**Team**: 6, Mert Siginc, Michael Müller

**Aufgabenaufteilung**:

1. <Aufgaben, für die Teammitglied 1 verantwortlich ist>,   
   <Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden>
2. <Aufgaben, für die Teammitglied 2 verantwortlich ist>,   
   <Dateien, die komplett/zum Teil von Teammitglied 2 implementiert/bearbeitet wurden>

**Quellenangaben**:

* <http://erlang.org/doc/apps/stdlib/index.html>
* <http://users.informatik.haw-hamburg.de/~klauck/verteiltesysteme.html>

**Bearbeitungszeitraum**:

**Aktueller Stand**: <Welche Teile der Software sind fertig inklusive Tests, welche sind fertig, aber noch nicht getestet, welche müssen noch implementiert werden>

**Änderungen des Entwurfs**: <Vor dem Praktikum auszufüllen: Welche Änderungen sind bzgl. des Vorentwurfs vorgenommen worden.>

**Entwurf**: <Entwurf nach den bekannten SE-Richtlinien und den Vorgaben gemäß Aufgabenstellung.>

Ergebnis des Prozesses des Definierens von Architektur, Komponenten, Schnittstellen, Abstrakten Datentypen und anderen Charakteristika eines Systems oder einer Komponente. Dient als einziges Dokument bei der Implementierung!

Anforderungsermittlung:

* (Nicht) Funktionale Anforderungen
* Anwendungsfälle
* Userstories

Modellierung:

* UML
* OO/funktionales Design
* Design & Architektur Pattern
* GRASP

Wie ist die Architektur aufgebaut?

Da hier nur Stationen senden und empfangen, und alle gleichwertig sind, ist es eine vollkommene Peer-To-Peer Kommunikation.

Wie sieht der (Funk-) Kanal aus?

Der Kanal ist zeitlich in Frames eingeteilt, die wiederum in Slots eingeteilt sind.

Alle Nachrichten dürfen nur über diesen Kanal laufen und nur diese dürfen gelesen (also empfangen) werden. Der Kanal ist über einen bestimmten Socket anzusprechen.

Der Kanal selbst ist nicht zu implementieren, sondern spiegelt das Verhalten innerhalb der Stationen wieder. Jede Station hat also gewisser maßen einen eigenen Kanal, der dann über die Laufzeit und der Synchronisation der Uhren zu einem einheitlichen Kanal wird.

* Frames
  + Länge: 1 Sekunde
  + Hält 25 Slots
  + In einem Frame können N Nachrichten gesendet werden, wobei N >= 0 und N <= Anzahl der Stationen.
  + Fängt ab dem 01.01.1970 um 00:00:00 Uhr (UTC) an zu arbeiten (Jede Sekunde ein Frame zu haben).
    - Frame 1 beginnt: 01.01.1970 um 00:00:00
    - Frame 2 beginnt: 01.01.1970 um 00:00:01
    - ….
* Slots
  + Länge: 40 MS
  + In dem Slot werden N Nachrichten, wobei N >= 0 und N <= Anzahl der Stationen, geschickt.
* Socket
  + IP: 225.10.1.2
  + Port: 15000 + Teamnummer (z.b. Port 15003)
* Kollisionen

Kollisionen entstehen, wenn mindestens 2 Stationen im selben Slot senden. Ziel ist es letztendlich einen Komplett Kollisionsfreien Nachrichtenverkehr zu haben, in dem in jedem Slot einmal gesendet wird, wenn Stationenanzahl = Slotanzahl in einem Frame.

Zudem sendet jede Station maximal einmal in einem Frame. Ist ein kollisionsfreier Nachrichtenverkehr erst einmal passiert darf es nie wieder zu Kollisionen kommen.

Ist die Stationenanzahl < Slotanzahl, so sind dann dementsprechend viele Slots ohne Nachricht. Bei Stationenanzahl > Slotanzahl wird in jedem Frame zu Kollisionen kommen da ja nicht genügend Slots vorhanden sind.

Wie sehen die Nachrichten aus?

* TTL = 1
* Gesamtlänge zu jeder Zeit 34 Byte:

Byte 0: Stationsklasse (A oder B)

Byte 1 – 10: Stationsname (Bildet mit Byte 11-24 die gesamten Nutzdaten)

Byte 11 – 24: Restliche Nutzdaten

Byte 25: Slotnummer, in dem die Station im nächsten Frame senden wird.

Byte 26 - 33: Zeitpunkt des sendens in MS, 8-Byte Integer, Big Endian, UTC.

Wie sieht die Station aus?

* Stationsname (Atom):
  + team<TEAMNUMMER>-<STATIONSNUMMER>
  + Beispiel für Team 06, Station 01: team06-01
* Wie bekommt man einen Slot und was geschieht danach?

Ein Slot findet man, in dem man folgenden Ablauf beachtet:

1. Alle kollisionsfreien Nachrichten auf deren Slotnummern überprüfen
2. Unter allen übrigen Slotnummern (im nächsten Frame) eine auswählen
3. Slotnummer in Nachricht einfügen und erst einmal davon ausgehen, dass es zu keiner Kollision kommen wird

Ergibt sich dann im nächsten Frame im ausgewählten Slot keine Kollision so wird der Slot behalten und kein neuer gesucht. Es darf nun zu keinen Kollisionen, in diesem Slot, mehr kommen.

Kommt es dagegen zu einer Kollision muss wieder einen **kompletten** Frame zugehört werden um in den dann kollisionfreien Nachrichten wieder eine freie Slotnummer zu finden.

* Wie sieht die Datenquelle aus?

Ist ein vorgegebenes Programm. Dieses wird über Standard Out (die Konsole) periodisch 24-Byte Nutzdatenpakete generieren (siehe Byte 1-24 der Nachrichten).

Verwendet wird dann pro Station nur das aktuellste generierte Paket. Wie greifen wir den Output des Programms ab, Pipe?

* Wie sieht die Datensenke aus?

Die Datensenke ist vorerst über den minimalsten weg zu realisieren. In diesem Fall stellt das Standard Out (in die Konsole) dar.

* Wie sieht die Interne Uhr aus?

Sie zeigt die aktuelle Zeit (für die Station), ausgehend von UTC, an. Sie wird konstant synchronisiert mit den Uhren der anderen Stationen. Dies geschieht dann über die empfangenen kollisionsfreien Nachrichten von Typ A Stationen.

Typ A und Typ B Stationen und deren Uhren unterscheiden sich nur darin, dass Typ A Stationen zu Beginn mit einer Abweichung gestartet werden.

* Wie sieht der Empfänger aus?

Der Empfänger ist zu jeder Zeit aktiv und hört auf die gesendeten Nachrichten im Kanal. Entsteht eine Kollision in einem Slot so werden die Nachrichten behandelt als hätte es sie nie gegeben.

Die kollisionsfreien Nachrichten werden daraufhin weiter untersucht, zum einen zur Synchronisation der Zeit und zum anderen um selbst einen Slot im nächsten Frame zu finden. Dies geschieht unabhängig von den anderen Stationen.

* Wie sieht der Sender aus?

Der Sender sendet zu einem bestimmten Zeitpunkt ein Nachrichtenpaket.

Wie läuft das ab?