Containing

Gemaakt door:
Groep 5
Joshua Bergsma
Remco de Bruin
Melinda de Roo
Arjen Pander
Jeffrey Harders
Yme van der Graaf

Inhoudsopgave

1. Protocol

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 d.m.v. 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 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 kopieën van een struct/class in verschillende projecten precies dezelfde class in het geheugen oplevert. Omdat deze "native" manier niet werkt moet er een tussenformaat bedacht worden, er zijn verschillende opties:

- **Een tekst 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 hebben voor een tekst formaat gekozen, hieronder een lijst van tekstformaten 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, 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 programmeurs al enige ervaring heeft met JSON.

2. Algoritmes

Om het programma goed te laten werken zijn er natuurlijk ook algoritmen nodig voor de optimalisatie. Onder andere optimalisatiealgoritmen en sorteer algoritmen.

In de simulator wordt gebruik gemaakt lineaire algebra. Om de animaties van de kranen volgens een bepaalde snelheid te laten bewegen, wordt de afstand berekent tussen de twee vectoren waartussen de haak van de kraan moet bewegen.

Voor het zoeken van de kortste route voor de AGV wordt het A-star algoritme gebruikt. Het is ons ten eerste aangeraden om dit algoritme te gebruiken. Ten tweede is het een nieuwer algoritme dan Dijkstra.

Naast de lineaire algebra en het A-star algoritme is een sorteer functie ook handig. De sorteer functie wordt gebruikt voor het sorteren van de aankomst- en vertrekdata van de containers. Met deze functie zal het programma gemakkelijker door de list heen gaan zonder te hoeven zoeken naar data.

3. Methoden

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

- Eclipse (Kepler Service Release 1) / IntelliJ (13.0)
- jMonkeyEngine (JME3) 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 (8.0.16846) en Blender (2.69).

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

De communicatie gaat via Facebook Chat en Skype.

4. Keuzes

XML
Het laden van de xml bestanden gaat d.m.v. de jaxb library, er wordt hierbij gebruik gemaakt van de MOXy implementatie, dit geeft ons toegang tot simpele xpath expressies en voorkomt dat we nested classes moeten maken. Deze library maakt het ons erg makkelijk, er wordt een class aangemaakt met boven elke attribuut "@XmlElement" (attribuut namen moeten overeenkomen met het xml bestand). De enige methode-aanroep die van belang is: "containers = (ContainerSetXml) jaxbUnmarshaller.unmarshal(xmlContentStream)". Het meerdere keren roepen van deze methode geeft geen bijwerkingen, we gaan ervan uit dat het een pure methode is.

Onze class die het laden van de xml file afhandelt heeft de volgende methoden die van belang zijn:

1. "public ContainerSetXml parse(String path) throws FileNotFoundException"
2. "private ContainerSetXml filterWrongInstances(ContainerSetXml set)"

*. (toekomstige) private methoden die helpen met het filtreren van foute/dubbele containers

Het meerdere keren roepen van parse(path) zou geen enkel probleem moeten opleveren (behalve mogelijke performance), de return waarde is puur afhankelijk op de content van het xml bestand, er wordt dus geen state opgeslagen.

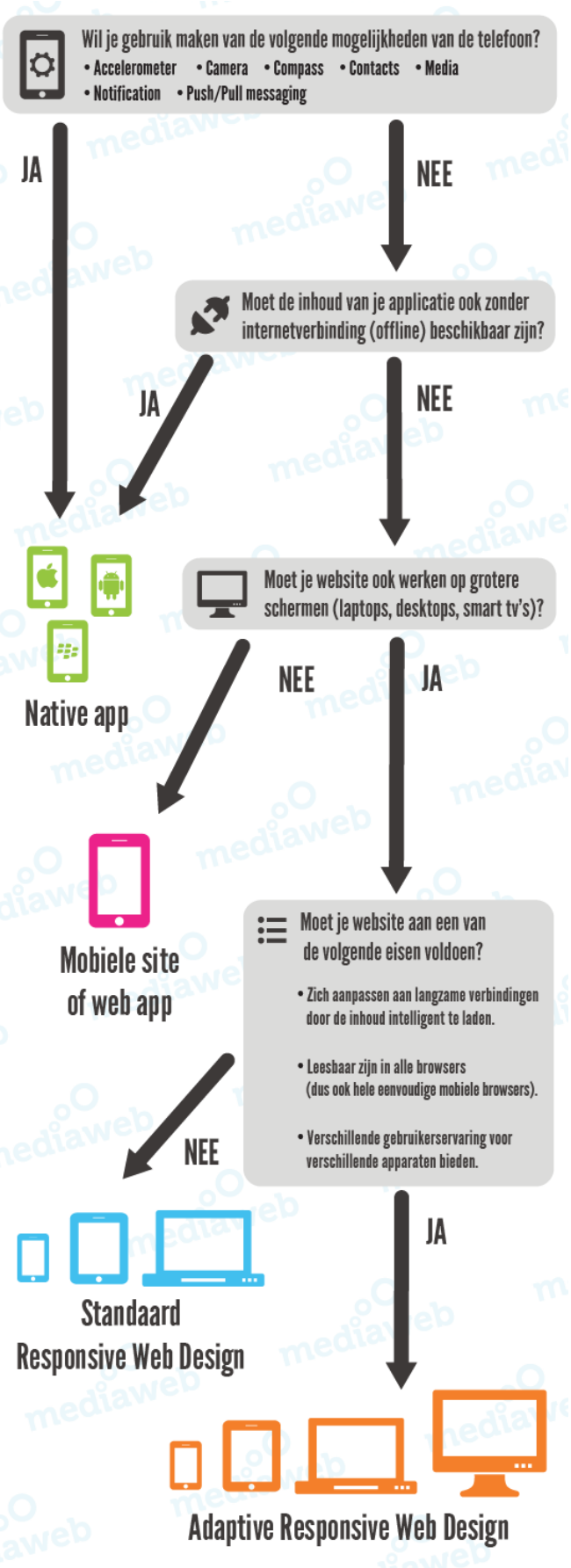
Waarom niet een 44ft container?
In de opdrachtomschrijving komen wij tegen dat we het beste er van uit kunnen gaan dat alle containers de standaard formaten van een 44ft container hebben.

Het probleem daarmee is dat 44ft helemaal niet een standaard maat is en daarom de hoogte en breedte nergens duidelijk vindbaar zijn, maar die van 40 en 45ft containers juist wel. Wij hebben er daarom voor gekozen om de 40ft te gebruiken, die veel gebruikelijker is en programmeer technisch geen verschil uit zou maken.

De afmetingen van een 40ft container zijn:
12,192m lang, 2,438m breed en 2.491m hoog.

Waarom niet een mobiele applicatie?
In de opdrachtomschrijving wordt een mobiele applicatie verplicht maar na overleg met W. van der Ploeg hebben wij ook akkoord gekregen voor het maken van een responsive website i.p.v. een mobiele applicatie.

De voornaamste reden hiervoor is dat je platform onafhankelijk bent. Het schrijven van een mobiele applicatie moet voor elk platform op een verschillende manier worden ontwikkeld. Dit nadeel heeft een responsive website niet. Verder geeft de keuzeboom (afbeelding 1) op de volgende pagina een goed beeld waarom en wanneer je welke optie kiest.



Afbeelding 1.