Distributed Systems - Project Communication

Marcel Marti
09-929-019
marmarti@student.ethz.ch

Thomas Meier
09-928-995
thomeier@student.ethz.ch

ABSTRACT
Ziel des Kommunikationslayers ist es, das Versenden von Daten zwischen Server und Client so zu abstrahieren, dass sich layer, die auf dem Kommunikationslayer aufbauen nicht mehr um die Details des Sendens und Emfpangens kümmern müssen und nichts von der Implementierung wissen müssen oder zu sehen bekommen.

Um das zu erreichen definieren wir vorab einige Protokolle, die innerhalb der Projektgruppe noch einmal zur Absprache kommen sollten. Danach definieren wir ein erstes Interface mit welchem die verschiedenen Gruppen arbeiten sollten bis wir eine Implementierung liefern können.

Registrierungsprotokoll

Wir hatten innerhalb der Projektgruppe bereits besprochen, dass sich Benutzer mit einem Namen, jedoch ohne ein Passwort anmelden. Um sicher zu gehen, dass die Registrierung funktioniert muss ein Protokoll her. Da wir uns darauf geeinigt haben mit dem TCP zu arbeiten, müssen wir uns nicht um Nachrichtenverlust kümmern, sondern können davon ausgehen dass Nachrichten ankommen solange Sender und Empfänger verfügbar sind.

Wir nehmen an, dass nach einem Verfügbarkeitsverlust keine Nachrichten mehr gesendet werden können. Ansonsten sende eine Deregistrierungsnachricht und beende.

Client

1. Sende dem Server eine Registrierungsnachricht.
2. Warte auf Bestätigung.
3. *Beim Erhalten der Bestätigung gehe davon aus, dass der Client registriert ist.*
4. Falls nach einer Timeoutdauer keine Bestätigung eingegangen ist, starte den Vorgang von vorne und erhöhe den Timeoutzähler um eins.
5. *Falls der Timeoutzähler einen bestimmten Wert n erreicht hat, betrachte den Registrierungsvorgang als gescheitert und melde der Layerschnittstelle Verbindungsprobleme.*

Server

1. Empfange Registrierungsnachricht.
2. *Sende Bestätigung, falls Verarbeiten geglückt und betrachte Client fortan als registriert.*
3. Sende Fehlermeldung sonst.

Probleme mit dem Protokoll

Sollte die Verbindung nachdem der Server die Registrierungsnachricht erhalten und verarbeitet hat nicht mehr verfügbar sein, wird der Client nach seinem Timeout versuchen eine neue Registrierungsnachricht zu schicken. Der Server kann diese Nachricht nun aber nicht von der Registrierungsnachricht eines neuen Clienten unterscheiden Register und muss dem Client eine Fehlermeldung senden.

Lösungen für das Problem

- Eindeutige Identität mit dem Namen mitsenden.
- Am Protokoll nichts ändern sondern Zombieclients vom Server handhaben lassen. Siehe dazu in Deregistrierungsprotokollen.

Allgemeine Nachrichtenprotokolle

Allgemeines Senden von Informationen

Informationen allgemein werden in fire-and-forget Manier gesendet. Inkonsistenzen durch wider erwarten verloren gegange Nachrichten werden durch den Server oder Clienten gehandhabt und nicht von der Kommunikation.

Inkonsistenzen

Inkonsistenzen im Spielzustand werden nicht vom Kommunikationslayer gehandhabt, sondern von der Server- und Clientlogik. Ansätze dazu sollten in der Gruppe diskutiert werden.

Mein Vorschlag wäre es, jedem Zug der in einem Sudokuspiel ausgeführt wird eine Nummer einer monoton steigenden Folge zu geben. Bei jeder Nachricht wird der letzte Zug den der entsprechende Teilnehmer erhalten hat als Nummer mitgesendet. Sollte der Server nun entdecken, dass die Nummer nicht mehr mit dem Serverstatus übereinstimmt werden alle fehlenden Züge erneut gesendet. Der Client dagegen verwirft alle Züge, die seine Zugnummer nicht direkt fortsetzen.

Deregistrierungsprotokoll

Deregistrierung

1. Empfange Deregistrierungsnachricht.
2. Teile der Layerschnittstelle den Fehler mit.
3. *Betrachte den Teilnehmer als nicht registriert. Schliesse die Verbindung.*

Abmeldung

1. Sende Deregistrierungsnachricht an den Zielclienten.

2. *Betrachte den Teilnemer als nicht registriert. Schliesse die Verbindung.*

Client: Zombieprotokoll

Dieses Protokoll kümmert sich um den Fall, dass der Server keine Verbindung mehr zum Client hat und dies vor dem Unterbruch nicht mitteilen konnte.

1. Führe für den Server ein Timeout mit.

2. Wird das Timeout erreicht bevor eine Nachricht vom Server eingegangen ist, betrachte den Server als tot und leite eine Deregistrierung ein.

3. Leite den Fehler and die Layerschnittstelle weiter.

4. Wird eine Nachricht vom Server empfangen, setze das Tiemout zurück und reagiere auf die Nachricht.

Register

Server: Zombieprotokoll

Dieses Protokoll kümmert sich um Clients die keine Verbindung mehr haben und keine Chance hatten dies vor dem Unterbruch mitzuteilen.

1. Für jeden registrierten Client wird ein Timeout mitgeführt.

2. Sollte dieses Timeout erreicht werden, sende dem Client eine Nachricht die eine Bestätigung verlangt.

3. Falls die Bestätigung empfangen wird, setze das Timeout für den Teilnehmer zurück.

4. Sollte nach einem weiteren Timeout keine Bestätigung erhalten worden sein, leite eine Deregistrierung ein.

5. Entferne den Clienten.

Crash

Bei einem Crash wird eine Deregistrierungsnachricht gesendet und beendet. Sollte die Deregistrierungsnachricht nicht ankommen, wird der nun tote Client vom Zombieprotokoll gehandhabt.

Auswirkungen des Zombieprotokolls

Sollte ein Client sich registriert haben, davon aber nichts wissen, sorgt das Zombieprotokoll dafür, dass der fehlerhaft registrierte Name nach einer gewissen Zeit (Der Timeoutzeit) wieder freigegeben wird. Das bedeutet insbesondere, dass bei einem fehlerhaften Loginversuch der Name des einzuloggenden für die Timeoutdauer nicht verfügbar ist und dem entsprechenden Benutzer auch als soches angezeigt wird. ***Das könnte insbesondere bei mehreren Fehlerhaften Anmeldungen äusserst frustrierend für den Betroffenen sein.***

NACHRICHTENFORMAT

Währned das Nachrichtenformat für reservierte Nachrichten, insbesondere jene der oben genannten Protokolle, fix ist und keinen Spielraum zulässt, können für Nachrichten, die Spielinformationen übermitteln sollen, beliebige Formate definiert werden. Die einzige Grundvoraussetzung die allen Nachrichtenformaten zu Grunde liegt ist, dass das erste Feld immer den Nachrichtentyp bestimmt. Zudem soll in Absprache mit der Projektgruppe eine Nachrichtenhierarchie eingeführt werden, die ähnlich zu einer Klassenhierarchie gemeinsame Eigenschaften von Nachrichtenformaten in einem Basisformat vereint.

Noch nicht fixierte Formate

- **Registrierungsnachricht:**
  {"MessageType":"Register", "Name":*NAME*}
- **Deregistrierungsnachricht *Unfixed*:**
  {"MessageType":"Deregister", "Name":*NAME*}
- **Bestätigung:** {"MessageType":"ACK"}
- **Verneinung:** {"MessageType":"NACK"}
- **Fehlermeldung:**
  {"MessageType":"Error", "ErrorType":*TYPE*, "Msg":*MESSAGE*}
- **Bestätigungsaufforderung:**
  {"MessageType":"Request", "RequestType":"ACK"}

Diese Formate sind insofern noch nicht fixiert, als dass zusätzliche Einträge hinzukommen könnten wie sich nach der Besprechung mit der Projektgruppe noch herausstellen wird.

Basisformat

Das Basisformat enthält wie schon beschrieben den Nachrichtentypen. Allgemein wird eine Typenhierarchie innerhalb einer Nachricht an erster Stelle aufgebaut:

{"MessageType":*TYPE*, "NextType":*TYPE*, "NextType":*TYPE*, ..., *OTHER-INFORMATION*}

Weitere Formate

Weitere Formate werden nach Absprache mit der Projektgruppe hinzugefügt und an die Anforderungen innerhalb des Teams angepasst.

API

Sowohl für den Server, wie auch für den Clienten werden Interfaces bereitsgestellt, an denen wir nach dem Release nichts mehr verändern werden. Zudem wird für jeden Nachrichtentyp eine Klasse erstellt sowie eine zusätzliche Klasse für jeden Fehlertyp. Die genauen Nachrichtenklassen werden sich in der nächstens Projektgruppenbesprechung ergeben.