New & Cool

In de twee vorige delen van deze JavaFX serie stonden verschillende aspecten van dit platform voor het ontwikkelen van interactieve desktopapplicaties centraal. Inmiddels heeft Sun Microsystems de nieuwe, indrukwekkende versie 1.2 van JavaFX uitgegeven met veel nieuwe functionaliteit die het platform een grote stap voorwaarts positioneert. Als afsluiting van deze serie over JavaFX daarom een overzicht van de mogelijkheden van het grafische FXD-bestand, de aspecten van dataopslag en communicatie die JavaFX biedt en de toekomstverwachting van deze technologie.

JavaFX: On the road...

Mogelijkheden van het grafische FXD-bestand

Ronald van Aken
is werkzaam bij Accenture Technology Solutions en specialiseert zich op het gebied van Java-, BPM- en SOA-oplossingen.

Tijdens JavaOne was JavaFX een belangrijk onderwerp. De topman van Oracle benadrukte zelf ook het belang van het platform en de technologie. De verbeteringen die in versie 1.2 zijn toegevoegd, zien er goed uit. Na een hobbelige start bereikt de technologie denk ik het niveau voor de industrie om deze echt serieus te nemen. Enkele belangrijke toevoegingen aan JavaFX in versie 1.2 zijn:

1. Nieuwe visuele controls. Het palet is uitgebreid met de meest gangbare widgets om interactieve applicaties mee te bouwen. Waar we voorheen eigenlijk, Swing buiten beschouwing gelaten, over maar één enkele control beschikten (TextBox). De controls zijn opgenomen in het 'common' profile, zodat ze op alle platformen kunnen worden gebruikt. De API die hiervoor wordt gebruikt, is volledig geënt op javafx.Scene en heeft geen afhankelijkheden met/van de Java AWT.
2. Gebruik van grafieken. Er is een specifiek stuk API toegevoegd om het gebruik van grafieken mogelijk te maken binnen applicaties.
3. Data persistentie. Binnen versie 1.2 is het mogelijk om op een platformonafhankelijke manier de data op te slaan. De applicatielogica heeft daardoor geen kennis nodig van het onderliggende platform (mobiel, desktop).
4. Snelheid van de applicaties. Voor versie 1.2 zijn diverse zaken verbeterd en aangepast in de compiler. Uiteindelijk leidt dit tot minder bytecode en een aanzienlijke verbetering in de performance van de applicatie. Dit zal een terugkerend aspect zijn voor de volgende releases.

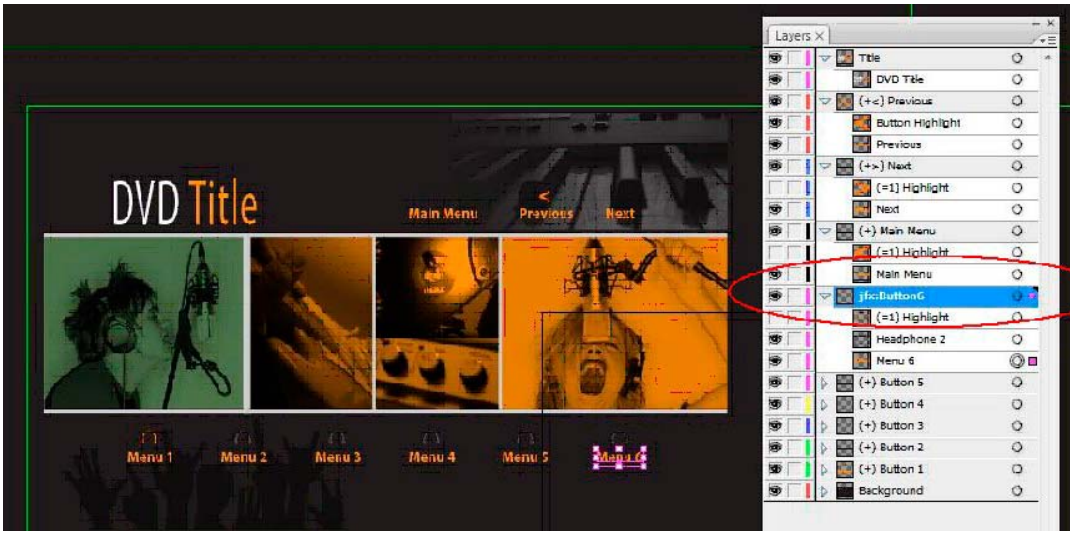
Daarnaast zijn kortere ontwikkelperioden aangekondigd voor nieuwe releases, zodat de industrie de nieuwe ontwikkelingen snel kan inzetten. Al met al mooie vooruitzichten waarmeeJavaFX een stap vooruit zet om als totaaloplossing te worden ingezet voor het realiseren van mooie grafische applicaties op het Java platform.

FXZ en FXD bestanden

Dat JavaFX visueel is georiënteerd, zal ondertussen duidelijk zijn. Het gebruik van multimediabestanden is makkelijk te integreren en de formaten die worden ondersteund, zijn onder andere AVI, JPEG, PNG en MPEG. Naast het afspelen en tonen van de multimediaformaten, is er ook nagedacht over de manier waarop multimedia geïntegreerd kan worden binnen JavaFX-applicaties. Hiervoor heeft Sun een nieuw grafisch formaat, genaamd FXZ, gespecificeerd. De reden voor het specificeren van een eigen bestandsformaat is dat hierdoor op het JavaFX-platform interactie mogelijk is tussen de elementen die zijn gedefinieerd in het grafische bestand en de JavaFX-code. Dit bestand is dus niet bedoeld als nieuw grafisch bestandsformaat, maar als een brug tussen grafische tools en het JavaFX-platform. Met het aangemaakte FXZ bestand kunnen we als JavaFX-ontwikkelaar de grafische objecten benaderen als reguliere JavaFX-objecten! Een FXD is een tekstueel (XML) bestand met daarin de ID's van alle elementen uit het grafische bestand en de benodigde informatie om deze weer te geven. Het FXD-bestand is onderdeel van een FXZ-bestand. Dit is een gecomprimeerd (ZIP) bestand met daarin het FXD-bestand en de grafische bestanden die onderdeel uitmaken van het grafisch ontwerp in de Adobe-applicatie.

Integratie van FXZ

Momenteel kunnen we FXZ-bestanden maken met behulp van de JavaFX Production Suite (voorheen Project Nile). Deze suite bevat een plugin voor Adobe

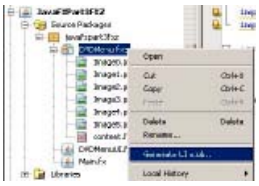


Figuur 1: Gebruik van Adobe illustrator voor FXZ bestanden.

Illustrator Creative Suite 3 (CS3) en Photoshop om FXZ bestanden aan te maken.

Met deze functionaliteit kunnen we een achtergrondplaatje, waarin delen van de applicatie zijn verwerkt, eenvoudig integreren met de JavaFX script logica. Door gebruik te maken van layers in het grafische bestand en elke layer een specifieke sleutel (ID) te geven, worden ze beschikbaar binnen de JavaFX omgeving als objecten.

In figuur 1 is te zien hoe layers worden gedefinieerd op basis van een grafisch plaatje in Adobe illustrator. Ik heb in dit voorbeeld één van de standaardvoorbeelden genomen die door Adobe worden meegeleverd. In Netbeans kan het aangemaakte FXZ bestand aan een JavaFX-project worden toegevoegd. Het enige wat nu nog gedaan moet worden, is een UI stub bestand aanmaken, dit bied vervolgens gemakkelijk toegang tot de elementen in het FXZ-bestand. Het stub bestand wordt aangemaakt door het FXZ-bestand te selecteren in Netbeans en de optie 'Generate UI stub' te selecteren.



Figuur 2: Aanmaken van UI stub in Netbeans voor het FXZ bestand.

Door het aanmaken van het Stub bestand worden de objecten die de prefix 'jfx:' hebben in het FXZ bestand via variabelen beschikbaar gesteld. Een voorbeeld van een Stub bestand:

```
override public var url = "{__DIR__}DVDMenu.fxz";

public-read protected var Button6: Node;
```

```
override protected function contentLoaded() : Void {
        Button6=getNode("Button6");
}

/**
* Check if some element with given id exists and write
* a warning if the element could not be found.
* The whole method can be removed if such warning is
 not required.*/
protected override function getObject( id:String) :
 Object {
    var obj = super.getObject(id);
    if ( obj == null) {
 System.err.println("WARNING: Element with id {id} not
found in {url}");
    }
    return obj;
}
```

Voorbeeld 1: Gegeneerde code van de Stub.

Het Stub-bestand bevat de variabele (Button6) voor de aangemaakte layer id in het Illustrator-bestand. Daarnaast is het aangemaakte Stub-bestand een afgeleide klasse van FXDNode. Hierdoor is het Stub bestand direct aan het Stage object toe te voegen en te activeren in de applicatie. Dit laatste is te zien in het volgende voorbeeld.

Door de functionaliteit van FXZ-bestanden ontstaat er een totaal nieuwe dimensie aan mogelijkheden om visuele aspecten gemakkelijk toe te passen in een JavaFX-applicatie. Door deze werkwijze met verschillende bestanden en gescheiden verantwoordelijkheden ontstaat de mogelijkheid om binnen projecten verschillende expertises parallel te laten werken; dit kan een groot voordeel zijn! Zo kunnen de grafische ontwerpers parallel aan de vormgeving werken, JavaFX-ontwikkelaars specifieke GUI logica realiseren en de Java-ontwikkelaars zich tegelijkertijd concentreren op de koppeling en communicatie tussen client-server software services. Dit was de oorspronkelijke belofte van JSP's , maar JSP's hebben het lang zonder professionele editor moeten doen. De integratie met de Adobe-software zal helpen dit doel te realiseren aangezien dit de

Er ontstaat een nieuwe dimensie om visuele aspecten toe te passen in een JavaFX-applicatie.

Je kunt specificeren hoeveel data de applicatie mag wegschrijven.

mainstream tooling is voor grafische ontwerpers. De toekomst zal het leren hoe deze strategie zal uitpakken voor het JavaFX-platform.

Gebruik van datapersistentie

Data is een essentieel element binnen applicaties en kan binnen JavaFX op een aantal manier worden geïntegreerd met de applicatielogica. Aangezien het zonder problemen mogelijk is om vanuit FX Javacode te kunnen aanroepen, kan er voor gekozen worden om de datalaag te implementeren in Java met het gebruik van de huidige frameworks die veel worden toegepast. Er kan echter ook voor worden gekozen om gebruik te maken van webservices en op die wijze de applicatie van de nodige datafunctionaliteit te voorzien. Binnen FX is als standaardmechanisme de klasse Storage beschikbaar gemaakt. Dit is een speciale klasse die zorgt voor de mogelijkheid om op een platformonafhankelijke manier (mobiel, desktop) gegevens op te slaan. De klasse kan worden gebruikt binnen elke omgeving om data lokaal op te slaan, maar kan ook worden gebruikt in applicaties die in zogenoemde onveilige omgevingen worden opgestart (browsers). De klasse is in versie 1.2 toegevoegd dus zorg voor een laatste versie van de SDK wanneer hiermee geëxperimenteerd wordt! Toevoegde waarde van het gebruik is dat ontwikkelaars zich niet druk hoeven maken over specifieke opslagkenmerken die kunnen gelden voor verschillende platformen. Hierdoor kan een ontwikkelaar op een uniforme wijze werken met dataopslag ongeacht het platform waar de applicatie op draait.
Een eenvoudig voorbeeld van hoe we klasse kunnen gebruiken:

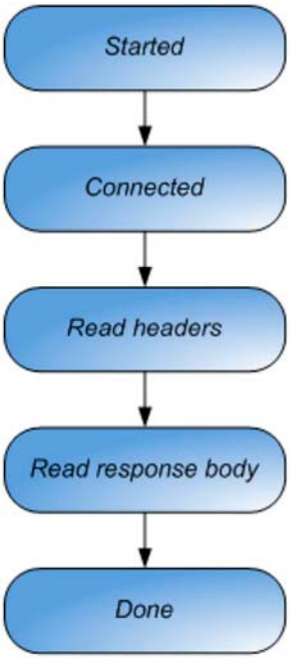
```
var storage = Storage { source: "testfile.txt" };
var res = storage.resource;
{
   var outputStream = res.openOutputStream(true);
   outputStream.write("test".getBytes());
   outputStream.close();
};
```

Voorbeeld 2: Gebruik van Storage klasse.

Het voorbeeld zal de tekst lokaal in <user home dir>\Sun\JavaFX\Deployment\storage\muffin opslaan. Dit is echter afhankelijk van het platform waarop gewerkt wordt . Aangezien dit de verantwoordelijkheid is voor de JVM, is het voor de ontwikkelaar niet van belang
Het is vanuit veiligheidsperspectief mogelijk om verdere beperkingen in te stellen. Zo is het bijvoorbeeld te specificeren hoeveel data weggeschreven mag worden door de applicatie. Vooral bij applicaties die gegevens wegschrijven naar een webserver is dit belangrijk.

XML en HTTP in JavaFX

Kijkend naar de mogelijkheden om gebruik te maken van webservices binnen JavaFX, zijn er twee keuzes. De eerste is om de webservice vanuit reguliere Java-code aan te roepen en deze code te gebruiken in de JavaFX-applicatie. De andere mogelijkheid is om gebruik te maken van de JavaFX API die zich vooral richt op het asynchroon versturen van http-berichten. Dit model sluit aan bij de architectuur van Web 2.0 applicaties die veel gebruik maken van AJAX (Asynchrounous Javascript And Xml)
Binnen het domein van Web 2.0 zijn RESTful webservices een graag geziene gast. Deze webservices leveren een architectuur voor webcommunicatie die volledig gebaseerd is op hoe het internet in de basis is opgezet: gevraagde informatie is gekoppeld aan een URL. RESTful web services worden volgens dit principe gedefinieerd. Voor deze webservices is binnen JavaFX een API beschikbaar gemaakt die het ons mogelijk maakt een HttpRequest te versturen. HTML-ontwikkelaars zullen dit object goed kennen, zeker wanneer AJAX-technologieën in een webapplicatie worden verwerkt. De variant die in Java FX beschikbaar is gemaakt, is hiermee vergelijkbaar en zorgt voor een eenvoudige manier om een verzoek via het http-protocol te versturen en vervolgens de reactie af te handelen.
Binnen JavaFX wordt het object HttpRequest gebruikt om asynchroon berichten te versturen. Het object vereist een locatie en een methode. Programmeurs die ervaring hebben met het coderen van AJAX javascript zullen veel overeenkomsten in dit object zien. Wanneer we een verzoek via dit object versturen, dan ondergaat het HttpRequest een aantal stadia. Deze zijn geïllustreerd in voorbeeld 3.



Voorbeeld 3: Status verloop HttpRequest.

Het uitwerken van een HttpRequest is relatief simpel en een voorbeeld hiervan is te zien in Voorbeeld 2.

```
import javafx.io.http.HttpRequest;
import java.lang.Exception;

public class ExampleWebRequest {

function sendMessage(){
         var httpRequestError: Boolean = false;

         var request: HttpRequest = HttpRequest {
              location: "http://eenURL/applicatie"
              method: HttpRequest.GET

              onDone: function() {
              }
         }
         request.start();
    }
}
```

Voorbeeld 4: Gebruik van HttpRequest object.

Het codevoorbeeld toont hoe de locatie en de methode worden gedefinieerd en vervolgens het verzoek wordt verzonden door middel van de start-operatie.
Het antwoord van de webservice wordt afgevangen door een event handler te definiëren voor de verschillende communicatiestadia. Hiermee krijgen we een goede controle over het verloop van de communicatie. In voorbeeld 3 wordt een event handler gedefinieerd.

```
function sendRequest(){
          var httpRequestError: Boolean = false;

         var request: HttpRequest = HttpRequest {
              location: "http://eenURL/applicatie"
              method: HttpRequest.GET

              //handles data results
              onInput: function(input: java.io.
InputStream) {
                   try {
                        var parser = EmployeeParser{};
                        employees = parser.parse(input);
                        if(parser.errorMessage.length() >
0) {
                    httpRequestError = true;
                        }
                   } finally {
                        input.close();
                   }
              }
```

Voorbeeld 5: Event handlers in HttpRequest.

Een belangrijk object om het antwoord te verwerken, is PullParser. Dit object lijkt erg op de functionaliteit van een SAX parser. In Voorbeeld 4 is te zien hoe een parse methode gedefinieerd wordt als implementatie voor de PullParser.

```
public function parse(input: InputStream): Employee[] {

         var employees: Employee[];
         var employee: Employee;


// Parse the input employee data (XML) and instantiate
Employee object
         def parser = PullParser {
              input: input
              onEvent: function(event: Event) {
                   //starting to process the flight
element
                   if (event.type == PullParser.START_
ELEMENT) {


 if(event.qname.name == "employee" and event.level == 1)
{
                             employee = Employee{ };
                        }
                   } //else if it is an end of the
element

// check what kind of element it is and store it
                   else
                   if (event.type == PullParser.END_
ELEMENT) {

// it is a container level element - employee, insert
it into the employees array

 if(event.qname.name == "employee" and event.level == 1)
{
                             // insert employee in
employees;
                        } //
                        else

if(event.qname.name == "name" and event.level == 2) {
                             employee.name = event.text;
                        }
                   }
              }
         }
         parser.parse();
         return employees;
    }
```

Voorbeeld 6: gebruik van PullParser.

Bovenstaand voorbeeld toont het gebruik van de PullParser om JSON of XML data te interpreteren. Het is een eenvoudig eventgedreven model dat veel lijkt op SAX, maar daarentegen veel eenvoudiger is. Door de events START_ELEMENT en END_ELEMENT af te vangen, kan de data stukje voor stukje worden verwerkt en omgezet naar bijvoorbeeld een objectmodel.

Tot slot

Het doel van deze serie over JavaFX was een goed beeld te geven over dit platform. Dat JavaFX niet iets is om op een maandagochtend uit de kast te trekken om het direct toe te passen, is daarmee hopelijk duidelijk geworden. De potenties van dit platform zijn groot en bij aanhoudende gedrevenheid zal het een waardevolle aanvulling vormen op de huidige beschikbare Rich Internet Application (RIA)-oplossingen. Naar mijn mening is JavaFX het GUI platform van de toekomst en daarmee een goede vervanger voor Swing applicaties. Een kritische noot daarbij is dat het platform langzaam op start is gekomen en zeker in vergelijking met de concurrentie van Adobe en Microsoft nog wel de nodige aandachtspunten kent. Denk hierbij aan integratie met andere tooling en eenvoudige ontwikkeling van user interfaces (zoals dat onder andere mogelijk is bij Adobe Flex). Toch zie ik overwegend de grote potentie van dit platform en voorzie ik dat JavaFX een prominente rol gaat innemen. Zowel als oplossing voor RIA's, als een volwaardig alternatief voor het ontwikkelen van Java user interfaces. «

JavaFX is het GUI platform van de toekomst en een goede vervanger voor Swing applicaties.