

Android toepassing

Groep 7

Blog <http://forceflow.ulyssis.be/mume09/>

Bart Tuts, 2de master Computerwetenschappen, optie gedistribueerde systemen

Jeroen Baert, 2de master Computerwetenschappen, optie mens-machine communicatie,

Laurens De Vocht, 1e master Computerwetenschappen, optie mens-machine communicatie

Jan Grauwels, 1e master Computerwetenschappen, optie mens-machine communicatie

Abstract—Dit is het verslag voor opgave 2 van multimedia, van groep 7. Wij hebben een Android-toepassing uitgewerkt voor uitgewerkt voor het genereren en grafisch representeren van playlists op feestjes, op basis van de informatie in de Last.FM profielen van de aanwezigen. de belangrijkste uitdagingen waren het binnenhalen van informatie van Last.fm op de achtergrond zodat het het gebruik van de applicatie niet verstoort en het implementeren van de visualisatie op de beperkte graphics API.

Ingediend op: 2 november 2009

1. Idee

Het kern-idee van onze applicatie is behouden. Het doel van de applicatie is nog steeds dat een groep mensen die samenkomen, op basis van de gegevens in hun last.fm accounts kunnen luisteren naar muziek die ze allemaal aangenaam vinden.

In onze vorige applicatie konden gebruikers zich inschrijven via de desktopapplicatie die we in flex hadden geïmplementeerd. Het is echter niet zo evident om alle gasten op een bepaald feestje zich te laten inschrijven op de dure Androidphone van de gastheer of de dj. Daarom hebben we ervoor gezorgd dat er op een server een lijst van verschillende parties wordt bijgehouden en een lijst van gebruikers.

Op elke Androidphone die onze app geïnstalleerd heeft kunnen parties toegevoegd worden en kunnen gebruikers zich inschrijven op een of meerdere feestjes. De gastheer of dj hoeft dan gewoon te synchroniseren met de centrale server. Een bijkomend voordeel daarbij is dat het ook mogelijk is om onze desktopapplicatie ook te laten samenwerken met deze centrale server. Zo kunnen ook gasten van een bepaald feestje die niet in het bezit zijn van een Androidphone zich inschrijven.

We hebben onze applicatie zo opgebouwd dat de meerwaarde van deze gedistribueerde versie van onze app ook zit in het feit dat het nu mogelijk is dat het inschrijven kan gebeuren voor dat het feestje effectief begint. Aan elke party die is opgeslagen in de database wordt ook een tijdstip en een plaats gekoppeld. Onze app zou eenvoudig kunnen worden uitgebreid met maps en eventueel een notificatie als de party bijna begint. Omdat dit laatste eigenlijk bijkomstige features zijn, hebben we ervoor gekozen om deze niet te implementeren. We voorzien gewoon het framework om deze later eenvoudig te kunnen toevoegen. Daarnaast

hebben we ons gefocust op het aanbieden van dezelfde functionaliteit als de desktopversie van onze app.

2. Storyboard

Het storyboard is in essentie hetzelfde gebleven als bij de flex-applicatie. De features en de algemene opeenvolging van acties is dezelfde gebleven, hoewel het uitzicht van de applicatie uiteraard aangepast werd aan een handheld device.

Het voornaamste verschil in de interface is dat alles gesplitst is over verschillende schermen. Dit betekent dat nu de userlist, playlist, visualisatie en modify elk afzonderlijk schermvullend zijn. Via een overzichtslijst die bij de start van de applicatie wordt getoond is het mogelijk om te navigeren tussen deze verschillende componenten. Van de ene component naar een andere navigeren gebeurt altijd via de overzichtslijst. En vanuit een component kan er naar deze overzichtslijst genavigeerd worden door op de *back*-knop van de android phone te drukken. Vooral voor de visualisatie was het belangrijk dat het volledige scherm kon benut worden omdat deze anders snel onoverzichtelijk zou worden.

Het was oorspronkelijk ook de bedoeling om een extra component *Statistics* te voorzien als tegenhanger (en eventuele back-up) voor de visualisatie. Deze zou een aantal tabs bevatten die een tekstuele representatie van de gegevens zou bevatten die in de visualisatie grafisch worden voorgesteld. Zo zou bvb. één tab kunnen aangeven hoeveel artiesten de gebruikers gemeenschappelijk hebben met de rest van de gebruikers. Een andere tab zou bvb. kunnen dienen om in een lijst van artiesten aan te duiden hoeveel gebruikers een bepaalde artiest in hun top 10 hebben staan. Deze component is uiteindelijk niet geïmplementeerd hoewel de overzichtslijst van de user-interface wel een item *Statistics* bevat.

In deze overzichtslijst is er ook een item *Modify* voorzien zoals ook in het oorspronkelijke storyboard getekend staat en die ook dezelfde functionaliteit zou hebben. Maar net zoals vorige keer is deze extra component niet geïmplementeerd.

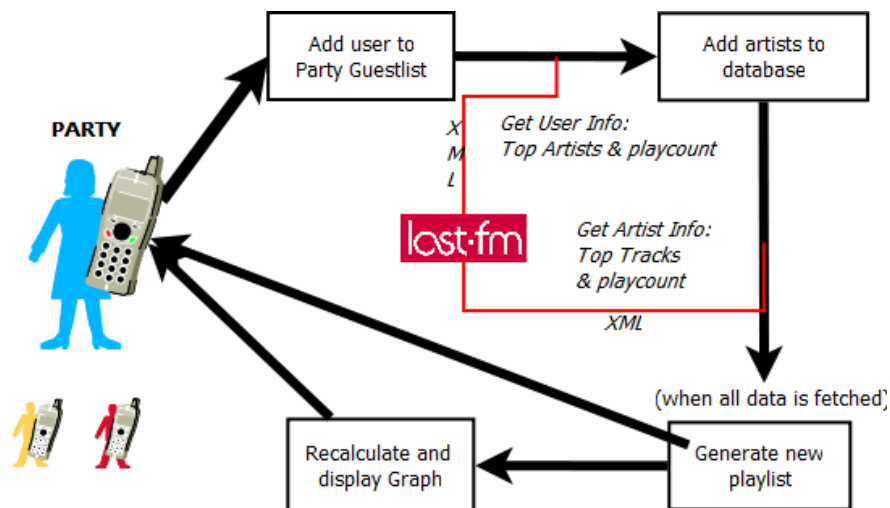
Om een Last.fm-gebruiker toe te voegen moet de gebruiker nu op een knop in het opties-menu klikken van de userlist waarna er een dialoogvenster tevoorschijn komt dat om een Last.fm-gebruikersnaam vraagt (i.p.v. een tekstveld dat permanent in de userinterface aanwezig is zoals in de flex-applicatie).

Ook de playlist kan gegenereerd worden door op de knop *Update* in het opties-menu van de playlist te klikken.

Omdat de schermresolutie van een de android phone veel te klein is om een volledige graaf te laten zien, werd overgeschakeld op een zogenaamde 'roamer'. Een voorbeeld hiervan is de constellation roamer van Asterisq. Hierbij is er 1 centrale node, omringt door een aantal burens, die op hun buurt mogelijk weer een aantal burens hebben, enz. Indien op een van de niet-centrale nodes wordt geklikt, wordt die node de nieuwe centrale node en worden enkel zijn burens nog getoond. In plaats van alle nodes te tonen, toont men enkel een subset van

nabijgelegen nodes van de huidige centrale node. Via het klikken op burens kan toch de volledige graaf genavigeerd worden. Het nadeel van de roamer is dat het algemene beeld van de graaf verloren gaat. Het voordeel is dat onbeperkt grote grafen zonder probleem genavigeerd en (lokaal) getoond kunnen worden. In de implementatie werd omwille van de beperkte schermgrootte gekozen om enkel de rechtstreekse burens te tonen. Wanneer een gebruiker centraal staat worden zijn top 10 artiesten getoond als burens. Wanneer een artiest centraal staat worden de gebruikers bij wie deze artiest in de top 10 staat getoond als burens.¹

3. Software-Ontwerp



Luisterdata

Net zoals in de Flex-implementatie wordt ook hier Last.FM-luisterdata gebruikt. Gezien de beperkte downloadsnelheden van een Android-toestel - mobiele netwerken zijn immers niet altijd even snel/betrouwbaar als een traditionele kabelverbinding - werd echter wel rekening gehouden met het feit dat er niet evenveel data zou worden opgevraagd. Dezelfde API methodes worden aangeroepen:

- **user.GetTopArtists**(username, api_key, (optional) timespan)
Bij het toevoegen van een nieuwe gebruiker.
- **artist.GetTopTracks**(artistname, api_key, (optional) timespan)
Bij het toevoegen van een nieuwe artiest. (Als gevolg van een nieuwe gebruiker)

Voor deze API-methodes is geen user authentication nodig, een API key volstaat. Dit is een bijkomend voordeel, aangezien de authenticatiemethode² voor de Last.FM API nodeloos ingewikkeld is en we aan een Android-implementatie hiervan veel tijd zouden verloren zijn.

1. <http://asterisq.com/products/constellation/roamer/demo>

2. <http://www.last.fm/api/authspec>

XML Ophalen

Het kiezen van een geschikte http-component op het Android-platform om de XML-informatie binnen te halen werd bemoeilijkt door het feit dat Android standaard 2 verschillende HttpClient-gerelateerde packages meeleverd: *java.net.http.** en *apache.http.**. Waarom blijft ons een raadsel, gezien de aangeboden klassen niet onderling compatibel zijn. Wij hebben uiteindelijk geopteerd voor de Apache-packages, gezien de hoeveelheid beschikbare documentatie over het gebruik van deze componenten.

XML Parsen

Bij de Flex-implementatie was het mogelijk om XML te parsen op een **Document Object Model (DOM)**-manier: de hele XML file werd ingelezen in het geheugen en voorgesteld als een tree, waarop kon genavigeerd/gezocht worden. Deze manier van parsen is echter een slecht idee op een apparaat dat beperkt is in geheugenruimte, zoals een Android phone. Door een middelgroot XML bestand (30 kb) om te zetten naar een XMLTree kan je al gauw een heel deel van het (bovendien gedeeld!) geheugen vullen. Het is dan ook geen verrassing dat de Android API geen methodes aanbiedt om een XML-stream om te zetten naar een Tree of een Structured List.

Als alternatief voor DOM-parsing bestaat er Serial Access Parsing, geïmplementeerd in Android door middel van SAX³ (Simple API for XML). Het komt erop neer dat tijdens het lezen van een XML-bestand er events worden gestuurd wanneer een tag geopend/gesloten wordt. Het was dan aan ons om een gepaste handler te definiëren die door middel van interne status-booleans bijhield welke tags momenteel open/gesloten waren, om zo de juiste informatie uit de XML-stream te extraheren.

Asynchrone Tasks

Aangezien het ophalen en parsen van artiest/track-data veel tijd vraagt en deze acties bovendien vaak moeten uitgevoerd worden (*Worst Case Scenario*: 1 nieuwe gebruiker = 50 nieuwe artiesten) was het belangrijk dat deze handelingen asynchroon verliepen, zodat de UI thread met de visualisatie niet zou 'hangen' bij elke aanvraag. Na het doorlopen van de Android-tutorials werd eerst gedacht om het ophalen van de data als een Android Service te implementeren. Na verder onderzoek bleek echter dat dit niet de gewenste functionaliteit bood: we willen de data immers niet volledig *los* van een activity ophalen: we hebben de data nodig in de Visualisatie, dus we willen ze graag ophalen via die Activity, maar niet noodzakelijk in dezelfde thread.

Sinds Android 1.5 is echter een oplossing beschikbaar: *AsyncTasks*. Dit zijn aparte entiteiten die in een aparte worker thread een actie kunnen uitvoeren. Deze activiteiten kunnen enkel van in de UI-thread opgeroepen worden - Android handelt verder de (vaak frustrerende) threads-logica af. Tijdens het implementeren van onze eigen Asynchrone Tasks (1 voor elke soort API-call en XML-parsing pass) ontdekten we een fout in de Google Android Development Blog⁴. De AsyncTask klasse is templated, en hoort dus aangemaakt te worden

3. http://en.wikipedia.org/wiki/Simple_API_for_XML

4. <http://android-developers.blogspot.com/2009/05/painless-threading.html>

met de juiste types, hetgeen in de blogpost niet gebeurt. Enkele uurtjes en een gesprek op het #android-dev chatkanaal later was de fout opgehelderd:

```
<forceflow> They don't specialize the types in the blogpost
<@ctate> man. I need to ask Romain what's up with that.
<forceflow> The blogpost code is wrong, I think. It needs to implement the given template.
<@ctate> Indeed, it is. I would recommend actually specializing the AsyncTask, should be fine then.
<@ctate> I will chastise Romain the next time I see him :)
<forceflow> Are you a Google employee, ctate?
<@ctate> yes
```

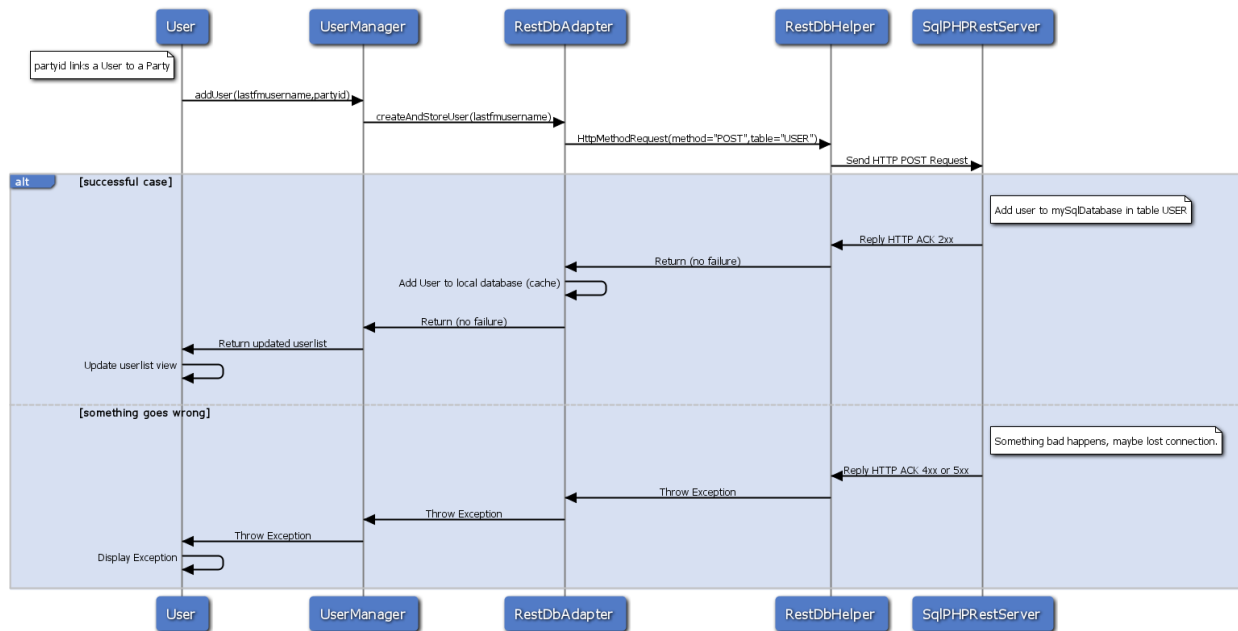
Persistentie

Het opslagaspect bestaat uit 2 delen: een lokale database (werkt als soort cache) en een remote database. Alle bevestigingen worden (GET Requests) gedaan op de lokale SQLite database die voorzien is door het Androidplatform. Deze database kan echter gesynchroniseerd worden met een remote database in MySQL. Daarbij wordt de lokale database gelijkgezet met de remote versie. Bij het toevoegen van een gebruiker of een party zal deze eerst worden toegevoegd aan de remote server en als deze operatie slaagt zal deze ook worden bijgehouden in de lokale database. Door te werken met een lokale database die slechts op tijdstippen wordt gesynchroniseerd die de gebruiker zelf kiest kan worden vermeden dat de applicatie op momenten heel traag functioneert telkens de lijst van gebruikers wordt opgevraagd.

De interactie met de database gebeurt via een middlewareplatform RESTful. Dit houdt in dat de operaties op de MySQL database niet via een JDBC connectie rechtstreeks verlopen maar via een PHP script dat draait op dezelfde server. Dit PHP script transformeert de database tot gewone xml bestanden. Op deze xml bestanden kunnen gewone HTTP operaties gebeuren die verwerkt worden door het PHP-script. Bij het ophalen van informatie moeten de xml-files geparst worden, dit gebeurt op een volledig analoge manier als bij lastfm-gegevens. Dit php-script hebben we niet zelf geïmplementeerd maar aangepast voor onze applicatie. Dit kan hier: <http://phprestsql.sourceforge.net/> teruggevonden worden. Dit is heel handig aangezien Android geen manier voorziet om een connectie met een MySQL database te openen. Het enige nadeel van deze methode is dat de overdracht stateless is. Als er iets foutloopt moet de volledige transactie herbeginnen. De enige feedback van de server is dan ook de HTTP-acknowledgement met de statuscode. In het onderstaande sequencediagram wordt deze aanpak duidelijk.

We hebben het framework van onze applicatie zo gemaakt dat er een abstracte klass DbAdapter is waardoor het heel eenvoudig is om een andere soort interactie met een remote database te implementeren. Zelf hebben we zo LocalDbAdapter voorzien. Deze is volledig hetzelfde als de RestDbAdapter, alleen dat hierbij de gegevens niet worden gesynchroniseerd met een remote server.

Voorbeeldje met sequencediagram:



Interface / GUI

Er is gekozen om via één overzichtslijst (de PartyVis-klasse op het klasse-diagram) tussen de verschillende componenten (Userlist, Playlist, Modify, Visualisation, Statistics) van de userinterface te navigeren. Deze componenten zijn allen android activities. Er kan vanuit een activity terug genavigeerd worden naar deze overzichtslijst door op de *back*-knop van de android phone te drukken (zoals vermeld bij het story-board).

Voor de acties die per activity mogelijk zijn is er geopteerd om menu-items te maken in het opties-menu van de respectievelijke activity. Dit is een standaard menu dat het android-platform aanbiedt en dat zichtbaar wordt wanneer er in een bepaalde activity op de knop *menu* van de android phone wordt gedrukt.

Per activity is de layout gedefinieerd in een XML-file. Ook voor de data (zoals bvb. strings) die in de userinterface wordt weergegeven is er gekozen om deze in XML-files te definiëren. Op deze manier zijn de operaties, die in de activities gedefinieerd zijn, mooi gescheiden van de layout en de data. Wat het dus gemakkelijker maakt om snel de layout of data van een activity aan te passen zonder dat de volledige activity herschreven moet worden.

Een alternatief (en het oorspronkelijke) ontwerp voor de userinterface was om zonder overzichtslijst (PartyVis) te werken. Het was dan de bedoeling om in het opties-menu van elke activity een verwijzing naar alle andere activities te zetten. Al gauw bleek echter dat dit onoverzichtelijker was en meer implementatie-werk vereiste.

Visualisatie

Er wordt gekozen om voor de visualisatie geen gebruik meer te maken van bewegende veer-achtige verbindingen omwille van onze twijfels i.v.m. performantie. We hebben enkele websites met javascript en java applet voorbeelden geopend in zowel de emulator als op de echte android phone en deze bleken erg traag en schokkend te bewegen. Hoewel een rechtstreekse java-implementatie zeker sneller zou zijn geweest, wilden we het risico niet lopen 40 uur te developen om dan vast te stellen dat de hardware de berekeningen niet aankan.

Een eerst blik op het beschikbare animatie-framework van android leerde ons ook dat dit vooral geschikt is voor animaties die voor compilatie worden gedefinieerd. Er moet namelijk een klasse of xml file gemaakt worden die de animatie en zijn parameters volledig beschrijft. Implementatie van complexe, voortdurend veranderende animaties zoals een verensysteem leek ons dan ook vrij omslachtig. Een fancy animatie is ook geen prioriteit in ons applicatie. Op vrijdag 30 oktober kregen we een demo te zien van een ander groepje. Zij hadden besloten om wel een bewegend verensysteem te implementeren. De demo toonde aan dat het wel mogelijk is een vrij performante animatie te maken (het systeem bestond weliswaar maar uit 1 veer met 2 nodes op de uiteinden), hoewel onze vermoedens over de werklust ervan bevestigd werden. Het team stelde namelijk dat ze al hun development tijd tot op dat moment gebruikt hadden om de animatie werkende te krijgen.

De visualisatie component bestaat in de eerst plaats uit een visualisationManager. Hierin wordt ingesteld welke gebruiker / artiest afgebeeld moeten worden. De visualisatiemanager is verder verantwoordelijk voor het omzetten van de business objects naar nodes en ook het plaatsen van de nodes op een grid. Een Android applicatie gebruikt normaal een adapter om informatie in het model automatisch in de view te plaatsen. Deze adapters zijn echter vooral bedoeld om bvb. een gegeven ArrayList van Strings in een ListView in de GUI te plaatsen. Gezien de zeer specifieke en vrij complexe plaatsing van de objecten op een canvas, was het gebruik van een adapter voor de visualisatie component niet mogelijk.

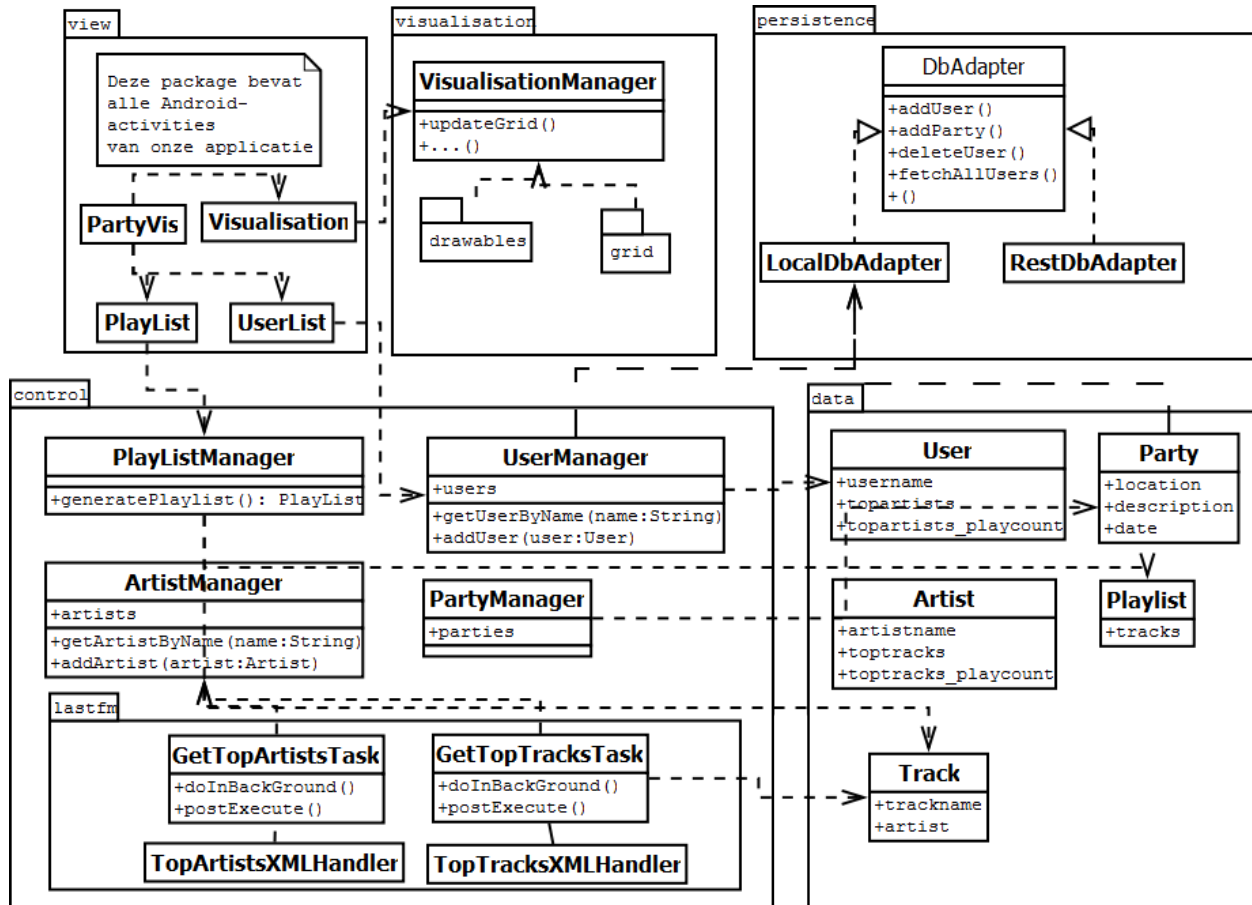
VisualizationGrid staat in voor de opslag van de tekenbare nodes, de lay-outing van het grid op een canvas en het genereren van een gezamenlijk Drawable object van alle de nodes en edges. met lay-outing wordt bedoeld het berekenen van de breedte van kolommen en hoogte van rijen, alsook de nodige random rondom op basis van het aantal rijen/kolommen en de schermresolutie. VisualizationGrid berekent op basis van de schermresolutie en de minimale grootte van een node ook hoeveel rijen en kolommen en maximaal op het scherm passen.

Het VisualizationGrid heeft geen gewone matrixvorm. Er zijn weliswaar rijen en kolommen maar de oneven rijen zijn een halve rijhoogte lager gezet dan de oneven rijen. De reden hiervoor is dat labels met artiestennamen al snel vrij veel plaats kunnen innemen en door elke oneven kolom een halve rijhoogte lager te plaatsen, overlappen deze labels niet. Het alternatief is de kolommen breder maken, en dus smaller, maar dat is op een apparaat met zo'n beperkte resolutie iets wat we zoveel mogelijk willen vermijden.

De visualisatie wordt geupdatet wanneer:

- Een nieuwe gebruiker is toegevoegd en zijn top artiesten gedownload zijn.
- In de visualisatie op een niet-centrale node wordt geklikt
- In het visualisatie-menu de "focus on" optie wordt gebruikt

Klassendiagramma



Het klassendiagramma bevat enkele overeenkomsten met dat van de Flex-implementatie:

- Er wordt nog steeds gewerkt met het MVC-principe (**M**odel **V**iew **C**ontroller). Onze packagestructuur is ingedeeld volgens deze werkwijze.
- Het **data** package bevat alle klassen die als enige functie het bijhouden van data hebben: *User*, *Artist* en *Track* zijn equivalent aan de Flex-iteratie van het programma, bijkomende klassen zijn *Playlist* en *Party*.
- In het package **control** zitten de klassen die de logica van de applicatie implementeren
 - *ArtistManager*, *UserManager* en *PartyManager* zorgen ervoor dat er voor elke user, artist en party slechts 1 object aanwezig is in de applicatie. Bij deze managers moeten gebruikers, artiesten en feestjes geregistreerd en gederegistreerd worden.

- *PlaylistManager* genereert een playlist op aanvraag.
- In het package **control.lastfm** zitten de asynchrone tasks die dienen om de Last.FM data binnen te halen en te parsen: *GetTopArtistTask* en *GetTopTracksTask*.
- In het package **visualisation** zitten de klassen om de *visualisation* activity te ondersteunen.
- In het package **view** zitten alle activiteiten: hierin wordt de flow van de applicatie en de layout van de GUI gedefinieerd.
- In het package **persistence** zitten alle adapters die nodig zijn om data te *persisten*: door het gebruik van een gepaste databaseadapter kan data worden weggeschreven naar een lokale of remote database.

4. Implementatie in Android

Interface / GUI

Zoals vermeld bij Software-Ontwerp is er gekozen om de layout en data in XML-files te definiëren. Dit gaf aanleiding tot een aantal problemen.

Eén van deze problemen was dat de code moeilijker te debuggen was aangezien de errors zich vaak/meestal voordeden in de XML. Het was dan moeilijk om te vinden wat er nu precies misliep in de XML. Zo bleek het bvb. dat een *ListActivity* vereist dat de layout-XML-file een specifieke default-textview moet bevatten, iets wat in één regel in de API van android beschreven staat en waar nogal gauw wordt overgelezen. De methode om dit soort problemen op te lossen was dan ook vaak heel nauwkeurig in de API gaan lezen om geen details over het hoofd te zien.

Een ander moeilijkheid was soms om specifieke data op te halen uit de data-XML-files. Zo kan men bvb. een array definiëren in een XML-file maar dan was het bvb. niet mogelijk om één specifiek item van die array op te vragen in de activity. Waardoor deze items dan toch moesten gedefinieerd worden in de activity zelf.

Een probleem van totaal andere aard was dat de standaard componenten die android aanbiedt voor de userinterface vaak te kort schoten op gebied van aanpasbaarheid. Zo kan er bvb. gebruik gemaakt worden van de standaard tabs die android aanbiedt maar deze namen te veel plaats in voor onze applicatie omdat er per tab ruimte werd voorzien voor een pictogram te plaatsen (iets waar wij geen behoefte aan hadden). Het was dan echter niet mogelijk om de hoogte van deze tabs simpelweg te verkleinen. Zodat er uiteindelijk zelf klassen en xml-layouts moesten geïmplementeerd worden om kleinere tabs te voorzien.

Visualisatie

De burens van de centrale node dienen uniform maar toch enigzins random verdeeld te worden over het scherm. Hiertoe werd volgend algoritme uitgewerkt: Een grid van n vakjes wordt initieel voorgesteld als n intervallen met lengte 1 die langs elkaar op een rechte worden geplaatst. we kiezen een random floating point getal in het interval $[0, n[$. Indien dit bvb 12,423 is plaatsen we in vakje 13 de eerste node. Vervolgens stellen we de breedte van interval 13 in op 0. De lengte van de intervallen van de directe burens in het grid wordt door x gedeeld en die van de burens van de burens wordt door $x/2$ gedeeld. Er wordt dan een nieuw random getal gekozen in het totale interval en gekeken in welk interval deze ligt. Aangezien de intervallen van de directe en indirecte burens van de vorige keuze kleiner zijn geworden is de kans ook kleiner dat de volgende node hierin geplaatst wordt. Omwillen van tijdsgebrek werd dit algoritme niet geïmplementeerd, maar vervangen door een manueel

ingestelde volgorde. Het implementeren, integreren en finetunen van het algoritme zoals het hier hierboven beschreven wordt, zou ongeveer 3 uur kosten. Met finetunen wordt bedoeld een waarde voor x zoeken die op de meeste resoluties een vrij goede verdeling geeft.

De visualisatie klassen zijn origineel opgebouwd vanuit het idee dat de applicatie zou kunnen draaien op android phones met verschillende schermresoluties. De belangrijkste stukken code (bvb de VisualizationGrid klasse) zijn hier inderdaad toe in staat. Op een aantal plaatsen, buiten de visualisatie component wordt er echter expliciet of impliciet uitgegaan van de standaard schermresolutie en een grid van 5 kolommen en 8 rijen. De reden hiervoor is opnieuw tijdsbesparing. Het aanpassen van alle code om te kunnen werken op alle mogelijk schermresoluties bedraagt zo'n 4 uur.

Moeilijkheden bij visualisatie

- De Android graphics API lijkt weinig matuur. Veel van de zelfgeschreven klassen in het project zijn klassen waarvan men zou verwachten dat ze zich al in het graphics framework bevinden. Stel dat men 10 identieke cirkels wilt tekenen. Dan moet men 10 ovalShapes maken, deze elk toevoegen aan een eigen ShapeDrawable waaraan men de boundingbox van de cirkel meegeeft en dan de kleur en lijndikte instellen door voor elk van de 10 apart `shapeDrawable.getPaint().setColor()` en `shapeDrawable.getPaint().setStrokeWidth()` op te roepen. Om dit te incapsuleren creeerde ik een klasse CircleDrawable waaraan je gewoon positie, straal en een vooraf gedefinieerd Paint kunt meegeven, wat veel gemakkelijker en dynamischer werkt. Je kan bvb met 1 lijntje code van Paint object wisselen.
- Wanneer in de `onDraw()` methode `canvas.drawRect()` wordt gedaan, wordt er nog niets getekend. Alles wordt blijkbaar in een queue of zo geplaatst en achteraf ineens getekend. Ettelijke uren werden verloren doordat een redelijk ingewikkeld stuk code helemaal niet leek te werken. Het stuk code was echter volledige juist. Het probleem zat in een fout verderop, waar ik me van bewust was. Deze deed echter de applicatie crashen vlak voor het einde van de `onDraw` methode, waardoor er niets getekend werd.
- Er werd ook redelijk wat tijd verloren aan een mogelijk verwant probleem. Om toch een beetje animatie te voorzien werd gepoogd bij het aanklikken van een node, alle andere nodes uit te laden faden, de aangeklikte node naar het centrum te transleren en dan de nieuwe burens in te faden. De methode `startAnimation()` lijkt echter niets te doen. na het beëindigen van de `onDraw` methode gebeurt er helemaal niets. wnr men dan op een willekeurige plaats klikt, krijgen we plots het eindresultaat van de animatie te zien. Zelfs bij animaties die meer dan een minuut moeten duren wordt hetzelfde vastgesteld. Het zou kunnen dat dit ook iets te maken heeft met het oproepen van `startTransition` binnen `onDraw`, hoewel dit niet bevestigd is. De enige mogelijke workaround voor dynamisch gecreeerde animaties lijkt deze incapsuleren, opslaan in een queue, vanuit een andere thread voortdurend pollen en animaties gesynchroniseerd met de eerste thread (GUI thread) te starten.

- Voor click events wordt overal verwezen naar de interface `onClickListener`. Deze is echter ontoereikend omdat enkel wordt gemeld dat er op een bepaalde View geklikt is. Het is niet mogelijk op te vragen op welke x en y coördinaat deze klik gebeurde. Tijdens de demo's op vrijdag 30 oktober was er gelukkig een ander groepje dat nu ook een applicatie heeft gebouwd zoals onze spring graph. Zij waren op hetzelfde probleem gestoten en hadden de methode `onTouchListener(Event)` gevonden. Deze dient eigenlijk voor het afhandelen van 'stroke' events (met de vinger over het scherm vegen) maar kan ook voor clicks worden gebruikt. Het meegegeven event argument bevat de x en y coördinaat op de canvas waar geklikt werd. Zij waren zo vriendelijk deze informatie met ons te delen, waarvoor dank!

5. Resultaat

Conceptueel

Hoewel bepaalde delen van onze applicatie, zoals de roamer-visualisatie of het afhalen van last.fm data ongetwijfeld niet uniek zijn, is ons doel toch steeds geweest een unieke, nuttige applicatie voor de gebruiker te creëren en niet een bestaande applicatie te repliceren op het android platform. We denken dat we hier met onze PartyVis applicatie ook op het Android platform opnieuw in geslaagd zijn. Het basisconcept van de applicatie is bewaard gebleven en we denken dat we de verschillende features goed hebben aangepast aan het Android platform.

Implementatie

Bijna applicatie features werken maar de implementatie is geen onverdeeld succes.

De Last.fm data retrieval component is volledig operationeel.

De playlistgeneratie is volledig operationeel.

De visualisatie is operationeel en vervult de taak die hij moet vervullen, maar door toevoeging van enige animatie en finetuning van het uitzicht van de nodes had hij er wel een stuk aantrekkelijker kunnen uitzien.

De modify en statistics menu's zijn niet geïmplementeerd wegens tijdstekort.

De persistence component werkt zeer goed wat betreft het opslaan en bijhouden van informatie. Er is helaas wel nog een onvindbaar memory leak. Af en toe duiken er ook wat problemen op bij het opstarten als de opgeslagen informatie terug wordt ingeladen en de bijhorende business objecten worden gecreëerd.

Teamwork

Tijdens de vorige iteratie bleek het soms zeer moeilijk om geschikte werkmomenten te vinden, omwille van onze zeer verschillende uurroosters. Daarom besloten we deze iteratie minder af te spreken en meer individueel te werken. Ieder kreeg een component toegewezen en werkte deze uit. Tijdens het weekend en op maandag dienden de componenten nog te worden geïntegreerd tot een volledige applicatie. Hierbij bleek echter dat niet alle componenten even grondig getest waren. Er waren ook enkele features waarvan het niet heel duidelijk was in welke component ze thuishoorden en die waarschijnlijk als gevolg daarvan niet volledig afgewerkt waren. Dit leidde tot een redelijk lange en frustrerende integratiesessie. Voor de volgende iteratie zullen we daarom terug

meer regelmatig overleg plegen, ideeën aftoetsen bij de andere teamleden en een meer gedetailleerdere werkverdeling opstellen. We zullen ook bekijken of het niet haalbaar is telkens met op z'n minst 2 in hetzelfde lokaal te werken of eventueel zelfs pair programming toe te passen. Deze techniek is immers zeer geschikt om bovenstaande problemen te vermijden.

6. **Over Android**

Platform

Het bestaan van een free, open source developmentplatform voor de android phones is uiteraard op zich al mooi. Het betekent dat de wereld van applicaties op handheld devices opengaat voor open source developers. Het android framework is goed voorzien om snel vrij eenvoudige applicaties te maken op basis van de aangeboden interface-componenten. Indien men echter wil afwijken van deze standaard componenten wordt het al snel een stuk ingewikkelder. De graphics API lijkt in elk geval nog niet voldoende matuur om zelf custom interfaces en interactieve animaties te tonen.

Voor het bouwen van eenvoudige applicaties die werken met tekst of statische afbeeldingen en waarvan de interface opgebouwd is uit standaard componenten, lijkt Android zeer aangeraden. Voor volledig custom-build applicaties lijkt het toch nog wat te vroeg. Een van de grote architecturale veranderingen in Android 2 is overigens een nieuwe graphics architecture dus het lijkt erop dat ik niet alleen sta in mijn kritiek op het huidige framework.⁵

Development tools

De android plugin en de bijhorende tools zoals logcat en de emulator vertonen. Als een applicatie meermaals na elkaar wordt gecompileerd en gestart dan werkt soms de emulator niet. Ook de logCat tool stopt soms plots met het weergeven van berichten. Het gebeurde ook soms dat de berichten wel getoond werden, maar al snel weer verdwenen, alsof iemand elke seconde op 'clear' drukte. Deze quircks werden op verschillende computers met verschillende OS's vastgesteld. Een eigenschap die veel irritatie heeft opgewekt bij de teamleden is dat de plugin weigert een programma te compileren wanneer een van de klassen nog fouten bevat, ook wanneer deze klasse op geen enkele manier in het project gebruikt wordt. Dit verhindert de mogelijkheid om even een stuk code testen terwijl de klasse nog niet volledig geïmplementeerd is. Het is ook spijtig dat het niet mogelijk is het scherm van de Android te draaien. Men moet kiezen tussen een horizontale of verticale emulator. Hoewel de android eclipse-plugin niet het niveau haalt van de java-functionaliteit van eclipse, kan globaal gezien toch worden gesteld dat de plugin vrij deftig werkt.

7. **Extra**

Plaats een video-opname op jullie blog die het gebruik van jullie toepassing illustreert. Plaats daar ook de code.

5. <http://developer.android.com/sdk/android-2.0-highlights.html>

8. Besluit

Er is bij de implementatie van dit project gepoogd om alle features van de flex applicatie over te nemen en nog een aantal extra features toe te voegen. De werkdruk lag hierdoor misschien iets te hoog om binnen de beschikbare tijd een mooi afgewerkte implementatie te leveren.

Wat betreft de visualisatie zouden we zeker langer zoeken naar bestaande bibliotheken voor roamer. Zelf implementeren was omwille van de ruwe kantjes van de API veel werk met aanvaardbaar resultaat.

Wat betreft de persistentie blijkt dat opslag voor android niet zo triviaal is. Zo bleek het niet mogelijk om gewoon een connectie te openen rechtstreeks met een sql-database. Het alternatief, de middleware RESTful waar we voor gekozen hebben is echter zeer waardevol gebleken. Het belangrijkste nadeel, dat we niet goed konden inschatten is dat dit toch enige vertraging introduceert bij het toevoegen van gebruikers of parties en het synchroniseren met de server.

Appendix

	Bart Tuts	Jeroen Baert	Jan Grauwels	Laurens De Vocht	Totaal
Storyboard	1	1	1	1	4
Ontwerp	5	5	5	5	20
Implementatie visualisatie	27	0	0	0	27
implementatie GUI	0	0	20	2	22
implementatie Last.FM data retrieval	0	20	0	0	20
implementatie persistence and model integratie	0	3	0	30	33
	6	2	4	4	16
Verslag schrijven	8	5	4	3	20
Totaal	47	36	34	45	162
