint8_t	Grobe Beschreibung des Fehlers
Detail:	Char-Array	nicht null-terminierter (length - 1) Bytes lang; Leer möglich!

3 Client

3.1 Programmabläufe

3.1.1 Hauptprogramm

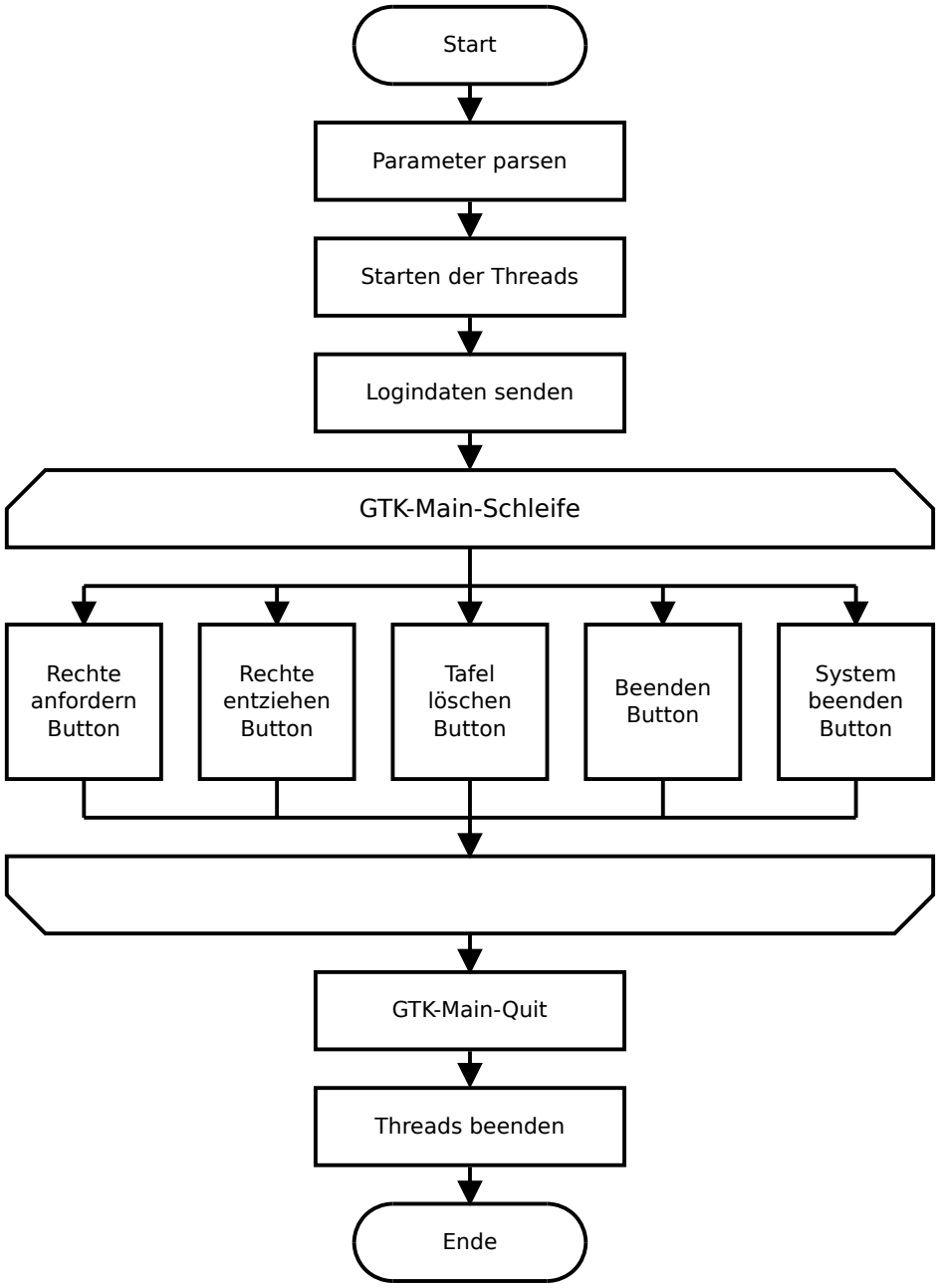

Abbildung 11: Ablauf Hauptprogramm

3.1.2 Command-Thread

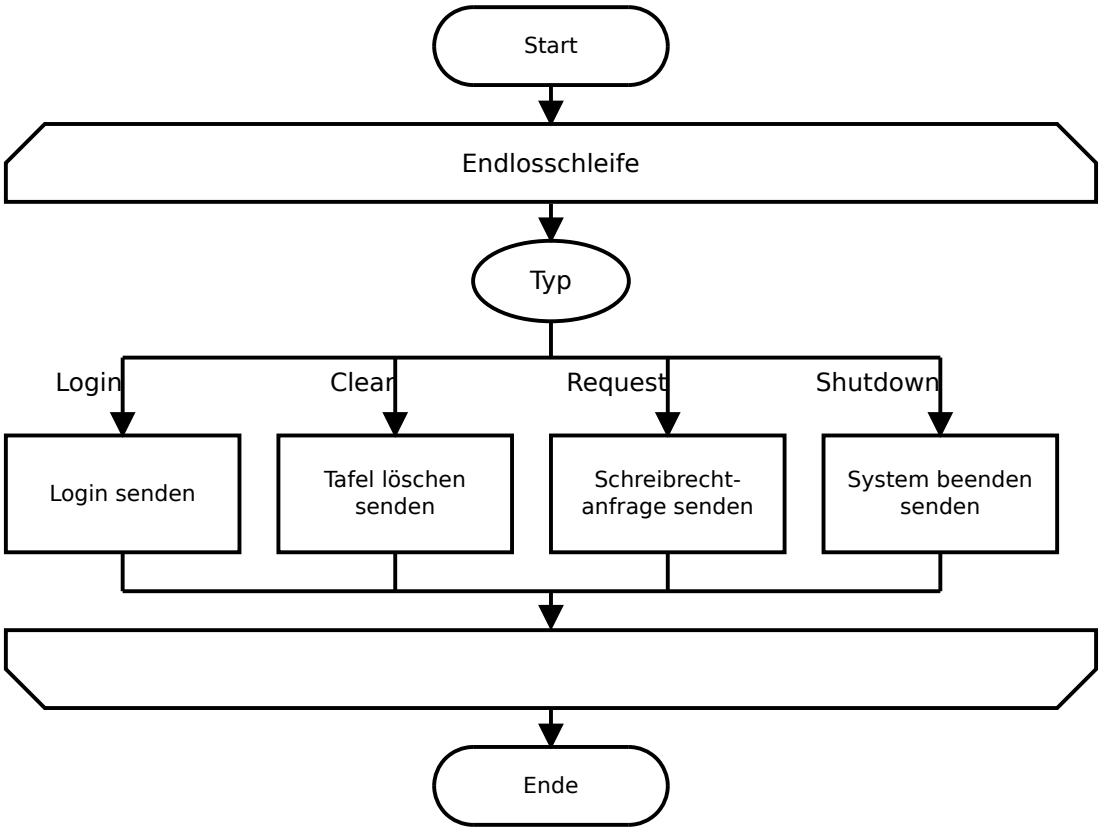

Abbildung 12: Ablauf Command-Thread

3.1.3 Live-Agent

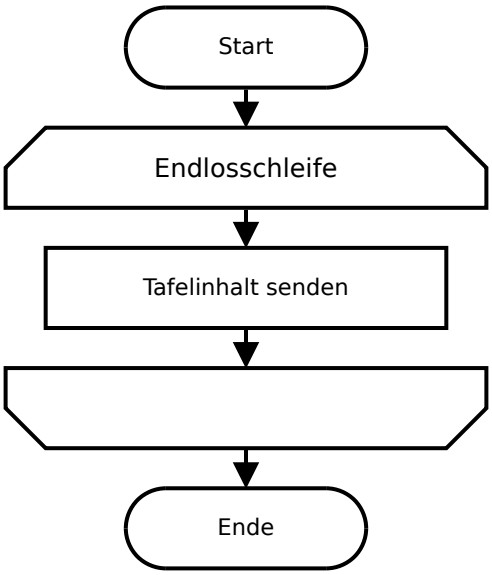

Abbildung 13: Ablauf Live-Agent

3.1.4 Listener-Thread

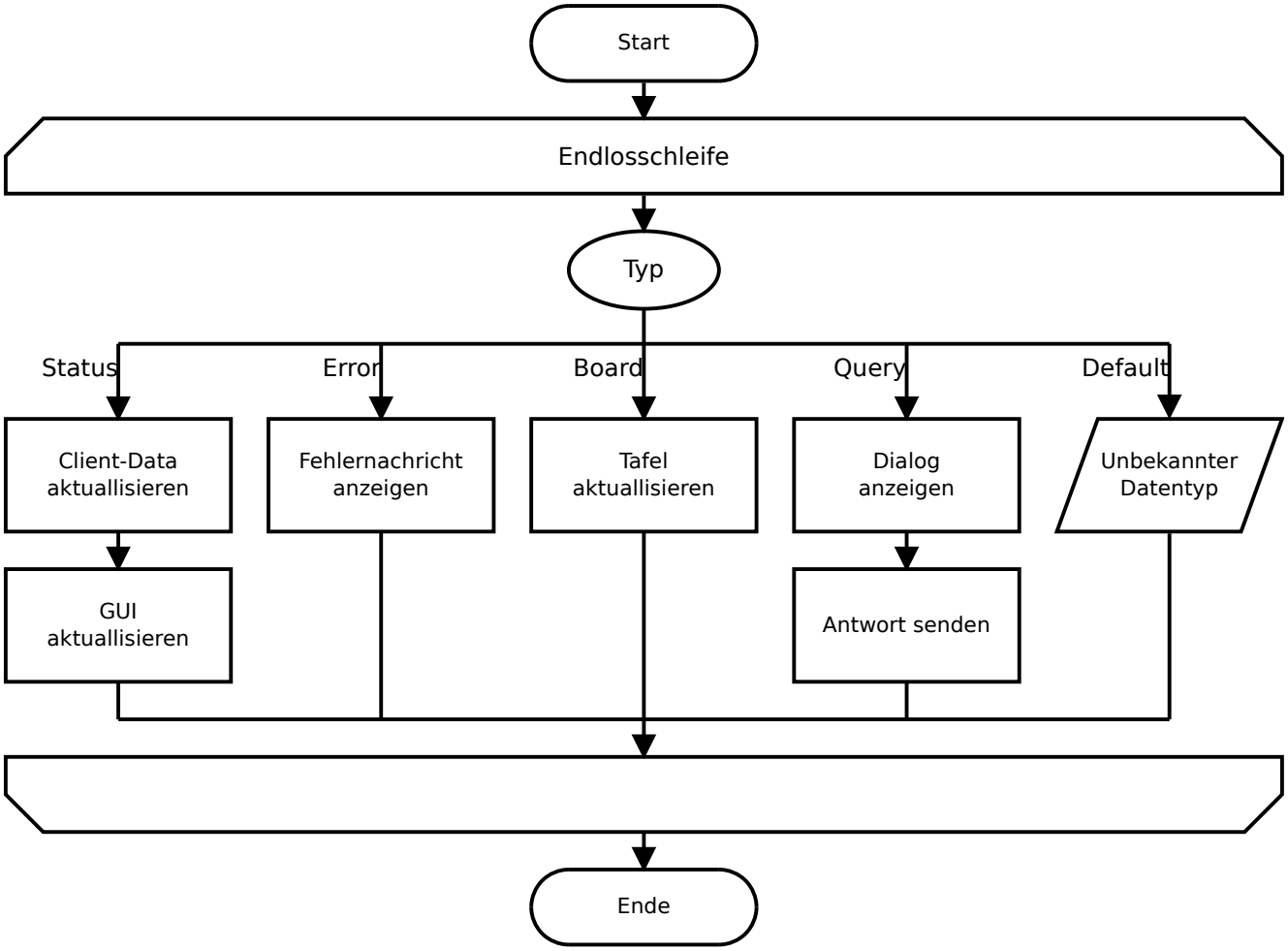

Abbildung 14: Ablauf Listener-Thread

3.2 Funktionshierarchie

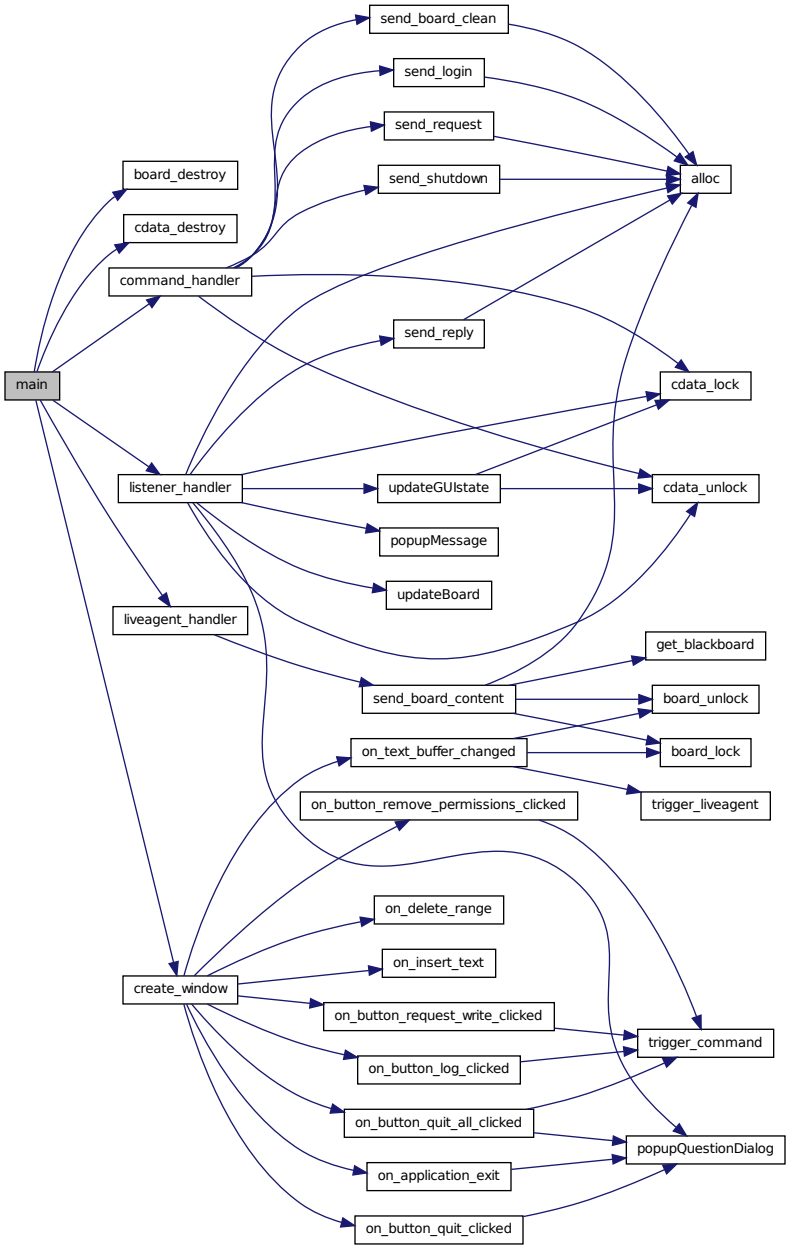

Abbildung 15: Funktionshierarchie Client

3.3 Modulhierarchie

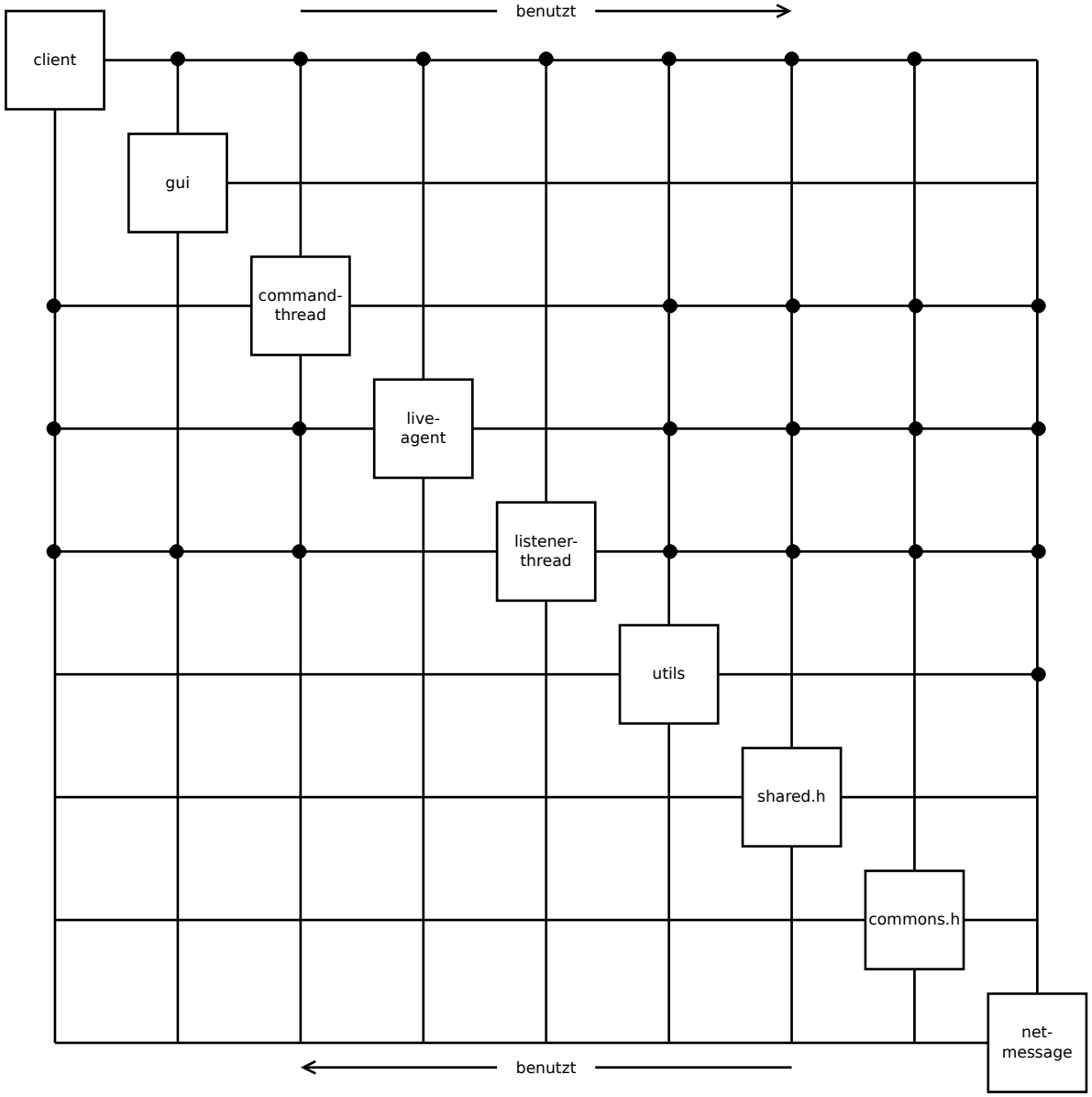

Abbildung 16: Modulhierarchie Client

3.4 Quellcode

Der Quellcode ist auf der CD zu finden.

4 Server

4.1 Programmabläufe

4.1.1 Hauptprogramm

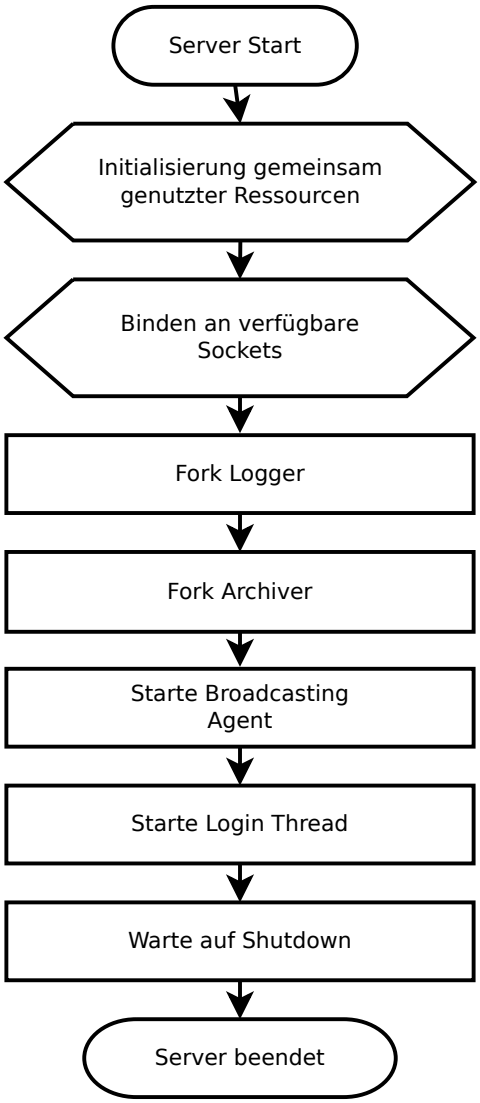

Abbildung 17: Ablauf Hauptprogramm

4.1.2 Login-Thread

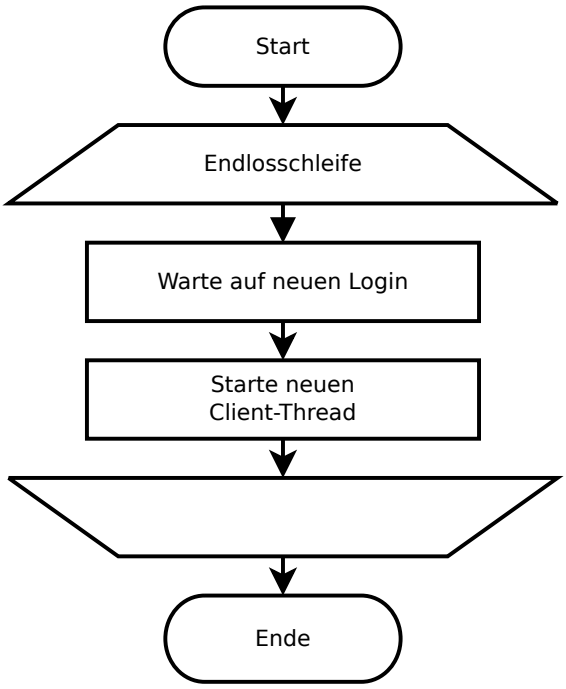

Abbildung 18: Ablauf Login-Thread

4.1.3 Client-Thread

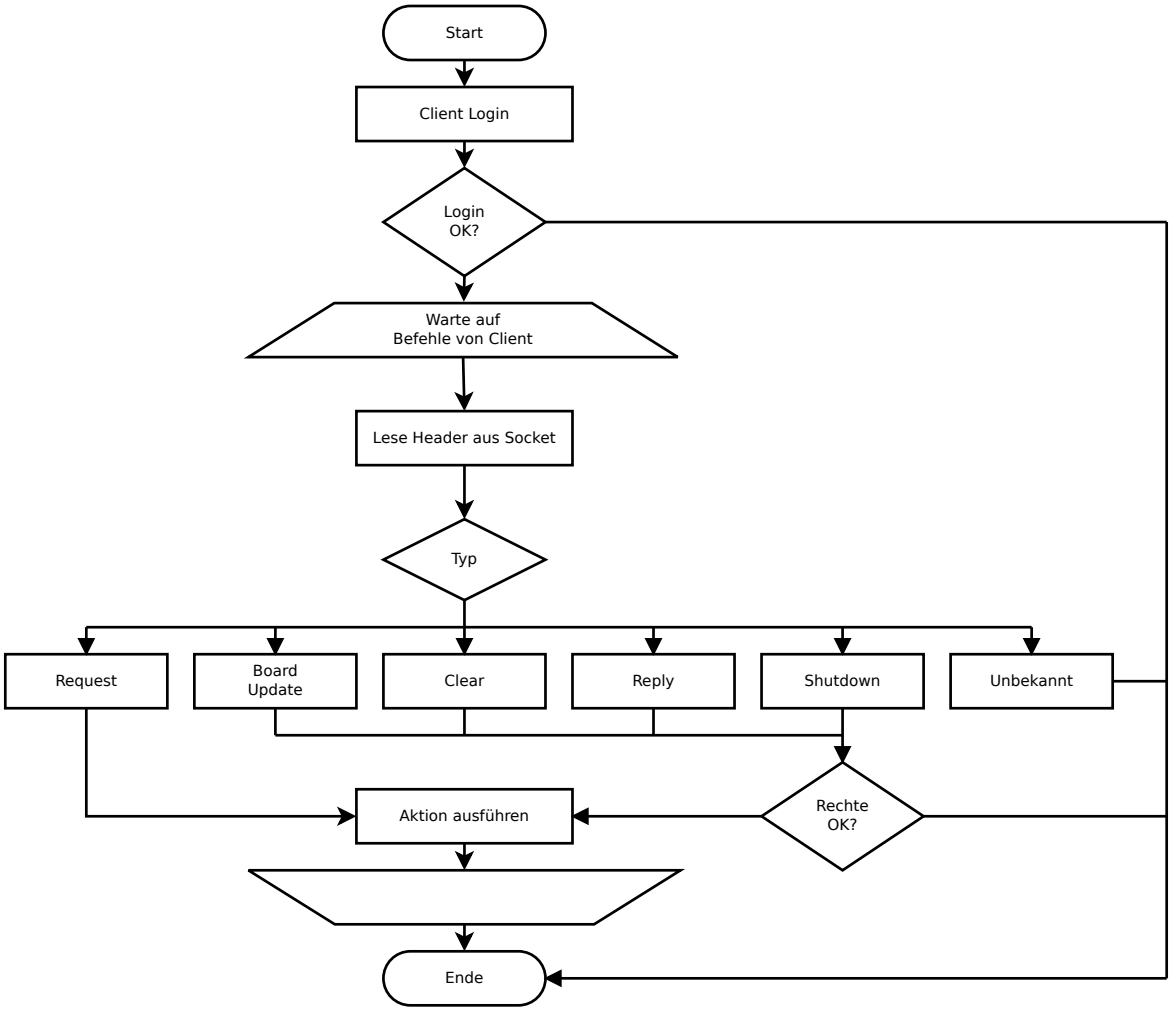

Abbildung 19: Ablauf Client-Thread

4.1.4 Broadcasting-Agent

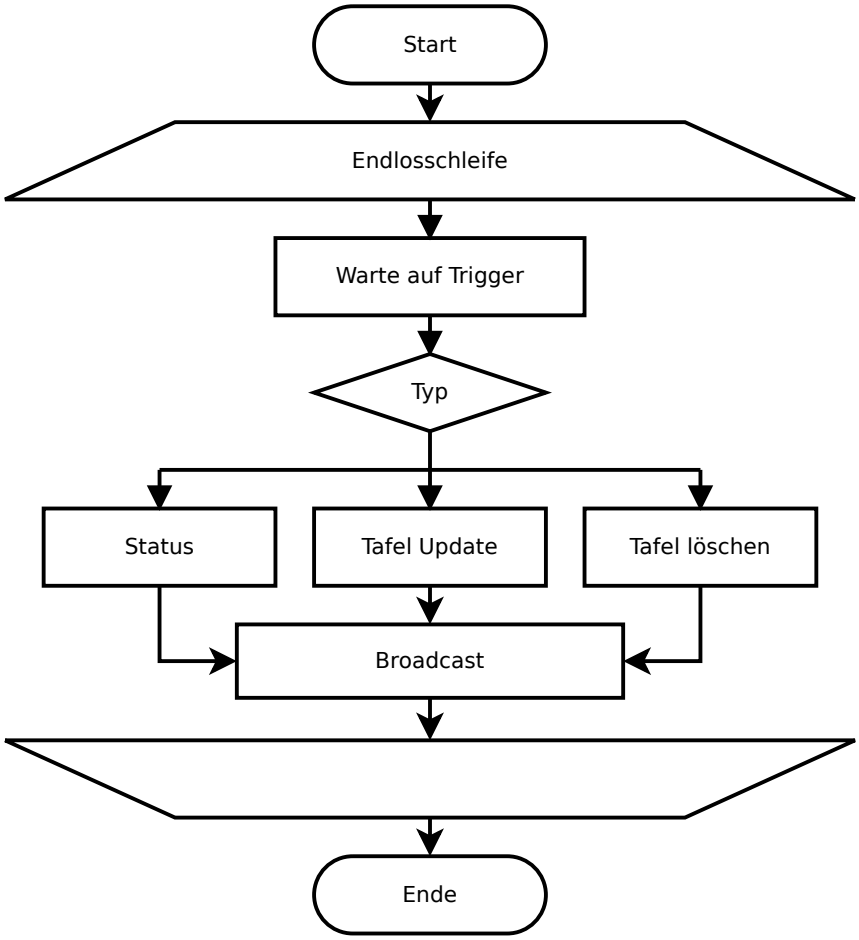

Abbildung 20: Ablauf Broadcasting-Agent

4.2 Funktionshierarchie

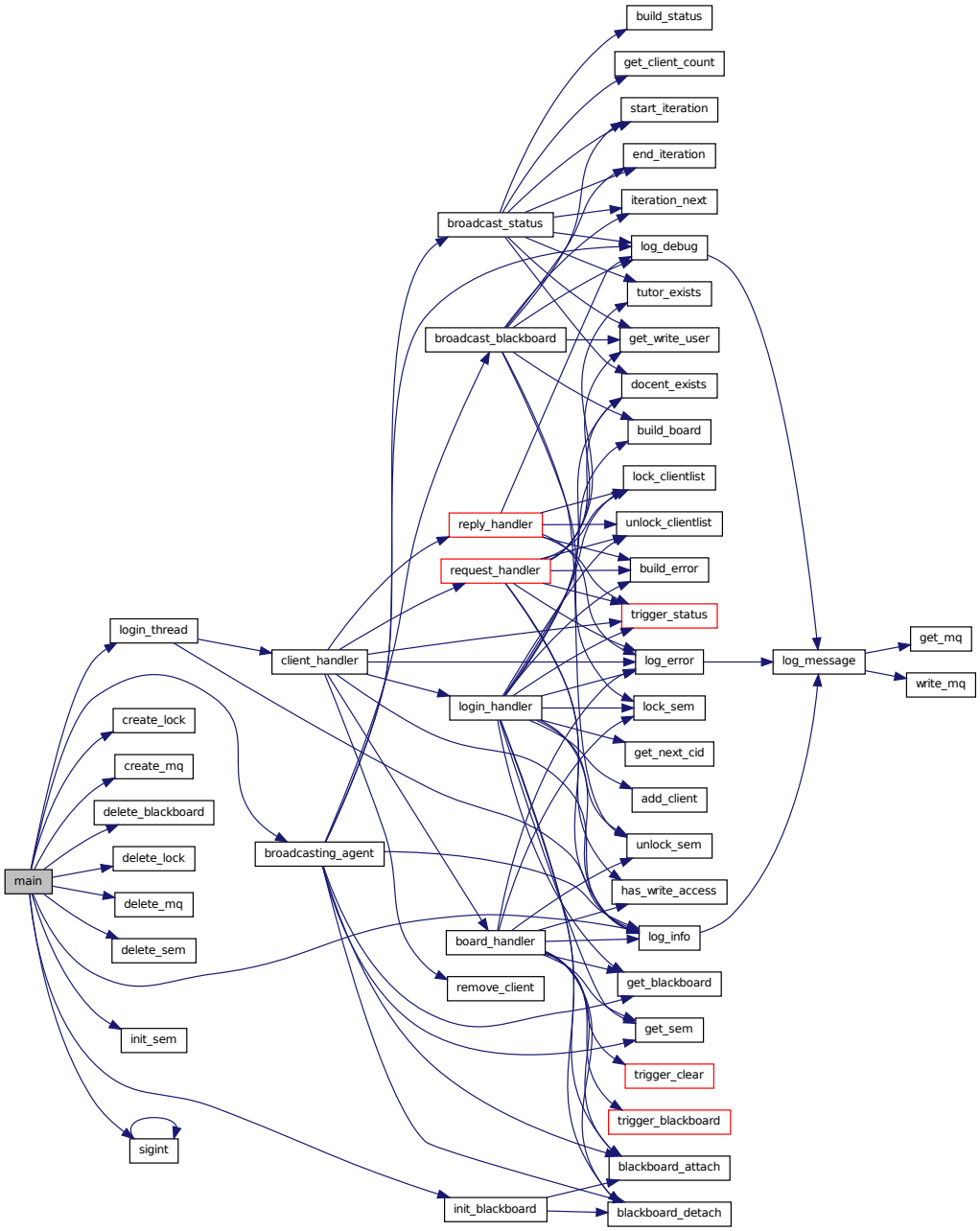

Abbildung 21: Funktionshierarchie Server

4.3 Modulhierarchie

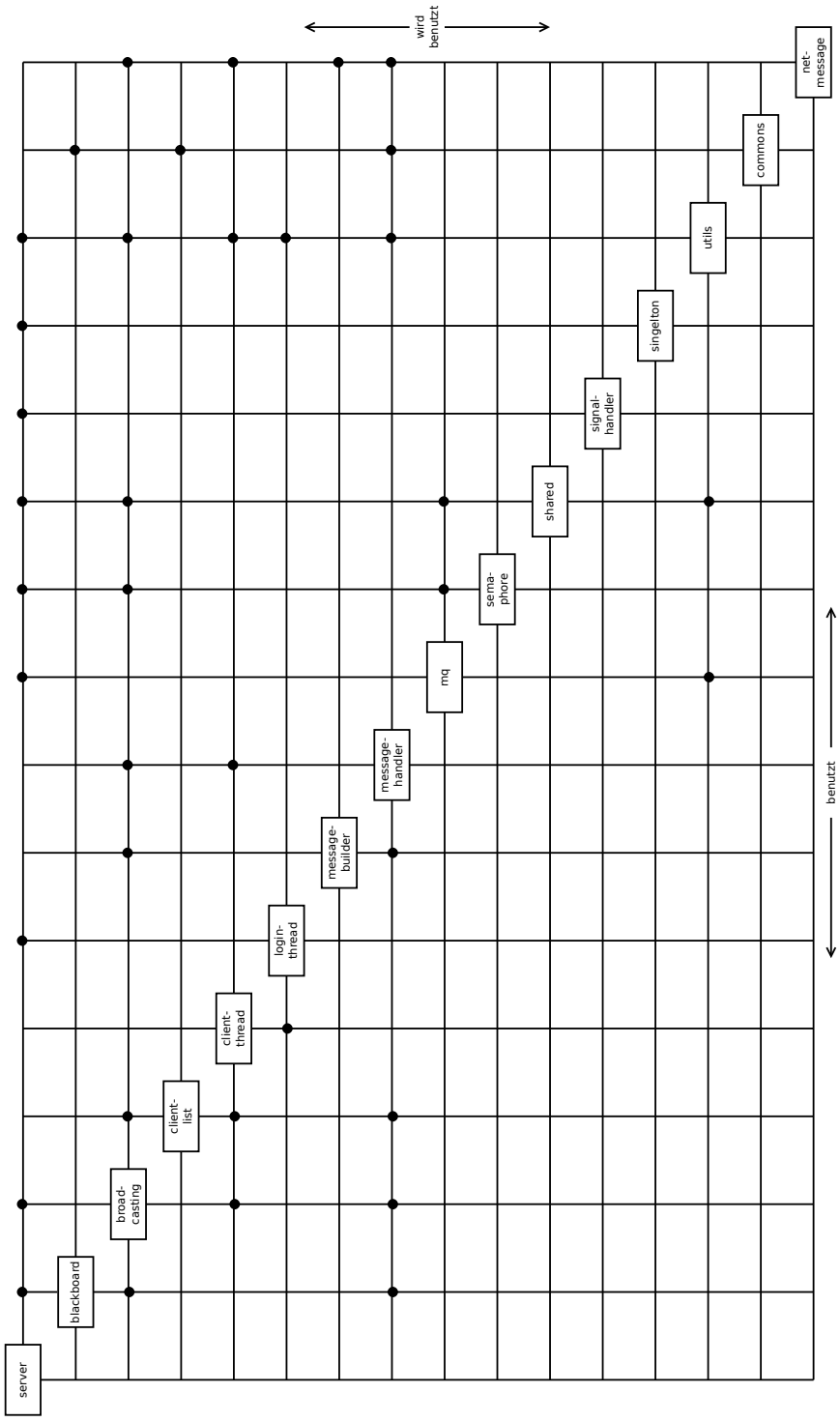

Abbildung 22: Modulhierarchie Server

4.4 Quellcode

Der Quellcode ist auf der CD zu finden.

5 Logger

5.1 Programmablauf

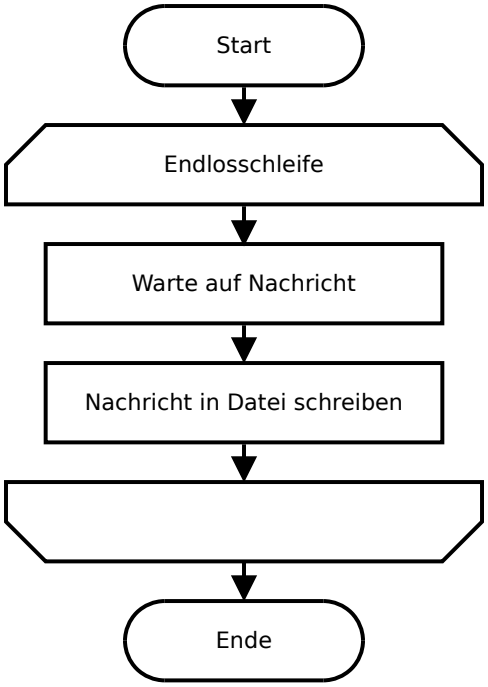

Abbildung 23: Programmablauf Logger

5.2 Funktionshierarchie

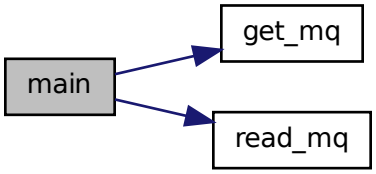

Abbildung 24: Funktionshierarchie Logger

5.3 Modulhierarchie

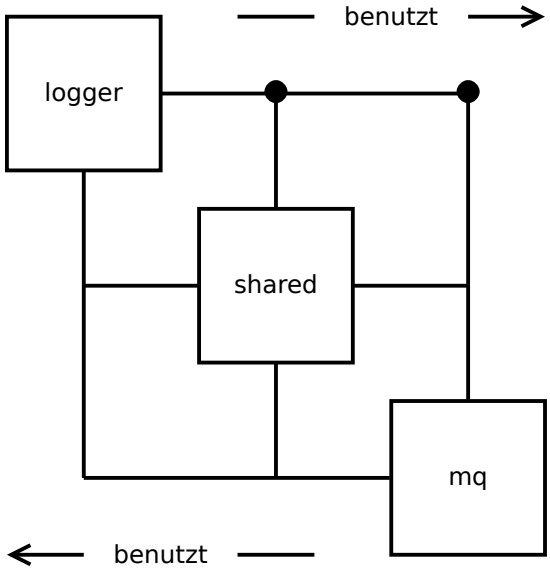

Abbildung 25: Modulhierarchie Logger

5.4 Quellcode

Der Quellcode ist auf der CD zu finden.

6 Archivierer

6.1 Programmablauf

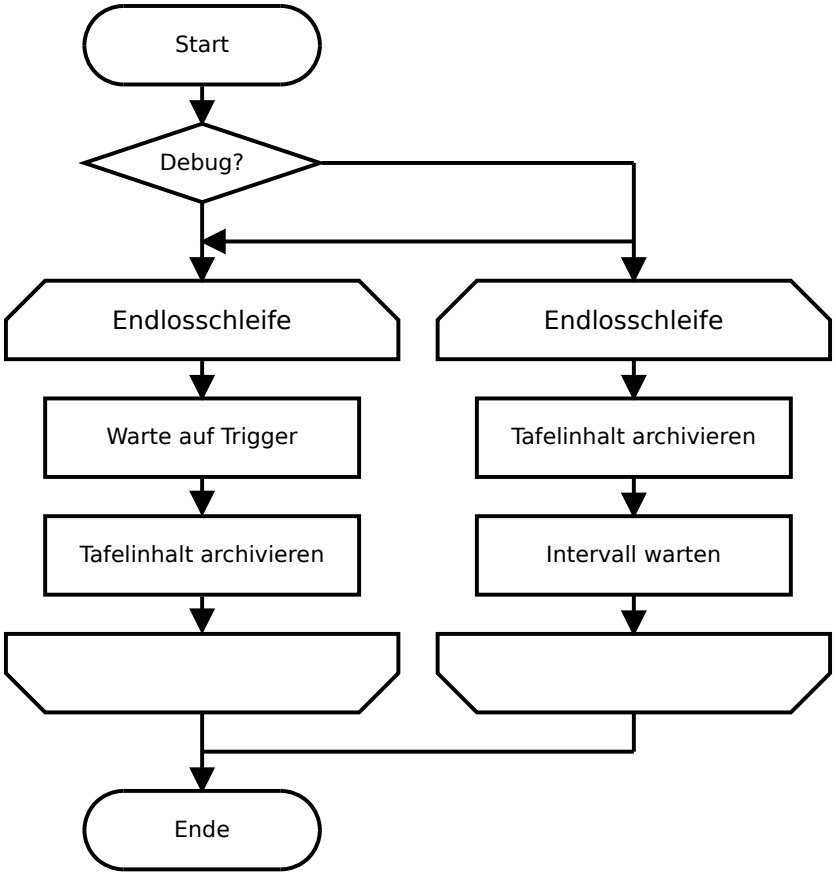

Abbildung 26: Programmablauf Archivierer

6.2 Funktionshierarchie

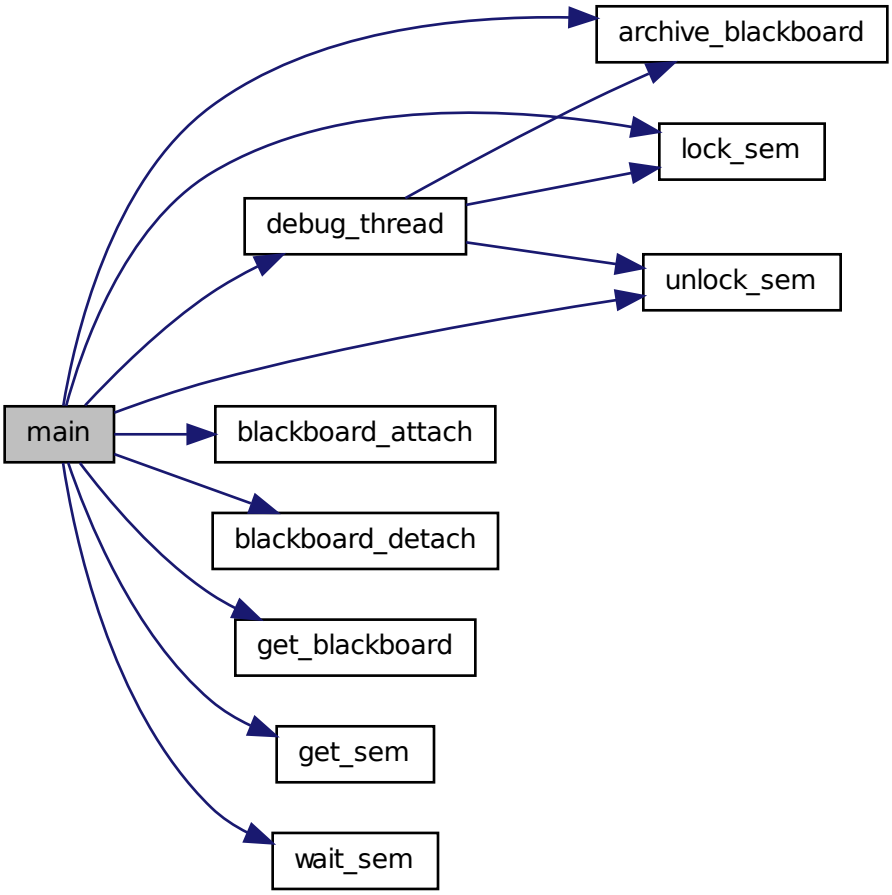

Abbildung 27: Funktionshierarchie Archivierer

6.3 Modulhierarchie

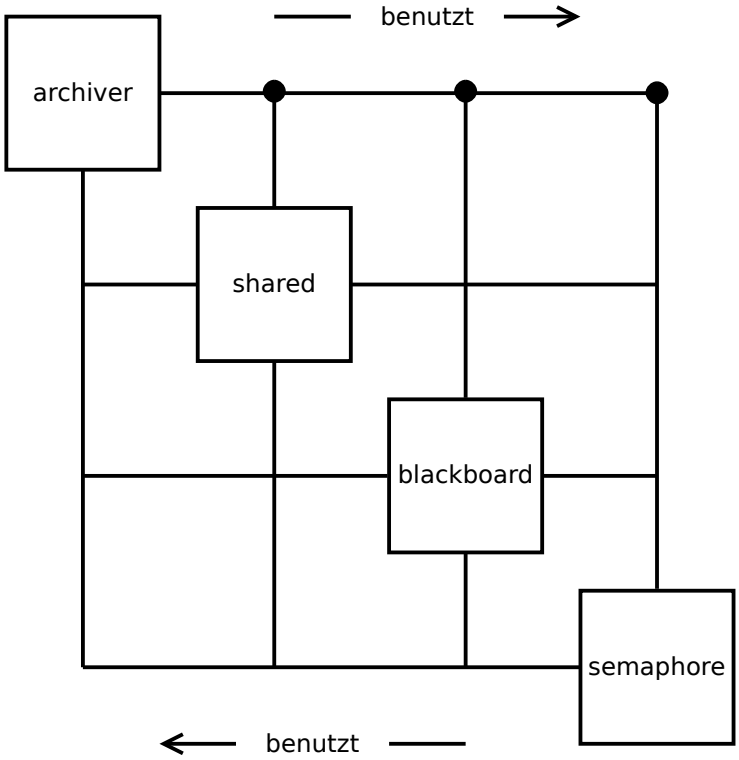

Abbildung 28: Modulhierarchie Archivierer

6.4 Quellcode

Der Quellcode ist auf der CD zu finden.

7 Client-Liste

7.1 Datenstrukturen

Die Daten von jedem Client wird in der Struktur `client_data` gespeichert. Diese einzelnen Datenstrukturen werden dann in einer doppelt verketteten Liste gespeichert.

```
struct client_data {
  uint16_t cid;
  int sfd;
  uint8_t role;
  char name[];
}
```

```
struct cl_entry {
  struct client_data *cdata;
  struct cl_entry *next;
  struct cl_entry *previous;
}
```

7.2 Exportierte Funktionen

```
void lock_clientlist ()
void unlock_clientlist ()
void add_client (struct client_data *cdata)
int remove_client (int sfd)
int docent_exists ()
int tutor_exists ()
struct cl_entry *get_write_user ()
void set_write_user (struct cl_entry *user)
struct cl_entry *get_docent ()
struct cl_entry *get_user_sfd (int sfd)
struct cl_entry *get_user_cid (uint16_t cid)
int has_write_access (int sfd)
int is_docent (int sfd)
uint16_t get_next_cid (void)
int get_client_count (void)
struct cl_entry *start_iteration ()
struct cl_entry *iteration_next ()
void end_iteration ()
```

8 Tafel

8.1 Datenstrukturen

Die Tafel wird als Char-Buffer mit 1200 Zeichen im Shared-Memory angelegt.

8.2 Exportierte Funktionen

```
1 int get_blackboard (key_t key)
2 char *blackboard_attach (int shmid)
3 void blackboard_detach (char *bboard)
4 int init_blackboard (key_t key)
5 void delete_blackboard (int shmid)
```

9 Zusammenfassung und Fazit

Das Praktikum 'Systemprogrammierung' hat uns einen tieferen Einblick in die systemnahme Programmierung von Client-Server-Anwendungen gegeben.
Beim ersten Systementwurf war uns (und vielen anderen) nicht genau klar, was gewünscht war und vor allem wie man das Problem angehen sollte. Eine Verknüpfung mit der Vorlesung 'Software-Engineering' wäre hier wünschenswert und hilfreicht gewesen. Desweiteren ist das Verhältnis von Arbeitsaufwand zu Credits um ein vielfaches Höher als in anderen Praktikas bzw. Vorlesungen.
Die selbstständige Problemlösung war ein wichtiger Bestandteil aus dem wir viel lernen konnten. Außerdem war das Praktikum eine gute Möglichkeit, die theoretischen Kenntnisse aus der Vorlesung 'Betriebssysteme' praktisch anzuwenden.

A Petrinetz

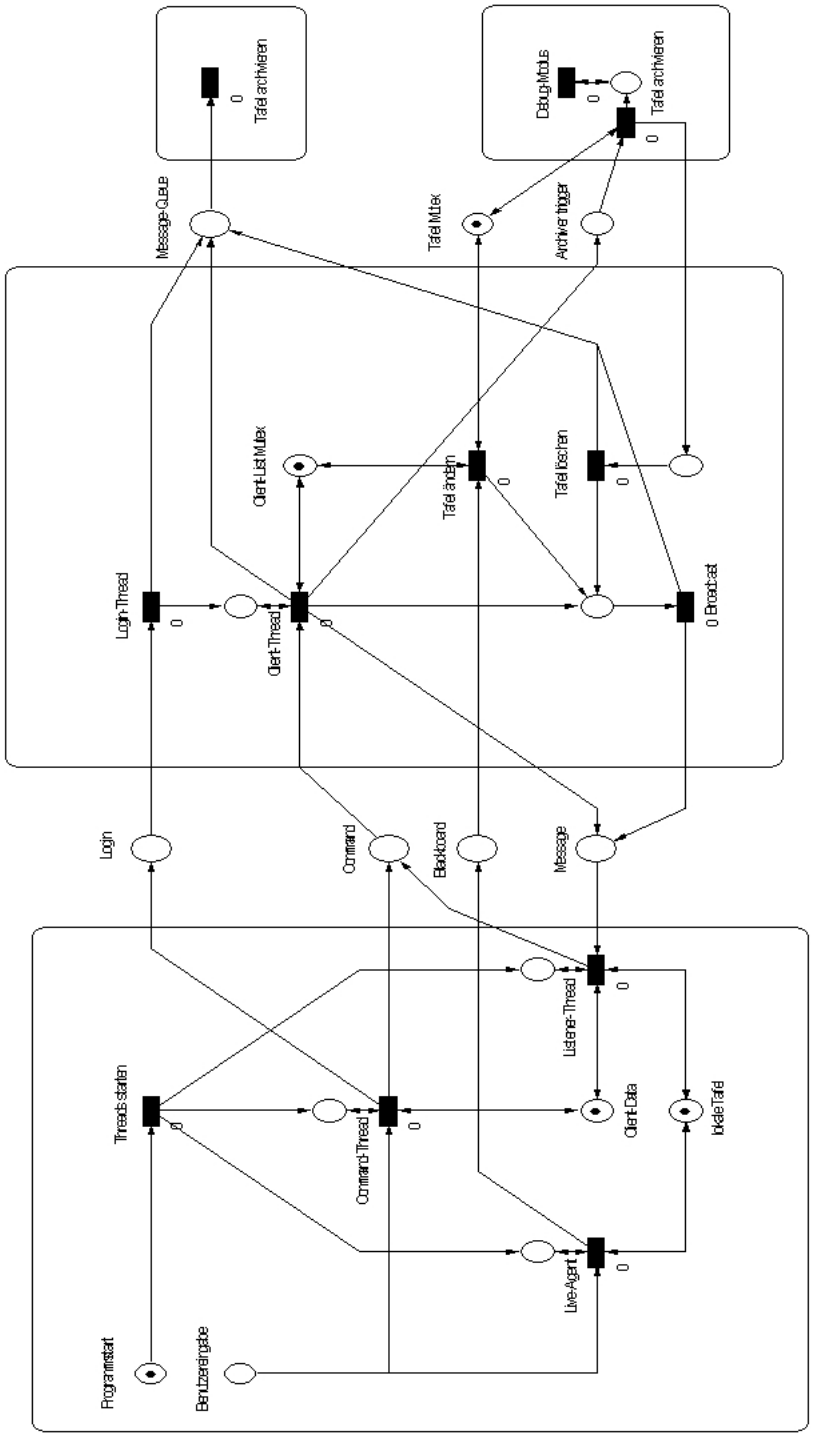

Abbildung 29: Petrinetz