

Multitouch toepassing

Groep 6

Blog <http://mumetech.wordpress.com/>

Laurens Van Keer, 3^{de} bachelor Informatica

Bram Truyaert, 3^{de} bachelor Informatica

Abstract—Dit is het verslag van groep 6 voor de vierde en laatste opgave van multimedia. Wij hebben een multitouch-toepassing uitgewerkt, die de muziekverzameling van de gebruiker kan visualiseren in een netwerk en nieuwe muzieknummers kan voorstellen. De belangrijkste uitdagingen waren het leren werken met het MT4J framework, de implementatie van een realistische *spring graph* en het implementeren van multitouch gestures. Uit dit project leren wij vooral dat ons uitgebreid ontwerp het implementatie proces veel vlotter heeft laten verlopen.

Ingediend op: 31 december 2009



1 IDEE

Het doel van onze mobiele applicaties voor het Android platform en de iPhone (resp. *GeoTrack* en *iGeoTrack*) was het (geografisch) localiseren van andere Last.fm gebruikers met een gelijkaardige muzieksmaak. Dit werd mogelijk gemaakt dankzij de GPS-functionaliteit van de platformen en onze eigen server. We vonden dit idee wel interessant om te implementeren op mobiele apparaten, maar voor dit multitouch project is het uiteraard minder geschikt.

Een ander idee was een multitouch versie van onze Flex toepassing: deze applicatie - ook *GeoTrack* genaamd - visualiseerde en vergeleek land per land de muziekvoorkeur van Last.fm gebruikers over de wereld. Het moeilijkste probleem bij een (multitouch) implementatie van dit idee, is het integreren van een geschikte API voor het weergeven en bewerken van kaarten (*Google Maps* bijvoorbeeld). Voor MT4J¹ bestaat er een dergelijke API: *Modest Maps*². Hoewel deze bibliotheek erg slecht gedocumenteerd is, zou het dus perfect mogelijk zijn dit idee opnieuw te implementeren.

We hebben echter besloten een volledig nieuwe idee uit te werken, omdat we vonden dat het *GeoTrack* concept van onze Flex toepassing de multitouch functionaliteit niet optimaal kan benutten. Ons nieuwe idee was oorspronkelijk een multitouch interface voor de open source mediaspeler en -manager genaamd *Songbird*³. Deze interface zou zowel de metadata van de lokale MP3's moeten inladen, als de externe data van de Last.fm API. Dankzij een externe service zoals Last.fm kan er dus meer data over de muziekcollectie opgevraagd worden en kunnen er suggesties voor nieuwe muzieknummers gemaakt worden.

We zijn afgestapt van het idee om deze interface voor *Songbird* te maken, omdat dit alleen maar extra complexiteit toevoegt aan de applicatie en onze focus zou kunnen "afleiden" van één van de belangrijkste doelen van deze opdracht: het leren werken met een multitouch framework. We wilden wel nog een toepassing maken die de lokale muziekcollectie van de gebruiker visualiseert en suggesties maakt voor nieuwe muziek. Het inladen en (tijdelijk) bijhouden van de MP3 bestanden zou nu gebeuren door één of meerdere muziekbestanden in onze applicatie te slepen, i.p.v. door communicatie met de *Songbird* API.

Een lastig probleem bij deze applicatie is het bedenken van een intuïtieve en aantrekkelijke gebruikersinterface. De toepassing moet de lokale en externe (suggesties door Last.fm) muzieknummers kunnen visualiseren en de gebruiker moet door deze collectie kunnen navigeren en per nummer details kunnen opvragen. Om dit te realiseren, gebruikten we het concept van de *spring graph*. De knopen in deze graaf stellen de muzieknummers voor. Wanneer twee nummers overeenkomen (bepaald door de Last.fm API), dan zijn ze verbonden via een boog. De gebruiker kan het netwerk uitbreiden, door op een knoop te tikken. Met ieder muzieknummer wordt ook een knop geassocieerd, die een detailvenster over het muzieknummer opent. Door middel van multitouch gestures, kan de gebruiker doorheen het netwerk navigeren.

¹ "Multitouch For Java" - het open source Java ontwikkelingsplatform dat we voor dit multitouch project gebruiken. Zie ook <http://mt4j.org/>.

² Zie ook http://mt4j.org/mediawiki/index.php/Modest_Maps_Example_-_Source_Code en <http://modestmaps.com/>

³ <http://www.getsongbird.com/>

2 STORYBOARD

Nadat de applicatie wordt opgestart, is het de bedoeling dat één of meerdere MP3 bestanden worden ingeladen. Dit kan op verschillende manieren. Een eerste is het voorzien van een simpel "Open" dialoogvenster dat standaard in de meeste programma's te vinden is. Dit zou echter geen gebruik maken van de multitouch functionaliteit. Daarom opteerden we voor het *file drop* idee: via multitouch gestures kunnen MP3 bestanden in de applicatie gesleept worden.

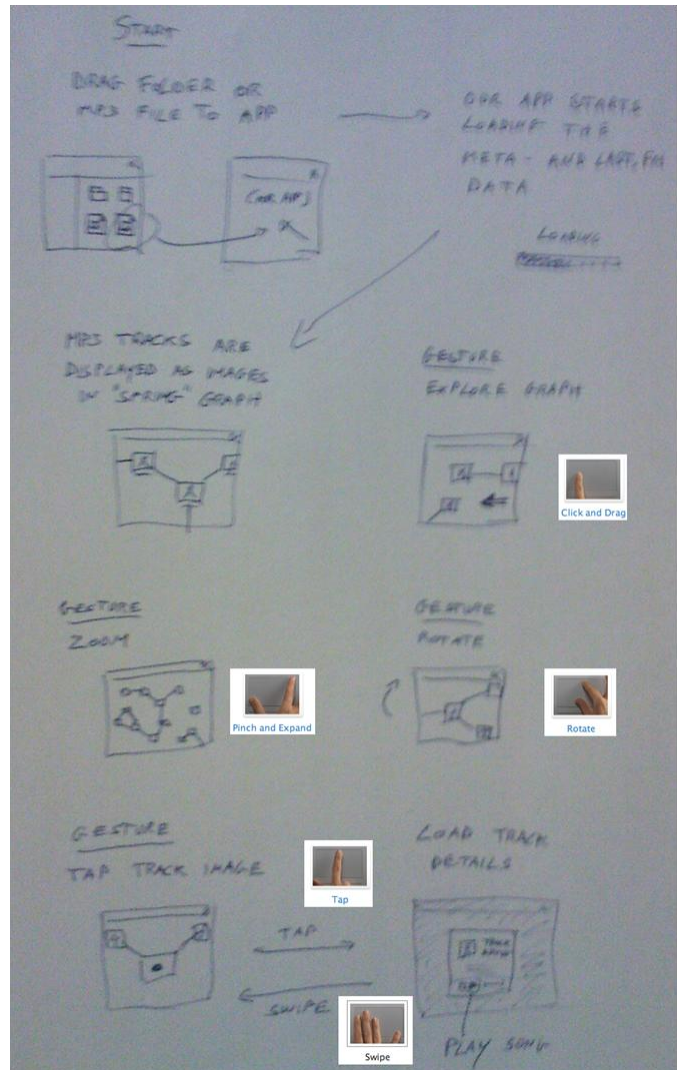
Vervolgens wordt de MP3 metadata ingelezen en wordt m.b.v. de Last.fm API meer informatie afgehaald. Daarnaast wordt met dezelfde API ook een lijst van gelijkaardige muzieknnummers afgehaald.

Nu worden alle nummers met een bijpassende afbeelding⁴ in een netwerk voorgesteld. De knopen (*nodes*) in de graaf zijn muzieknnummers, die met elkaar verbonden worden als ze gelijkaardig zijn. De verbindingen tussen de knopen worden *springs* genoemd, omdat ze zich als veren gedragen, zodat het netwerk gepositioneerd en geordend kan worden. Dit positioneren zal gebeuren m.b.v. algoritmen die gebaseerd zijn op fysische wetten zoals de wet van Hooke.

Met behulp van multitouch gebaren (*gestures*) kan het netwerk gemanipuleerd worden. Zo kan de gebruiker het netwerk verkennen, door eenmaal te tikken en vervolgens te slepen. Er is ook ondersteuning voor de "klassieke" *zoom* en *rotate* acties.

Als de gebruiker eenmaal tikt op een afbeelding, zou de metadata van het nummer, de data van Last.fm en een muziekspeler weergegeven moeten worden. Één manier om dit te doen is bijvoorbeeld een minimalistische weergave onderaan de graaf. We hebben echter besloten om de informatie in een nieuwe *scene* te openen, omdat dit met het MT4J framework eenvoudiger is. Dit vinden we een "stijlbreuk" met de weergave in een netwerk, maar we namen onszelf voor de visueel aantrekkelijkere oplossing pas te implementeren als het belangrijkste werk (de programmalogica en de visualisatie van de graaf) verricht is. Helaas heeft dit laatste ons veel tijd gekost (zoals verder in dit verslag wordt toegelicht) en zijn we bij de "simpelere" oplossing gebleven.

Een extra feature die in figuur 1 niet staat getekend, is het dynamisch uitbreiden van het netwerk. Door op een afbeelding te blijven duwen, wordt de graaf vergroot met muzieknnummers gelijkaardig aan het "aangerakte" nummer. Opnieuw wordt deze informatie van Last.fm gehaald.



Figuur 1 - Het storyboard.

⁴ Als afbeelding voor een muzieknnummer wordt ten eerste gekeken of de overeenkomstige Last.fm pagina een album cover heeft. Zo niet, dan wordt een foto van de artiest gebruikt.

3 HARDWARE-ONTWERP

We hebben zelf een multitouch box gemaakt, gebaseerd op het ontwerp dat we gezien hebben in de les. Deze box bestaat uit enkele eenvoudige componenten: een doos, een webcam en een aanraakoppervlak. Het oppervlak bestaat uit twee verschillende delen: een plaat uit plexiglas, met daarop een blad papier (met dezelfde grootte als de plaat) erop bevestigd. Dit papier vormt het eigenlijke oppervlak dat aangeraakt kan worden.

De opbouw van deze box was heel eenvoudig. Als eerste werd de camera in de box geplaatst. Door een opening aan de onderkant van de doos loopt de USB-kabel uit de box. Vervolgens moest het plexiglas en het papier nog op maat gesneden worden. Ten slotte werden de drivers voor de camera geïnstalleerd en werd de camera gekalibreerd in CCV⁵.



Figuur 2 - De webcam in de doos.



Figuur 3 - De doos.

⁵ Community Core Vision, een open source multitouch tracker. <http://ccv.nuigroup.com/>

4 SOFTWARE-ONTWERP

Tijdens het ontwerp van ons project hebben we ook enkele *use cases* gedefinieerd (zie appendix 2), om de gewenste functionaliteit voor onszelf klaar en duidelijk te maken. Op basis van deze use cases en het storyboard identificeerden we vrij snel enkele componenten die zeker in ons programma aanwezig zouden moeten zijn. Allereerst gebruiken we de *TUIO*⁶ abstractie laag om multitouch apparaten te ondersteunen. Daarnaast is er nog een framework nodig om in Java onze applicatie te kunnen schrijven, aangezien we beide ons met deze taal het comfortabelst voelen. Het bekendste Java multitouch framework is *MT4J*⁷. Dan zijn er nog componenten nodig om *file drops* mogelijk te maken, functionaliteit om MP3 metadata in te laden, Java *bindings* om Last.fm API oproepen te maken en code om een spring graph te kunnen tekenen.

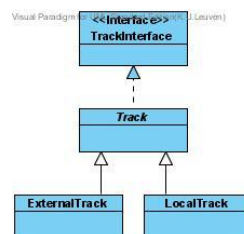
We hebben ook Python als taal overwogen, omdat het Python multitouch framework *PyMT*⁸ een grotere community en dus meer ondersteuning lijkt te hebben dan *MT4J*. Bovendien vinden we dat met Python visueel aantrekkelijkere programma's geschreven kunnen worden. Toch hebben we niet voor deze taal gekozen, omdat we er veel minder ervaring mee hebben en we het ons niet konden veroorloven om enkele tientallen extra uren te spenderen aan het leren (correct) werken met de taal.

De structuur van *TouchTrack* hebben we volgens het *MVC design pattern*⁹ ontworpen. Dit houdt in dat het project uit drie belangrijke delen bestaat: het "model" gedeelte dat alle programmologica bevat, de "views" die de user interface (UI) vormen en de "controllers" die de interactie tussen de UI en de models mogelijk maken. Deze structuur heeft als grote voordeel dat ons project erg modulair en dus gemakkelijk uitbreidbaar wordt. Het ontwerp wordt nu verder uitgelegd a.h.v. deze onderdelen. In appendix 3 staat een volledig¹⁰ UML diagram.

Model (programmologica)

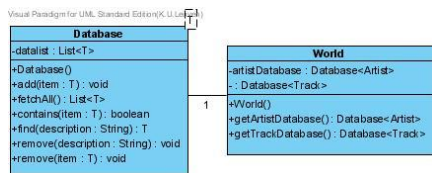
Een muzieknnummer stellen we intern voor m.b.v. de abstracte klasse *Track*. Deze heeft twee subklassen *ExternalTrack* en *LocalTrack*, omdat een nummer ofwel als MP3 lokaal beschikbaar is, ofwel ingeladen werd met enkel Last.fm data en dus waarschijnlijk niet in de bibliotheek van de gebruiker zit.

Om Tracks aan te maken, gebruiken we het *factory method* ontwerppatroon. In ons ontwerp is dit dus een klasse *TrackFactory* met een publieke, statische methode *createTrack*. Dit maakt het gemakkelijk voor de controller om *Track* objecten van het juiste type te maken.



Figuur 4 – Track klassen.

We hebben besloten dat de UI nooit rechtstreeks de modelklassen mag aanspreken. Dit gebeurt ofwel via de controllers, ofwel via interfaces die enkel specifieke methodes van een modelklasse beschikbaar stellen. Bijvoorbeeld, *Track* implementeert de interface *TrackInterface* om methodes zoals het opvragen van de titel, de afbeelding, ... voor de UI toegankelijk te maken. Zo kan de UI niet per ongeluk het interne model inconsistent maken.



Figuur 5 – Bijhouden van alle tracks en artists.

Artiesten worden voorgesteld met een simpele *Artist* klasse, met bijbehorende *ArtistInterface* om dezelfde redenen als voor de *TrackInterface*.

De collectie van alle model objecten moet natuurlijk ergens worden bijgehouden. Hiervoor hebben we een generische *Database* klasse ontworpen. Referenties naar de Database objecten met artiesten en met muzieknnummers worden bijgehouden in de *World* klasse.

View (UI)

De user interface bestaat uit drie delen: gesture listeners, graph components en scenes.

Een scene is een subklasse van de *MT4J* klasse *AbstractScene*. Deze klassen hebben elk een *canvas* waarop "getekend" kan worden. Ze stellen dus eigenlijk de zichtbare vensters met multitouch functionaliteit voor. *TouchTrack* bestaat uit slechts twee scenes: *TrackGraphScene* en *TrackDetailScene*.

⁶ <http://www.tuio.org/>

⁷ <http://www.mt4j.org/>

⁸ <http://pymt.txzone.net/>

⁹ Model View Controller. Zie <http://nl.wikipedia.org/wiki/Model-view-controller-model>

¹⁰ Met uitzondering van de test cases en exception klassen.

Controller

Eerst overwogen we meerdere *use case controllers*¹¹, maar het leek ons eenvoudiger en intuïtiever slechts twee controllers te voorzien: een *ArtistController* en een *TrackController*. Deze voorzien dus operaties om tracks of artiesten aan te maken, gelijkaardige tracks in te laden, tracks verwijderen, enzovoort... Controller operaties aanvaardden enkel primitieve types als argumenten en geven uitsluitend primitieve types terug of interfaces naar modelklassen (*ArtistInterface* of *TrackInterface*). Zie het UML diagram in appendix 3 voor alle operaties.

5 IMPLEMENTATIE

De implementatie van de programmatica (model) en van de controllers verliep vrij vlot, aangezien dit los staat van de multitouch functionaliteit die voor ons wel nieuw is. De user interface en het tekenen van de spring graph verliep veel stroever. Doorheen de hele implementatie hebben we niets aan het oorspronkelijke ontwerp veranderd, op enkele *gestures* na en een extra mogelijkheid om de graaf dynamisch uit te kunnen breiden (zoals in deel 2 over het storyboard reeds uitgelegd werd). In wat volgt, leggen we onze implementatie per onderdeel van het ontwerp uit.

touchtrack.model

De “modelklassen” die de logica bevatten, worden in de *touchtrack.model* package afgezonderd. Deze package bevat de *Database* en *World* klassen, die beschreven werden in deel 4 van dit verslag. Verder bevat het pakket nog de subpackages *touchtrack.model.artist* en *touchtrack.model.track* voor resp. Artist en Track gerelateerde klassen. De implementatie van al deze klassen is behoorlijk rechttoe rechtaan.

touchtrack.controller

De controllers kunnen interageren met de modelklassen. Bv. de methode *TrackController.createTrack* zal *TrackFactory.createTrack* aanroepen en het resulterende Track object toevoegen aan de (Track) database in het *World* object.

touchtrack.util

Het inladen van de MP3's gebeurt ook los van de “view”: hiervoor hebben we een klasse *TrackLoader* aangemaakt in de *touchtrack.util* (utilities) package. Deze klasse hoeft enkel de controllers op te roepen om Tracks toe te voegen en maakt gebruik van de Tritonus¹² API om metadata in te laden.

In de *TrackLoader* klasse worden ook de Last.fm oproepen gemaakt (*TrackLoader.getSimilarTracks*). We gebruiken een eenvoudige Java bibliotheek¹³ die Last.fm API bindings voorziet. Een klein probleem met deze library is dat sommige Last.fm API functies ontbreken: een voorbeeld is het opvragen van het “match” percentage van twee gelijkaardige muzieknnummers (vb. “Free Bird” van Lynyrd Skynyrd komt volgens Last.fm 35.54% overeen met “Fortunate Son” van CCR). Deze functie hebben we zelf toegevoegd en nadien een eigen gecompileerde versie van de bibliotheek gebruikt. Deze is terug te vinden in *TouchTrack/lib/last.fm-bindings-touchtrack.jar*.

De *util* package bevat daarnaast nog een klasse om MP3's af te spelen. Deze is eveneens gebaseerd op de Tritonus API.

Ten slotte bevat het utilities pakket nog een *FileDrop*¹⁴ klasse, om nieuwe MP3's die in de applicatie worden gesleept te detecteren. Bij de constructie van een *FileDrop* object wordt een “listener” object meegegeven, dat nieuwe MP3's kan inladen m.b.v. de *TrackLoader* klasse.

touchtrack.view

De user interface implementeren was beslist het moeilijkste deel van dit project. Een eerste obstakel was de gebrekkige documentatie van MT4J. De makers van dit framework bieden helaas slechts een beperkt aantal voorbeelden aan en minimale “Java docs”. De “community” achter MT4J lijkt niet erg groot, zodat elders voorbeelden en ondersteuning zoeken moeilijk was. Voor ons kwam het dus neer op *trial and error* tijdens het zoeken naar hoe MT4J scenes in elkaar zitten.

¹¹ Één controller per use case. Een use case controller bevat uiteraard enkel operaties die bij de use case hoort. Bv. *LoadSimilarTracksController*.

¹² <http://www.tritonius.org/>

¹³ <http://www.u-mass.de/lastfm/doc>

¹⁴ Geschreven door Robert Harder. <http://iharder.sourceforge.net/current/java/filedrop/>

Een volgend probleem was het tekenen van het netwerk. Hiervoor probeerden we eerst het open source framework JUNG¹⁵ te gebruiken, maar deze bleek achteraf niet geschikt te zijn voor ons project. JUNG is namelijk eerder bedoeld voor de analyse en visualisatie van data en ondersteunt ook geen *spring graph* waarin de nodes dynamisch van positie veranderen. Dus besloten we zelf een spring graph te implementeren en te tekenen.

Alle componenten van het netwerk staan in de *touchtrack.view.graph* package. Instanties van de klasse *TrackNode* worden geassocieerd met een *TrackInterface* en stellen dus muzieknnummers voor. Een *TrackSpring* verbindt de nodes en een *TrackGraph* bevat een lijst van alle nodes en springs, naast enkele methodes om deze te positioneren. Deze drie klassen uit de *graph* package kunnen zichzelf tekenen op het scherm: bij een node is dat de afbeelding, met de Track titel eronder en bij een spring is het een eenvoudige lijn.

Om de knopen en bogen in de graaf te positioneren, gebruiken we de fysica bibliotheek *traer.physics*¹⁶. Met deze bibliotheek wordt in *TrackGraph* een zogenaamd *particle system* aangemaakt. Elke *particle* komt overeen met een *TrackNode*. Dit systeem zorgt voor een goeie positionering van de knopen en de bogen. De lengte van een boog zegt hoe veel de muzieknnummers overeenkomen (zoals bepaald door Last.fm). De dikte ervan of de kleur heeft geen betekenis. Een node heeft een blauwe rand, wanneer die een *LocalTrack* voorstelt.

Een laatste probleem was het implementeren van de *gesture listeners*. Dit zijn dus klassen (uit de *touchtrack.view.gesture* package) die gedetecteerde multitouch gebaren moeten afhandelen. Gelukkig gebeurt de detectie van enkele "standaard gestures" door het MT4J framework zelf: dit zijn gebaren zoals tikken, tikken en slepen, zoomen, roteren, ... Het implementeren van een listener is dan ook niet erg moeilijk, ondanks de slechte documentatie.

We hadden echter wel problemen met een "tap listener" op de nodes. Zo'n *TrackNode* hebben we namelijk als een zogenaamde *MComponent* gedefinieerd, waarop normaal heel gemakkelijk een listener kan worden gedefinieerd, m.b.v. een *addGestureListener* methode. Dit werkte niet, dus hebben we ervoor gekozen om bij iedere "tap gesture" te checken in welke component de opgevangen x en y coördinaten liggen en vervolgens de correcte actie uit te voeren. Dit gebeurt in de klasse *TrackGraphGestureListener*.

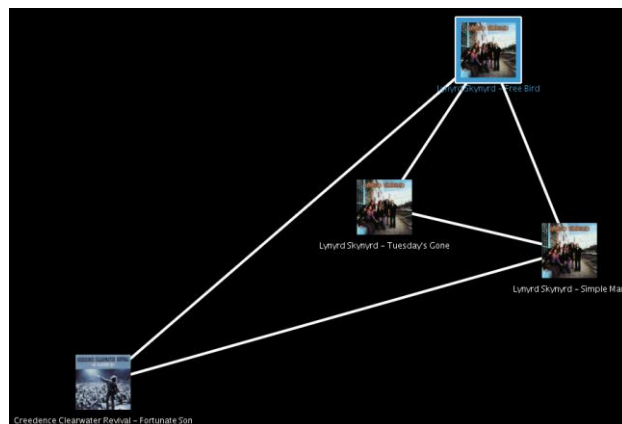
Alle netwerk componenten worden getekend op de *TrackGraphScene* uit de package *touchtrack.view.scene*. Wanneer een *TrackNode* wordt aangeraakt, dan wordt naar de *TrackDetailScene* gegaan, waar wat meer informatie over het muzieknnummer wordt getoond (indien beschikbaar).

6 RESULTAAT

Toegegeven, het bekomen resultaat heeft momenteel weinig praktisch nut. Het enige wat TouchTrack momenteel doet, is het visualiseren van een collectie MP3's en het toevoegen van gelijkaardige nummers aan het netwerk. De details van een muzieknnummer die bekeken kunnen worden door op een node te tikken, bestaan momenteel enkel uit een herhaling van de afbeelding, de titel en extra info gehaald van de Last.fm pagina. Aanvankelijk was het de bedoeling dat een MP3 ook zou kunnen worden afgespeeld. De functionaliteit om dit te doen is er wel (*Track.play()*), maar helaas is er nog geen *stop()* functie...

Toch zijn we tevreden met de werkende *spring graph*, die met dynamische gegevens kan werken en gemanipuleerd kan worden met multitouch gestures. Het was voor ons een goeie oefening op visualisatie, software ontwerp en het werken met verschillende (Java) frameworks.

TouchTrack is niet gebaseerd op een bestaande toepassing, maar na het zien van enkele demonstraties van andere groepen, ontdekten we dat er een Java applicatie¹⁷ bestaat die informatie van Last.fm afhaalt en voorstelt in een *spring graph*. De applicatie is weliswaar geen multitouch toepassing, maar de manier om de graaf te tekenen is gelijkaardig; de ontwikkelaar maakt ook gebruik van de *traer.physics* bibliotheek. Een (multitouch) toepassing met hetzelfde doel als TouchTrack vonden we echter niet.



Figuur 6 – Een TouchTrack netwerk.

¹⁵ <http://jung.sourceforge.net/>

¹⁶ <http://www.cs.princeton.edu/~traer/physics/>

¹⁷ <http://pages.swcp.com/~atomboy/lastfmgraph/>

7 OVER MULTITOUCH

Open source multitouch frameworks en toepassingen wonnen de laatste jaar aan populariteit en dat is te merken aan de kwaliteit. Toch ondervinden we dat een typisch probleem met open source software hier terug opduikt: doordat er een relatief groot aanbod is aan open source multitouch projecten van hobbyisten, is de *community* nogal verspreid. Dit heeft als gevolg dat de documentatie en ondersteuning voor die frameworks erg minimaal is, zodat ze een hoge “instapdrempel” hebben. De gebrekkig documentatie was dus duidelijk ons grootste obstakel in deze opdracht. Vele oplossingen werden d.m.v. *trial and error* gevonden.

Nog een “probleem” bij deze opdracht is dat het bedenken van een creatieve multitouch applicatie (rond het thema muziek) veel moeilijker is, dan het maken van een toepassing voor mobiele platformen zoals de Android. Een mogelijke oorzaak is volgens ons dat het gebruik van standalone multitouch apparaten nog niet zo wijd verspreid is, met uitzondering van mobiele apparaten met een touch screen.

Het ontwikkelen van multitouch mashups heeft dan weer als voordeel dat er voor heel veel programmeertalen open source bibliotheken bestaan die de interactie met het TUIO protocol verzorgen (zie de “Software” pagina van de TUIO.org site). Ontwikkelaars hoeven dus geen nieuwe taal te leren en hoeven zich niet bezig te houden met *low level* programmeerwerk. Dit verbaasde ons eerlijk gezegd, omdat we er vóór dit project nog geen idee van hadden dat “hobbyisten” multitouch toepassingen konden bouwen, die kunnen concurreren met *Microsoft Surface*¹⁸.

8 BESLUIT

Dit project hebben we anders aangepakt dan de vorige opdrachten, in de zin dat we ons deze keer bijna volledig aan het oorspronkelijk ontwerp hebben gehouden. We hebben onze tijd genomen om een haalbaar idee te bedenken en om dit in een storyboard te gieten. Daarna identificeerden we enkele *use cases* (met dank aan de lessen MOP) en bedachten we een software ontwerp gebaseerd op het *MVC design pattern*. Ten slotte implementeerden we ieder component in Java, met behulp van het MT4J framework en de pak andere bibliotheken die we in dit verslag vermeld hebben.

De implementatie verliep doorgaans vrij vlot, omdat we heel goed wisten wat ons uiteindelijke doel was. De grootste problemen ontstonden bij het implementeren van de *spring graph*. De oorzaak ligt volgens ons bij de gebrekkige documentatie van MT4J, *traer.physics*, ... Oplossingen vonden we voornamelijk via *trial and error*.

Ondanks dat het resultaat weinig praktisch nut heeft, zijn we er toch vrij tevreden van. Het voldoet bijna volledig aan ons oorspronkelijk ontwerp en we hebben behoorlijk wat bijgeleerd over visualisatie, multitouch frameworks en software ontwerp.

¹⁸ <http://www.microsoft.com/surface/>

APPENDIX 1 – TIJDSBESTEDING

	Laurens Van Keer	Bram Truyaert	Totaal
Experimenteren met Processing	3	15	18
Storyboard	1	0	1
Ontwerp	5	1	6
Multitouch doos maken	0	3.5	3.5
Modelklassen implementeren	3	1	4
Controllers implementeren	2	0	2
Tekenen van de graph	32	4	36
Inladen Track metadata (util)	6	12	18
Verslag schrijven	8	0.5	8.5
Video maken	0	2	2
Totaal	60	39	99

APPENDIX 2 – TOUCHTRACK USE CASES

Use Case: Load MP3

Primary Actor: The user of the system

Basic Flow:

1. The user drags and drops an MP3 file into the application.
 2. The system loads and processes the metadata from the file.
 3. The system calls the Last.FM API to load extra data (the image e.g.) about the given track.
 4. The system calls the Last.FM API to load similar tracks into the database.
 5. The system shows the user a graph in which the selected track is the "base" node and in which the similar tracks are "neighbour" nodes.
-

Use Case: Load Similar Tracks

Primary Actor: The user of the system

Basic Flow:

1. The user selects a node (track).
 2. The system calls the Last.FM API to load similar tracks into the database.
 3. The system shows the user an EXTENDED graph in which the selected track is the "base" node and in which the similar tracks are "neighbour" nodes.
-

Use Case: Load Track Details

Primary Actor: The user of the system

Basic Flow:

1. The user selects a node (track).
2. The user selects an option to show the track details.
3. The system shows the user a window with the track details.

Appendix 3 – Volledige UML diagram

