**Spilihp**

2012

**Architectuur Document**

**S33b**

**Remi Arts, Simon v. Amstel, Luc Martens, Jan Verhagen, Mitch v. Bijsterveld, Pierre Butela**

Inhoud

[H1 Inleiding 2](#_Toc343508719)

[Context 2](#_Toc343508720)

[Applicatie 2](#_Toc343508721)

[Doel van dit document 2](#_Toc343508722)

[H2 Domeinmodel 3](#_Toc343508723)

[Klassendiagram 3](#_Toc343508724)

[H3 Persistentie 5](#_Toc343508725)

[Eigenschappen van objecten m.b.t. persistentie 5](#_Toc343508726)

[Date opslag 5](#_Toc343508727)

[H4 Communicatie 6](#_Toc343508728)

[Attributen m.b.t. communicatie 6](#_Toc343508729)

[Manier en momenten van communicatie 6](#_Toc343508730)

[H5 Realisatie niet-functionele eisen 7](#_Toc343508731)

[Betrouwbaarheid 7](#_Toc343508732)

[Performance 7](#_Toc343508733)

[Beveiliging 7](#_Toc343508734)

[Schaalbaarheid 7](#_Toc343508735)

[H6 Component diagram 8](#_Toc343508736)

[Component diagram 8](#_Toc343508737)

[Koppeling tussen componenten 8](#_Toc343508738)

[Services per component 9](#_Toc343508739)

[Allocatie van objecten 9](#_Toc343508740)

[Remote objecten 9](#_Toc343508741)

[H7 Deployment 10](#_Toc343508742)

[Deployment diagram 10](#_Toc343508743)

[H8 Specificatie van interfaces 11](#_Toc343508744)

[Interface IWedstrijd: 11](#_Toc343508745)

[Interface IGebruiker: 12](#_Toc343508746)

[Interface ISpeler: 12](#_Toc343508747)

[Interface ILobby: 13](#_Toc343508748)

# H1 Inleiding

Context  
De beginnende firma NetGames is een jong bedrijft wat de markt van computersgames wil gaan betreden. NetGames heeft een multiplayer game bedacht die ze graag gerealiseerd willen hebben. Het project bestaat uit een 3 player airhockey game. Het spel moet vanuit aparte pc’s/laptops kunnen worden gespeeld.

Applicatie  
De gedistribueerde game kan door drie verschillende spelers worden gespeeld, iedere speler heeft een pc of laptop nodig om de game te kunnen spelen. Het spel wordt via internet met elkaar in verbinding gebracht.   
Op een driehoekig bord zal het spel worden gespeeld, en elke speler kan het spel spelen door een edge te laten bewegen. Het spel heeft tien rondes en in elke ronde verschijnt er een puck(bal) vanuit middelpunt van het veld. Door die puck te raken of slaan, kunnen punten worden gescoord. Als er wordt gescoord, krijgt de speler die als laatste de puck heeft geraakt er een punt bij en een nieuwe ronde gaat meteen beginnen. Alle spelbewegingen van de gesimuleerde game kunnen real-time zichtbaar gemaakt worden op het beeldscherm van de pc of laptop. Tijdens het spel kunnen de spelers tegelijk chatten in de wedstrijdchat lobby. Chatberichten, scores en ratingen tijdens de spelssessie worden opgeslagen. Na afloop van een spel kunnen alle chatberichten, scoren en ratingen worden bekeken.

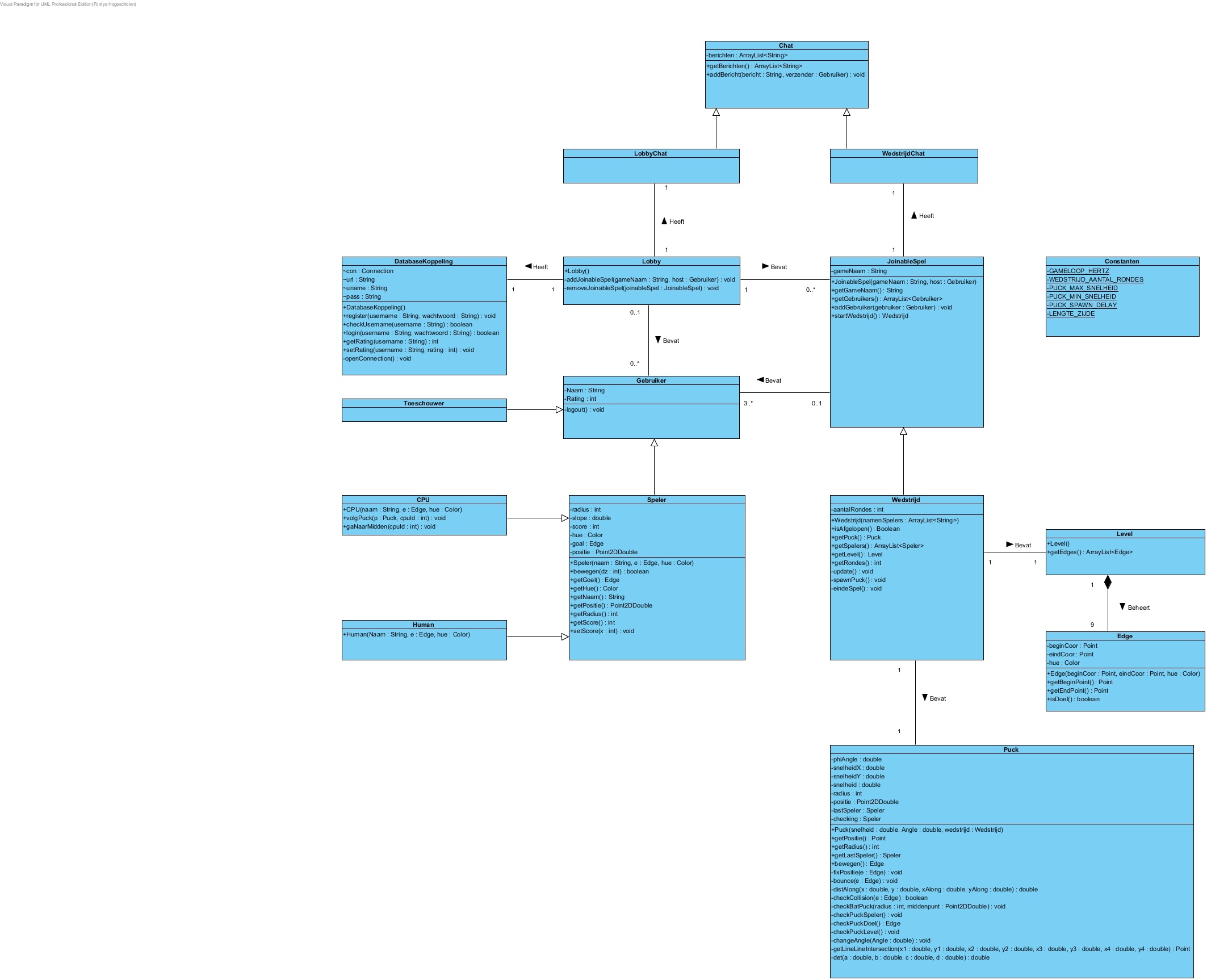
Doel van dit document  
In dit document gaan we de architectuur(technische ontwerpen van software) voor de gemaakte game verantwoorden. Daarna gaan we alle ontworpen diagramen en interfaces voor het airhockey game beschrijven met betrekking tot de mogelijkheid om onderling (de verschillende onderdelen) te communiceren. Verder gaan we zeven aspecten van de architectuur beschrijven:

* Domeinmodel
* Persistentie
* Communicatie
* Realisatie van niet-functionele eisen
* Componenten
* Deployment

# H2 Domeinmodel

In dit hoofdstuk gaan we toelichting geven op onze gemaakte keuzes bij het klassendiagram. Het domeinmodel dat in dit hoofdstuk wordt getoond, zullen we ook in onze applicatie gaan toepassen.

## Klassendiagram

****Figuur 1: Klassendiagram van het domeinmodel.

In figuur 1 is het klassendiagram van het domein afgebeeld. Hierin worden de volgende klassen onderscheiden:

* **Lobby**: algemene representatie voor de weergave van wedstrijd actie tijdens een spel.
* **Chat:** representeert een chatbox, die zowel lobbychat als wedstrijdchat kan omvatten.
* **LobbyChat**: representeert een lobbychat dat door de toeschouwers en mogelijk ook door spelers kan worden gebruikt
* **WedstrijdChart**: representeert een wedstrijdchat die door de spelers tijdens de wedstrijd kan worden gebruiken
* **Wedstrijd:** representeert een wedstrijd of juist een spel die bezig is. Een wedstrijd heeft een aantal rondes(10).
* **JoinableSpel:** representeert een spel dat gejoind is. Een joinablespel heeft een naam en gebruiker die dat spel heeft gejoind.
* **Level:** representeert een niveau. Een level bevat en beheert de edges.
* **Edge:**  is de visualisering van de speler in het spel. Een edge wordt bestuurd door een speler. Een edge heeft een positie en snelheid.
* **Puck:** Een puck wordt gezien als een bal tijdens het spel. Een puck heeft een positie, snelheid en radius.
* **Gebruiker:** representeert een deelnemer aan de applicatie. Een gebruiker heeft een naam en een rating. Een gebruiker kan de rol van speler, toeschouwer, human of CPU aannemen.
* **Speler:** representeert een deelnemer aan het airhokeyspel. Een speler kan een spel spelen.
* **CPU:** representeert een computer. Een computer kan in het spel als robot speler worden gebruikt.
* **Human:** representeert een persoon. Een human kan een spel spelen.
* **Toeschouwer:** representeert een deelnemer die niet speelt. Een toeschouwer kan een spel bekijken en chatten met deelnemers in het spel.
* **Databasekoppeling:** Een databasekoppeling verzorgt voor de connectie tussen de applicatie en de database.

# H3 Persistentie

In dit hoofdstuk wordt de persistentie van de objecten beschreven.

## Eigenschappen van objecten m.b.t. persistentie

* Chat: alle berichten.
* JoinableSpel: gameNaam, gebruikers in de game, spelers in de game, chat.
* Wedstrijd: aantalRondes, gebruikers in de game, spelers in de game, chat.
* Lobby: chat, JoinableSpel.
* Gebruiker: Naam, Rating.
* Edges: Coordinaten, Color.
* Puck: Snelheid, richting, positie, lastPlayer aangeraakt, radius.

## Date opslag

Gegevens van de gebruikers(Naam, wachtwoord en rating) worden opgeslagen in de database. Voor deze oplossing is gekozen omdat in het geval van stroomuitval of servercrash de gegevens niet verloren gaan.   
Overige gegevens als aantal gespeelde rondes, aantal doelpunten of online gebruikers worden runtime in code opgeslagen. Na een wedstrijd wordt de rating per speler aangepast in de database.

# H4 Communicatie

In dit hoofdstuk wordt de communicatie van en tussen objecten beschreven. Hierbij wordt aangegeven welke attributen van welke objecten moeten worden doorgegeven. Hoe de communicatie wordt gerealiseerd (bv. met RMI). En wanneer de communicatie plaatsvindt (push/pull, welk object is leidend?). Hoe en wanneer er communicatie plaats vind kan verschillen per attribuut.

## Attributen m.b.t. communicatie

Tijdens een spel worden de volgende attributen gecommuniceerd:

* Wedstrijd:
  + Puck: positie
  + Speler: positie, dz (de beweging van een batje)
  + Chat: berichten
  + Gebruiker: bericht (in de chat)

In de lobby worden de volgende attributen gecommuniceerd:

* Lobby:
  + Chat: berichten
  + JoinableSpel: gameNaam
  + Gebruiker: Naam, bericht (in de chat)

## Manier en momenten van communicatie

Omdat communicatie voor het spel real-time moet verlopen hebben we gekozen om Remote Method Invocation (RMI) te gebruiken. Voor het doorgeven van de berichten van de chat objecten wordt het push principe gebruikt. We gebruiken ook voor het doorgeven van de coördinaten van alle objecten tijdens een spel het push principe. Ook voor het en het toevoegen van een nieuwe wedstrijd hebben we gekozen voor push. Voor het opvragen van alle beschikbare wedstrijden hebben we gekozen voor het pull principe. We gaan voor RMI gewoon de standaard poort (1099) gebruiken.

Elke instantie van Chat zal zodra deze via push een bericht krijgt van een Gebruiker deze ook meteen naar alle Gebruikers die bij dezelfde Chat zijn doorsturen. Na het aanroepen van de methode addBericht() van de klasse Chat door een Gebruiker zal dus elke tot die Chat behorende Gebruiker worden geinformeerd.

Bij elke update van een Wedstrijd zullen alle Spelers die het spel spelen en alle Toeschouwers die meekijken doormiddel van push een nieuwe set met coördinaten krijgen waarmee in de client het spel wordt getekend. Ook zal zodra een Speler beweegt doormiddel van push een bericht gestuurd worden naar de Wedstrijd. De Wedstrijd zal dit dan in de berekeningen verwerken.

Er zal ook in de lobby een via Push worden doorgegeven als een gebruiker een nieuwe Wedstrijd wil aanmaken. Voor het opvragen van alle wedstrijden gebruiken we een refresh knop waarbij er als er op wordt geklikt de lijst met JoinableSpellen wordt opgehaald.

# H5 Realisatie niet-functionele eisen

*In dit hoofdstuk wordt de realisatie van (overige) niet‐functionele eisen besproken. De volgende onderwerpen dienen ten minste te worden besproken: betrouwbaarheid, performance, beveiliging, schaalbaarheid. Eventueel kunnen nog extra niet‐functionele eisen worden besproken.*

## Betrouwbaarheid

Tijdens het spelen van het air hockey spel dienen op alle cliënt versies van het spel de posities van de puck en de overige spelers zo nauwkeurig mogelijk weer gegeven te worden. Door gebruik te maken van een push methode op de server krijgen alle cliënten te allen tijde de juiste posities tegelijkertijd toegestuurd. Wanneer een speler op de cliënt van positie veranderd zal dit doorgestuurd worden naar de server waarna deze een push zal uitvoeren zodat alle andere cliënten een update zullen krijgen en de posities gelijk getrokken zullen worden. Dit geldt zowel voor spelers als voor toeschouwers. Wanneer de server geen contact meer kan krijgen met een cliënt die als speler dient. Zal hij deze vervangen door een bot. Hiervan krijgen de overige cliënten een update van zodat zij ook weten dat er een bot speelt in plaats van de speler. Overigens zal het van belang zijn dat de tijden van de cliënten gelijk lopen. Hierdoor zal de animatie van de puck en speler gelijk moeten lopen.

Performance  
Het updaten van de posities zullen gebeuren door het RMI protocol Push principe. Hierdoor worden updates verstuurd wanneer deze er zijn via het internet. De performance van de game is erg afhankelijk van het internet verbruik en de zendtijd van de server. Dit is ook weer afhankelijk van het overige netwerk en dataverkeer.

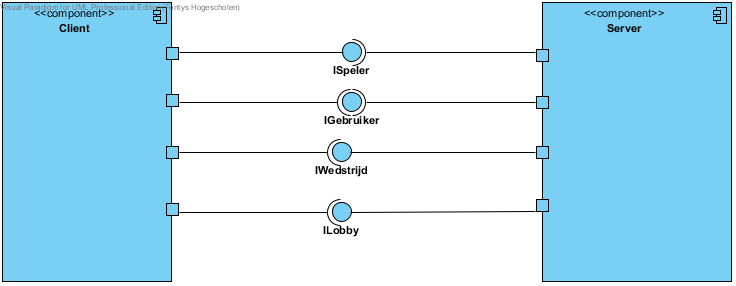
De chatbox en het spel zullen beide gebruik gaan maken van een push methode die zorgt dat alle cliënten dezelfde informatie snel ontvangen. Dit zal allemaal voor de nodige bandbreedte vragen. De daadwerkelijke benodigde bandbreedte wordt nog onderzocht.

Beveiliging  
Zoals het er nu uitziet zullen wij geen speciale maatregelen nemen om het onderscheppen van push data of manipuleren van scores tegen te gaan. Wel is het mogelijk om dit in een later stadium nog altijd aan te passen.

Schaalbaarheid  
Het gedistribueerde air hockey game zal in eerste instantie opgezet worden om met 2 spellen te kunnen spelen. Hiermee zal het uitgebreid getest gaan worden. Dit zal bestaan uit 1 game met daarin 3 spelers en 2 toeschouwers per game. In latere stadiums zal de game een x aantal games kunnen bevatten.

# H6 Component diagram

## Component diagram



Figuur2: component diagram

Het componenten diagram in figuur definieert 2 componenten met interfaces. De volgende componenten worden onderscheiden:

Client: Dit component representeert de software applicatie voor de klant.

Server: Dit component representeert de software applicatie van de game server.

## Koppeling tussen componenten

Voor de onderlinge real-time communicatie wordt Remote Method Invocation(RMI) gebruikt. De koppeling tussen de componenten wordt gerealiseerd middels 4 interfaces, te weten ISpeler, IGebruiker, IWedstrijd en ILobby. Omdat alle componenten informatie nodig hebben van alle andere componenten zijn alle aangeboden interfaces verbonden met alle bijhorende vragende interfaces.

## Services per component

Services worden gerealiseerd middels de volgende interfaces:

ISpeler:

* Verstuurt input van een gebruiker

IGebruiker:

* Vraagt gegevens van gebruikers.
* Verstuurt gegevens van gebruiker.

IWedstrijd:

* Verstuurt wedstrijdgegevens.

ILobby:

* Vraagt wedstrijdgegevens.
* Vraagt gegevens van gebruikers.
* Verstuurt wedstrijdgegevens.
* Verstuurt gegevens van gebruikers.

## Allocatie van objecten

Allocatie van objecten per component:

Client: instanties van ISpeler, IGebruiker en ILobby.

Server: instanties van IWedstrijd

## Remote objecten

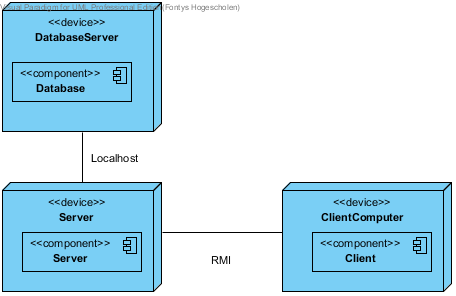
Remote objecten per component:

Client: instanties van IWedstrijd, ISpeler.

Server: instanties van IGebruiker en ISpeler.

# H7 Deployment

## Deployment diagram



Figuur3: deployment diagram

# **H8** **Specificatie van interfaces**

*In dit hoofdstuk wordt de specificatie van interfaces besproken. Voor ieder interface wordt per methode gedefinieerd:*

* *Naam methode*
* *Naam en type argumenten*
* *Precondities*
* *Type returnwaarde*
* *Beschrijving*
* *Aanleiding voor excepties*

## Interface IWedstrijd:

Methode naam: isAfgelopen  
Precondities: de instantie van wedstrijd is gestart.  
Type returnwaarde: Boolean  
Beschrijving: controleert of de instantie van wedstrijd beëindigt mag worden.  
Aanleiding voor Excepties: -

Methode naam: getPuck  
Precondities: de instantie van wedstrijd is gestart.  
Type returnwaarde: Puck  
Beschrijving: creëert een instantie van de Puck klasse.  
Aanleiding voor Excepties: -

Methode naam: getSpelers  
Precondities: een instantie van wedstrijd is aangemaakt.  
Type returnwaarde: ArrayList<Speler>  
Beschrijving: haalt alle spelers van een wedstrijd op.  
Aanleiding voor Excepties: -

Methode naam: getLevel  
Precondities: een instantie van wedstrijd is aangemaakt.  
Type returnwaarde: Level  
Beschrijving: creëert een instantie van de Level klasse.  
Aanleiding voor Excepties: -

Methode naam: getRondes  
Precondities: de instantie van wedstrijd is gestart.  
Type returnwaarde: int  
Beschrijving: haalt het aantal gespeelde rondes van een wedstrijd op.  
Aanleiding voor Excepties: -

Methode naam: Update  
Precondities: de gameloop is gestart.  
Type returnwaarde: void  
Beschrijving: ververst alle gegevens van een wedstrijd.  
Aanleiding voor Excepties: -

Methode naam: spawnPuck  
Precondities: de instantie van wedstrijd is gestart, een instantie van Level is gemaakt.  
Type returnwaarde: void  
Beschrijving: reset de positiewaarden van de Puck instantie.

Methode naam: eindeSpel  
Precondities: de instantie van wedstrijd is gestart, getrondes returnt dezelfde waarde als het maximaal aantal rondes van een wedstrijd.  
Type returnwaarde: void  
Beschrijving: beëindigt de instantie van wedstrijd.  
Aanleiding voor Excepties: -

## Interface IGebruiker:

Methode naam: Logout  
Precondities: de gebruiker zit in een sessie.  
Type returnwaarde: void  
Beschrijving: beëindigt de sessie van de client en verwijderd de instantie van gebruiker.

## Interface ISpeler:

Methode naam: bewegen  
Precondities:   
Type returnwaarde: Boolean  
Beschrijving: verandert de postitie van de Speler instantie gelijk aan de input van de gebruiker.  
Aanleiding voor Excepties:

Methode naam: getGoal  
Precondities: er is een instantie van Level gemaakt.  
Type returnwaarde: Edge  
Beschrijving: haalt de postitie van het doel op een Edge op.  
Aanleiding voor Excepties:

Methode naam: getHue  
Precondities: er is een instantie van Wedstrijd gemaakt.  
Type returnwaarde: Color  
Beschrijving: Haalt de kleur van de speler op.  
Aanleiding voor Excepties:

Methode naam: getNaam  
Precondities:   
Type returnwaarde: String  
Beschrijving: haalt de naam van de gebruiker op  
Aanleiding voor Excepties:

Methode naam: getPosition  
Precondities: er is een instantie van wedstrijd en level gemaakt.  
Type returnwaarde: Point2dDouble  
Beschrijving: haalt de positie van de gebruiker op.  
Aanleiding voor Excepties:

Methode naam: getRadius  
Precondities: er is een instantie van wedstrijd en level gemaakt.  
Type returnwaarde: int  
Beschrijving: berekent de radius van de bat van de gebruiker.  
Aanleiding voor Excepties:

Methode naam: getScore  
Precondities: er is een instantie van wedstrijd gemaakt.  
Type returnwaarde: int  
Beschrijving: haalt de score van de gebruiker in een wedstrijd op.  
Aanleiding voor Excepties:

Methode naam: setScore  
Precondities: Speler heeft een doelpunt gemaakt.  
Type returnwaarde: void  
Beschrijving: verandert de score van een speler.  
Aanleiding voor Excepties:

## Interface ILobby:

Methode naam: addJoinableSpel  
Precondities: er is een nieuwe instantie van JoinableSpel gemaakt.  
Type returnwaarde: void  
Beschrijving: voegt de nieuwe instantie van JoinableSpel toe aan de lijst.  
Aanleiding voor Excepties:

Methode naam: removeJoinableSpel  
Precondities: de te verwijderen instantie van JoinableSpel staat in de lijst met JoinableSpellen  
Type returnwaarde: void  
Beschrijving: verwijdert de instantie van JoinableSpel uit de lijst met JoinableSpellen.  
Aanleiding voor Excepties: