

Projekt: Rupert  
  
Software Architektur Dokument

Daunicht Nils (ndaunich@hsr.ch)

Fröhlich Adrian (a1froehl@hsr.ch)

Hochreutener Joel(jhochreu@hsr.ch)

Peisker Dominic(dpeisk@hsr.ch)

# Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Änderung | Autor |
| 30.03.2015 | 0.0 | Erstellung | Dominic Peisker |
| 02.04.2015 | 0.1 | Ergänzung | Dominic Peisker |
| 16.04.2015 | 1.0 | Einfügen Grafiken und Korrekturen | Dominic Peisker |
| 20.04.2015 | 1.1 | Anpassungen und Ergänzungen bei der logischen Architektur. | Nils Daunicht |
| 13.05.2015 | 2.1 | Anpassung Systemübersicht auf aktuellen Stand. | Nils Daunicht |
| 18.05.2015 | 2.0 | Packagediagram um aktuelles Ergänzt | Nils Daunicht |
| 19.05.2015 | 2.1 | Logische Architektur angepasst | Nils Daunicht |
| 27.05.2015 | 2.2 | Letzte Anpassungen | Nils Daunicht |

# Inhalt

[Änderungsgeschichte 2](#_Toc420660981)

[Inhalt 3](#_Toc420660982)

[1. Einführung 5](#_Toc420660983)

[1.1 Zweck 5](#_Toc420660984)

[1.2 Gültigkeitsbereich 5](#_Toc420660985)

[1.3 Übersicht 5](#_Toc420660986)

[2. Systemübersicht 6](#_Toc420660987)

[2.1 Allgemein 6](#_Toc420660988)

[2.2 Client 6](#_Toc420660989)

[2.3 Shared 6](#_Toc420660990)

[2.4 Server 6](#_Toc420660991)

[3. Design 7](#_Toc420660992)

[3.1 Homescreen 7](#_Toc420660993)

[3.2 Lobby 8](#_Toc420660994)

[3.3 Serverlobby 9](#_Toc420660995)

[3.4 Startphase 10](#_Toc420660996)

[3.5 Bewegungsphase 10](#_Toc420660997)

[3.6 Kampfphase 11](#_Toc420660998)

[3.7 UI Architektur 12](#_Toc420660999)

[4. Architektonische Ziele & Einschränkungen 13](#_Toc420661000)

[4.1 Security 13](#_Toc420661001)

[4.2 Tools 13](#_Toc420661002)

[4.3 Client-Server-Kommunikation 13](#_Toc420661003)

[4.4 Implementierung 13](#_Toc420661004)

[5. Logische Architektur 14](#_Toc420661005)

[5.1 client 15](#_Toc420661006)

[5.2 client.ui 15](#_Toc420661007)

[5.3 client.application 15](#_Toc420661008)

[~~5.4 client.messaging~~ 15](#_Toc420661009)

[5.5 client.domain 15](#_Toc420661010)

[~~5.6 client.resources~~ 16](#_Toc420661011)

[5.7 server 16](#_Toc420661012)

[5.8 server.domain 16](#_Toc420661013)

[~~5.9 shared.JMS~~ 16](#_Toc420661014)

[5.10 shared.com 16](#_Toc420661015)

[~~5.11 shared.domain~~ 16](#_Toc420661016)

[5.12 shared.game 16](#_Toc420661017)

[5.13 shared.resource 16](#_Toc420661018)

[6. Klassendiagramm 17](#_Toc420661019)

[7. Deployment 17](#_Toc420661020)

[8. Datenspeicherung 18](#_Toc420661021)

[9. Grössen und Leistung 19](#_Toc420661022)

# Einführung

## Zweck

Das Dokument beschreibt die Architektur unseres Projektes. Mit diesem Dokument soll der Aufbau des Projektes verständlich sein und die Implementierung vorgenommen werden können.

## Gültigkeitsbereich

Das Dokument ist während des SE2 Projektes im Frühlingsemester 2015 und im Zusammenhang mit der Arbeit am Projekt Rupert gültig.

## Übersicht

Zuerst wird im Kapitel Systemübersicht eine Übersicht und Einführung in die Architektur des Projektes gegeben. Diese Übersicht wird im folgenden Kapitel verfeinert und mit Zielen und Einschränkungen ergänzt. Das Kapitel logische Architektur stellt danach die gesamte logische Architektur der Software detailliert dar. Im Kapitel Deployment wird anhand eines Deployment Diagrammes aufgezeigt, wie unsere Software auf den verschiedenen Geräten verteilt sein wird. Weiter wird aufgezeigt, wie wir die Speicherung der permanenten Daten realisieren werden. Zum Schluss werden noch Angaben über die Grösse und Leistung der Software gemacht.

# Systemübersicht

## Allgemein

Um das Projekt übersichtlich zu halten und nicht Code, welcher sowohl Client als auch Server verwenden, doppelt schreiben zu müssen, teilen wir unser Projekt in drei Hauptpackete auf. Diese heissen Client (enthält die Client-Logik), Server (enthält die Serverlogik) und Shared, welches die von beiden benötigte Logik enthält.

|  |  |  |
| --- | --- | --- |
| Client | Shared | Server |
| Präsentation Layer |  |  |
| Application Layer | Application Layer |  |
| Domain Layer | Domain Layer | Domain Layer |
|  | Data Layer |  |

Im Projekt root liegt ein Resource-Ordner „img“. Dort liegen sämtliche Bilder die im Projekt benötigt werden.

## Client

Der Client besteht aus drei verschiedenen Schichten. Der Präsentation Layer, später auch UI genannt,ist für die Interaktion mit dem Benutzer verantwortlich. Auf dem Application Layer werden die Informationen aus dem Präsentation Layer verarbeitet und das Messaging betrieben. Der Domain Layer beinhaltet die Business Logik des Clients.

Beinhaltet den Einstiegspunkt ins Program des Clients (main).

## Shared

Das Package Shared beinhaltet einen Apllication-, Domain– und ein Data Layer. Im Application Layer befindet sich eigentlich nur JMS-Logik, welche für das messaging benötigt wird. Der Domain Layer beinhaltet Datenklassen, die vom Client und Server benötigt werden. Im Data Layer Befinden sich Klassen, welche auf Bilder aus dem „img“ Ordener zugreifen.

## Server

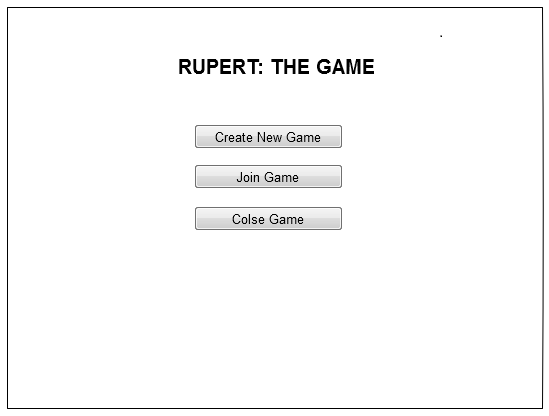
Der Server besteht nur aus dem Domain Layer, in dem sich die Spiellogik und die Steuermechanismen des Spiels befinden.

Beinhaltet den Einstiegspunkt ins Program des Servers (main).

# Design

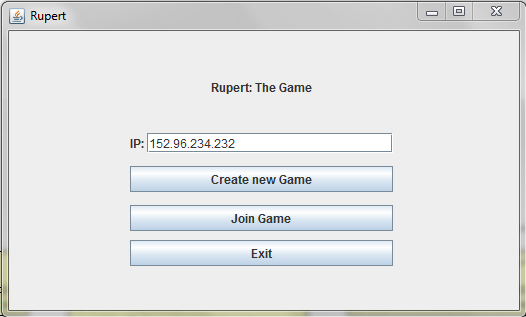
## Homescreen

Stand 16.04



Auf Grund des Wegfallens der Lobby, ist es nötig die IP direkt anzugeben. Daher wird ein Textfeld hinzugefügt.

Implemented



## Lobby

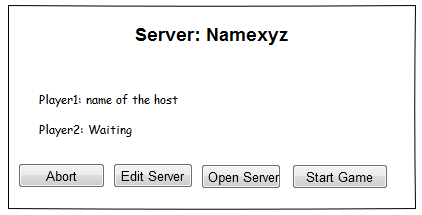
Die Lobby wird auf Grund der zu knapp bemessenen Zeit in dieser Version nicht implementiert und fällt weg. Sie wird durch direkte Eingabe der IP ersetzt (siehe Homescreen).

Stand 16.04



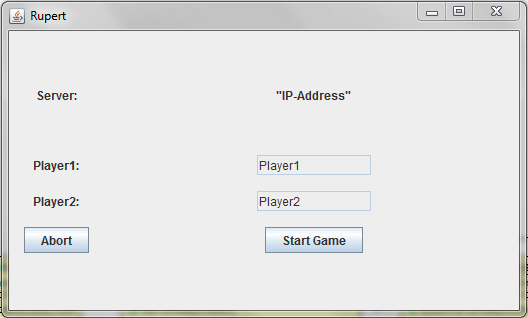
## Serverlobby

Stand 16.04



Das Ändern der Server Parameter wird in dieser Version nicht implementier und wird deshalb im GUI nicht angeboten. „Open Server“ fällt weg, da die Lobby nicht implementiert wird (siehe Lobby).

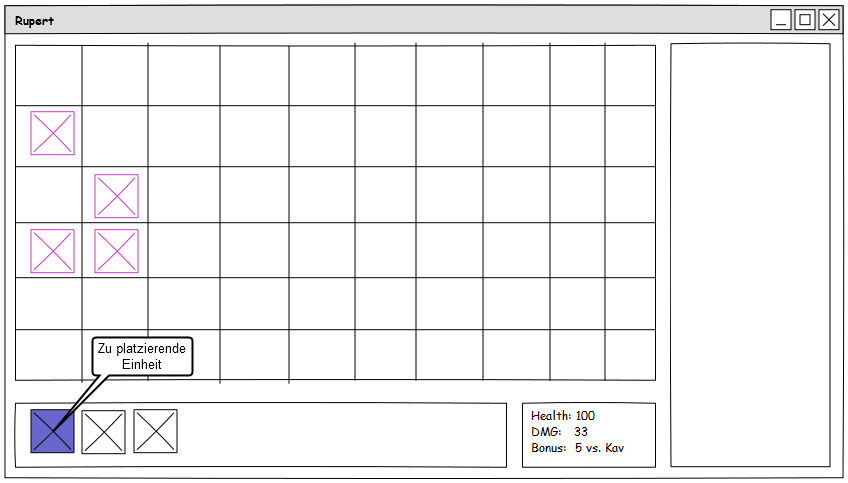
Implemented



## Startphase

Die Startphase wird auf Grund von Zeitmangel in dieser Version nicht implementiert und fällt weg. Die Phase wird übersprungen.

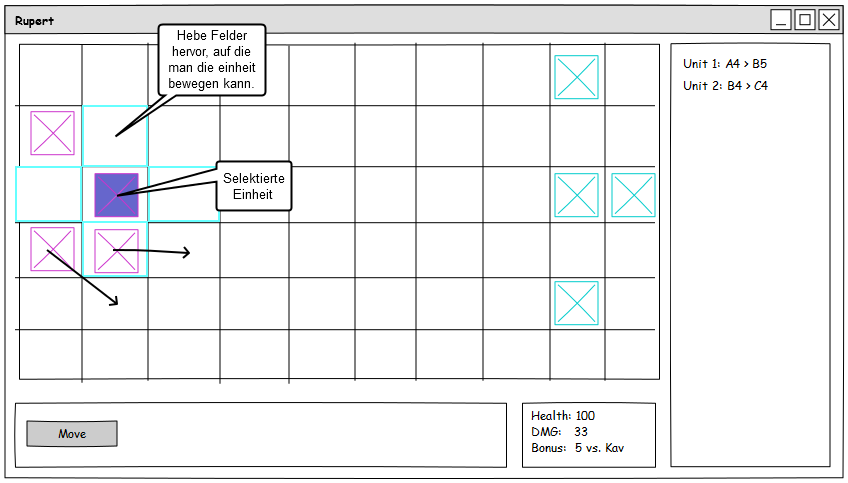
Stand 16.04



## Bewegungsphase

Bewegungen werden durch Mausklick initiiert.

Implemented

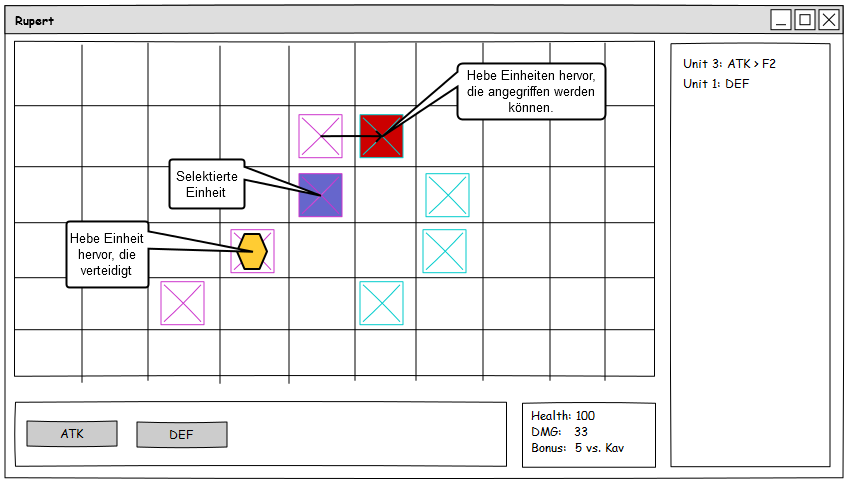


## Kampfphase

Angriffe können nach dem Auswählen einer Einheit mit einem Mausklick auf eine andere initiiert werden.

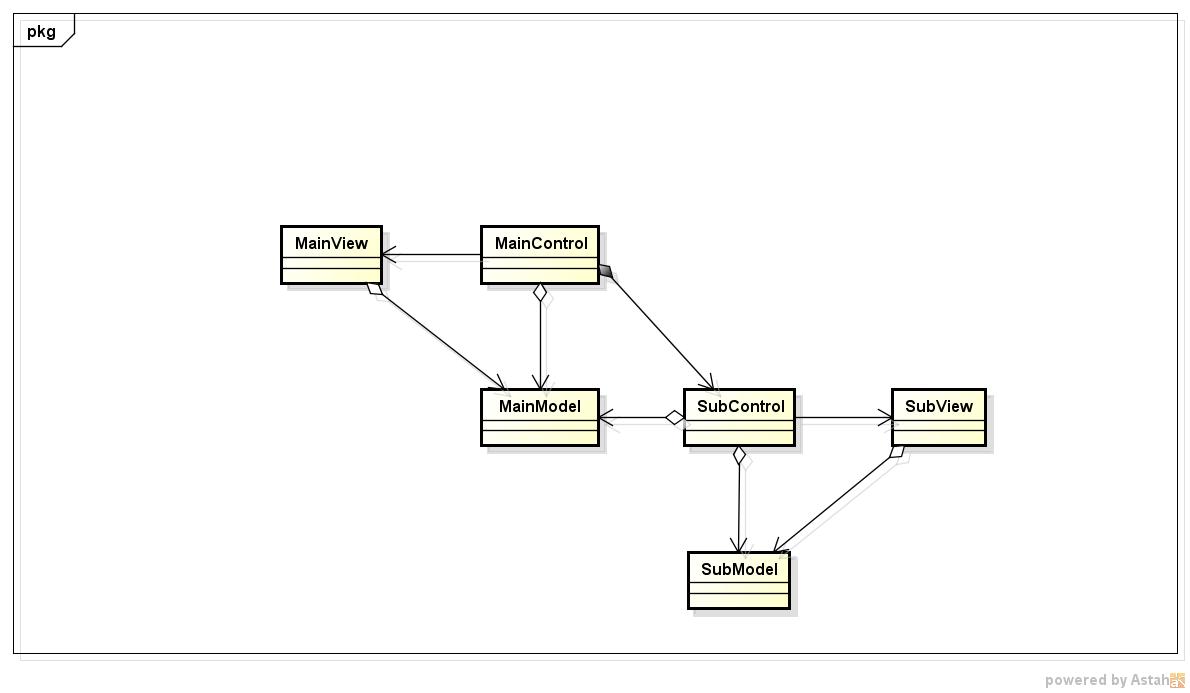
Das hervorheben von angreifbaren Einheiten fällt weg und wird durch das hervorheben aller Felder im Angriffsbereich ersetzt (als Referenz siehe Bewegungsphase).

Implemented



## UI Architektur

Wir setzen auf eine MVC-Implementierung nach folgendem Schema.



Wir werden eine Main View haben mit verschiedenen Sub Views. Die Sub Controller werden vom Main Controller gesteuert. Um den Zugriff auf geteilte Daten zu vereinfachen, können Sub Controls auf das Model der Main Control zugreifen.

Wird nur angewendet um die Daten des Spiels und die Steuerung der Oberfläche sauber zu trennen.

# Architektonische Ziele & Einschränkungen

## Security

Wir haben uns entschieden, bei unserem Projekt nicht auf die Security zu achten. Theoretisch ist es so zwar möglich, dass sich ein Client durch Modifikationen einen unfairen Vorteil verschaffen könnte. Da das Spiel jedoch über ein lokales Netzwerk gespielt wird und die nötigen Securitymassnahmen sehr aufwändig wären. Da es sich um ein Spiel handelt und sich so der anrichtbare Schaden auf maximal eine gewonnene Partie entfällt ist es verkraftbar die Implementation von weiteren Sicherheitsmassnahmen zu vernachlässigen.

Wärend der Implementation wurde festgestellt, dass die Sicherheit vernachlässigbar ist. Es ist maximal möglich seinen eigenen Client in einen korrupten Zustand zu bringen und somit das Spiel ungültig zu machen. Es werden nur festgelegte Werte über das Netzwerk übertragen, durch die kein Schaden beim Mitspieler entstehen.

## Tools

Als Entwicklungstool wird Eclipse verwendet. Für die Realisierung der Client-Server-Kommunikation setzen wir auf JMS mit Active MQ. Das UI wird mit dem Eclipse-Plugin WindowBuilder (http://download.eclipse.org/windowbuilder/WB/release/R201406251200/4.4/) designed. Wir verwenden stan4j (http://update.stan4j.com/ide) zur Überprüfung der Code-Qualität.

## Client-Server-Kommunikation

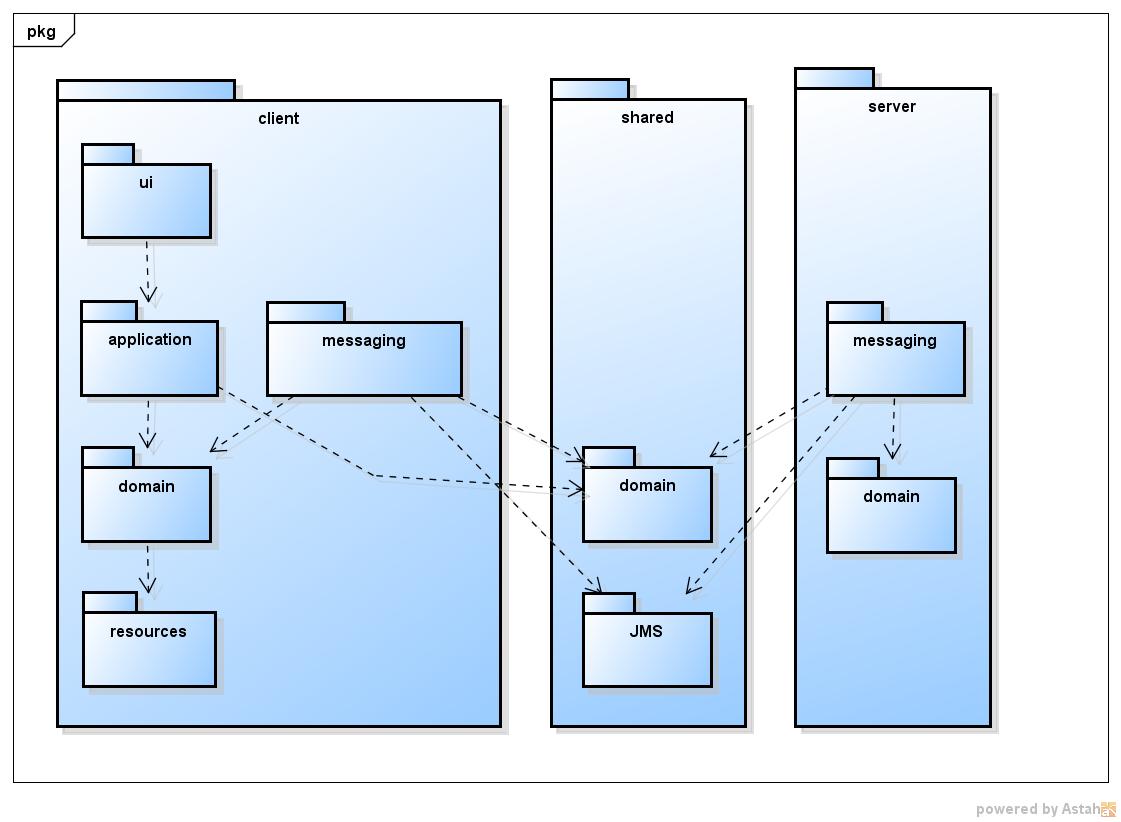
Die Kommunikation zwischen Client und Server soll asynchron sein.

## Implementierung

Wir haben uns entschieden, das Game als ein Java Projekt zu entwickeln. In diesem Projekt sollen sämtliche Daten des Spiels enthalten sein, vom Code bis zu den Grafiken. Am Ende wird ein Einziges JAR-File benötigt pro Client.

# Logische Architektur

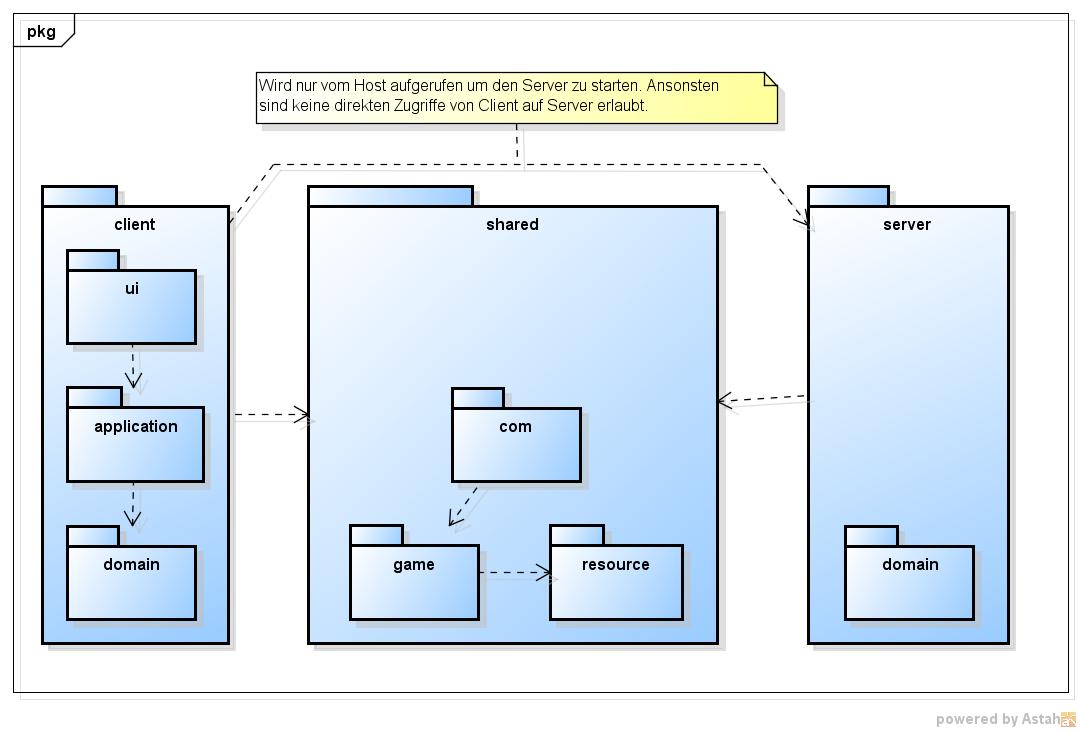
Stand 16.04



* „JMS“ wurde nachträglich in „com“ umbenannt.
* „resources“ wurde in „resource“ umbenannt und in das shared package verschoben.
* „messaging“ wurde aus Client und Server entfernt und im „shared.com“ vereinigt, da die Zugriffe von Server und Client immer die Selben sind.
* „shared.domain“ wurde in „game”, „com“ und „resource“ aufgeteilt. In „game” befindet sich das Gerüst des Spiels.

Genaure Beschreibungen der Sub-packages folgen.

Implemented 5



## client

Hier sollte lediglich der Einstiegspunkt in den Client (main) existieren. ~~Gegebenenfalls könnten hier Konfigurations-Files liegen.~~ Der Client darf genau einen Zugriff auf den Server machen. Dieser wird benötigt um den Server für den Host zu starten.

## client.ui

Ist vor allem für die Umsetzung der Darstellung und der Interaktion mit dem User zuständig. Die enthaltenen Files definieren das Zeichnungsverhalten und die Listener werden generiert.

## client.application

Hier kommen alle Elemente hinein, welche für die Interaktion zwischen UI-Elementen zuständig sind, wie Application-Controller.

## ~~client.messaging~~

~~Hier werden hauptsächlich Klassen anzutreffen sein, die für die Kommunikation zwischen Server und Client benötigt werden.~~

## client.domain

~~Hier werden sich Klassen befinden, welche für die Darstellung benötigt werden, sowie die Lobby-Logik.~~

Hier werden Speicherstrukturen für das UI und Logik für den Client abgelegt.

## ~~client.resources~~

~~Enthält alle Grafiken und allenfalls Konfigurations-Files~~

## server

Hier sollte lediglich der Einstiegspunkt in den Server (main) existieren. ~~Gegebenenfalls könnten hier Konfigurations-Files liegen.~~

## server.domain

Hier werden Speicherstrukturen, welche der Client nicht sehen soll und Logik des Servers abgelegt.

## ~~shared.JMS~~

~~Ist das Adapter-Pattern für JMS.~~

## shared.com

Hier liegen Klassen, welche vom JMS benötigt werden oder zur Kommunikation zwischen Server und Client benötigt werden.

## ~~shared.domain~~

~~Hier werden primär übertragbare Speicherstrukturen zum Datenaustausch zwischen Client und Server abgelegt.~~

## shared.game

Dieses Paket beinhaltet die gesamte Spielstruktur, die dem Client und dem Server bekannt sein sollten.

## shared.resource

Hier befinden sich Klassen, die Zugriff auf im Projekt abgelegte Bilder benötigen und im Projekt benötigte Klassen und Enums welche keine weitere Logik anbieten.

# Klassendiagramm

Die Klassendiagramme werden mit dem eclipse Plugin ObjectAid (<http://www.objectaid.net/update>) erstellt. Um die Übersicht zu bewahren sind die Diagramme als .jpg abgelegt.

Sub Packages werden bis zum package-Scope visualisiert.

Main Packages werden nur im public-Scope visualisiert.

## client

<classdiagram_client.jpg>

## client.ui

<classdiagram_client_ui.jpg>

## client.application

<classdiagram_client_application.jpg>

## client.domain

<classdiagram_client_domain.jpg>

## server

<classdiagram_server.jpg>

## server.domain

<classdiagram_server_domain.jpg>

## shared.com

<classdiagram_shared_com.jpg>

## shared.game

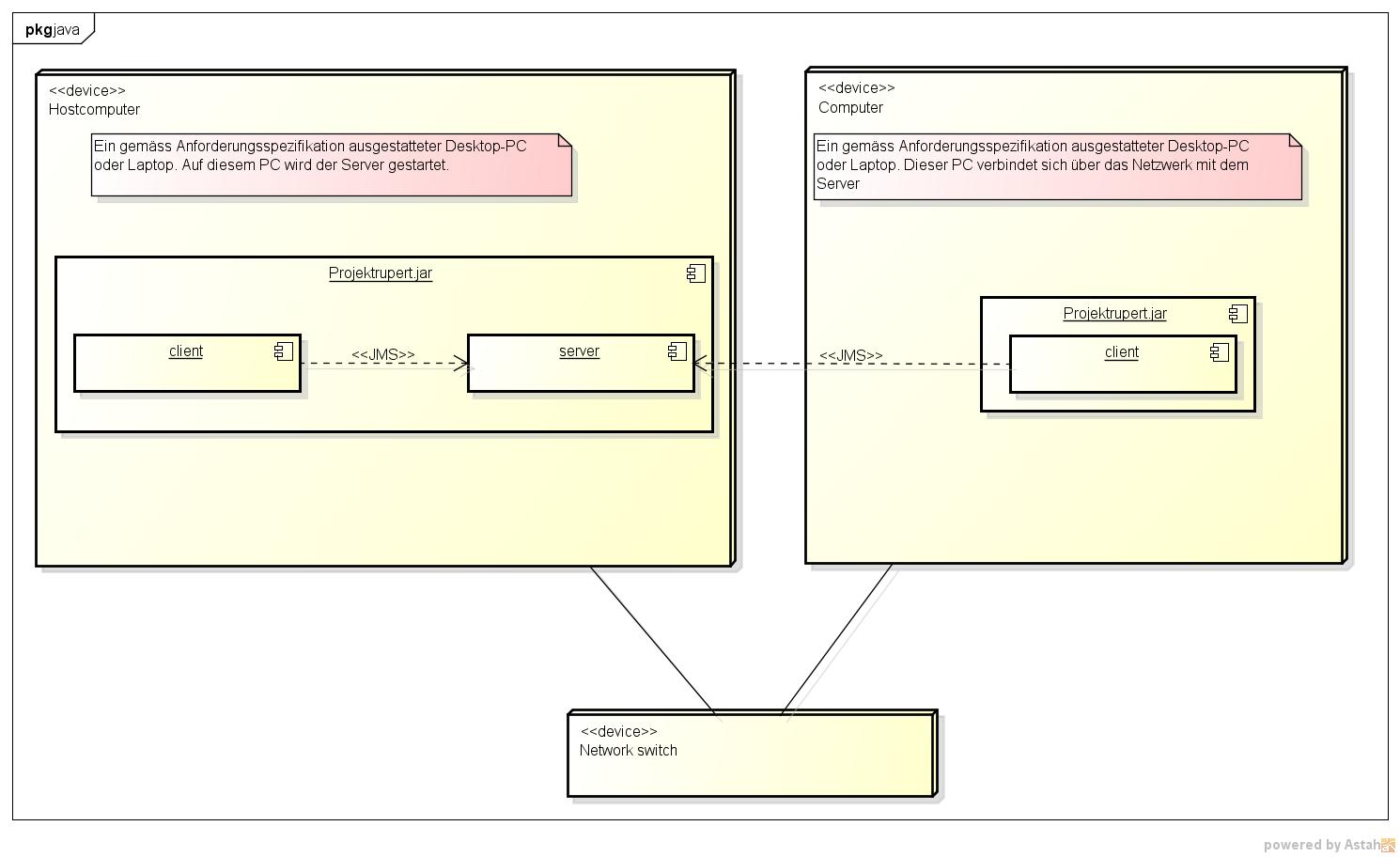
<classdiagram_shared_game.jpg>

## shared.resource

<classdiagram_shared_resource.jpg>

# Deployment

Implemented 6



So sieht das Diagramm im laufenden Betrieb aus.

Das JAR-File selber respektive der Client wird jedoch immer zusammen mit dem Server ausgeliefert, damit jeder Client auch ein Spiel hosten kann (siehe Diagramm -> Hostcomputer).

# Datenspeicherung

Von den Spielen selbst werden keine Daten gespeichert. Die einzigen Daten die gespeichert werden sind permanente Daten wie Grafiken. ~~Diese werden direkt mit dem Programm mitkompiliert und liegen somit zusammen mit dem Code als Binärdatei im .jar File.~~ Sie befinden sich im JAR-File und werden als Resource im Projekt eingebunden. Es ist somit möglich, die Bilder bei jedem Client seperat anzupassen und zu ändern.

# Grössen und Leistung

Die Architektur kann theoretisch eine beliebige Anzahl Clients verwalten. Allerdings wird im Zuge dieses Projekts die Anzahl User auf Zwei Anwender pro Partie/Server Begrenzt.

Die Architektur benötigt in der ausgelieferten Version noch einige kleinere Anpassungen um effektiv mehr als zwei Spieler zu unterstützen.