

THE – NETWORK – SIMULATOR

A powerful experience

Projektkurs Mathe – Physik - Informatik
2016/17

Dokumentation von
Felix Stach

Inhaltsverzeichnis

1. Der Projektkurs

2. Vorarbeit

2.1 Osi-Schichtenmodell

2.2 UDP/ TCP

2.3 IP-Adressen/ IP-Routing

2.4 Sonstige Inhalte

3. Beschreibung

3.1 Ziel

3.2 Struktur

3.3 System

3.4 Klassen

4. Umsetzung (Software)

4.1 Eclipse (IDE)

4.2 Dropbox

4.3 GitHub

5. Arbeitsweise

6. Probleme

6.1 Motivation

6.2 Modell des Nichtbeschwerens

6.3 Klarheit/ Komplexität

6.4 Zeiteinteilung

6.5 Reflexion

7. Fazit

8. Literaturverzeichnis

1. Der Projektkurs

Diese Dokumentation beschäftigt sich mit dem Projektkurs 2016/17 zum Thema MPI – Mathe - Physik - Informatik. Für diesen habe ich mich mit Katja L. Zu einem Team zusammengeschlossen, in welchem wir uns gemeinsam an ein selbst gewähltes Thema begeben würden um ein Jahr lang unsere Idee umsetzen, und dann Ende Juni/ Anfang Juli ein angemessenes Endergebnis präsentieren können. Das ganze Projekt lief unter der Aufsicht von Herrn Bachran. Der Kurs lief letzten Endes gänzlich anders ab als erwartet, aber warum genau, wie es dazu gekommen ist und was ich alles daraus lernen und mitnehmen konnte, darum geht es später.

Ganz Anfang hat sich natürlich die Frage nach unserem Team gestellt. Wir haben die Möglichkeit bekommen in solchen zu arbeiten, und da ich eine solche Arbeitsweise mag lag die Entscheidung dazu nah. Es gab noch verschiedene andere infrage kommenden Mitschüler aus dem Kurs, aber da Katja und ich beide wussten, dass wir gerne etwas im Bereich Physik/ Informatik machen wollten haben wir uns ziemlich schnell dazu entschieden zusammenzuarbeiten.

Nun hat sich natürlich die Frage gestellt, was wir denn als Thema nehmen wollten. Dass es wohl ein Thema aus der Informatik sein würde war uns zwar klar, aber genauer hatten wir eigentlich noch keine Vorstellung davon. Bei dem ersten Treffen hatte ich, wie es gerne mache, erst einmal aus verschiedenen kleineren Ideen eine größere „zusammengebastelt“, welche mir auch sehr gefallen hat. Unerwarteterweise haben wir den Plan aus gewissermaßen unbekannten Gründen sehr schnell verworfen; Warum genau kann ich im Nachhinein gar nicht sagen. Auf jeden Fall haben Katja und ich uns in den darauffolgenden Stunden sehr viel mit Herr Bachran über mögliche Themen unterhalten, und schlussendlich sind wir dann bei unserem Thema, dem NETWORK SIMULATOR, also der Umsetzung einer Netzwerksimulation in Java, stehen geblieben.

Da weder Katja noch ich uns zuvor in irgendeiner Form mit dem Thema Netzwerke befasst hatten und die paar Einblicke, die uns Herr Bachran gegeben hat erst mal interessant aussahen, entschieden wir uns für genau dieses Thema. Für ein gesamtes Jahr würden wir uns also mit diesem Inhalt auseinandersetzen, das war zumindest der Plan. Doch bevor wir anfangen konnten mit der Planung und der darauffolgenden Implementation mussten wir uns zuvor noch ein paar wichtige Inhalte ansehen, die für unsere Arbeit wichtig waren.

2. Vorarbeit

Um unser Thema verstehen zu können und um eine Vorstellung zu bekommen womit wir überhaupt arbeiten haben wir uns am Anfang des Projektkurses, noch in der Phase in der wir uns unser Thema ausgesuchten, zuerst ein paar wichtige Themen angesehen, welche für unser Thema sehr hilfreich sein sollten. Darum geht es in diesem Abschnitt.

2.1 Osi-Schichtenmodell

Das Osi-Schichtenmodell (oder Osi-Modell „*Open Systems Interconnection Model*“) ist ein seit 1983 bei der International Telecommunication Union (ITU) eingeführter Standard, welcher die Kommunikation über verschiedene technische Ebenen ermöglicht und diese begünstigt. Alle Aufgaben einer Schicht sind eng begrenzt.

Um uns zu überlegen auf welcher Ebene unser Simulator fungieren sollte mussten wir uns zunächst alle Ebenen ansehen, die uns zur Verfügung standen. Dazu verwendeten wir dieses Modell.

Schicht 1	Bitübertragungsschicht (Physical Layer)
Schicht 2	Sicherungsschicht (Data Link Layer)
Schicht 3	Vermittlungsschicht (Network Layer)
Schicht 4	Transportschicht (Transport Layer)
Schicht 5	Sitzungsschicht (Session Layer)
Schicht 6	Darstellungsschicht (Presentation Layer)
Schicht 7	Anwendungsschicht (Application Layer)

Wir haben uns grob damit beschäftigt welche Aspekte die jeweilige Schichten ausmachen und sind dann zu dem Schluss gekommen, dass wir uns auf keine spezielle Schicht einigen sollten um uns die Umsetzung verständlicher und einfacher zu machen. Da dieses Modell für unsere späteren Ergebnisse nicht mehr wichtig war haben wir uns auch nicht weiter damit beschäftigt, weshalb hier keine weiteren Erklärungen folgen.

2.2 UDP/ TCP

Das UDP („User Datagram Protocol“) ist ein Protokoll, welches das Versenden von Datenpaketen in IP-basierten und verbindungslosen Netzwerken ermöglicht. Es wurde ab 1977 entwickelt.

Bei dem TCP („Transmission Control Protocol“) handelt es sich um eines, welches die Austauschmöglichkeiten von Datenpaketen von mehreren Komponenten in einem Netzwerk beschreibt. Es unterscheidet sich zum UDP in der Form, dass es Datenaustausch in beide Richtungen zwischen zwei Komponenten zulässt. Dieses wurde ab 1973 entwickelt

Hier handelt es sich um zwei weitere Begriffe, die für uns theoretisch wichtig gewesen wären, hätten wir unseren ursprünglichen Plan noch erweitert. Da wir uns aber stets auf einer sehr minimalistischen Ebene aufgehalten haben sind diese Begriffe und deren Funktionen so nicht weiter für uns interessant geblieben.

2.3 IP-Adressen/ IP-Routing

Eine IP-Adresse ist ein digitaler Code, welcher Komponenten in einem Netzwerk adressierbar macht und auf dem Internetprotokoll (IP) basiert. Heutzutage sind IPv4-Adressen gängig, welche aus vier Zahlen (von jeweils 0 bis 255) und jeweils Punkten zwischen diesen bestehen. IPv4-Adressen werden durch eine 32-Stellige Binärzahl beschrieben. Für eine weitere Form, die IPv6-Adressen ist eine Binärzahl mit 128 Stellen erforderlich.

Unter IP-Routing versteht sich der Vorgang, bei dem das Internetprotokoll (IP) die Route zum Zielkomponenten berechnet um Daten sowohl über physikalischer Verbindung als auch über Übertragungssysteme zu vermitteln.

Beides sind genau wie die beiden zuvor sehr große, ausgedehnte und komplexe Themengebiete über welche es viel zu Dokumentieren und erklären gibt, aber da auch diese beiden Themen für uns am Ende überhaupt keine Wichtigkeit hatten haben wir uns auch damit nicht weiter beschäftigt.

2.4 Sonstige Inhalte

Sonst gibt es nicht mehr viel hinzuzufügen. Wir besprachen noch die Begriffe „Encapsulation“, also die Einbettung von Datenteilen, und die Beziehungen „Aggregation“ („is-part-of“-Beziehung) und die ähnliche „Komposition“.

3. Beschreibung

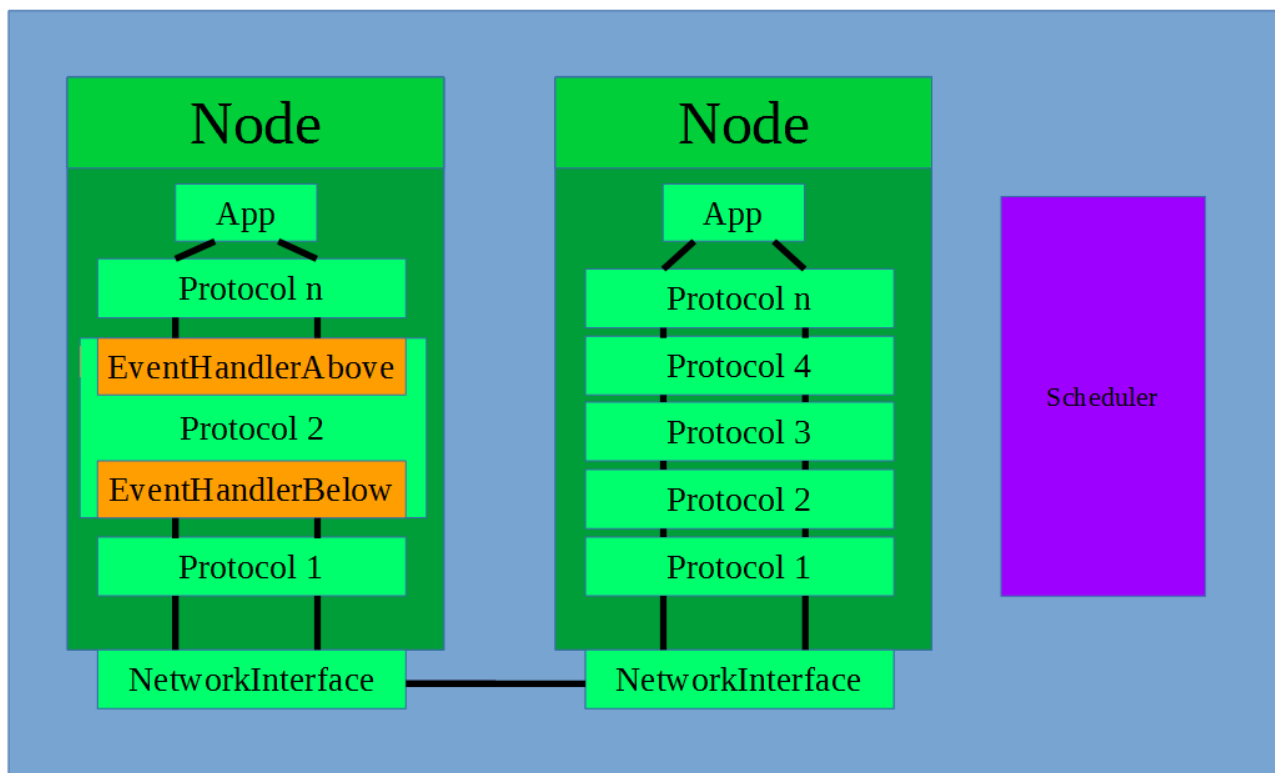
In der ersten Zeit des Projektkurses haben wir uns nicht nur überlegt was der Simulator können sollte, sondern auch wie wir das ganze modellieren wollen. Im Folgenden wird nun also beschrieben wie unser Projekt aussehen sollte, welche Ziele wir verfolgten, welche Strukturen wir uns überlegt haben und welche Klassen dazu nötig waren.

3.1 Ziel

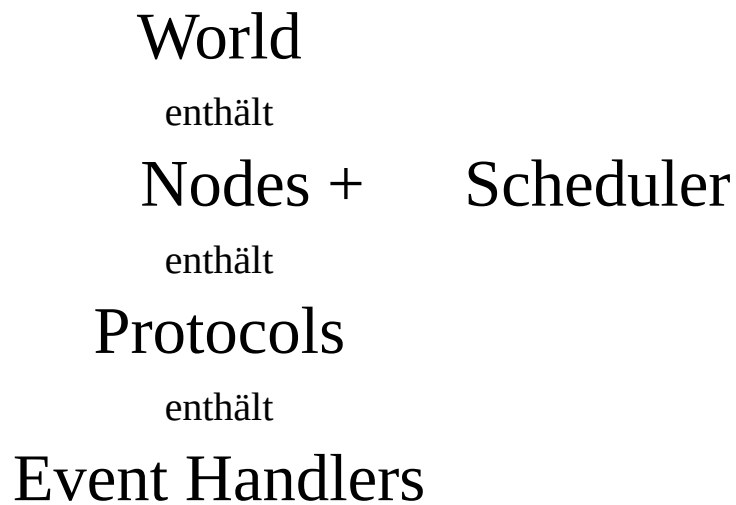
Das Ziel unseres Projektes war es am Ende ein funktionierendes Konstrukt an Klassen und Code zu entwickeln, das es ermöglichen sollte ein Datenpaket mittels einem sogenannten „Event“ von einem Medium (= Node) zu einem Anderen zu schicken. Da wir am Anfang noch nicht wirklich wussten wo uns das Projekt hinbringen würde war das allerdings nur der vorläufige Plan, welcher sich allerdings später auch nicht mehr geändert hat.

3.2 Struktur

In dieser Abbildung sind die wichtigsten Klassen und deren Beziehung untereinander dargestellt. Dazu kommen noch einige später erläuterte Klassen.



Veranschaulichung:



3.3 System

Das System sollte wie folgt funktionieren:

Alle Objekte befinden sich in einer Welt („World“). In diese Klasse befindet sich ebenfalls die Main-Methoden, mit der die Prozesse gestartet werden (Elemente erzeugen und Befehle absetzen). In der Welt befinden sich Knoten („Node“), welche Medien darstellen und alle von der Welt erfasst sind. Diese Knoten speichern eine Liste verschiedener Typen von Protokollen („Protocol“), welche alle hintereinander gelistet sind. Zu diesen gehören das einfache Protokoll („SimpleProtocol“), die App („App“), und ein Netzwerk-Interface („NetworkInterface“). Jedes dieser Protokolle (mit Ausnahme der App) enthält zwei EventHandler („EventHandler“; „handlerAbove“, „handlerBelow“), von denen jeder eine Methode zum erhalten (RECEIVE) und Senden (SEND) von Events („Event“) besitzt. Die EventHandler haben die Funktion diese weiterzuleiten. Durch einen SEND-Befehl wird jenes Event zum nächsten Protokoll weitergeleitet, ein RECEIVE-Befehl leitet das Event zum entgegengesetzten EventHandler weiter, bis dieses entweder bei einem Netzwerk-Interface oder einer App ankommt. In dem Event sind Zeitstempel („timeStamp“), der aktuelle EventHandler, der EventType („EventType“) und ein Datenpaket („DataPacket“) gespeichert. Eine Ausnahme der Protokolle stellt die App dar; die App kennt nur den unteren EventHandler („handlerBelow“). Im eigentlichen Sinne sollte die Liste der anderen bekannten Knoten entweder als Liste im Netzwerk-Interface gespeichert werden oder beim Knoten selbst. Für unser Beispiel-Szenario, dass ein Knoten ein Datenpaket zu einem Anderen schickt, wurde die App noch einmal aufgeteilt in „SenderApp“ und „ReceiverApp“. Außerdem kommen zu diesen Klassen noch verschiedene Typen von Schemulern hinzu, welche alle Events verwalten und sortieren.

Dazu gehören das Interface „Scheduler“, welches durch einen einfachen Scheduler „simpleScheduler“ implementiert wird sowie einen komplexeren weiteren Scheduler, den sogenannten „Minimum-Heap-Scheduler“. Im Folgenden Abschnitt wird jede Klasse näher erläutert.

3.4 Klassen

„World.java“

Die Klasse World organisiert das ganze System und enthält daher auch die „public static void main“. Außerdem sind auch alle vorhandenen Knoten in einem Vektor „Knotenliste“ gespeichert, ein Scheduler ist eingetragen und in der main-Methode werden alle Knoten und dazugehörige Protokolle erzeugt, die für die Simulation wichtig sind.

„Node.java“

Die Klasse Node speichert ihren Ort (Koordinaten x und y), die Reichweite des Signals („rangeOfSignal“) und hat außerdem das dazugehörige Netzwerk-Interface und eine Liste von Protokollen gespeichert. Beim Erzeugen werden beide Koordinaten übergeben und die Klasse besitzt Funktionen zum Hinzufügen eines Protokolls und zum Verbinden („Attaching“) eines weiteren Knoten.

„Protocol.java“

Die Klasse Protocol fungiert als abstrakte Oberklasse für spezifiziertere Protokolle.

Hier werden das obere und untere Protokoll angegeben, ebenso der obere Eventhandler und der untere Eventhandler.

„EventHandler.java“

Die Klasse EventHandler ist ein Interface, welches von Protokollen implementiert wird. In diesem wird nur die Methoden processEvent („Event“) vorgegeben.

„SimpleProtocol.java“

Die Klasse SimpleProtocol erbt von der Klasse Protocol. Sie stellt eine einfache Form eines Protokolls dar. In ihr existieren die Klassen „SimpleProtocolHandleAbove“ und „SimpleProtocolHandleBelow“ welche die Klasse EventHandler.java implementieren. Die beiden EventHandler können jeweils Events verarbeiten („Process Events“). Je nach Eventtyp können diese ein Event empfangen oder senden. Wird ein Event mit dem Typ SEND übergeben, dann wird dieses an das nächste Protokoll weitergeleitet. Hat es den Typ RECEIVE, so wird es an den anderen EventHandler im selben Protokoll übergeben.

„EventType.java“

Die Klasse EventType ist eine Enumeration und enthält die möglichen Event-Typen (SEND und RECEIVE), welche vom Event verwendet werden.

„Event.java“

Die Klasse Event enthält ausschließlich die Eigenschaften eines solchen. Dazu gehören der Zeitstempel „ts“ (in Sekunden), der aktuelle EventHandler „h“, der EventType „t“ (verwendet EventType), und das dazugehörige DataPacket „p“.

„DataPacket.java“

Die Klasse DataPacket speichert die Eigenschaften eines Datenpakets. Diese haben wir allerdings nie besprochen und daher auch nicht implementiert.

„NetworkInterface.java“

Die Klasse NetworkInterface erbt von der Klasse Protocol.java und stellt das verbindende Element zwischen einem Knoten (Node) und einem Anderen dar. Sie enthält dennoch zwei EventHandler, wobei der untere auch mit dem unteren EventHandler eines anderen Knoten verbunden ist.

„ReceiverApp.java“ / „SenderApp.java“

Die Klassen ReceiverApp und SenderApp erben beide von der Klasse Protocol.java und stellen die jeweils letzten Klassen in einer Node dar. In einem Beispiel-Aufbau der Welt enthält Node1 eine ReceiverApp als letztes Protokoll und Node2 eine SenderApp als letztes Protokoll. Diese Klassen kennen nur einen unteren EventHandler, da es nichts mehr nach oben weiterzuleiten gibt.

„Scheduler.java“

Die Klasse Scheduler ist ein Interface, welches die Methoden für die Implementierenden Klassen angibt. Dazu gehören processNext(), next(), schedule(Event e), cancel(Event e), und cancelAll().

„SimpleScheduler.java“

Die Klasse SimpleScheduler implementiert die Klasse Scheduler und stellt ein einfaches System für einen solchen dar.

„MinimumHeapScheduler.java“

Die Klasse MinimumHeapScheduler implementiert ebenfalls die Klasse Scheduler.java und stellt ein fortgefahrenes System für einen solchen dar.

4. Umsetzung

Wie schon aufgegriffen wussten wir am Anfang an vielen Stellen selber nicht, wie wir vorgehen sollten und womit wir arbeiten würden. Nachdem wir also nach einigen Überlegungen wussten wie wir wir ungefähr arbeiten wollten und welche Programme mit den dazugehörigen Funktionen uns zur Verfügung standen, haben wir beschlossen folgende Software zu verwenden:

4.1 Eclipse IDE



Eclipse IDE ist eine kostenlose plattformunabhängige „Open Source“ Programmierumgebung, welche wir in der Entwicklungsversion „Neon“ (4.6.3) zur Programmierung sämtlicher Klassen, Interfaces und weiteren Inhalten verwendet haben.

Dass wir mit Java arbeiten wollen, da waren wir uns ziemlich schnell einig, da diese eigentlich auch die einzige Programmiersprache ist in der wir uns einigermaßen auskannten und welche wir daher auch verwenden konnten. Nun ist es so, dass man auch bei nur einer Programmier-Sprache die Möglichkeit hat verschiedene Programmierumgebungen zu verwenden. Neben dem von uns letzten Endes verwendeten Eclipse Neon standen uns auch noch diverse andere Programmierumgebungen zur Verfügung. Zum Beispiel der klassische Java-Editor, welchen ich schon aus meinem Informatik Leistungs-Kurs kannte, die Software Greenfoot, welche für einfache grafische Darstellung gedacht war und BlueJ, welches eine sehr praktische Oberfläche zum Erstellen und Inspizieren von Objekten und Einsehen derer Beziehungen untereinander bietet.

Da uns Eclipse IDE aber am meisten Funktionen bereitgestellt hat, unter Anderem einen sehr ausgedehnten Debugger zum Finden und korrigieren von Problemen, haben wir uns für diese Plattform entschieden. Außerdem gibt es dort auch ein direkt eingebautes Plugin für eine unmittelbare Verbindung mit unserem GitHub-Verzeichnis, welches uns viel manuelle Arbeit erspart hat, da man über wenige Aktionen sofort die Dateien aus diesem Verzeichnis mit seinen eigenen Dateien ersetzen konnte. Genauso konnte man auch seine lokalen Dateien, falls diese nicht mehr aktuell waren, durch jene aus dem Verzeichnis ersetzen. Das war für uns ein sehr wichtiger Grund diese Plattform zu verwenden.

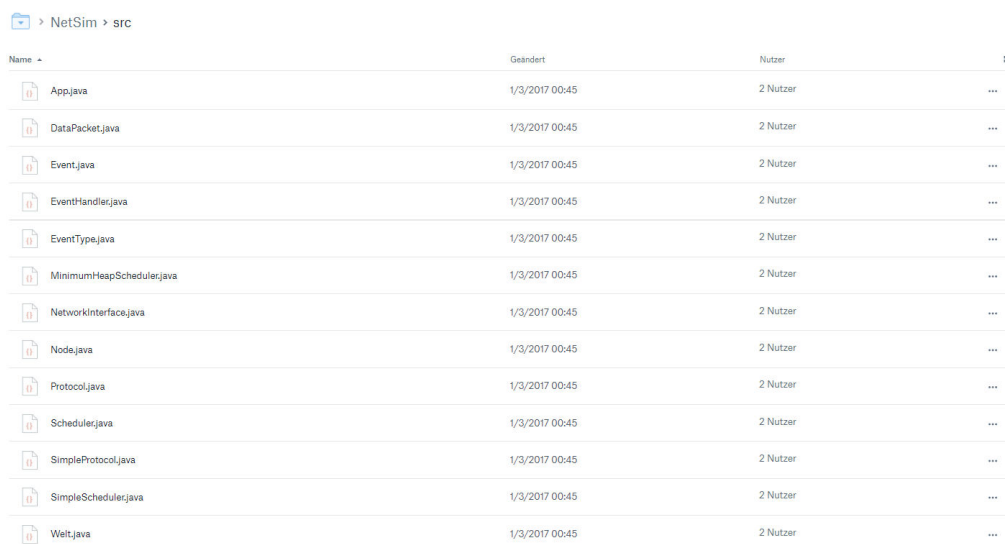
4.2 Dropbox



Dropbox ist ein seit 2007 eingeführter FileHosting-Dienst welcher in der Basisversion kostenlos ist. Entwickelt von Dropbox Inc ist diese Software zum Backup und zur Synchronisation von Dateien gedacht.

Wie schon erwähnt haben wir später nur noch GitHub verwendet, und dass das nötig war ist uns auch relativ schnell aufgefallen. Zuvor aber haben wir mit diesem Dienst gearbeitet, da es auch bei den Teilnehmern der Robotik AG (zu denen ich gehöre) schon seit Jahren Standard ist und dort perfekt funktioniert. Also war es erst einmal naheliegend diese Möglichkeit auch für den Projektkurs zu benutzen.

Nachdem es in den ersten Stunden auch keine großen Probleme gab mussten wir allerdings feststellen, dass es durchaus ein sehr aufhaltendes Problem gab, denn im Gegensatz zur Robotik AG, bei der wir stets mit Laptops arbeiten auf denen ebenfalls Dropbox installiert ist, war uns das mit den Schulrechnern nicht möglich, da sonst jeder auf unser Verzeichnis hätte zugreifen können. Wäre dieses Problem nicht gewesen hätten wir einfach die Dateien direkt im Dropbox Ordner bearbeiten können, aber so mussten wir uns jedes Mal bei Dropbox online anmelden, die alten Dateien löschen, dann die neuen wieder hinzufügen und das Ganze hat sich auf dauer als sehr mühsam erwiesen.



Name	Geändert	Nutzer	
App.java	1/3/2017 00:45	2 Nutzer	...
DataPacket.java	1/3/2017 00:45	2 Nutzer	...
Event.java	1/3/2017 00:45	2 Nutzer	...
EventHandler.java	1/3/2017 00:45	2 Nutzer	...
EventType.java	1/3/2017 00:45	2 Nutzer	...
MinimumHeapScheduler.java	1/3/2017 00:45	2 Nutzer	...
NetworkInterface.java	1/3/2017 00:45	2 Nutzer	...
Node.java	1/3/2017 00:45	2 Nutzer	...
Protocol.java	1/3/2017 00:45	2 Nutzer	...
Scheduler.java	1/3/2017 00:45	2 Nutzer	...
SimpleProtocol.java	1/3/2017 00:45	2 Nutzer	...
SimpleScheduler.java	1/3/2017 00:45	2 Nutzer	...
Welt.java	1/3/2017 00:45	2 Nutzer	...

Die Übersicht im Dropbox-Verzeichnis

4.3 GitHub



GitHub ist ein von GitHub, Inc entwickelter und im Jahr 2008 gestarteter Online-Dienst zur gemeinsamen Entwicklung und Umsetzung von Projekten. Beschrieben als „kollaborative Versionsverwaltung“ und unter dem Motto „Build software better, together.“ bietet es sämtliche Funktionen, die für eine gelungene Kooperation nötig sind.

Als Alternative zu Dropbox hat uns Herr Bachran also diese Plattform vorgeschlagen. Obwohl ich den Namen schon vorher häufig gehört hatte kannte ich GitHub bis dahin eigentlich noch gar nicht, Katja ebenso. Als wir uns diese Plattform ansahen ist mir gleich aufgefallen, dass diese wesentlich schwieriger zu bedienen war als Dropbox, aber durch das praktischen Einbindungs-Plugin von GitHub in Eclipse wurde uns ein effizienter Workflow ermöglicht.

FelixStach / NetworkSimulator

Watch

0

Star

0

Fork

0

<> Code

Issues 0

Pull requests 0

Projects 0

Wiki

Settings

Insights

Branch: master

NetworkSimulator / src /

Create new file

Upload files

Find file

History

katja.leibold Vergleich fertig

Latest commit 95e93e8 22 days ago

..

DataPacket.java

test for upstream

3 months ago

Event.java

260417

2 months ago

EventHandler.java

260417

3 months ago

EventType.java

erste Version

4 months ago

KatjaTest.java

erste Version

4 months ago

MinimumHeapNode.java

erste Version

4 months ago

MinimumHeapScheduler.java

richtig eingerückt

29 days ago

NetworkInterface.java

test for upstream

3 months ago

Node.java

test for upstream

3 months ago

Protocol.java

WARNING: head commit changed in the meantime

3 months ago

ReceiverApp.java

260417

2 months ago

Scheduler.java

erste Version

4 months ago

SenderApp.java

260417

2 months ago

SimpleProtocol.java

260417

2 months ago

SimpleScheduler.java

260417

2 months ago

Vergleich.java

Vergleich fertig

22 days ago

World.java

260417

2 months ago

Die Übersicht im GitHub-Verzeichnis

5. Arbeitsweise

Zu Beginn des Sommers einigten wir uns auf folgende Arbeitsteilung: Während Katja einen Minimum-Heap-Baum programmiert, welcher dann von meinem Programm verwendet wird, konzentriere ich mich auf die Kreierung und Implementierung von allen anderen Elementen. Zwar haben wir beide das selbe GitHub-Verzeichnis verwendet und auf die selben Dateien zugegriffen, hatten aber insgesamt sehr wenig mit der Arbeit des Anderen zu tun.

Die Umsetzung und Programmierung dieses Projekts hat sich über ein ganzes Jahr hingezogen, da es für uns ein sehr kompliziertes Thema war, bei dem man auch ohne Hilfe nur schwer alleine zuhause weiterarbeiten konnte. Den ursprünglichen Plan für den Netzwerk Simulator haben wir daher auch nicht (wie vorher evtl. eingeplant) noch erweitert. Ich habe mich das Jahr über stets bemüht das System dieser Ebene zu verstehen, aber dennoch war es mir häufig aufgrund von derzeit noch fehlendem Wissen (z.B. über Listen/ Vektoren) kaum möglich solche Datenstrukturen zu implementieren.

Am Ende kam also auf den ersten Blick absolut nichts brauchbares dabei heraus, das ganze Jahr über habe ich quasi nichts geschafft, bin immer auf der Stelle herum getreten und habe mehrmals die selben Fragen gestellt, ohne die Antworten wirklich zur Kenntnis zu nehmen. Im nächsten Kapitel wird noch sehr genau beschrieben, warum sich der Kurs am Ende keinesfalls als „nicht brauchbar“ herausgestellt hat, was die Probleme für einen Ursprung hatten und was ich daraus lernen konnte.

6. Probleme

Der Projektkurs lief gänzlich anders ab als erwartet. Nachdem wir uns im Sommer 2016 auf unser Thema, den einfach bezeichneten „Network Simulator“, geeinigt hatten, fingen wir direkt an die Klassen mitsamt der Methoden zu programmieren. Aufgrund unseres Arbeitsstiels hat sich die ganze Entwicklung als sehr schwierig herausgestellt, diese Unterkapitel beschreiben genauer an welchen Problemen es gescheitert ist und was ich in Zukunft ändern kann.

6.1 Motivation

Zurückbrachtet fingen die Probleme schon bei der Motivation, also am Anfang an. Katja und ich haben uns sehr schnell auf unser Thema geeinigt; aber eigentlich wusste keiner von uns wirklich was uns erwartet. Folglich hat sich unser Thema leider als sehr unpassend und uninteressant für mich herausgestellt, anders gesagt war es das längste, schwierigste (und langweiligste) Thema mit dem ich mich bisher im Bereich Informatik beschäftigt habe. An dieser Stelle mag man sich fragen, weshalb ich dies denn nicht im Verlauf des Jahres angemerkt habe. Und da beginnt auch schon eins der wichtigen Probleme.

6.2 Modell des Nichtbeschwerens

Über die Jahre habe ich mir (freiwillig/ automatisch) angewöhnt die Dinge und Tatsachen so hinzunehmen wie sie sind, ohne mich groß darüber zu beschweren. Dies hat stets auch so ziemlich alles betroffen was für mich nicht nur sehr langweilig oder nervig war, sondern auch sehr schwierig und komplex. Ein Beispiel dafür war mein Lateinunterricht, welchen ich vier Jahre lang „abgearbeitet“ und mich selten groß darüber beschwert habe. Seid der Grundschule ließ sich dieses System auf alles mögliche anwenden; In der Grundschule fand ich diese gut und habe sie nach Abschluss vermisst, aber schon schnell danach realisierte ich für mich, dass ich jene nur so gut fand, weil ich mich nicht darüber geärgert hatte wie anstrengend und „nervig“ sie war. Genau das gleiche passierte dann auch mit diversen Fächern welche mir schwer gefallen sind, dem Bus, der immer viel zu überfüllt war und einigem mehr.

Insgesamt hat sich dieses „Modell des Nichtbeschwerens“ immer sehr gut ausgezahlt. Ohnehin hätte es wohl sowieso nichts geändert wenn ich mich ausgiebig über ein Fach geärgert hätte, welches ich nicht abwählen konnte, einen Bus, der auch nicht häufiger fahren kann und weiteren Aspekten. Also habe ich gelassen. Was uns natürlich zu einem sehr wichtigen Punkt bringt... diesem Projektkurs.

Es scheint als habe ich in diesem Kurs einen ganz wesentlichen Aspekt vergessen... dass man die freie Wahl des Themas hatte. Da ich nun nach zehn Jahren Schule das Gefühl hatte, eben jene würde nie enden, hat sich dies anscheinend auch stark auf meine kreative Initiative ausgewirkt und ich habe nur mit dem Gefühl gearbeitet am Ende irgendein Ergebnis zu haben; ohne wirklich darüber nachgedacht zu haben welches. Trotz der öfteren Anmerkung vom Anfang, dass man bei Problemen möglichst schnell dem Projektleiter die Situation schildern sollte, habe ich es nicht getan. Eine Erkenntnis, welche wahrscheinlich vieles in meinem zukünftigen „Karriereweg“ ändern wird. Aber das war am Ende natürlich nicht das einzige Problem. Weitere folgten.

6.3 Klarheit/ Komplexität

Ein ganz wesentliches Problem kommt hinzu. Trotz meines Informatik Leistungskurses fiel es mir kontinuierlich sehr schwer, das gesamte System zu überblicken. Die Beziehung der Klassen untereinander zu verstehen, sowie einen generellen Plan zu haben. Obwohl ich bei ziemlich jedem Treffen meines Teams Mittwoch nachmittags anwesend war, musste ich dennoch sehr oft Funktionsweisen von Systemen nachfragen (weitere Anmerkungen dazu später) und die gleichen Fragen häufig mehrmals stellen, da ich mir die „Lösungen“ nicht gemerkt und keine entsprechenden Notizen angefertigt habe. Noch dazu kam, dass wir des öfteren mit Datenstrukturen gearbeitet haben welche mir (zu diesem Zeitpunkt) noch unbekannt waren wie (Informatik bezogene) Listen und Vektoren. Ich bin mir immer noch nicht ganz sicher was an dieser Stelle richtig gewesen wäre, angesehen der vielen Stunden die wir allein im Informatik LK gebraucht haben um alles komplett zu verstehen, aber es wäre sicherlich sehr sinnvoll gewesen mir die Strukturen selber noch einmal anzusehen und zu versuchen diese zu verstehen, denn es gibt ja immer noch das Internet.

6.4 Zeiteinteilung

Und auch das hat sich letzten Endes als größeres Problem herausgestellt. Wie gesagt: Ich habe mich kaum darauf konzentriert, an was ich da eigentlich arbeite, ich wusste nur, dass ich daran arbeite. Anders gesagt: Ich habe quasi einfach nur jeden Mittwoch an irgendetwas gearbeitet was aktuell für mich interessant aussah und habe mir/ uns nie wirklich einen Zeitplan oder ähnliches überlegt, da ich davon ausgegangen bin, dass das ganze aufgeht. Dem war am Ende natürlich nicht so; und was jetzt am Ende des Schuljahres als Ergebnis herausgekommen ist beschreibt sich selbst ganz gut als ein paar Klassen mit ein paar Zeilen Code, der mir stellenweise immer noch sehr schwer fällt zu verstehen. Für weitere Projekte wäre wahrscheinlich ein Zeitplan ganz am Anfang des Projektes sinnvoll, der beschreibt wann welche Stufe der Entwicklung erreicht sein soll und wann das gesamte Projekt fertig gestellt sein muss.

6.5 Reflexion

Seid dem Gespräch vor einigen Wochen habe ich immer wieder darüber nachgedacht was genau ich jetzt mit dem „Ergebnis“ des Projektkurses anfangen soll, welche Schlüsse ich daraus ziehen kann und warum es wohl so gelaufen ist, wie es nun mal passiert ist. Viele der Aspekte habe ich zwar schon genannt, aber diesem einen möchte ich noch einmal besondere Beachtung schenken: „Dem Nachfragen“. Hier habe ich es mit einem so wichtigen Aspekt zu tun, dass dieser wahrscheinlich den gesamten Ausgangs des Projekt hätte ändern können.

Insgesamt ist dies am Ende so aufgefallen, dass ich Probleme nach gewisser Zeit einfach nicht mehr genauer erläutert haben wollte, und mir anscheinend einfach dachte: *„Das verstehst du schon“*. Interessant für mich an dieser Stelle natürlich: Warum?

Rückblickend auf einige Jahre Schule lässt sich diese Frage wohl relativ einfach beantworten. Trotz dass ich mit vielen verschiedenen Lehrer(inne)n zu tun hatte war eine Reaktion gelegentlich immer gleich. Ich habe anscheinend JEDEN (bevorzugt in der Sekundarstufe 1) Lehrer und JEDE Lehrerin mit meinen Fragen und dem Nachfragen von für mich komplizierteren Problemfällen so sehr genervt, dass sich der Mut dennoch nachzufragen über die Zeit immer weiter vermindert hat. Noch dazu kommt, dass ich da über die Zeit sowieso immer vorsichtiger wurde, weil ich nicht mit allzu vielen Mitschülern Kontakt hatte und das ganze dadurch nicht auch noch negativer ausfallen sollte.

Aber ich denke das ist vielleicht gar nicht mehr so wichtig. Meine Aufgabe in der Schule ist den Unterricht zu verstehen, und die Aufgabe der Lehrer ist es, dass wir den Unterricht verstehen. Eigentlich sollte dem in der Hinsicht nichts im Wege stehen.

Ich habe mir mal den Vortrag von einem Professor aus der Informatik angehört. Sein Kommentar war, dass sich seiner Meinung nach viel zu wenige Studenten aufgrund von Verständnisproblemen melden als müssten. vor allem diese Aussage hat mich daran erinnert wie wichtig es ist immer wieder zu Fragen, wenn man es noch nicht verstanden hat. Das ist mir nun klargeworden.

7. Fazit

Jetzt, da ich am Ende des Jahres und damit auch am Abschluss dieser Dokumentation stehe denke ich, dass ich nicht nur diverses im Bezug auf größere Projektarbeiten gelernt habe, sondern mir jetzt auch vieles über meine Arbeitsweise bewusst geworden ist. Aber nicht nur über die Dinge die mir schwerfallen, sondern auch über Dinge welche ich gut kann und mir leicht fallen. Zum Beispiel bin ich häufig sehr kreativ und habe viele Ideen, mit denen ich auch viel erreichen kann wenn richtig umgesetzt. Dazu gehört natürlich auch an diesen festzuhalten, wenn ich ein Potenzial darin sehe, wie anfangs beschrieben. Da ich mich im Moment sehr verstärkt mit meiner Zukunft auseinandersetze (Besuch von Universitäten und Praktika) glaube ich auch an dieser Stelle eine wichtige Erfahrung gemacht zu haben, die mir später bei der Spezialisierung meines Studiums/ Berufs eine große Hilfe sein wird. Auf der anderen Seite weiß ich nun aber auch was ich ändern kann, dazu gehören das häufigere „Anfertigen von Mitschriften“, eine (angemessene) Meinungsäußerung und mehr Selbstinitiative.

Es hat zwar sehr lange gedauert, aber letzten Endes ist für mich aus „A terrible Experience“ eine „powerful Experience“ geworden, die mir rückblickend, so wie nun mal alles abgelaufen ist, möglicherweise sogar wesentlich mehr mitgegeben hat als wenn alles nach Plan gelaufen wäre. Eine Erfahrung, welche unerwartete Situationen in der Zukunft vielleicht sogar besser beschreibt als eine solche in der Alles funktioniert und es keine Probleme gibt...

~ Felix Stach

7. Literaturverzeichnis

- Elektronik-Kompendium (IP-Routing):
<https://www.elektronik-kompendium.de/sites/net/0903151.htm>
- Wikipedia (Routing):
<https://de.wikipedia.org/wiki/Routing>
- Wikipedia (Osi-Modell):
<https://de.wikipedia.org/wiki/OSI-Modell>
- Wikipedia (UDP/TCP):
https://de.wikipedia.org/wiki/Transmission_Control_Protocol
https://de.wikipedia.org/wiki/User_Datagram_Protocol
- Wikipedia (GitHub):
<https://de.wikipedia.org/wiki/GitHub>
- Wikipedia (Dropbox):
<https://de.wikipedia.org/wiki/Dropbox>
- Wikipedia (Eclipse):
[https://de.wikipedia.org/wiki/Eclipse_\(IDE\)](https://de.wikipedia.org/wiki/Eclipse_(IDE))
- Wikipedia (IP):
<https://de.wikipedia.org/wiki/IP>
- Wikipedia (Aggregation):
[https://de.wikipedia.org/wiki/Aggregation_\(Informatik\)](https://de.wikipedia.org/wiki/Aggregation_(Informatik))