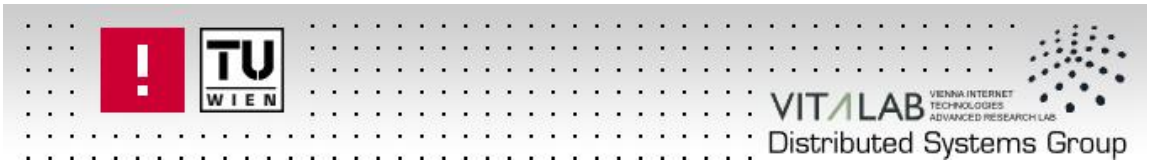

184.159 VU Software Architekturen SS11

Exercise 1

SWAG

Eugen Dahm, 0625325, 534

e0625325@student.tuwien.ac.at
27. März 2011

Inhaltsverzeichnis

1 Deployment Diagram

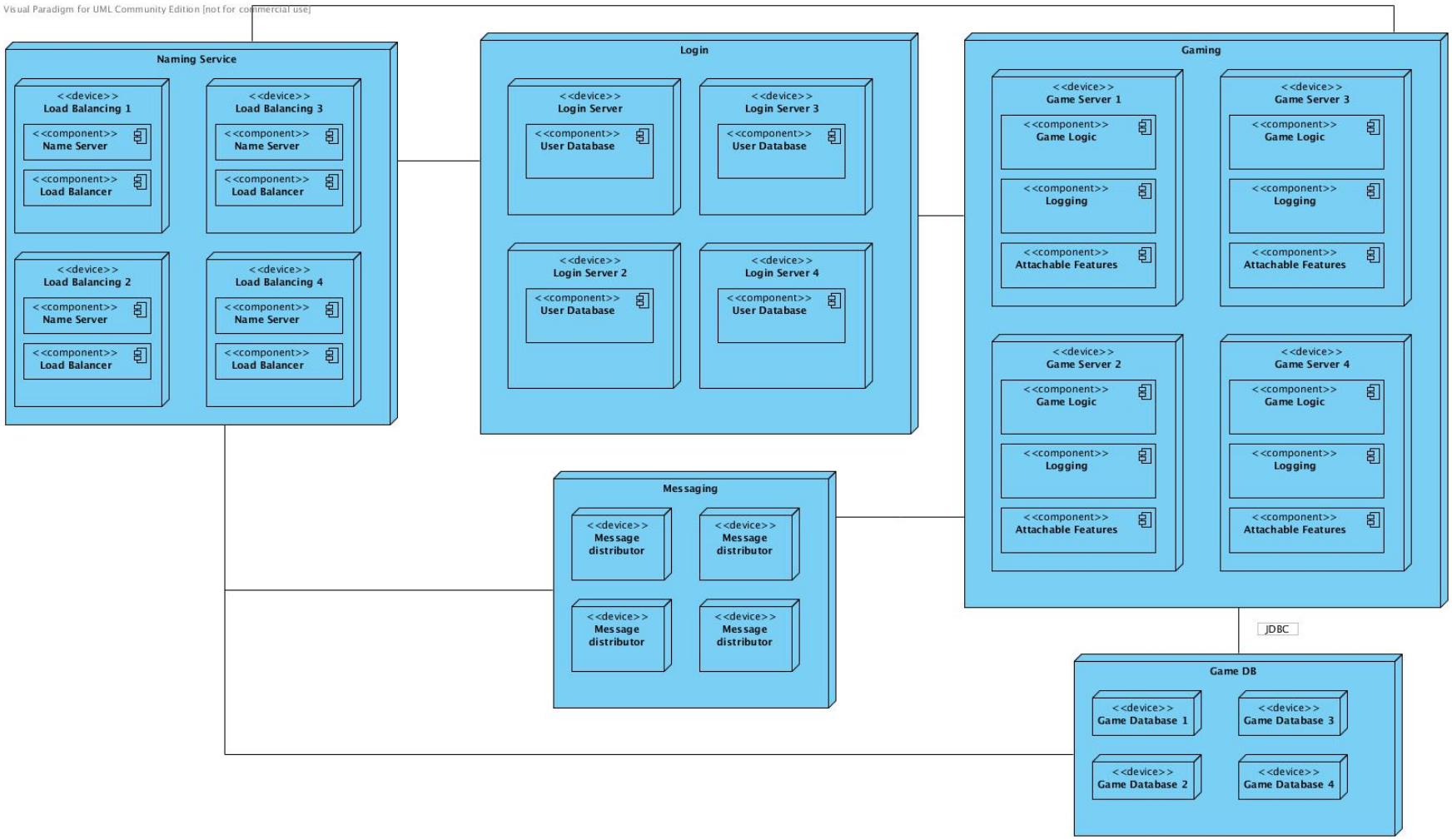

Abbildung 1: deployment diagram

1.1 Uptime

Since the requirements demand a reliable architecture, we have to do a few calculations to meet the 99.99% uptime. We have to take all physical devices (nodes) into account. Assumption: For the sake of convenience, we don't take repairs into account - we don't care about nodes that need to be repaired. According to the requirements, the probability of one node failing is 10% - a factor of 0.1.

1.1.1 Calculation to meet 99.99% uptime

$$99.99\% = 0.9999 \quad \text{uptime} \tag{1.1}$$

$$1 - 0.9999 = 0.0001 \quad \text{failure rate} \tag{1.2}$$

$$0.1^x = 0.0001 \quad \text{logarithm to find the x} \tag{1.3}$$

FOUND THE X →x $$\ln(0.1) = \ln(0.0001) \tag{1.4}$$

$$x = \frac{\ln(0.0001)}{\ln(0.1)} = 4 \text{ nodes} \tag{1.5}$$

Conclusion: We need at least 4 nodes, to meet the 99.99% uptime requirement.

1.1.2 Naming Service

4 Naming Service Nodes ensure a failsafe access to the gameservers.

$$ns\_uptime = 1 - 0.1^4 = 1 - 0.0001 = 0.9999 \tag{1.6}$$

1.1.3 Login Server

$$login\_server = 1 - 0.1^4 = 1 - 0.0001 = 0.9999 \tag{1.7}$$

1.1.4 rest

The Messaging, Gaming and the Game DB node uptimes can be calculated analogously.

1.1.5 overall Uptime

$$(ns\_uptime + login\_server \tag{1.8}$$

$$+ messaging + gaming + game\_db)/5 \tag{1.9}$$

$$= \frac{0.9999 * 5}{5} \tag{1.10}$$

$$= 0.9999 = 99.99\% \text{ uptime} \tag{1.11}$$

1.2 Architectural Decisions

1.2.1 Naming Service decision

For our reliable, and scalable architecture, we have to decide, how the gameservers should be accessable.

Issue	How are the gameservers reliably accessable by a big number of users
Decision	Setup a separate cluster of naming servers with its own load balancers.
Assumptions	A user doesn't create too much load - a user only sends a few requests to the Naming Service per login
Argument	A distinct naming service, reduces the overall number of necessary load balancers, since they only have to be replicated 4 times, instead of replicated load balancers per service
Implications	-

Tabelle 1: Naming Server Structure

1.2.2 separated databases

Since the user information isn't required during the game, it is plausible to save all user data separated from the game-data and the messages.

Issue	How can we separate user-data from game-data
Decision	Setup a separated login cluster, which stores all userdata in its own databases
Assumptions	User data will not be needed during a running game, therefore separation isn't a problem
Argument	A Separated database for the user-information makes it possible for a user to change his information without sending requests to a gameserver
Implications	The login Server needs an interface to the game database, in order to list all maps, a user is currently playing on

Tabelle 2: separated databases

2 Component Diagram

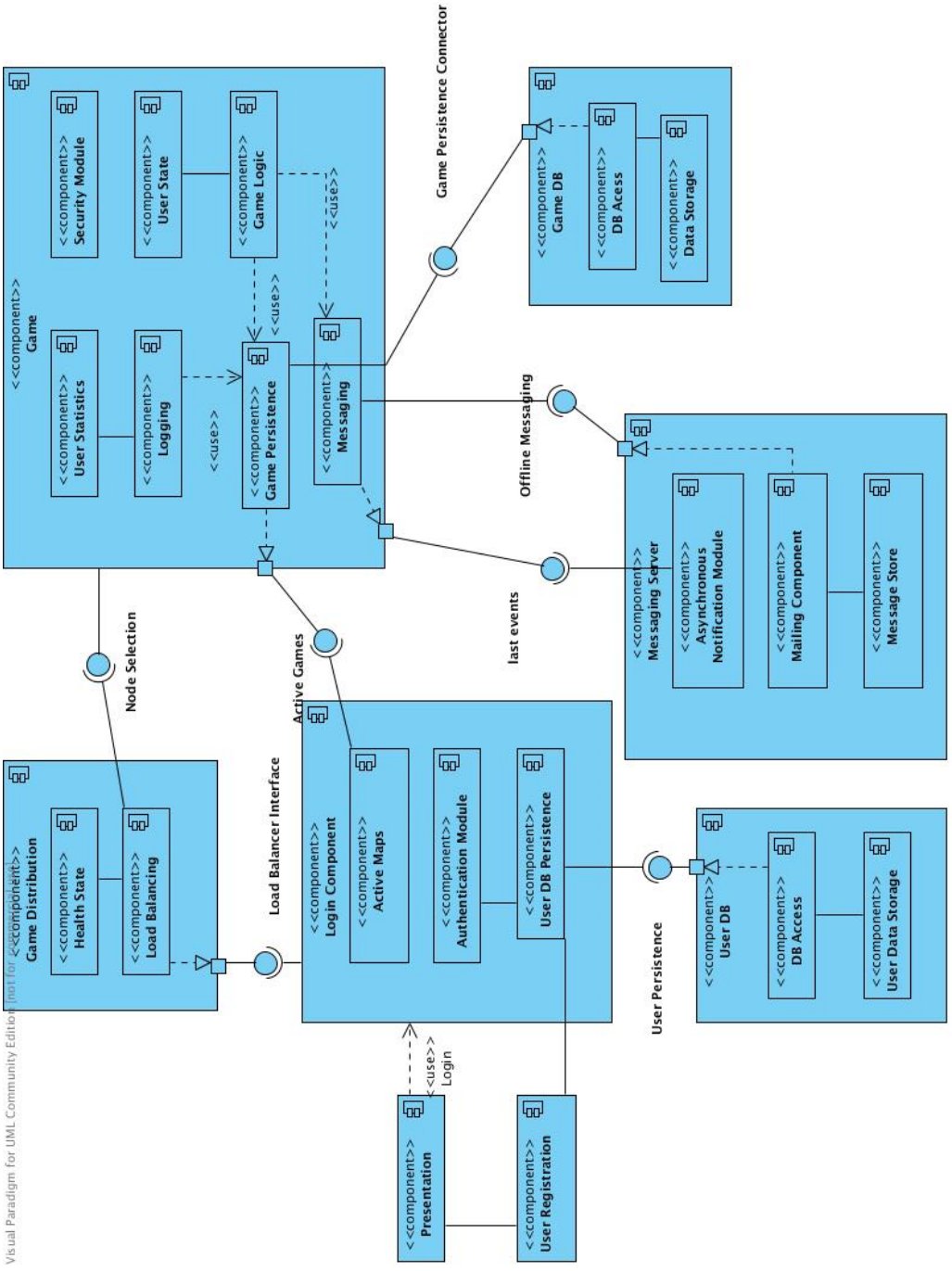

Abbildung 2: component

3 Database Model

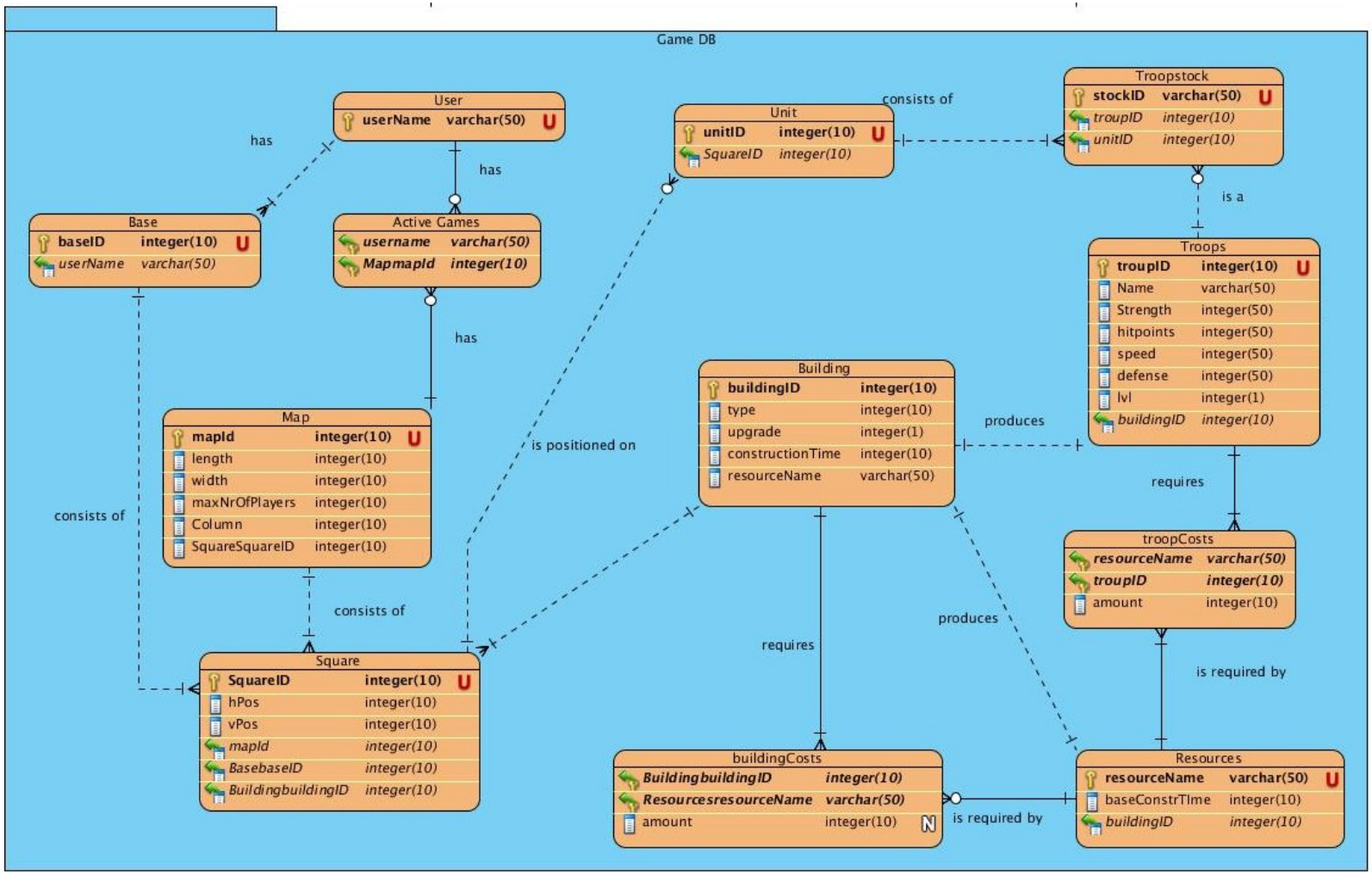

Abbildung 3: component

3.1 User and Messaging

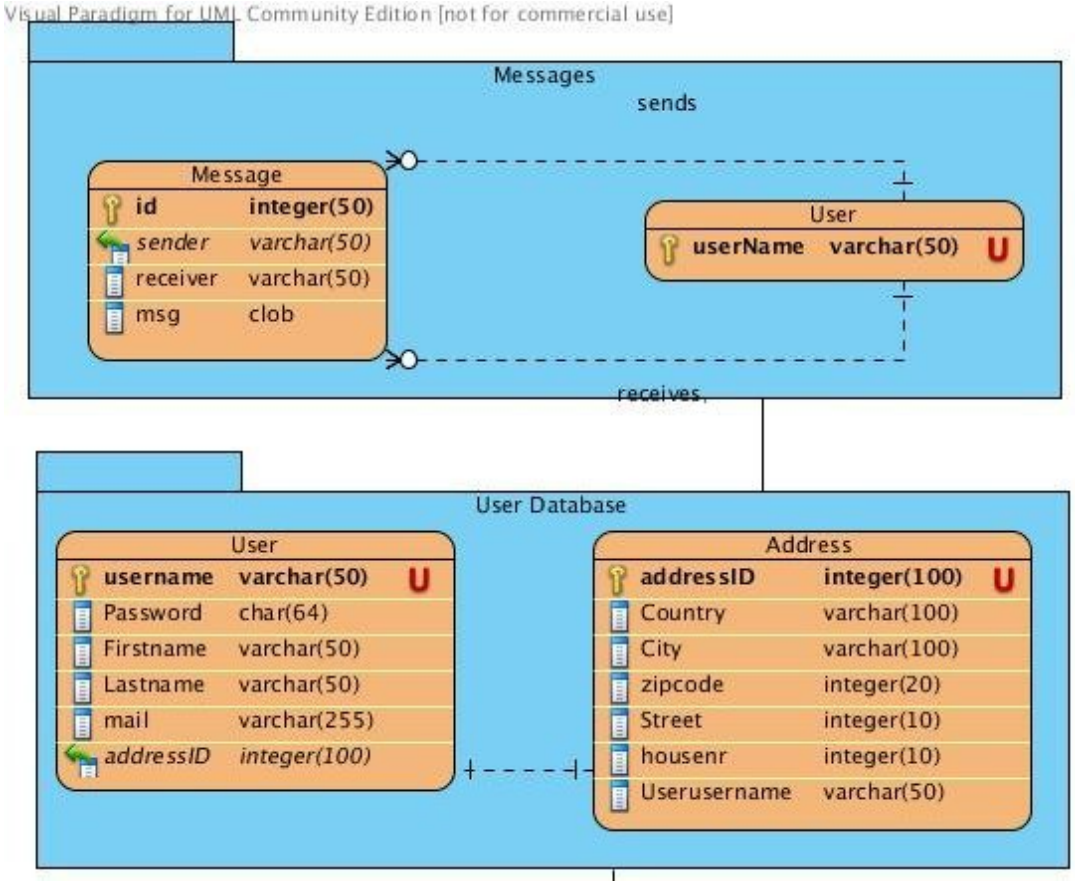

Abbildung 4: component

4 Core Use Cases

Use Case	Register
Description	A user registers an account in the system, with which he can login and play on a chosen map
Assumption	The user has a valid email address
Post condition	The user has a registered account

Tabelle 3: register account use case

Use Case	Login
Description	A user can login into the game with is previously chosen credentials, consisting of a username and a password
Assumption	The user has registered an account with his chosen credentials - user,pass, email etc.
Post condition	a user is logged in

Tabelle 4: login use case

Use Case	Select Game
Description	A user can choose one of his active games, or join an already running game
Assumption	The user has an account. The user is logged into the system
Post condition	the user has joined one of his active games or any other running game

Tabelle 5: select Game use case

Use Case	Logout
Description	A user can can logoff at any time without loosing any data
Assumption	The user is playing on some map/maps
Post condition	the user has logged off

Tabelle 6: logout use case

Use Case	Quit Account
Description	A user can disable/delete his account at any time. All saved data will be deleted
Assumption	The user has an active game account
Post condition	all user data is deleted and the user cannot login anymore

Tabelle 7: Quit Account use case

Use Case	send in-game messages
Description	A user can send in-game messages to any user on any map
Assumption	The receiver of the message has to be online
Post condition	The user/users received the message

Tabelle 8: in-game messaging use case

Use Case	send offline messages
Description	A user can send messages to any user, even offline users. A user receives such messages as emails
Assumption	the receiver isn't currently online
Post condition	the user has received the email

Tabelle 9: offline messaging

Use Case	create building
Description	A player can build any of the predefined buildings in any of his bases
Assumption	the player has enough recourses/money to build the building
Post condition	building is ready to use

Tabelle 10: create building use case

Use Case	send troops
Description	A player can send his troops as unit to any square on the map
Assumption	the player has troops
Post condition	the troops have arrived at the selected square

Tabelle 11: send troops use case

Use Case	attack base
Description	a player can attack any enemy base on the map.
Assumption	the player has sent his troops to an enemy base
Post condition	the user has destroyed the enemy base, or has been defeated

Tabelle 12: attack base

4.1 Assumptions

4.1.1 Attacking a base

A player can attack an enemy base, after he defeated all troops, stationed in this base. According to the specification, an enemy base can be attacked after this player has no more troops on the map, which isn't reasonable.

4.1.2 Upgrading

Every building can be upgraded - resource buildings, troop buildings etc. Each one has a base level, and 2 possible upgrades. It doesn't make any sense to have upgradeable resource buildings but not upgradeable troop buildings.

4.1.3 explicitly quitting game

When a user explicitly quits a game, he looses all his bases and they are free for all to take over. Each building takes its time to be taken over and the player receives all recources stored in a building.

5 Patterns and other decisions

5.0.4 Security

1. Passwords are stored as SHA-512 hashes. SHA-512 is a cryptographically secure hash algorithm, which won't be broken in the near foreseeable future. Since the NIST want's to replace the SHA-2 family by SHA-3, which will be ready by 2012, those hashes can be optionally replaced by SHA-3.

2. All parts of the security/authentication system are easily replaceable, to incorporate recent security best practices.

5.0.5 Notification System

1. Offline messaging is reliable in a sense that sending of emails is failsafe. It doesn't make sense to incorporate acknowlegement of emails by the receiver.

2. Notifications have different priorities. A user receives all messages while beeing offline as email. As a fallback, the user receives the most important offline messages additionally as in-game notifications when he logs in again.

5.1 patterns

5.1.1 User Interaction

Issue	We want to separate the user interaction from the of the code base
Decision	Utilization of the MVC- Pattern
Assumptions	
Argument	It splits the code into 3 parts. • Model: data representation, database etc. • View: the graphical representation - how the user sees the game • Control: the component which controls the flow. It interacts with the model and the view.
Implications	The gui is extensible

Tabelle 13: User Interaction

5.1.2 game control

Issue	We need one or more controller components which handle the whole game
Decision	Utilization of the Singleton Pattern
Argument	A Singleton Controller gets instantiated only once and can be easily used to control the whole game afterwards
Implications	Static variable, exposing the control to virtually every class

Tabelle 14: game controller

Issue	We need a way for the messaging server to know, when a user is online again.
Decision	Using implicit invocation/ a publish-subscribe pattern for async. callbacks
Argument	The game controller doesn't need to wait for a response. The controller only notifies the messaging server that the user is back online. The messaging server then sends all missed events via a callback
Implications	

Tabelle 15: publish subscribe/ callback