**Containing**



**Gemaakt door:**

Joshua Bergsma

Remco de Bruin

Melinda de Roo

Arjen Pander

Jeffrey Harders

Yme van der Graaf

Inhoudsopgave

[1. Protocol 3](#_Toc372726578)

[2. Algoritmes 4](#_Toc372726579)

[3. Ontwikkelomgevingen 5](#_Toc372726580)

# 1. Protocol

Communicatie tussen server/client:

De communicatie tussen de "controller" en "simulator" is typisch een vorm van

server/client, dit gaat via het netwerk.

Communicatie via het netwerk gaat door middel van sockets, er word zo eerst

een poort opengezet in de server die vervolgens wacht tot 1 (of meerdere?)

client een verbinding aangaat.

Er word dan gepraat via deze connectie dmv TCP packets.

UDP packets hebben we ook overwogen maar de voordelen maken niet echt veel uit

voor ons project, wanneer de packets groot worden is het vooral onhandig om

UDP te gebruiken.

Communicatie via een netwerk komt altijd meer bij te kijken dan je verwacht,

zo kan de server niet zomaar een dump van een class sturen en verwachten dat

de client dit goed kan opnemen, de structuur van de class intern kan

verschillen door het gebruik van een andere(nieuwere) compiler of gewoon omdat

de JVM het leuk vind om de class anders instancieren, deze kans is vooral

groot als je verschillende implementaties gebruikt zoals proprietary

oracle/openJDK.

Je mag dus nooit uitgaan dat een exacte code kopieen van een struct/class in verschillende projecten precies dezelfde class in het geheugen opleverd.

Omdat deze "native" manier niet werkt moet er een tussenformaat bedacht worden, er zijn verschillende opties:

\* Een text formaat

dit zal makkelijk zijn om te lezen en zal minder problemen opleveren, er zijn standaard formaten waarna classes geserialiseerd kunnen worden, dit

kan heel veel code schelen en zal waarschijnlijk ook eleganter wezen. Een nadeel is dat de packets erg groot kunnen worden en dus onnodig veel

bandbreedte en performance gaan kosten.

\* een binary formaat

dit is niet leesbaar zonder een hex-editor of dergelijke editor, het protocol moet ook compleet van scratch bedacht worden. De beloning kan groot zijn als performance en bandbreedte belangrijk.

We kozen een text standaard, hieronder een lijst van textformaten die we

hebben overwogen:

\*XML

Volgens velen heeft XML als voordeel dat het snel ge(de)serialized kan

worden en dat het leesbaar is, hier zijn wij het niet mee eens, ook is het

onnodig groot, wat misschien niet belangrijk is als er lichte compressie

overheen word gegooit, maar dat is weer een extra onnodige stap.

\*JSON:

Dit is een compact formaat, erg leesbaar en de "standaard" voor het

(de)serializen van/naar javascript objecten, sinds wij van plan zijn om een

web app te maken met mogelijk een android app als wrapper is het belangrijk om

alvast rekening te houden met de javascript code.

Javascript kan overweg met XML en JSON, en de bestaande code die de Container

XML file deserialiseerd kan ook overweg met JSON.

Voor ons was het duidelijk dat JSON veel minder problemen zou opleveren, ook

niet onbelangrijk is dat onze netwerk programmeur al ervaring had met JSON

# 2. Algoritmes

# 3. Methoden

Tijdens dit project gaan wij gebruik maken van de volgende programmeeromgevingen:

* Eclipse / IntelliJ
* jMonkeyEngine (geïntegreerd in Eclipse / IntelliJ)

Deze omgevingen worden gebruikt voor het ontwikkelen van de Controller (Server), de Simulator (Client) en de Management Interface (Android). Dit zal worden geschreven in programmeertaal Java.

In het geval wij kiezen voor een Responsive Website i.p.v. een Android applicatie wordt Sublime Text 2 aan het lijstje met programmeeromgevingen toegevoegd. Deze zal dan in programmeertalen HTML, CSS, JavaScript en wie weet nog stukjes PHP worden geschreven.

Naast bovenstaande wordt er ook gebruik gemaakt van de software Google Sketchup en Blender.

Voor het delen van de code / documenten wordt GitHub gebruikt.

De communicatie gaat via Facebook Chat en Skype.