Universität Bremen
FB 3 – Informatik
Dr. Karsten Hölscher
TutorIn: Dr. Karsten Hölscher

Software-Projekt 2 – SoSe 2020

VAK 03-BA-901.02

Architekturbeschreibung

Clara Maria Odinius	odinius@uni-bremen.de
Habib Mergan	habib1@uni-bremen.de
Kevin Santiago Rey Rodriguez	kev_rey@uni-bremen.de
Liam Hurwitz	hurwitz@uni-bremen.de
Mehmet Ali Baykara	baykara@uni-bremen.de
Miguel Alejandro Caceres Pedraza	mcaceres@uni-bremen.de

Abgabe: 31. Mai 2020 — Version 1.1

Inhaltsverzeichnis

Version und Änderungsgeschichte

Die aktuelle Versionsnummer des Dokumentes sollte eindeutig und gut zu identifizieren sein, hier und optimalerweise auf dem Titelblatt.

Version	Datum	Änderungen
0.1	22.04.2020	Dokumentvorlage als initiale Fassung kopiert
0.2	05.05.2020	Anwendungsfälle hinzugefügt
0.3	09.05.2020	Verzeichnis für Definitionen, Akronyme und Abkürzungen hinzugefügt
0.4	12.05.2020	erste Gedanken zur Globalen Analyse notiert
0.5	12.05.2020	Einflussfaktoren hinzugefügt
0.6	13.05.2020	Einflussfaktoren bearbeitet, Problemkarten hinzugefügt
0.7	14.05.2020	Schreibstil überarbeitet
0.8	14.05.2020	Konzeptionelle Sicht hinzugefügt
0.9	14.05.2020	weitere Problemkarten erstellt
1.0	15.05.2020	Problemkarten überarbeitet
1.1	16.05.2020	Einflussfaktoren überarbeitet
1.2	16.05.2020	Globale Analyse überarbeitet
1.3	18.05.2020	Zweck der Architekturbeschreibung hinzugefügt
1.4	18.05.2020	Abkürzungen Alphabetisch sortiert
1.5	20.05.2020	Globale Analyse überarbeitet
1.6	21.05.2020	Konzeptionelle Sicht überarbeitet
1.7	21.05.2020	Ausführungssicht überarbeitet
1.8	21.05.2020	Referenzen hinzugefügt
1.9	21.05.2020	Ausführungssicht überarbeiet
2.0	22.05.2020	Ausführungssicht Beschreibung ergänzt
2.1	22.05.2020	Modulsicht png dem Dokument hinzugefügt
2.2	22.05.2020	Zustandsdiagram png dem Dokument hinzugefügt
...		

1 Einführung

1.1 Zweck

Diese Architekturbeschreibung beschreibt die grundlegenden Strukturen unseres Spiels „Even Geiler Than Geiler Than Faster Than Light“. Zu den Lesern gehören unser Püfer und Tutor Karsten Hölscher, die Gruppenmitglieder und weitere Interessierte der Fachrichtung Informatik. Der Leser soll hiermit zum einen den Kontext der Arbeit verstehen, sowie den Aufbau der Anwendung, vom Groben bis ins Detail, über viele Abstraktionsebenen hinweg. Somit wird nicht nur die technische Realiesierung des Spiels erklärt, sondern auch die allgemeinen Anforderungen an das Projekt und in diesem Zusammenhang getroffene Entscheidungen. Darüber hinaus, hilft uns eine Architekturbeschreibung dabei, das Projekt zu planen und strukturiert durchzuführen.

1.2 Status

Das Dokument beschreibt unseren ersten Entwurf und wird in der Implementierung umgesetzt.

1.3 Definitionen, Akronyme und Abkürzungen

Begriff	Definition
Software-Architektur	Beschreibt die grundlegende Komponenten eines Softwaresystems.
Architektursicht (View)	Repräsentation eines ganzen Systems aus der Perspektive einer kohärenten Menge von Anliegen (IEEE P1471, 2002).
Anwendungsfälle	Spezifiziert eine beliebige Menge von Aktionen, die ein System ausführen muss, damit ein Resultat stattfindet, welches für mindestens einen Akteur von Bedeutung ist.
Framework	Programmiergerüst, welches den Rahmen der Anwendung bildet. Es umfasst Bibliotheken und Komponenten.
GUI	Steht für Graphical User Interface und kennzeichnet eine grafische Schnittstelle, über die ein Mensch mit einer Software interagiert.
Libgdx	ist ein Java-Framework für plattformunabhängige Spieleentwicklung.
IDE	integrierte Entwicklungsumgebung - Hilft bei der Bearbeitung von Projekten in der Softwareentwicklung.
Versionskontrolle	Hochgeladenen Versionen werden festgehalten und können wieder hergestellt werden.
HTTP	Steht für Hypertext Transfer Protocol, einem zustandslosen Protokoll zum synchronen Versenden von Informationen über Rechnernetze.
Interface	Schnittstelle.
Javadoc	Software-Dokumentationswerkzeug für die Programmiersprache Java.
JavaEE	(Java Platform Enterprise Edition) Eine Spezifikation für die transaktionsbasierte Ausführung von in Java programmierten Anwendungen.
JUnit	Frameworke für Tests für die Programmiersprache Java.
HTTPS	Durch Verschlüsselungstechniken gesichertes HTTP.
Maven	Java Programme können standardisiert und verwaltet werden.
ibrary	Sammlung von verschiedensten vorgefertigten Methoden oder Klassen.
Multiplayer	Spielmodus, mit mehr als einem Spieler zur gleichen Zeit.
Singleplayer	Spielmodus, mit einem Spieler, der gegen einen computergesteuerten Gegner spielt.

Paketdiagramm	Strukturdiagramm der UML, stellt die Verbindung zwischen Paketen, Paketimports bzw. Verschmelzungen und deren Abhängigkeiten dar.
Problemkarte	Beschreiben Probleme im Zusammengang der Einflussfaktoren und stellen Lösungen bzw. entsprechende Strategien dar.
Sequenzdiagramm	Strukturdiagramm der UML, stellt den Austausch von Nachrichten zwischen Objekten mittels einer Lebenslinie dar.
Klassendiagramm	Strukturdiagramm der UML, stellt die statischen Strukturen eines Systems dar.
Komponentendiagramm	Strukturdiagramm der UML, stellt Komponenten und deren Schnittstellen dar.
TCP	Das Transmission Control Protocol ist ein Netzwerkprotokoll, das definiert, auf welche Art und Weise Daten zwischen Netzwerkkomponenten ausgetauscht werden sollen.
UDP	Das User Datagram Protocol, kurz UDP, ist ein minimales, verbindungsloses Netzwerkprotokoll, das zur Transportschicht der Internetprotokollfamilie gehört.
UML	Steht für Unified Modeling Language und ist eine grafische Modellierungssprache zur Spezifikation, Visualisierung, Konstruktion und Dokumentation von Modellen für Softwaresysteme.

1.4 Referenzen

- Architekturbeschreibung DataColorado Software-Projekt 2 WiSe 2019/20
- Architekturbeschreibung RainersRaiders Software–Projekt 2 WiSe 2016/17
- Architekturbeschreibung SuperSaiyans Software-Projekt 2 SoSe 2019
- Hinweise zur Abgabe der Architekturbeschreibung für Software–Projekt 2 (Stand 03.05.2019)
- Kick-Off-Folien (Stand 20. April 2020)

1.5 Übersicht über das Dokument

2 Anwendungsfälle

2.1 Anmeldung

Ein User startet das Spiel. Ein Fenster, in dem sich der User mit Password und Namen einloggen kann, öffnet sich. Wenn er die richtige Daten einträgt. kann das Spielt weiter geführt werden. Falls die Daten nicht richtig sind, bekommt der User einen Hinweis darauf und wird geben es erneut zu versuchen.

2.2 Multi-Spieler

Wenn der User angemeldet ist, dann kann er sehen, welche der andere User ebenfalls verbunden sind und hat entweder die Möglichkeit gegen ander User zu spielen oder aber ein Spiel gegen den Computer zu starten.

2.3 Spielt-Starten

Im ersten Fenster des Spiels kann der Benutzer das Raumschiff, die Waffen, welche er verwenden möchte, die Besatzung und den Schwierigkeitsgrad des Spiels auswählen. Anschließend startet das Spiel.

2.4 Spielt-Verlauft

Zu beginn, kann der Spieler die Besatzung in den von ihm gewählten Positionen/Sektionen des Raumschiffes positionieren. Dann kann er die Sprungtaste drücken, um die Zielplaneten auszuwählen. Der Benutzer kann dies in jeder Runde wiederholen.

Wenn der Benutzer einen Zielplaneten auswählt, kann er auf Gegner treffen und nach einem Dialogfeld gegen diese antreten.

Sobald der Benutzer gewonnen hat, kann der Gegner sein Raumschiff reparieren, seine Besatzung heilen oder einen Sprung machen.

2.5 Kampf-Regeln

In einem Kampf gelten folgende Regeln:

- Wenn ein Kampf beginnt, kann der Benutzer seine Waffen ausrichten, um auf bestimmte Sektionen des gegnerischen Raumschiffes zu zielen.
- Waffen brauchen Zeit zum Laden und Schießen.
- Damit eine Waffe das Ziel treffen kann, muss der Gegner Schilde deaktiviert haben.

- Um die Schilde zu deaktivieren, müssen diese zuvor x mal getroffen worden sein. Je Treffer auf das Schild, verliert es an Stärke. Sobald die Stärke des Schildes Null ist, ist es deaktiviert.
- Waffen treffen zu bestimmten Wahrscheinlichkeiten das Ziel.
- Damit eine Waffe verwendet werden kann, muss die Sektion, in der die Waffe steht, funktionsfähig sein, sowie sich ein Besatzungsmitglied in selbiger Sektion aufhalten muss, damit diese auch bedient werden können.
- Jedes Mal, wenn eine Waffe ein Raumschiff trifft, verliert sie ihren Lebenspunkt.
- Sobald eines der beiden Raumschiffe keine Lebenspunkte mehr hat, hat es verloren.

2.6 Pausenmodus

Während eines Kampfes kann der Benutzer das Spiel in den Pausenmodus versetzen. Dies bedeutet, dass beide Schiffe aufhören zu schießen und die Besatzung aufhört, sich zu bewegen. Wenn das Spiel fortgesetzt wird, werden die Waffen wieder aktiviert und auch die Besatzung.

3 Globale Analyse

3.1 Einflussfaktoren

Legende:

++ Hohe Flexibilität und Veränderlichkeit

\+ Leichte Flexibilität und Veränderlichkeit

\- - Sehr geringe Flexibilität und Veränderlichkeit

\- Wenig Flexibilität und Veränderlichkeit

Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
O1 : Organisation			
O1.1 Time-To-Market			
Die Auslieferung erfolgt am 02.08.2020.	Keine Veränderlichkeit oder Flexibilität, da Vorgaben bestehen.	- -/ - -	Nicht alle Funktionen können realisiert werden bzw. implementiert werden.
O1.2 Architektur-Abgabe			
Die Auslieferung erfolgt am 31.05.2020.	Keine Veränderlichkeit oder Flexibilität, da Vorgaben bestehen.	- -/ - -	Durch den Zeitdruck könnte die Architektur mangelhaft werden. Wenn wir uns nicht genug Zeit lassen, könnten Aspekte, die relevant für die Architektur sind, vergessen werden.
O1.3 Entwickler			

Die Projektgruppe besteht aus 6 Entwicklern	Falls ein Entwickler (temporär) ausfällt kann ein anderer Entwickler einspringen. Keine neuen Entwickler können eingestellt werden. Gruppenmitglieder können wegfallen oder austreten	+/ - -	Die Architektur kann wegen Zeitmangel (es können nicht mehr als die ursprünglichen sechs Entwickler mitarbeiten) und fehlenden Fähigkeiten Mängel enthalten. Unvollständige Implementierung droht
O1.4 Fähigkeiten Entwickler			
Nicht alle Entwickler haben die gleiche Programmier-erfahrung	Keine Flexibilität, aber Veränderlichkeit, da man sich einarbeiten/recherchieren kann.	- -/ ++	Die Implementierung kann Mängel enthalten.

O1.5 Teamarbeit in Corona-Zeiten			
Das persönliche Treffen kann nicht stattfinden, aufgrund von Kontakt-beschränkung	Digitale Treffen über z.B. Discord. Keine Veränderlichkeit aufgrund der Gesetzeslage.	++/ - -	Missverständnisse können öfter auftreten. Teamarbeit könnte erschwert werden und dadurch die Qualität der der Architektur beeinträchtigen

Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
T1: Technik			
T1.1: Programmiersprache			
Java 8 oder höher ist vorgegeben.	Die Programmier-sprache wird vorgegeben. Wir können aber zwischen verschiedenen Versionen auswählen.	- -/ - -	Das Architektur muss in Java umgesetzt werden.
T1.2 Betriebssystem			
Die Anwendung muss auf den gängigen Betriebssystemen (MacOS, Windows, Linux) laufen.	Keine Veränderlichkeit oder Flexibilität, da Mindestanforderungen bestehen. Aber wir können noch weitere Betriebssysteme hinzufügen	- -/ - -	Bei der Implementierung müssen die gängigen Betriebssysteme berücksichtigt werden. Wenn die Anwendung auf mehr Betriebssystemen laufen soll, dann muss mehr Zeit investiert werden.
T1.3 Client-Server-Architektur			
Zur Implementierung muss Client-Server-Architektur benutzt werden.	Keine Veränderlichkeit oder Flexibilität, da Mindestanforderungen bestehen.	- -/ - -	Das Projekt muss eine Client-Server-Architektur haben.

T1.4: Framework libgdx			
Als Framework zur Erstellung der Oberfläche muss libgdx verwendet werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Der Client muss in libgdx umgesetzt werden. Der Aufwand steigt, weil die Gruppe ein neues Framework lernen müssen und daher können Mängel in der Implementierung entstehen.
T1.5: Persistenz			
Zur Speicherung der Daten soll eine leichtgewichtige relationale Datenbank verwendet werden.	Keine Flexibilität, da Mindestanforderungen bestehen. Veränderlich, da durch überlastete Server auf ein externes Datenbank-managementsystem umgesprungen werden muss.	- -/ +	Die Architektur muss angepasst werden, sodass wir eine leichtgewichtige relationale Datenbank verwenden können. Wir entscheiden uns für H2
T1.6: Build System			
Um das Projekt zu bauen muss ein Build-System benutzt werden.	Auswahl zwischen Maven und Gradle möglich. Keine Veränderlichkeit möglich, da Mindestanforderungen bestehen.	+/ - -	Das Projekt muss Maven oder Gradle-build fähig sein. Wir verwenden Gradle

T1.7: Wartbarkeit			
Die Software muss einfach zu warten sein.	Es sind keine Anforderungen bezüglich der Projektstruktur gegeben, daher können wir flexibel selbst entscheiden wie wir diese implementieren .	++/ ++	Die Architektur muss so aufgebaut sein, sodass die Software leicht gewartet und erweitert werden kann.

Einflussfaktor	Flexibilität und Veränderlichkeit	++/ - -	Auswirkungen
P1: Produktfaktoren			
P1.1: Spiel unterbrechen und Spielstand speichern			
Die Anwendung muss in der Lage sein, ein zuvor von einem bestimmten Benutzer gestartetes Spiel zu speichern und dem Benutzer die Möglichkeit zu geben, es zu einem anderen Zeitpunkt fortzusetzen.	Keine Flexibilität und keine Veränderlichkeit, da Mindestanforderungen bestehen.	- -/ - -	Wir müssen eine Datenbank integrieren um zu speichern.

P1.2: Schwierigkeitsstufen			
Für Benutzer müssen mindestens zwei unterschiedliche Schwierig-keitsgrade implementiert werden.	Keine Veränderlichkeit,da Mindestanforderungen bestehen.Es können aber mehr als zwei Schwierigkeitsgrade implementiert werden. Außer dem Karsten-Modus können wir den Schwierigkeitsgrad der anderen Stufe beliebig auswählen.	+ / - -	Das Spiel soll in mindestens zwei verschiedenen Schwierigkeits-stufen gespielt werden können. Pro weiterer Spielstufe ändert sich die Architektur.
P1.3: Multiplayer			
Der Server muss einen Multiplayer Modus ermöglichen.	Keine Flexibilität oder Veränderlichkeit, da Mindestanforderungen bestehen. Wir können es aber ermöglichen, dass mehr als zwei Spieler gleichzeitig spielen.	+/ - -	Die Anwendung darf nicht von gleichzeitiger Verwendung von zwei Nutzern überfordert sein. Falls mehr als zwei Spieler gleichzeitig spielen sollen, dann muss die Architektur angepasst werden.
P1.4: Singleplayer			
Der Server muss einen Singleplayer Modus ermöglichen, falls keine mensch-lichen Spieler vorhanden sind.	Keine Flexibilität oder Veränderlichkeit, da Mindestanforderungen bestehen.	- -/ - -	Implementierung eines computergesteuerten Gegners (NPC).
P2.0: Struktur und Eingeschaften des Raumschiffs			
P2.1: Aufteilung in Sektionen			

Pro Sektion gibt es die relevanten Systeme: Antrieb, Waffen, Schutzschild	Keine Flexibilität oder Veränderlichkeit, da, Mindestanforderungen bestehen. Wir können aber noch weitere Systeme hinzufügen	- -/ - -	Die Sektionen in den Raumschiffen müssen die relevanten Systeme, Antrieb,Waffen,Schutzschild, beinhalten. Falls wir weitere Systeme hinzufügen, dann ändert sich die Architektur.
P2.2: Schäden der Sektionen			
Jede Sektion kann beschädigt werden, wodurch die darin enthaltenen Systeme beschädigt werden.	Keine Flexibilität oder Veränderlichkeit, da Mindestanforderungen bestehen.	- -/ - -	Die Beschädigung der Sektionen hat weitläufige Auswirkungen auf viele andere Komponenten.

P2.3: Ressourcen des Spiels			
Ressourcen müssen implementiert werden, um die Spiellogik auszuführen, zum Beispiel: Geld Energie Hüllenintegrität	Keine Flexibilität und keine Veränderlichkeit, da Mindestanforderungen bestehen.	- -/ - -	Entscheidender Einfluss auf die Umsetzung des Spiels
P2.4: Raumschiff Eigenschaften			
Eigenschaften können durch Geld verbessert werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Hat Auswirkungen auf die Veränderlichkeit der Systeme (Waffen usw.).
P3.0: Besatzung			
P3.1: Raumschiff hat Besatzung			
Besatzungsmitglied kann sich in Sektionen aufhalten und Systeme beinflussen.	Keine Flexibilität und keine Veränderlichkeit, da Mindestanforderungen bestehen.	- -/ - -	Hat Auswirkungen auf die Umsetzung der Sektionen, auf die Funktionsfähigkeit der darin enthaltenen Systeme und auf die möglichen Spielzüge eines Spielers
P3.2: Besatzung Eigenschaften			
Die Besatzung des Schiffes kann Systeme/ Sektionen reparieren, kann sterben kann eingestellt/ angeheuert werden und hat Fähigkeiten	Nur flexibel in der Art der Umsetzung. Keine Veränderlichkeit, da Teil der Mindestanforderungen. Aber wir können noch weitere Fähigkeiten spezifizieren	- -/ - -	Hat Auswirkungen auf die Aktionen die ein Besatzungsmitglied ausführen kann.

P3.3: Besatzung Fähigkeiten			
Die Fähigkeiten der Besatzung können verbessert werden und beeinflussen die Systeme des Schiffes, auf dem sie arbeiten.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Hat Auswirkungen auf die Aktionen die ein Besatzungsmitglied ausführen kann.
P4.0: Universum			
P4.1: Struktur des Universums			
Das Universum muss Stationen und Planeten haben, die vom Raumschiff des Spielers durchquert werden.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss ermöglichen, dass sich Raumschiffe im Universum Stationen und Planeten anfliegen können.
P4.2: Station/Planet Eigenschaften			
Jede Station oder jeder Planet kann Ereignisse haben, die das Spiel negativ oder positiv beeinflussen. Es gibt fünf verschiedenartige Ereignisse	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen.	- -/ - -	Die Architektur muss vorsehen, dass Ereignisse auf Stationen und Planeten Einfluss auf den Spielverlauf haben
P4.3: Feindliche Schiffe			
Es gibt mindestens drei verschiedene gegnerische Raumschiffe.	Keine Veränderlichkeit oder Flexibilität, da Teil der Mindestanforderungen..	- -/ - -	Die Architektur muss mindestens drei verschiedene Raumschiffe vorsehen

P5.0: Struktur des Kampfes			
P5.1: Taktische Entscheidungen			
Der Kampf mit einem gegnerischen Raumschiff erfolgt rundenbasiert.	Keine Flexibilität oder Veränderlichkeit, da Mindestanforderungen bestehen.	- -/ - -	Die Architektur muss ermöglichen, dass nur rundenbasierte Kämpfe stattfinden.
P5.2: Waffenverhalten			
Es muss entschieden werden auf welche Sektion des Gegners die Waffe schießen soll.	Keine Flexibilität oder Veränderlichkeit, da Mindestanforderungen bestehen.	- -/ - -	Die Architektur muss ermöglichen, dass entschieden werden kann auf welche Sektionen die Waffen schießen.
P5.3: Verteilen von Besatzungsmitgliedern			
Während eines Kampfes kann die Position der Besatzung auf dem Schiff verändert werden. Der Sektions-wechsel wird einige Zeit (Runden) in Anspruch nehmen.	Nur flexibel in der Art der Umsetzung. Keine Veränderlichkeit, da Mindestanforderungen bestehen.Wir können aber weitere Regeln aufstellen für die Verteilung.	- -/ - -	Hat Auswirkungen auf die Umsetzung der Sektionen, auf die Funktionsfähigkeit der darin enthaltenen Systeme und auf die möglichen Spielzüge eines Spielers

3.2 Probleme und Strategien

Problem 1: Gruppenmitglieder fallen weg
Beschreibung: Während des Projekts kann es dazu kommen, dass Gruppenmitglieder die Gruppe verlassen müssen oder eigenständig verlassen oder das Gruppenmitglieder temporär ausfallen. Das würde dazu führen, dass der Umfang des Projekts/Mindestanforderungen angepasst werden würden.
Einflussfaktoren: O1.1: Time-To-Market O1.2: Architektur-Abgabe O1.3: Entwickler
Strategien: S-1: Modularisierung S-2: Pair-Programming
Entscheidung: S-1

Problem 2: Zeitliche Unterschätzung des Aufwands
Beschreibung: Aufgrund von fehlender Erfahrung kann es dazu kommen, dass wir die Aufgaben und deren zeitlichen Aufwand unterschätzen. Das kann dazu führen, dass wir die Abgabetermine nicht einhalten können.
Einflussfaktoren: O1.1: Time-To-Market O1.2: Architektur-Abgabe O1.4: Fähigkeiten Entwickler
Strategien: S-1: Kein Fokus auf Mindestanforderungen. Features und Mindestanforderungen zeitgleich implementieren S-2: Fokus auf Mindestanforderungen und falls Zeit übrig bleibt, dann erst Features bearbeiten
Entscheidung: S-2

Problem 3: Covid-19
Beschreibung: In Zeiten der Corona-Krise können keine persönlichen Treffen stattfinden, aufgrund des Kontaktverbots und somit entfallen Teambuildung-Events und persönliche Treffen mit dem Tutor/Kunden.
Einflussfaktoren: O1.5: Teamarbeit in Corona-Zeiten
Strategien: S-1: Modularisierung S-2: Client-Server-Architektur
Entscheidung: S-1 und S-2

Problem 4: Missverständnisse bei Mindestanforderungen
Beschreibung: Die Gruppe entwickelt eine falsche Anforderung, aufgrund von Missverständnissen.
Einflussfaktoren: O1.1: Time-To-Market P1: Produktfaktoren
Strategien: S-1: Evolutionäre Prototypen entwickeln S-2: Inkrementelle Auslieferungen des Produkts S-3: Verschiedene Akzeptanztests durchführen
Entscheidung: S-1

Problem 5: Fehlende Erfahrung mit den Technologien
Beschreibung: Es kann sein, dass ein Teil oder die ganze Gruppe keine Erfahrungen in der Arbeit mit den vorgegebenen Technologien besitzen und somit in Verzug kommen mit den Abgabeterminen. Das kann zu mangelhaften Implementierungen führen.
Einflussfaktoren: O1.1: Time-To-Market O1.4: Fähigkeiten Entwickler T1: Technik
Strategien: S-1: Modularisierung. Einzelne Gruppenmitglieder fokussieren sich auf einzelne Technologien und schaffen sich Expertise und geben das Wissen weiter an andere Gruppenmitglieder S-2: Vorzeitiges finales Festlegen auf bestimmte Technologien um Änderungen der Architektur zu einem späteren Zeitpunkt zu vermeiden S-3: Entwicklung mit Technologien mit denen man schon Erfahrungen hat
Entscheidung: S-1

Problem 6: Persistenz der Daten
Beschreibung: Es muss eine Persistenz der Daten gewährleistet werden, damit Daten nicht verloren gehen bei beispielsweise Serverneustart und Spieler können ihre Spiele fortsetzen.
Einflussfaktoren: T1.5: Persistenz P1.1: Spiel unterbrechen und Spielstand speichern
Strategien: S-1: Aufbau einer Datenbankverbindung für Schreib-/Lese-Aktionen und danach Schließung der Datenbankverbindung S-2: Datenbankzugriffe gekapselt in der Komponente Persistence S-3: Externes Datenbankmanagementsystem
Entscheidung: S-2

Problem 7: Überlastung des Servers
Beschreibung: Der Server muss ermöglichen, dass mindestens zwei verschiedene Spieler das Spiel gleichzeitig spielen können.
Einflussfaktoren: T1.3: Client-Server-Architektur P1.3: Multiplayer
Strategien: S-1: Teilung der Anwendung in eine Client-Server-Architektur S-2: Client und Server sind enthalten im selben Programm S-3: Multithreaded Server implementieren
Entscheidung: S-1

Problem 8: Die Internet-Verbindung von einem oder beiden Spielern bricht ab im Multiplayer-Modus
Beschreibung: Wenn das Spiel im Multiplayer-Modus gespielt wird und ein oder beide Spieler zeitweise keine Internet-Verbindung haben, dann soll es ermöglicht werden, dass die Spieler ihr Spiel fortsetzen können bei hergesteller Internet-Verbindung.
Einflussfaktoren: T1.3: Client-Server-Architektur T1.5: Persistenz P1.3: Multiplayer P1.1: Spiel unterbrechen und Spielstand speichern
Strategien: S-1: Spielstände werden, in vorher festgelegten Zeitabständen, gespeichert um dann, nachdem die Internet-Verbindung wieder hergestellt wurde, das Spiel fortzusetzen S-2: Währendessen wird das Spiel pausiert und die Spieler kriegen darüber einen Bescheid sobald die Internet-Verbindung einseitig abbricht S-3: Das Spiel wird abgebrochen und der Gegenspieler gewinnt die Partie
Entscheidung: S-2

Problem 9: Die Internet-Verbindung bricht ab bei dem Spieler bei Einzelspieler-Modus
Beschreibung: Wenn das Spiel im Einzelspieler-Modus gespielt wird und der Spieler zeitweise keine Internet-Verbindung hat, dann soll es ermöglicht werden, dass das Spiel fortgesetzt werden kann, wenn die Internet-Verbindung wieder hergestellt wurde.
Einflussfaktoren: T1.3: Client-Server-Architektur T1.5: Persistenz P1.1: Spiel unterbrechen und Spielstand speichern
Strategien: S-1: Spielstände werden, in vorher festgelegten Zeitabständen, gespeichert um dann, nachdem die Internet-Verbindung wieder hergestellt wurde, das Spiel fortzusetzen S-2: Das Spiel wird pausiert bis die Internet-Verbindung wieder hergestellt wurde S-3: Das Spiel wird abgebrochen und der Gegenspieler gewinnt die Partie
Entscheidung: S-2

Problem 10: Komplexität der Implementierung
Beschreibung: Die Komplexität des Software und die Projektstruktur kann schnell unübersichtlich werden, wegen der Relationen zwischen den unterschiedlichen Objekten der Implementierung.
Einflussfaktoren: T1.7: Wartbarkeit
Strategien: S-1: Modulariserung der Software S-2: S.O.L.I.D, Prinzipien des OOP Designs
Entscheidung: S-1

Problem 11: Spielstände
Beschreibung: Die Spielstände müssen den jeweiligen Spielern zugeordnet werden. Spieler A kann das Spiel von Spieler B (weiter-)spielen.
Einflussfaktoren: T1.3: Client-Server-Architektur T1.5: Persistenz P1.1: Spiel unterbrechen und Spielstand speichern
Strategien: S-1: Jeder Spieler kann eigenen Account mit Passwort erstellen und die Spiele, die dieser Spieler gestartet hat, werden diesem Spiel zugeordnet S-2: Die Spielstände lokal speichern S-3: Den Spielstand als beispielsweise CSV-Datei abspeichern und exportieren und bei Fortsetzung des Spiels wieder importieren und weiter spielen
Entscheidung: S-1

Problem 12: Der Spieler kennt den Spielverlauf und die Spielregeln nicht
Beschreibung: Der Spieler spielt das Spiel zum ersten Mal und ist mit der Benutzeroberfläche überfordert und möchte möglicherweise erst die Spielregeln kennen bevor das Spiel gestartet wird.
Einflussfaktoren: P1: Produktfaktoren
Strategien: S-1: (Video-)Tutorial mit Stimme/Untertitel im Menü S-2: Anleitung während des Spiels mit Quick-Instructions
Entscheidung: S-1

4 Konzeptionelle Sicht

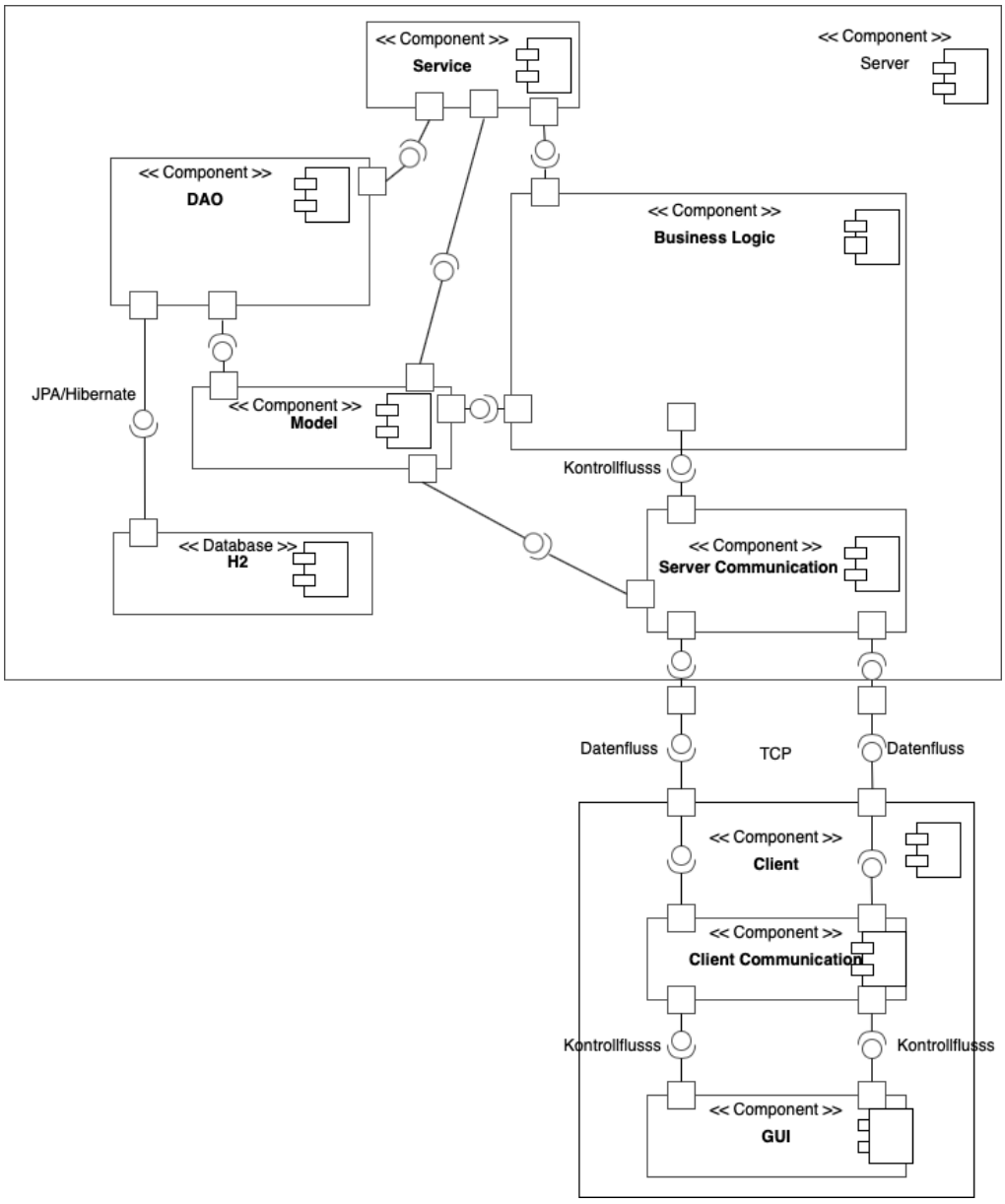

Abbildung 1: Konzeptionelle Sicht

In diesem Kapitel verwenden wir die konzeptionelle Sicht nach Hofmeister et al., um das System auf einer hohen Abstraktionsebene zu beschreiben. Das Computerspiel "Even Geiler Than Geiler Than Faster Than Light" wird durch eine Client-Server-Architektur realisiert.

H2: Die leichtgewichtige relationale Datenbank wird für die Speicherung von Daten und zum Persistieren unseres Moduls genutzt. Die Datenbank kommuniziert über

JPA/Hibernate mit dem DAO.

DAO: Das Data Acess Object ist ein Muster für die Gestaltung von APIs. Wir nutzen dieses Entwurfsmuster, um den Zugriff an die H2 Datenbank zu binden. Somit wird die Möglichkeit offen gehalten, die Datenbank auszutauschen, ohne den Code zu ändern. Außerdem wird die Programmlogik wird von den technisches Details der Datenspeicherung getrennt. Es nutzt die Schnittstellen zur Datenbank, zum Model und zur Service Komponente.

Business Logic: Die Business Logic Komponente beinhaltet die Funktionalität des Systems. In unserem Fall prüft sie die von den Clients übermittelten Aktionen auf Einhaltung der Regeln und Plausibilität. Sie nutzt die Schnittstellen Service, Model und Server Communication. Über die Schnittstelle zu Service kann diese Komponente schreibende und lesende Zugriffe ausführen. Hierbei haben wir die Strategie der Modularisierung umgesetzt (P1 Strategie 1).

Model: Das Model repräsentiert die Entities, mit all deren Eigenschaften und Funktionen, die wir definiert haben. Jeder Entity wird eine Tabelle in der Datenbank zugeordnet. Das Model hat jeweils eine Schnitstelle zu den Repositories, Service, Business Logic und Server Communication Komponente.

Service: Über diese Komponente werden lesende und schreibende Zugriffe auf persistente Daten in der Datenbank durchgeführt. Sie kapselt die Funktionen, die benötigt werden, um auf diese Daten zuzugreifen, sodass dann nur diese Komponente lesende und schreibende Aktionen durchführen kann. Auch hierbei haben wir die Strategie der Modularisierung umgesetzt (P1 Strategie 1).

Server Communication: Der externe Server kommuniziert mit dem Client über eine bidirektionale TCP-Verbindung. Hierbei haben wir die Strategie der Client-Server-Architektur umgesetzt (P7 Strategie 1). Des Weiteren übermittelt diese Komponente Aktionen vom Client an die Business Logic Komponente, um dort diese auf Regeln und Plausibilität zu untersuchen.

Client Communication: Diese Komponente stellt die Kommunikation zwischen des Clients und dem Server sicher und übermittelt Aktionen die von der GUI kommen an den Server und umgekehrt (P7 Strategie 1).

GUI: Die GUI registriert die Interaktionen, die zwischen dem Benutzer und der Anwendung aufkommen und repräsentiert die dafür notwendigen grafischen Symbole und Steuerelemente. Die Eingaben werden an die Client Communication weitergeleitet.Die Antwort des Servers wird über die Client Communication an die GUI geschickt und dort visualisiert . Diese Verbindung ist also bidirektional.

5 Modulsicht

5.1 Model

Folgende Klassen stellen die Datensicht dar. Jeder Klasse repräsentiert eine Entity in der Datenbank und seine Beziehungen. Einträge für diese Daten werden zum DAO-Modul umgeleitet und schließlich zur Datenbank gespeichert.

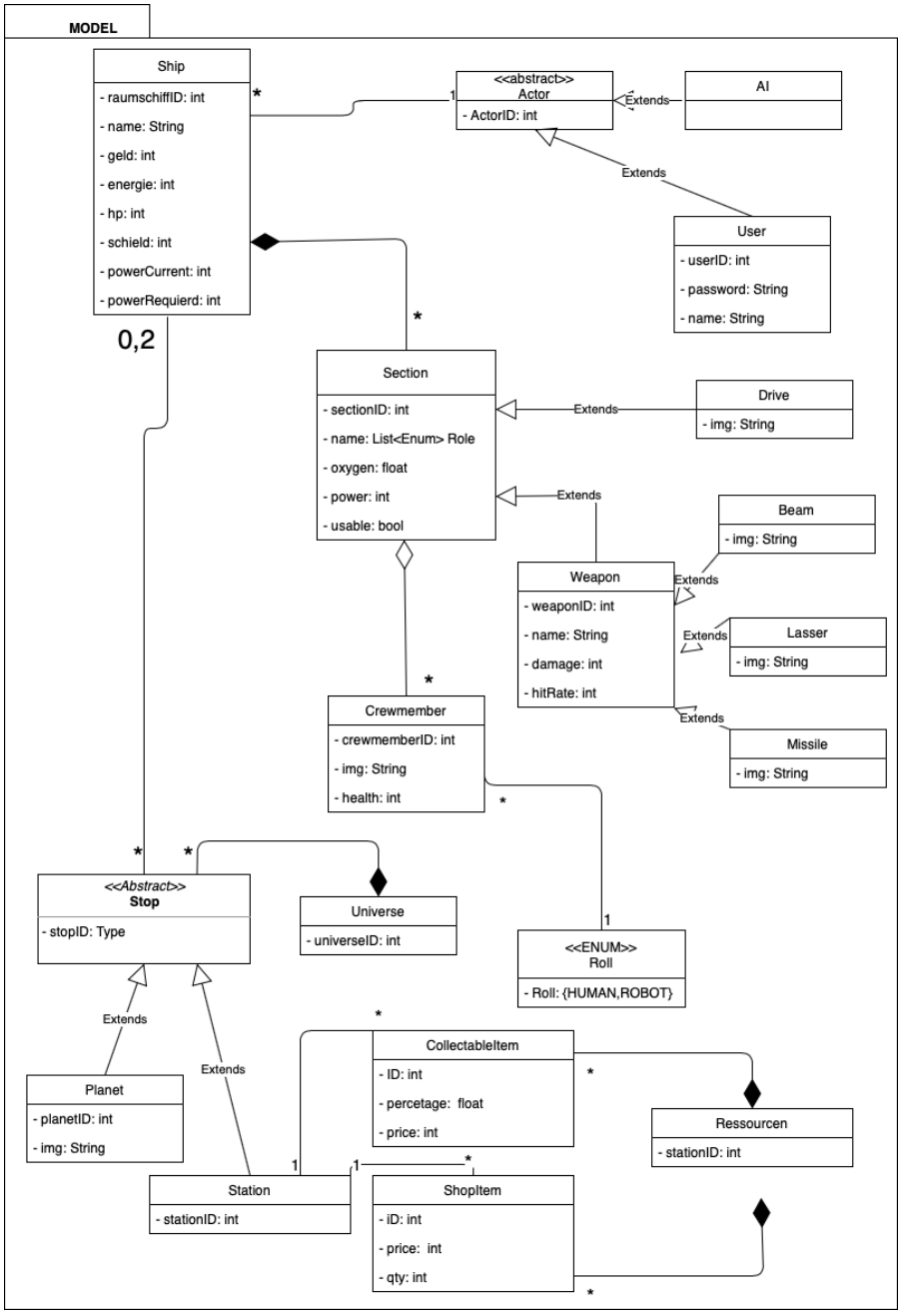

Abbildung 2: Model

5.2 Services

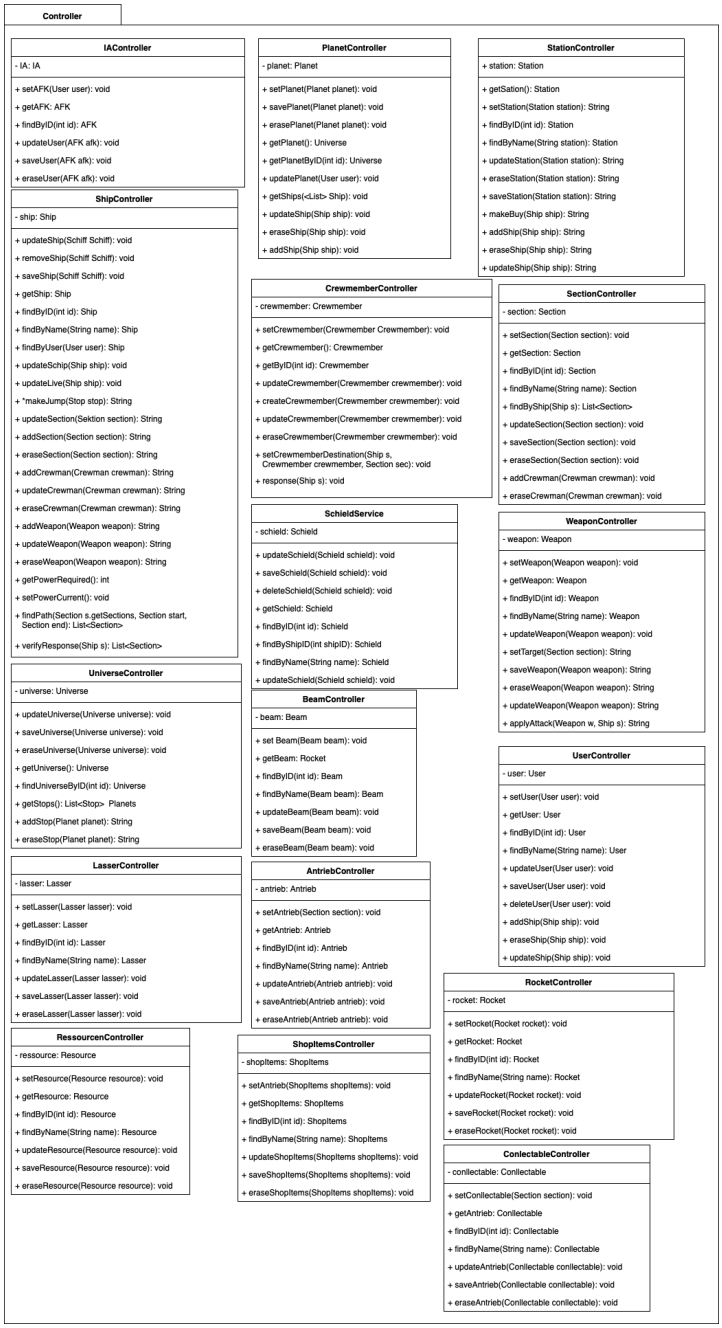

Abbildung 3: Service

5.3 Repository

Diese Klassen Repositories ermöglichen eine direkte Verbindung zur Datenbank und die Ausführung von Querry-Anfragen. Die Klassen implementieren Hibernate als Java Persistence API (JPA). Der Aufruf der Klassen erfolgt hauptsächlich über das Modul Service. Der Treiber und der Datenbank-Dialekt wird im Server in der Datei *applications.properties* konfiguriert.

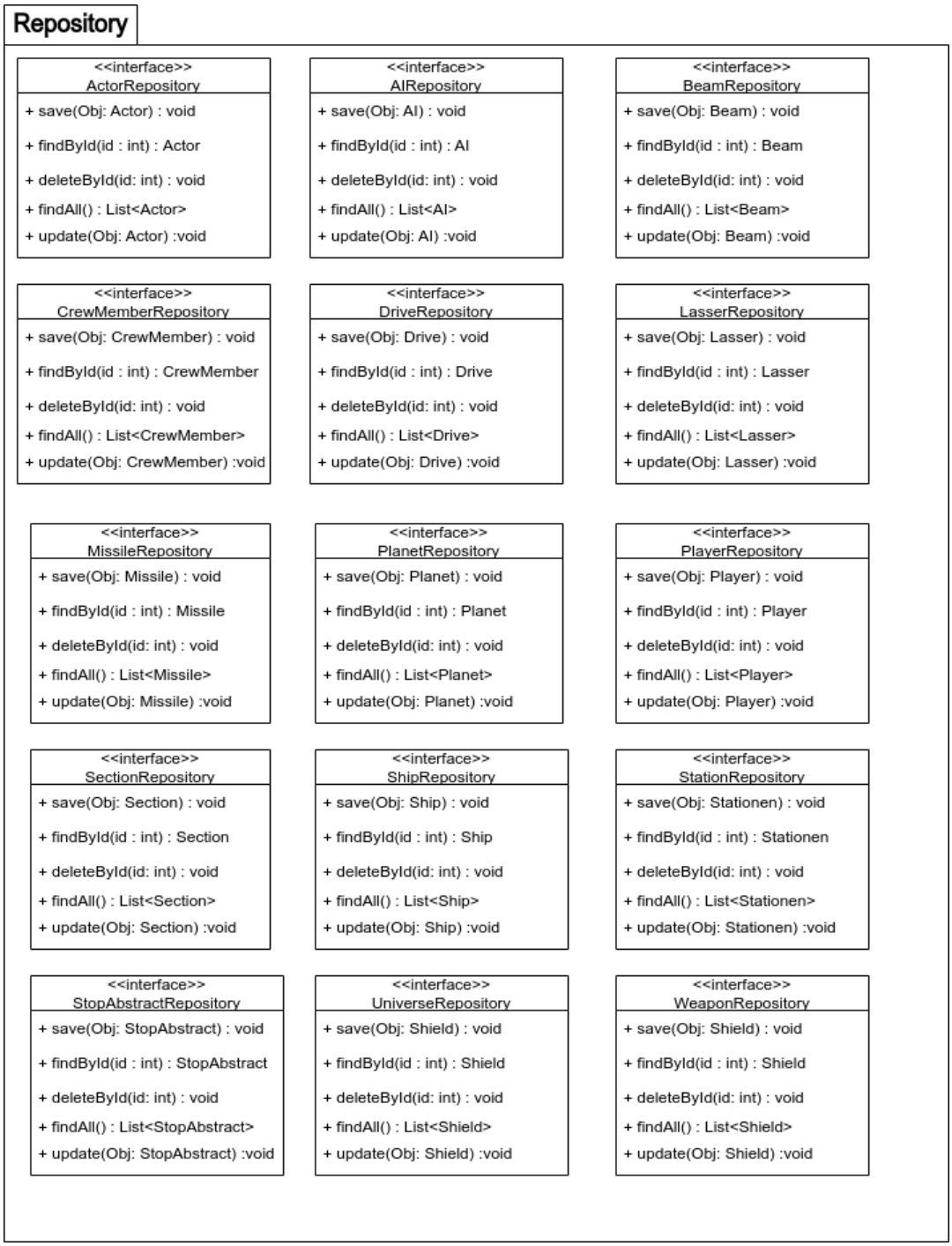

Abbildung 4: Repository

5.4 Zustandsdiagramm

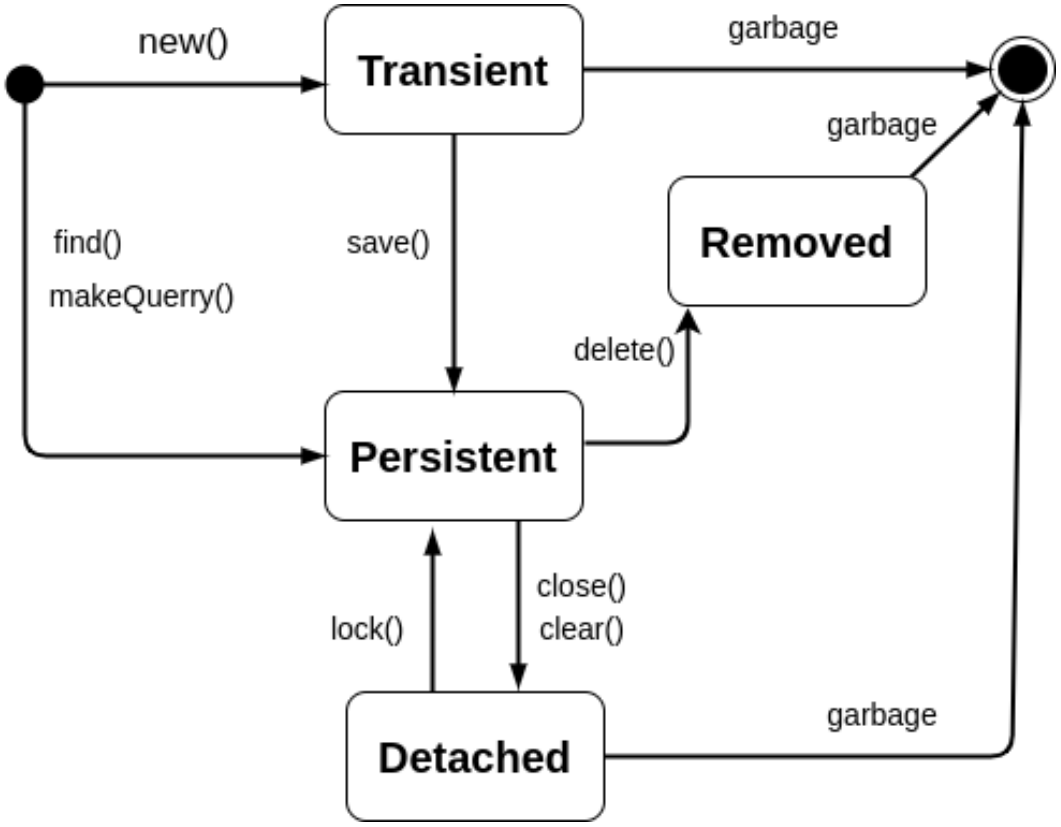

Abbildung 5: Zustandsdiagramm [1] http://www.eurolink.ch/jpa/ObjektZustandsDiagramm.pdf

6 Datensicht

Das Datenmodel ist so aufgebaut, dass Universe der Root-Knoten in unserem Daten Baum ist. Alles ist darin enthalten. Das Raumschiff Ship kann an Planeten oder an Stationen halten. An Stationen kann es einen Markt geben, sowie Collectable Item (Ressourcen), diese können benutzt werden, um das Raumschiff zu verbessern. Ein Schiff besteht aus Sectionen. Die Sectionen (Knoten) sind durch durchgängen (Kanten) miteinander Verbunden. Das Schiff ist also ein bidirektionaler Graph. Spezielle Abeiteilungen haben zusatzfähigkeiten. Hierzu gehört: Drive (Antrieb) und Weapon (Waffen). Die Waffen haben zusatz Attribute wie hitRate und dammage. CrewMember haben Fähigkeiten, die in Ihrer Rolle definiert sind und können sich in verschiedenen Sektionen aufhalten.

Das Schiff gehört einem Actor. Dein Actor ist endweder eine KI oder ein menschlicher Spieler.

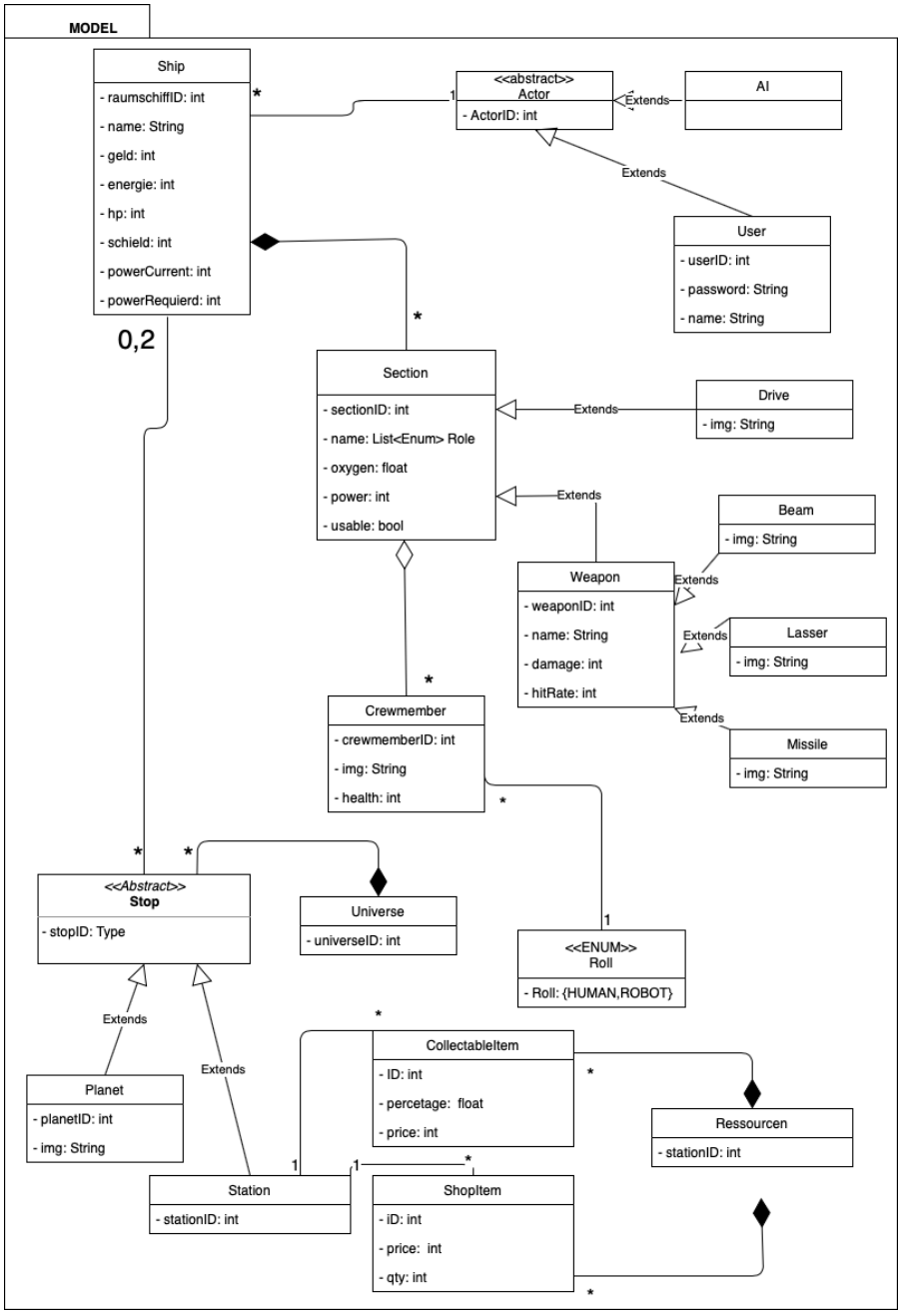

Abbildung 6: Datensicht

7 Ausführungssicht

In diesem Absatz wird die Ausführungssicht des Systems beschrieben. Diese enthält die Kommunikation bzw. die Interaktion der Komponenten und auch deren Hostrechner, auf denen die Komponenten laufen.
Die untenstehende Abbildung dient als Ausführungssicht für das Spiel "Even Geiler Than Geiler Than Faster Than Light" . Das Spiel wird mit dem Client-Server Architektur Pattern realisiert.

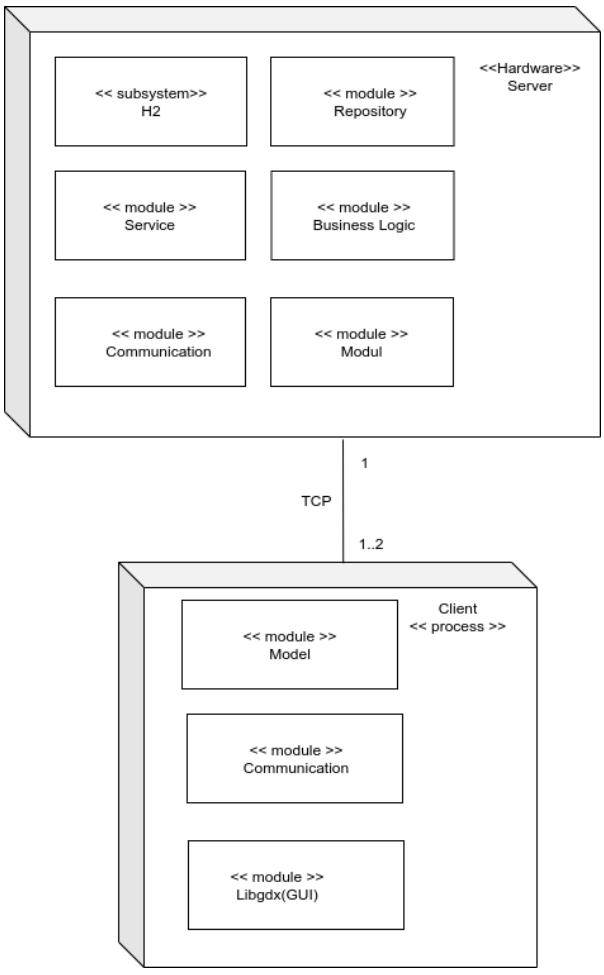

Abbildung 7: Ausführungssicht

Die Abbildung besteht aus zwei geteilten Boxen. Der oberste Teil repräsentiert den Server und enthält mehrere Module sowie Service, Datenbank(H2), Business Logic, Communication und Repository. Der Server kommuniziert durch TCP mit dem Client. Es können ein bis zwei Clients bzw. Spieler gleichzeitig mit dem Server kommunizieren. Der Server stellt die Verbindung mit der Datenbank sicher, indem er über das Service-Modul interagiert. Dadurch, dass wir Daten speichern müssen benötigen wir eine Datenbank. Mit Hilfe des Repository-Moduls werden die Daten persistiert und verwaltet. Die ausgewählte leichtgewichtige Datenbank(H2) wird mit dem Anwendungsserver auf dem selben Gerät laufen.
Der untere Teil repräsentiert den Client und besteht aus drei Modulen wie Model, Communication und GUI. Der Client wird durch TCP den Server ansprechen. Der Client interagiert mit der GUI, diese Eingaben werden dann über die Client Communication weiter an die Server Communication.

8 Zusammenhänge zwischen Anwendungsfällen und Architektur

8.1 Kampfrunde

In der Abbildung **??** ist zu entnehmen, wie Kämpfe ablaufen werden.

Als erstes kann der Spieler seine vorhandene Energie, falls er mag umverteilen. Falls er durch einen Angrif Energie verloren hat, so kann er jetzt den Sectionen die für in wichtig sind mit Strom versorgen. Der Spieler kann seinen Crew an neue Orte endsenden. Ihnen wird ein Pfad zur Ihrem Ziel berechnet. Als dann laufen diese dann los. Am Ziel angekommen führen sie Tätigkeiten aus, welche in der Section möglich sind.

Die mit Loop gekenzeichneten Kästchen sind Schleifen. Diese werden nicht oder mehrmals durchlaufen. Die Reihenfolge der Ausführung wird vom Spieler festgelegt.

Spätestens, wenn der Spieler keine Actionen durchführen kann, wird er die Kampfrunde beenden.

Jeder der drei Loops sind Eigenständige Sequenzen die in hier in einem Diagramm zusammengefasst worden sind.

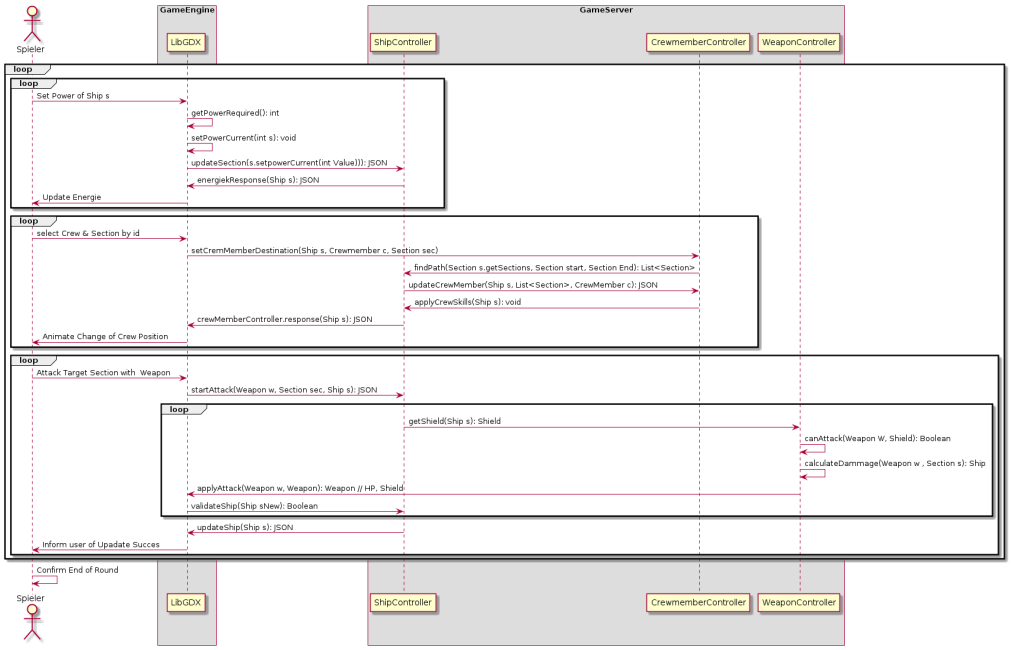

Abbildung 8: Kampf Runde

9 Evolution

Da das System als Projektarbeit der Veranstaltung Software-Projekt 2 entwickelt wird, ist nicht zu erwarten, dass das System nach Auslieferung des Systems erweitert oder abgeändert wird.

- Datenbank
  Bei der Weiterentwicklung des Spiels kann es dazu kommen, dass wir die leichtgewichtige relationale Datenbank austauschen müssen, aufgrund einer höheren Anzahl von Spielern. Da wir das DAO Entwurfsmuster benutzen lässt sich die Datenbank problemlos austauschen, ohne den Code zu ändern.
- Chat-Funktion
  Für die Echtzeit-Kommunikation zwischen Spielern kann das System auch durch eine Chat-Funktion erweitert werden. Dafür wird das WebSocket-Protokoll verwendet um eine bidirektionale Verbindung zwischen Clients und Server aufzustellen. Aufgrund unseres modularen Architekturaufbaus kann man diese Funktion leicht einbinden.
- Kompatibilität mit anderen Geräten
  Wenn die Anwendung auch auf anderen Geräten/Systemen laufen soll, dann kann man die Anwendung kompatibel machen zu anderen Geräten. Es müssten nur einige oberflächliche Optimierungen durchgeführt werden. Dies würde jedoch nur eine geringfügige Änderung an der Architektur haben.
- Highscore

Eine Highscore-Seite, die aufzeigt wie viele Punkte der Spieler oder alle Spieler erreicht haben ist keine Mindestanforderung, aber kann im Rahmen der Weiterentwicklung einfach umgesetzt werden. Diese Funktion hat eine direkte Auswirkung auf die Architektur. Eine neue Variable einsetzen in der Klasse User um durch den Spielverlauf die Variable zu verändern.

- Mehr als Zwei Spieler
  Unser Spiel ist für 2 Menschliche Spieler ausgelegt. Unser Datenmodel ist nicht auf 2 Spieler begrentzt. Intern ist die KI eine Implementierung der Schnitstelle Actor. Falls später Bedarf besteht, ist es Möglich ohne weitgehende Veränderung im Model mehr als 2 Menschliche Spieler zu unterstützen. Die größten Anpaßungen würden u.A. in dem Runden ablauf passieren. Alle festen Variablen, welche für 2 Spieler gelten, wurden angepasst werden müssen, so dass diese Dynmaisch sind. Sprich sich der Anzahl an Spieler anpassen. Es wäre wichtig genau darauf zu achten, dass es zu keinen Deadlocks kommt, welche wegen der erhöhten Spieler Anzahl geschehen könnten.