

Containing



Gemaakt door:

Groep 5

Joshua Bergsma

Remco de Bruin

Melinda de Roo

Arjen Pander

Jeffrey Harders

Yme van der Graaf

Inhoudsopgave

1. Inleiding	3
2. Protocol	4
3. Algoritmes	5
4. Methoden	6
5. Keuzes	7
6. UML	9

1. Inleiding

Het is de bedoeling dat er programma's worden geschreven die een goederenoverslag simuleren. Hiervoor worden XML bestanden ingelezen, die aangeven wanneer een schip, trein of vrachtwagen het terrein bereikt heeft, begint met laden en lossen, hoeveel containers er op het voertuig zitten en wanneer hij klaar is met laden en lossen. De verschillende containers die van het voertuig afkomen, worden vervolgens opgeslagen op het opslagterrein en anderen meteen over gedragen naar een ander voertuig, die de container weer verder vervoerd. Op het terrein staan kranen die de containers van de voertuigen afhalen, dit zijn railkranen. Zij verplaatsen de containers één voor één naar de AGV's, die ze naar hun volgende bestemming rijden. Dit kan het opslagterrein zijn, waar ze met een vrij beweegbare kraan op het terrein worden gestapeld, of een volgend voertuig, waar ze met een railkraan op het voertuig worden gestapeld. De data moet uiteindelijk ook weergegeven kunnen worden op een mobiel apparaat.

2. 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 wordt zo eerst een poort opgezet in de server die vervolgens wacht tot 1 (of meerdere?) Client een verbinding aangaat. Er wordt 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 vindt om de class anders instantiëren, 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 wordt 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 deserialiseert 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 enige ervaring heeft met JSON.

3. Algoritmes

Om het programma goed te laten werken zijn er natuurlijk een aantal algoritmen nodig. Onder andere een sorteer- en kortste route algoritme. In de simulator wordt ook gebruik gemaakt van lineaire algebra.

Lineaire algebra wordt gebruikt om de animaties van de kranen volgens een bepaalde snelheid te laten bewegen. Hiervoor wordt de afstand berekent tussen de twee vectoren waartussen de haak van de kraan moet bewegen.

PathFinding (kortste route algoritme)

In de client wordt bepaald welke route de AGV's nemen om hun bestemming te bereiken, omdat er vaak meerdere routes mogelijk zijn is er besloten om Dijkstra's algoritme te implementeren. Origineel waren we van plan om A-star te implementeren, omdat het nieuwer is en het ons werd aangeraden maar de performance die het mogelijk oplevert is niet van belang omdat er een route gekozen wordt tussen voor gedefinieerde routes i.p.v. het terrein op te splitsen in een grid, wat voor veel meer nodes zorgt.

Als performance nodig blijkt te zijn is het makkelijk om alsnog naar A-star te switchen, A-start is immers Dijkstra met een extra heuristics stap, d.m.v. deze heuristics wordt gegokt wat de beste route is zodat hele branches aan nodes overgeslagen kunnen worden. Ook zou het gebruik van A-star niet meer de kortste route garanderen, maar dat zou geen probleem moeten zijn, het is wel een extra nadeel die het gebruik van Dijkstra aanmoedigt.

Sorteeralgoritme

Naast de lineaire algebra en Dijkstra's algoritme is een sorteeralgoritme nodig. Er wordt sterk over nagedacht om hiervoor een soort van torens van hanoi algoritme voor te gebruiken. Dan wel aangepast / geoptimaliseerd voor onze doelen. Het sorteeralgoritme wordt gebruikt voor het sorteren van de aankomst en vertrekdata van de containers. Met behulp van dit algoritme (deze functie) zal het programma gemakkelijker door de lijst heen gaan zonder te hoeven zoeken naar data. Dit sorteeralgoritme wordt ook gebruikt voor het opslagterrein.

4. 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.

5. 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 opdrachtschrijving komen wij tegen dat we het beste er van uit kunnen gaan dat alle containers de standaard formaten van een 44ft container hebben.

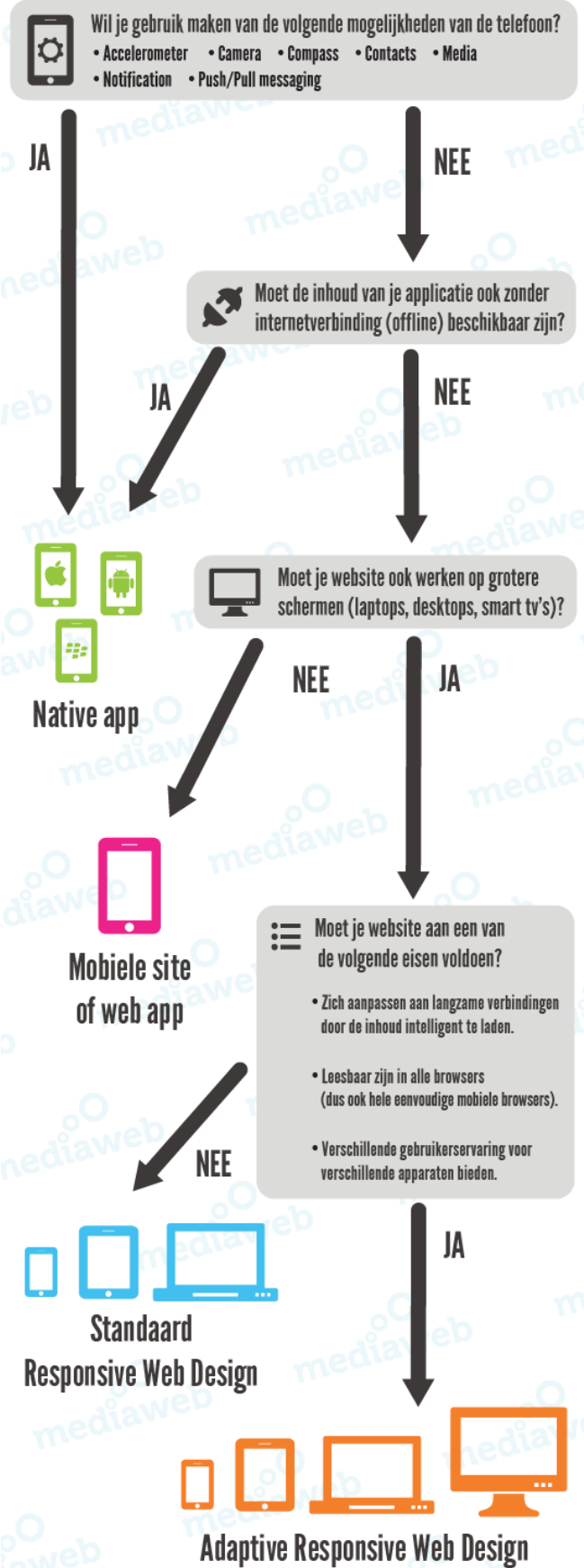
Het probleem daarmee is dat 44ft helemaal niet een standaard maat is en daarom de hoogte en breedte nergens duidelijk vindbaar zijn, maar die van 40 en 45ft containers juist wel. Wij hebben er daarom voor gekozen om de 40ft te gebruiken, die veel gebruikelijker is en programmeer technisch geen verschil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

Hieronder zijn de verschillende klasdiagrammen terug te vinden.

pkg server

```
classDiagram
    class Server {
        -thread : Thread
        -isRunning : boolean
        +enableHeartbeat : boolean
        +heartbeatTimeout : int
        +mainArgs : String[] : void
        +Server()
        +loginToStatisticsServer(host : String, name : String, password : String) : void
        +uploadStatistics() : void
        +getMessages() : String
        +sendMessage(message : String) : void
        +start(port : int) : void
        +stop() : void
        +restart(port : int) : void
        +run() : void
    }
    class FTPClient {
        -ftpClient
    }
    class ServerSocket {
        -serverSocket
    }
    class JPanel {
        -contentPane
        -serverPanel
        -boxpanel
        -textpanel
    }
    class JTextField {
        -userText
    }
    class JLabel {
        -SpeedText
        -clientsStatus
    }
    class JFileChooser {
        -fc
    }
    class Socket {
        -clientSocket
    }
    class JFrame {
        -frame
    }
    class JMenuBar {
        -menu
    }
    class JPasswordField {
        -passwordText
    }
    class JRadioButton {
        -enableHeartbeat
    }
    class JTextArea {
        -textArea
    }
    class ContainerSetXml {
        -containers {ordered}
    }
    class ContainerXml {
        +id : String
        +arrivalDay : int
        +arrivalMonth : int
        +arrivalYear : int
        +arrivalTill : String
        +arrivalFrom : String
        +leaveDay : int
        +leaveMonth : int
        +leaveYear : int
        +leaveFrom : String
        +leaveTill : String
        +arrivalPosX : int
        +arrivalPosY : int
        +arrivalPosZ : int
        +ownerName : String
        +nr : int
        +arrivalTransportType : String
        +leaveTransportType : String
        +arrivalCompany : String
        +leaveCompany : String
        +emptyWeight : int
        +fullWeight : int
        +contentName : String
        +contentType : String
        +dangerType : String
        +iso : String
    }
    class xmlParser {
        +mainArgs : String[] : void
        +loadPath : String : ContainerSetXml
        +parse(containerSet : ContainerSetXml) : PriorityQueue<Protocol>
        +containerXmlToContainer(container : ContainerXml) : Container
        +filterWrongInstances(set : ContainerSetXml) : ContainerSetXml
        +removeNxDuplicates(containers : List<ContainerXml>) : List<ContainerXml>
    }
    class PipLoginView {
        +name : String
        +password : String
        +server : Server
        +display() : void
        +placeComponents(panel : JPanel) : void
    }
    class Main {
        -Speed : int = 100
        -ClientStatus : String = "Offline"
        +mainArgs : String[] : void
        -Menu() : void
        -ServerPanel() : void
        -TextPanel() : void
        -InfoPanel() : void
    }
    Server "1" -- "*" FTPClient
    Server "1" -- "*" ServerSocket
    Server "1" -- "*" JPanel
    Server "1" -- "*" JTextField
    Server "1" -- "*" JLabel
    Server "1" -- "*" JFileChooser
    Server "1" -- "*" Socket
    Server "1" -- "*" JFrame
    Server "1" -- "*" JMenuBar
    Server "1" -- "*" JPasswordField
    Server "1" -- "*" JRadioButton
    Server "1" -- "*" JTextArea
    Server "1" -- "*" ContainerSetXml
    Server "1" -- "*" xmlParser
    Server "1" -- "*" PipLoginView
    Main "1" -- "*" Server
    Main "1" -- "*" JFrame
    Main "1" -- "*" JMenuBar
    Main "1" -- "*" JLabel
    Main "1" -- "*" JFileChooser
    Main "1" -- "*" JFrame
    Main "1" -- "*" JPasswordField
    Main "1" -- "*" JRadioButton
    Main "1" -- "*" JTextArea
    Main "1" -- "*" ContainerSetXml
    Main "1" -- "*" xmlParser
    ContainerSetXml "1" -- "*" ContainerXml
    xmlParser "1" -- "*" Main
```

[illegible][illegible]