**Literatuurstudie masterproef: Codegenerator van SceneBuilder-FXML naar Haskell**

**Aanleiding en achtergrond**

Hulpmiddelen voor ICT-gerelateerde problemen op te lossen ook zijn de laatste decennia sterk geëvolueerd. Hedendaagse programma’s met een grafische user interface (GUI) kunnen op een gemakkelijke manier gerealiseerd worden door studenten door onder andere JavaFX in combinatie met *SceneBuilder* te gebruiken. Hoewel JavaFX een krachtige tool is om applicaties te ontwikkelen, kunnen andere programmeer-paradigma’s, programmeertalen en platformen voordelen bieden die later worden besproken.

Tegelijkertijd zijn de hulpmiddelen voor ICT-gerelateerde problemen op te lossen ook geëvolueerd. Hedendaagse programma’s met een grafische user interface (GUI) kunnen op een gemakkelijke manier gerealiseerd worden door studenten door onder andere JavaFX in combinatie met *SceneBuilder* te gebruiken. In de opleiding industriële wetenschappen aan UHasselt/KULeuven op de campus Diepenbeek wordt de aanpak om GUI’s te ontwikkelen met JavaFX en SceneBuilder al met succes toegepast in het eerste bachelorjaar. Dit geeft duidelijk aan dat deze techniek gemakkelijk toepasbaar is voor beginners, waaronder de eerstejaarsstudenten van de opleiding.

**JavaFX en SceneBuilder**

JavaFX is een softwareplatform om (desktop)applicaties met een GUI te kunnen ontwikkelen die ondersteund worden door verschillende besturingssystemen waaronder Windows, Linux, MacOS, iOS en Android [1].

Een GUI wordt in JavaFX gedefinieerd als een boomstructuur van JavaFX *nodes* die de GUI vertegenwoordigen. Een *node* kan in de informatica beschreven worden als een onafhankelijke eenheid van een groter geheel. In de context van JavaFX, worden *nodes* gezien als componenten van *scene* (visueel component van de GUI) [2]. De JavaFX *nodes* kunnen gedefinieerd worden in een XML gemarkeerd tekstbestand van het type FXML. FXML-bestanden kunnen gemaakt en onderhouden worden met een GUI-tool, genaamd *SceneBuilder* waarbij studenten eenvoudig hun GUI-design kunnen beschrijven waarbij automatisch een FXML gegenereerd wordt die gebruikt kan worden om een JavaFX applicatie te ontwikkelen [3].

JavaFX, gebruikt de programmeertaal Java die op zich niet altijd de meest efficiënte taal is qua resources en energieconsumptie [3], [4]. Daarom is het interessant om een alternatief platform en/of programmeertaal te zoeken die de met-gebruiksgemak gegenereerde FXML van *SceneBuilder* kan omzetten naar een GUI-applicatie. Een kandidaat-programmeerparadigma om dit te realiseren is het functioneel programmeren, en meer bepaald de programmeertaal Haskell. Dit is de premisse van de onderzoeksgroep FunTTop waarbinnen deze masterproef zich situeert. FunTTop heeft immers als doelstelling om het gebruik van functioneel programmeren in het werkveld te verhogen en van Haskell, de academische functionele taal, in het bijzonder.

FunTTop staat niet alleen in deze visie. Verschillende grote bedrijven maken al gebruik van Haskell omdat het toestaat om code sneller en correcter te ontwikkelen, omdat meer programma-eigenschappen verifieerbaar zijn, maar ook omwille van de kracht waarmee programma’s in Haskell parallel uitgevoerd kunnen worden. Dit kan ertoe leiden dat bedrijven die zich zorgen maken over energieverbruik, zoals Facebook, functionele talen zoals Haskell gaan gebruiken voor efficiënte parallelle gegevenstoegang op hun servers [4], [5]. Ondertussen hebben reguliere programmeertalen zoals Java en C# functionele programmeerfuncties zoals lambda’s overgenomen omdat functionele talen talrijke voordelen bieden die later besproken zullen worden [6].

**Waarom Haskell over Imperatieve talen**

Haskell is een pure functionele programmeertaal waarbij het “functionele” betekent dat de bouwblokken van programma’s functies zijn. Een functie kan als argument aan een andere functie worden doorgegeven, als resultaat worden geretourneerd of worden toegewezen aan een variabele. Dit vermogen om functies als gegevens te behandelen, zorgt voor een hoger abstractieniveau en dus meer mogelijkheden voor hergebruik. Het “pure” aspect van Haskell is dat er geen bijwerkingen zijn wanneer een functie geëvalueerd wordt. De uitkomst van een functie hangt alleen af van de parameters en iedere evaluatie van die functie zal hetzelfde resultaat geven [4], [7].

Dat Haskell functioneel en puur is, is een eerste significant voordeel vergeleken met traditionele imperatieve programmeertalen. Bij Imperatieve programmeertalen worden programma’s beschreven als een sequentie van instructies die data muteren. Het muteren van data tussen de sequenties en functies van een imperatieve taal zorgt er soms voor dat bepaalde functies niet altijd hetzelfde resultaat teruggeven en daardoor onverwacht gedrag tonen [4].

Haskell is ook een *lazy* programmeertaal waarbij de “lazyness” refereert naar het niet strikt evalueren, ook wel *call-by-need* genoemd. Een uitdrukking wordt nooit geëvalueerd totdat het nodig is voor de evaluatie van een volgende uitdrukking. *Lazyness* zorgt voor een minimale hoeveelheid berekening die wordt uitgevoerd tijdens de uitvoering van het programma wat voor prestatie een pluspunt kan zijn in vergelijking met imperatieve programmeertalen [4], [7].

Recursie is de norm van programmeerstijl voor Haskell. Recursie vervangt de reguliere iteratieve lussen die toestandsmutaties van variabelen vereisen en zorgt ervoor dat complexe code aanzienlijk korter kan worden beschreven [4] Gecombineerd met het feit dat Haskell functies kan gebruiken zonder bijwerkingen zorgt ervoor dat een robuust programma gerealiseerd kan worden met weinig lijnen code vergeleken met imperatieve programmeertalen. Zo concludeerde de studie van [8] dat functionele talen zoals F# en Haskell in het algemeen beduidend beknopter waren. Haskell had volgens de studie van [8] gemiddelde twee keer minder lijnen nodig voor dezelfde taken van *Rosetta Code* en tot drie keer minder lijnen code dan C.

**Haskell is niet perfect**

Hoewel Haskell een robuuste foutongevoelige taal is, blinkt het toch niet op alle vlakken. Ten eerste zijn Haskell en functionele talen in het algemeen minder geheugenefficiënt vergeleken met talen zoals C en C++ [8]. Bovendien is Haskell niet altijd evident om aan te leren aangezien volgens [9] en [10] het concept van recursie en mathematische vaardigheid vereist is om Haskell op een vlotte manier te begrijpen. Daarnaast zijn de compiler *warnings* en errors niet altijd even duidelijk in Haskell [8]. Ten slot valt Haskell achter op het gebied van GUI bibliotheken aangezien vele bibliotheken al afgeschreven zijn en GUI’s ontwikkelen voor andere programmeertalen populairder is.

**De wereld van GUI’s in Haskell**

In de loop der jaren zijn er talloze GUI bibliotheken ontwikkeld voor Haskell, met een breed scala aan verschillende interfaces. Zo kent Haskell *high level* (functioneel en declaratief) en *low level* GUI bibliotheken [11]. Medium level bibliotheken kunnen eerder worden gezien als bibliotheken die functioneel zijn, maar toch een declaratieve smaak bezitten.

*Low level* bibliotheken en programmeertalen worden volgens Alan Perlis beschreven als “een programmeertaal waarbij de programma’s aandacht nodig hebben naar het irrelevante” [11, p. 42], [12]. Het irrelevante wordt volgens [11] beschreven als alle implementatiedetails die zich verschuilen achter het abstracte conceptueel model. In de realiteit van de wereld van Haskell GUI’s zijn dit de bibliotheken die als *wrapper* toegepast zijn rond een bibliotheek die in een imperatieve programmeertalen zoals C/C++ is geschreven [13]. In de context van programmeertalen en bibliotheken bevatten deze dunne *wrapper*s code die intern *application programming interface* (API) functies kunnen oproepen zonder de originele API van de onderliggende implementatie aan te passen. Enkele voorbeelden van deze bibliotheken zijn WxHaskell en GTK2Hs waarbij er op een eenvoudige, declaratieve manier de API-functies in Haskell kunnen worden opgeroepen van de van de bibliotheek die bijvoorbeeld in C++ is geschreven (WxWidgets en GTK+).

*High level* bibliotheken zorgen voor een duidelijke scheiding tussen de implementatie van de applicatie en de interface [14]. In de wereld van Haskell GUI’s zijn deze type bibliotheken gebaseerd op het *Functional Reactive Programming* (FRP) paradigma. Deze type bibliotheken staan eerder verder weg van het concept van typische *wrappers* die bij *lower* level bibliotheken gebruikt worden en staan meer bekend om de *reactive*, *event*-gedreven eigenschappen op een functionele manier te implementeren. Het *reactive* en *event*-gedreven duidt in deze context aan dat het programma alleen specifieke code uitvoert die wordt bepaald door de sequentie van events (muisklik, venster vergroten, …) [15].

In de volgende paragraaf zullen enkele populaire *low*, *medium* en *high*-level kandidaat bibliotheken besproken worden met hun voor en nadelen. Vervolgens zullen de belangrijkste kandidaat-bibliotheken verder in detail getest en bestudeerd worden om uiteindelijk een keuze te kunnen maken voor de bibliotheek die gebruikt zal worden voor de codegenerator.

**Enkele populaire bestaande Haskell GUI bibliotheken.**

1. **WxHaskell**

WxHaskell is een GUI-bibliotheek voor Haskell waarbij de bibliotheek zorgt voor de nodige *bindings* van Haskell naar WxWidgets, een cross-platform GUI bibliotheek geschreven in C++. *Bindings* kunnen in deze context beschreven worden als een verbinding tussen een programmeertaal en softwarebibliotheek. Ongeveer 75% van WxWidgets functionaliteit is overgenomen in WxHaskell. Volgens [10] worden WxHaskell GUI’s gedefinieerd met een imperatieve IO monad, die een “declaratieve smaak” hebben waardoor er toch een hoog niveau van abstractie wordt bereikt.

Een eerste voordeel van WxHaskell is dat de bibliotheek gebaseerd is op WxWidgets. WxWidgets wordt ondersteund door grote industriële spelers zoals *AVG*, *AMD*, *Lockheed* en *NASA* waardoor de bibliotheek onderhouden kan worden indien er nieuwe platform-specifieke features worden uitgebracht [10], [16]. Een tweede voordeel van WxWidgets, en dus WxHaskell is dat applicaties gemakkelijk omgezet kunnen worden naar verschillende besturingssystemen zonder de applicatie helemaal opnieuw te coderen aangezien het een cross-platform bibliotheek is [10]. Verder hebben WxWidgets applicaties een *native look and feel* wat betekent dat WxWidgets de widgets van het besturingssysteem gaat gebruiken waar het de applicatie op draait [10], [16]. Ten slotte kan WxHaskell met relatief weinig lijnen code geschreven worden en biedt het goede prestaties, korte compileertijden en weinig geheugenlekken [10], [17], [18].

WxHaskell kent ook nadelen. WxHaskell is in contrast tot WxWidgets minder onderhouden waardoor niet alle functionaliteit van WxWidgets *up to date* is. Zoals eerder vermeld wordt ook niet de volledige WxWidgets-functionaliteit ondersteund. Hoewel de compileertijd klein is, zorgt WxHaskell soms wel voor grote uitvoerbare bestanden na het compileren [10].

1. **FLTKHS**

FLTKHS is gebaseerd op FLTK, een *mature* cross platform C++ bibliotheek. FLTKHS is eerder low level, maar laat toe om *native* GUI’s in pure Haskell te maken [18].

FLTKHS kent als eerste voordeel dat er weinig *dependencies* (afhankelijkheden) zijn. Het is ook een van de gemakkelijkste bibliotheken om te installeren op Windows. Bovendien is *Fluid*, een GUI generator die oorspronkelijk bedoeld was voor FLTK-code te genereren sinds kort beschikbaar voor FLTKHS [18]. De Haskell en C++ *callbacks* lijken volgens [18] ook sterk op elkaar waardoor ontwikkeling vlotter kan verlopen.

Echter kent FLTKHS ook nadelen. Zo zijn volgens [18] de compileer- en link-tijden lang en niet optimaal. Hoewel FLTKHS als low level wordt beschreven is volgens [18] FLTKHS niet altijd even efficiënt. Er is ook geen duidelijke documentatie over FLTKHS beschikbaar, maar het goede nieuws is dat de bestaande documentatie van FLTK gebruikt kan worden om een oplossing in Haskell om te zetten.

1. **FranTK**

FranTK wordt volgens [14] beschreven als een bibliotheek om op een declaratieve (high-level) manier GUI’s te schrijven.

FranTK kent net als in JavaFX ook het principe van een *Listener* [14], [19]. Listeners zijn in FranTK krachtiger dan in Java omdat in FranTK Listeners geparametriseerd kunnen worden over elk type [14]. Er is dus geen dubbele code nodig om verschillende *Mouse*- en *ActionEvent*- *Listeners* te implementeren zoals in Java. Daarnaast wordt FranTK geïmplementeerd in een toolkit-onafhankelijke manier zodat het gemakkelijk te porten is op verschillende platformen en besturingssystemen [14].

Aangezien FranTK eerder high level is, is er geen mogelijkheid voor low level controle waardoor efficiency niet gegarandeerd kan worden. Bovendien is niet alle functionaliteit van GUI’s beschikbaar wat de creativiteits-mogelijkheden beperkt [14].

1. **GTK2Hs/gtk3**

GTK2Hs is een GUI-bibliotheek voor Haskell waarbij de bibliotheek zorgt voor de nodige Haskell *bindings* naar GTK+. GTK+ is een multi-platform toolkit om GUI’s te ontwikkelen, met een eerder medium tot low level interface [20], [21].

GTK2Hs heeft net als WxHaskell het grote voordeel dat het platform-onafhankelijk ontwikkeld kan worden. Echter is GTK2Hs op het gebied van actieve ontwikkeling en interfaces een groter voordeel vergeleken met WxHaskell.

Ten eerste wordt nog actiever gewerkt aan GTK2Hs dan WxHaskell. Ten tweede heeft GTK2Hs volgens experimenten van [22] een meer extensieve interface wat voor rijkere interface elementen zorgt vergeleken met WxHaskell. Bovendien heeft GTK+, en dus ook GTK2Hs een API beschikbaar met een duidelijke documentatie. Hoewel de API voor GTK+ is gemaakt, zou het voor GTK2Hs gebruikt kunnen worden aangezien GTK2Hs een *mapping* is van GTK+ naar Haskell [23]. Daarnaast biedt GTK2Hs het voordeel dat met *Glade*, een tool om visueel een GUI te maken waarbij Haskell code wordt gegenereerd, eenvoudig een GUI kan worden gemaakt [20]. Ten slotte zijn er inmiddels Haskell bindings ontwikkeld voor de laatste versie van GTK+ (GTK+3), genaamd gtk3.

Toch is GTK2Hs niet perfect. Het afhandelen en het werken met threads is vooral beperkt. Multithreading is ook niet ondersteund door GTK+, en dus ook niet door GTK2Hs. Haskell threads zouden standaard niet parallel uitgevoerd kunnen worden naast de GTK+ threads [22]. Hoewel er volgens [22] de oplossing bestaat om de threads van de GTK+ hoofd-thread te pauzeren om de Haskell threads een kans te geven, resulteert de oplossing in threads die zeer traag worden uitgevoerd. Gelukkig bestaat er een handleiding van [24] die snelle oplossingen biedt voor multithreading in GTK2Hs.

1. **gi-gtk**

gi-gtk is gelijkaardig aan GTK2Hs omdat het voor de *bindings* van Haskell naar GTK+ zorgt. Echter biedt gi-gtk enkele voordelen vergeleken met GTK2Hs. Ten eerste zijn de *bindings* van gi-gtk auto-gegenereerd in plaats van met de hand geschreven in GTK2Hs. Dit betekent dat de functionaliteit van GTK+ meer compleet zal zijn aangezien er meer *bindings* zijn voorzien vergeleken met GTK2Hs [25], [26]. Ten tweede werkt gi-gtk met out of the box met GTK+3, vergeleken met GTK2Hs die GTK+2 ondersteund (tenzij expliciet de gtk3-bindings worden gebruikt). Ten slotte biedt gi-gtk ook een declaratieve versie van *bindings* aan (gi-gtk-declarative), die declaratief programmeren in GTK+ mogelijk maakt.

1. **Fudgets**

Fudgets is een hoog level declaratieve GUI bibliotheek voor Haskell waarbij widgets als “fudgets” worden beschreven. Een *fudget* is een “functioneel equivalent van een widget, die in contrast tot andere GUI bibliotheken niet object-georiënteerd is” [15, p. 46]. Een fudget is tegelijktijdig ook een proces die kan communiceren met andere fudgets en met de buitenwereld [27].

Volgens [27] is uitbreiden van een programma evident aangezien *fudgets* op een hiërarchische manier gemakkelijk zijn uit te breiden: de bouwblokken zijn *fudgets* en het gehele programma is ook een *fudget*. De abstractie op een hoog level niveau van de fudgets als bouwblokken zorgt voor aparte, maar toch logischerwijze concept over een GUI.

1. **Qtah**

Qtah is een medium-level GUI bibliotheek voor Haskell die gebaseerd is op Qt 4/5. Op dit moment gebruikt Qtah veel classes van QtCore, QtGui, en QtWidgets [28]. Het is een jonge bibliotheek dat aan het groeien is. Daarnaast is het ook een bibliotheek dat makkelijk uit te breiden is door het toevoegen van functies of methodes in enkele regels code.

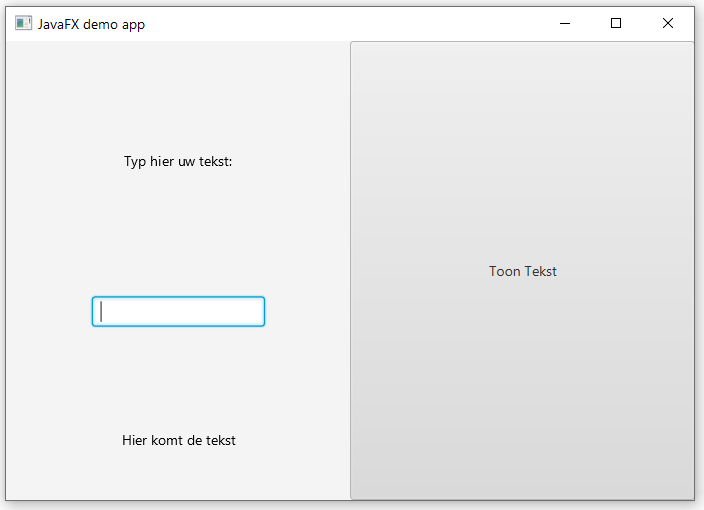
Qtah maakt gebruik van Hoppy om C++ code te binden. Hoppy is een foreign function interface (FFI) generator voor toegang tot C++ bibliotheken vanuit Haskell [29].

**Vergelijking verschillende bibliotheken**

Tabel 1: vergelijking van de verschillende Haskell GUI-bibliotheken

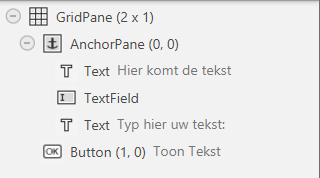
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Stijl** | **Multi platform** | **Onderhouden** | **Voordelen** | **Nadelen** |
| **WxHaskell** | Imperatief, met toch een declaratieve smaak (Medium-low-level) | Ja | Laatste grote release in 2017 | +Widgets hebben native-look en feel  +prestatie  +korte compileertijd  +gebaseerd op WxWidgets | -Grote binaries na compileren  -Niet 100% functionaliteit WxWidgets  -moeilijk te installeren |
| **FLTKHS** | (Medium)-low-level | Ja | Ja | +Haskell en C++ callbacks lijken op elkaar  +Custom widgets mogelijk door C++ methodes te *overriden*  +Gemakkelijk te installeren op Windows  +Duidelijke documentatie | -veel baseren op FLTK  -veel low-level implementatie  -lange compileertijden  -widgets hebben geen *native look* |
| **FranTK** | Declaratief, high-level | Alleen Linux en Windows | Nee | +Listeners geparametriseerd  +High level | -Efficiency en prestatie niet gegarandeerd door high-level |
| **GTK2Hs/gtk3** | Medium-low-level | Ja | Ja | +Actieve development  +Rijkere interface elementen vergeleken met WxHaskell  +grootste deel functionaliteit GTK+  +mogelijkheid voor laatste versie GTK+3 te gebruiken met gtk3  +Interessante tutorials te vinden | -Geen officiële support voor Multithreading (omwegen bestaan)  -Niet alle bindings voor alle GTK+ widgets |
| **gi-gtk** | Medium-low-level | Ja | Ja | +meer compleet, meer *bindings* voorzien dan GTK2hs  +mogelijkheid om declaratief te werken met gi-gtk-declarative  +veel tutorials en voorbeelden te vinden  +meer mogelijkheden om met threads te werken | -Weinig tutorials over te vinden |
| **Fudgets** | High-level | Alleen Unix-like systemen. Werkt het beste op GNU/Linux. | Laatste major release in 2000. | +Duidelijke documentatie  +Gemakkelijk hiërarchisch op te bouwen met fudgets | -Weinig voorbeelden en tutorials te vinden |
| **Qtah** | Medium-low level | Ja | Ja | +Gebaseerd op Qt  +Actieve development  +Veel uitgebreide functionaliteit Qt mogelijk | -Geen duidelijke en uitgebreide tutorials/demo’s, veel baseren op Qt  -Relatief jong  -Niet de volledige Qt bindings |

**Demo applicatie**

Om de kandidaat bibliotheken in detail te bestuderen zal er een demo-GUI applicatie nagemaakt worden in iedere bibliotheek. Vervolgens wordt de implementatie per bibliotheek besproken. Ten slotte worden op basis van de voor en nadelen de uiteindelijke bibliotheek gekozen die gebruikt zal worden voor de codegenerator.

Figuur 1: gewenste demo-applicatie in JavaFX

Figuur 1 toont de applicatie die uiteindelijk gemaakt zal worden. De bedoeling van de demo-applicatie is dat de gebruiker iets kan intypen en de ingetypte tekst van de gebruiker getoond zal worden na het drukken van de knop.

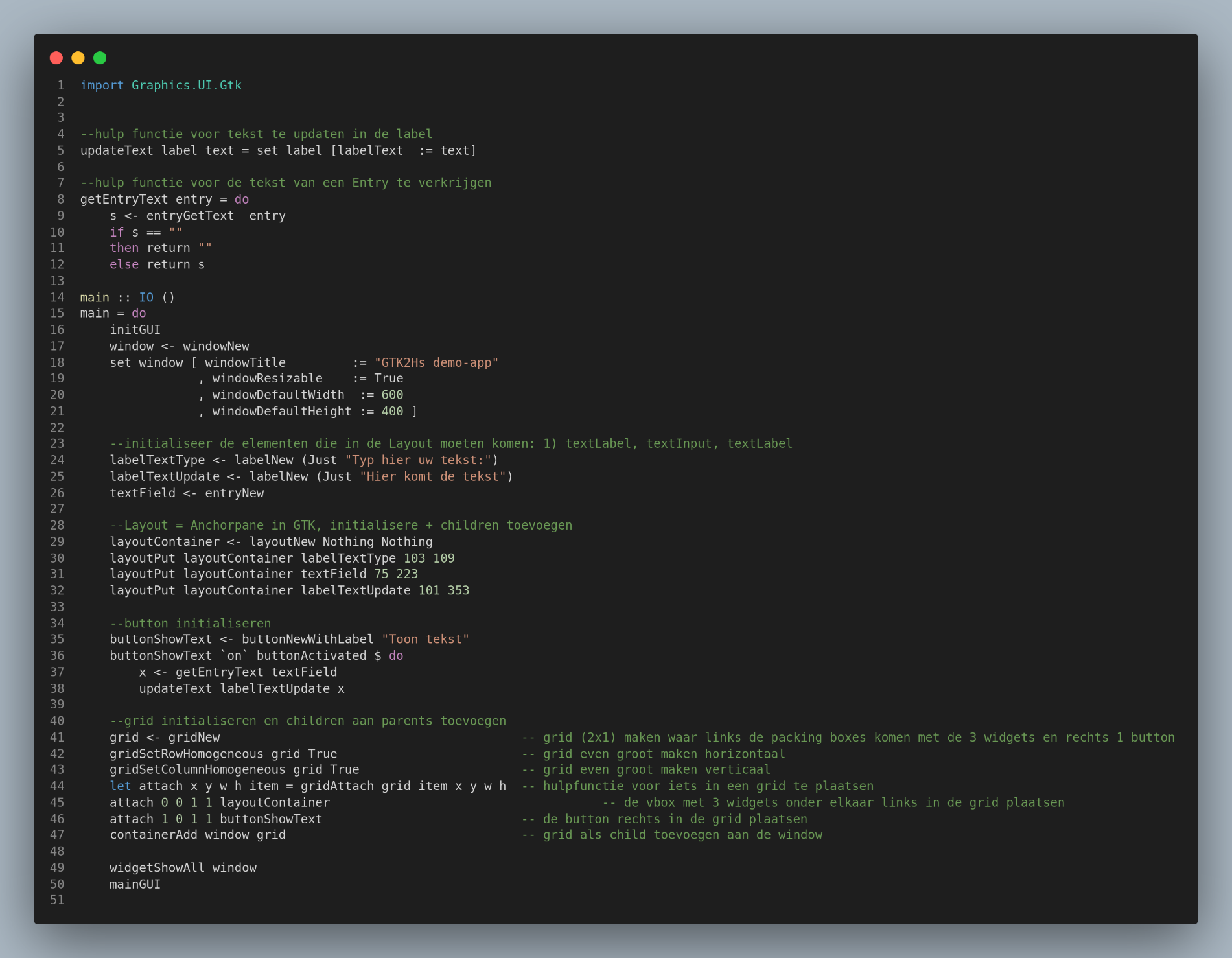
**Implementatie SceneBuilder voor JavaFX**

Figuur 2: hiërarchie van widgets voor JavaFX in SceneBuilder

Figuur 2 toont de hiërarchie van de verschillende widgets en grafische elementen die gebruikt worden in het demo-programma. Als top widget is er een *GridPane* van dimensie 2x1. De *GridPane* kan gezien worden als een tabel van cellen waarin andere widgets kunnen worden toegevoegd. In de rechter cel bevindt er zich de Button-widget die gebruikt wordt voor de tekst te tonen. In de linker cel bevindt er zich een *Anchorpane.* De *Anchorpane* is een grafisch element die het toelaat de elementen die zich in de *Anchorpane* bevinden, vrij rond te plaatsen in het gebied van de *Anchorpane* [30]. In de *Anchorpane* bevinden zich een het *TextField* element waarin de gebruiker tekst kan typen, en twee *Text*-labels.

**Implementatie van de demo-applicatie in de verschillende kandidaat GUI-bibliotheken**

1. **GTK2Hs**



1

2

3

Figuur 3: code van de implementatie demo-applicatie in GTK2Hs

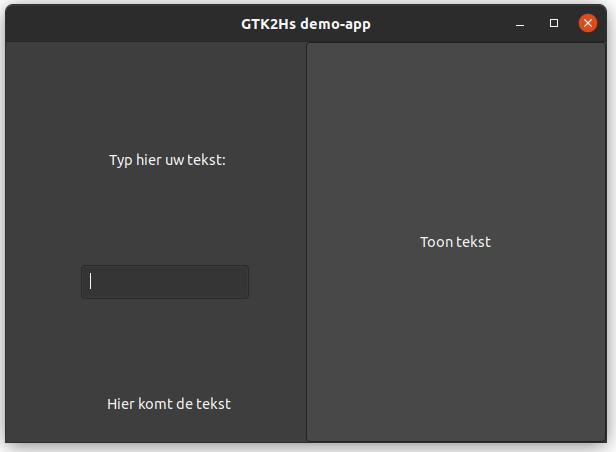
Figuur 3 toont de implementatie van de demo-applicatie in GTK2Hs, gebruikmakend van de gtk3 bindings. Bij de implementatie van de demo-applicatie in GTK2Hs vallen er enkele zaken op. Ten eerste heeft deze bibliotheek een medium-level stijl waarbij bepaalde elementen imperatief en declaratief worden afgehandeld. Zo wordt bijvoorbeeld het verkrijgen van de tekst uit de het tekstvak op een imperatieve manier afgehandeld m.b.v. de *callback* hulpfunctie *getEntryText.* Ook is het is het mogelijk om meerdere attributen van een bepaalde widget met de *set*-functie in te stellen zoals wordt gedaan voor de attributen van het venster (*window*). Ten slotte heeft een programma in GTK2Hs een vaste structuur:

1. *initGUI* functie aanroepen samen met een toplevel element (*window*) die de GUI initialiseert.
2. Widgets initialiseren en de *callbacks* definiëren
3. Afsluiten met de *widgetShowAll* functie die voor de benodigde allocatie zorgt samen met de mainGUI loop

|  |  |
| --- | --- |
| **JavaFX** | **GTK2Hs** |
| GridPane | Grid |
| Anchorpane | Layout |
| Text | Label |
| TextField | Entry |
| Button | Button |

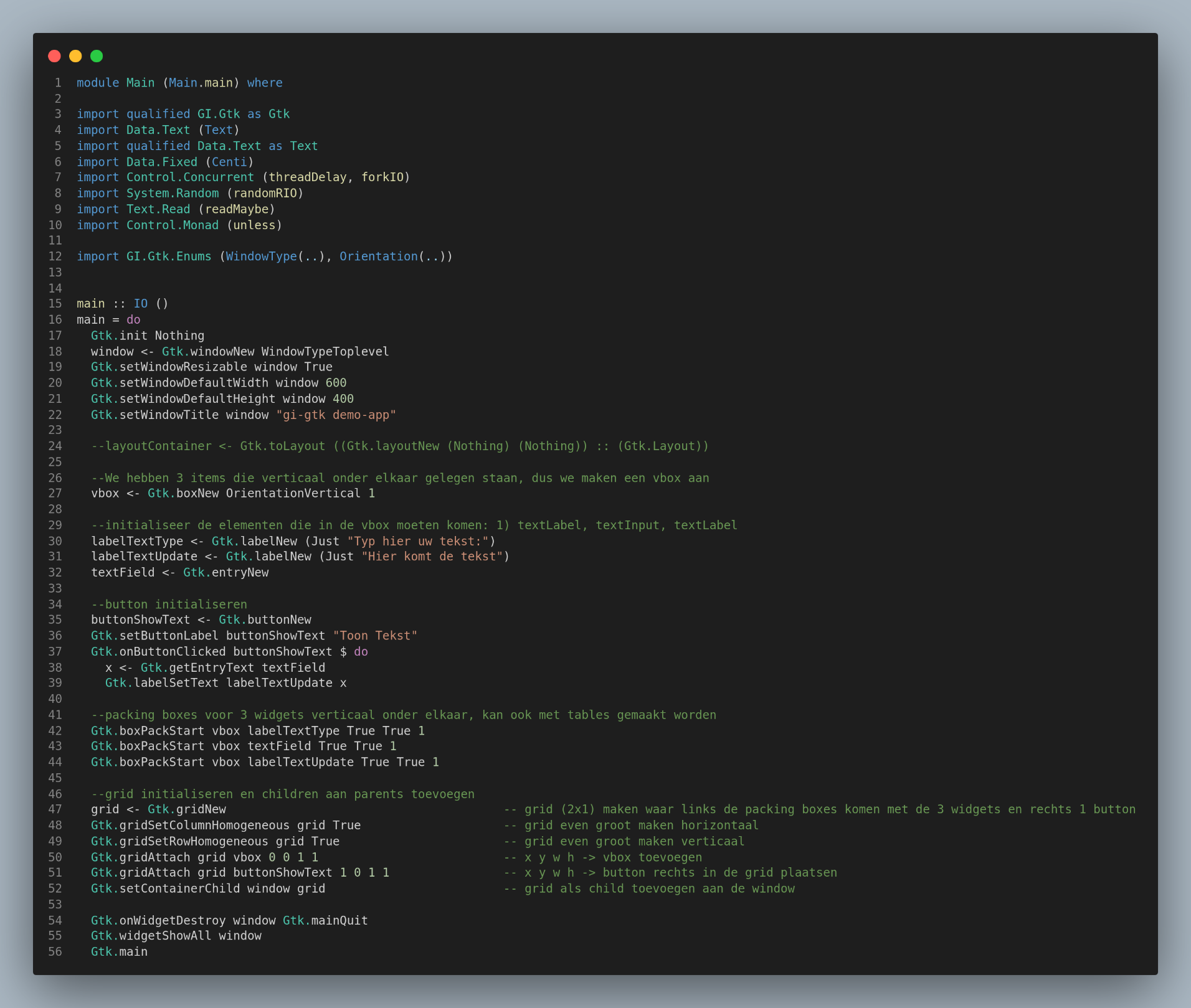
Tabel 2: tegenhanger van de JavaFX widgets in GTK2Hs

Tabel 2 toont de tegenhanger van iedere JavaFx widget in GTK2Hs. De GTK2Hs bibliotheek voorziet een directe tegenhanger voor iedere JavaFX widget die in het voorbeeldprogramma is gedefinieerd.

De GTK2Hs bibliotheek is op eerste zicht compleet en gemakkelijk mee te werken. Zo is ten eerste iedere JavaFX component die nodig is voor de voorbeeldapplicatie na te maken in GTK2Hs. Ten tweede kunnen de *callback* functies direct in het *main*-block gedefinieerd worden waarbij het aanroepen naar andere widgets in de *callback* functie mogelijk is. Zo is het mogelijk om de *callback* functie van de *Button* direct in het *main-*block te definiëren waarbij er op een eenvoudige manier de *textLabel* geüpdatet kan worden. Daarnaast zorgde de *set* functie ervoor dat de attributen van een bepaalde widget in één enkele lijn geïmplementeerd kunnen worden. Ten slotte heeft GTK2Hs het voordeel dat alle widgets een native look hebben, zoals is te zien in figuur 4.

Figuur 4: uiteindelijke demo-applicatie in GTK2Hs

1. **gi-gtk**



1

2

3

Figuur 5: code van de implementatie demo-applicatie in gi-gtk

Figuur 5 toont de implementatie van de demo-applicatie in gi-gtk. Ten eerste valt ook op dat de hulpfunctie *getEntryText* die gebruikt wordt in GTK2Hs, één directe functie geïmplementeerd kan worden. Daarnaast valt op dat er geen aparte *set*-functie is om parameters van een widget in te stellen. Alle eigenschappen van widgets, zoals de *window* worden lijn per lijn geïnitialiseerd. Ten slotte komt de structuur van het programma grotendeels overeen bestaande uit dezelfde drie stappen bij de implementatie van GTK2Hs:

1. *Gtk.init* functie aanroepen samen met een toplevel element (*window*) die de GUI initialiseert.
2. Widgets initialiseren en de *callbacks* definiëren
3. Afsluiten met de *widgetShowAll* functie die voor de benodigde allocatie zorgt samen met de *Gtk.main* loop

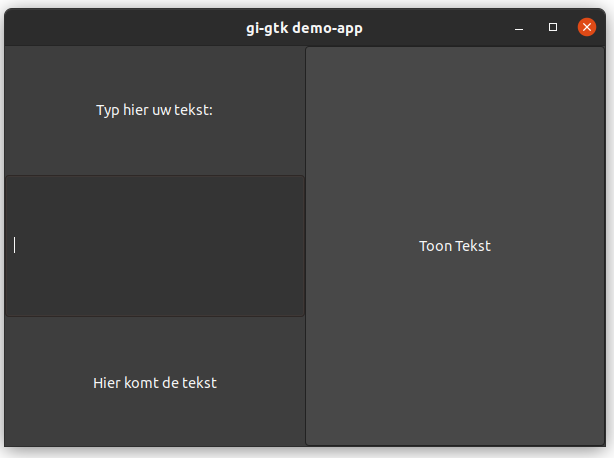
|  |  |
| --- | --- |
|  |  |

Tabel 3: tegenhanger van de JavaFX widgets in gi-gtk

|  |  |
| --- | --- |
| **JavaFX** | **gi-gtk** |
| GridPane | Grid |
| Anchorpane | Layout |
| Text | Label |
| TextField | Entry |
| Button | Button |

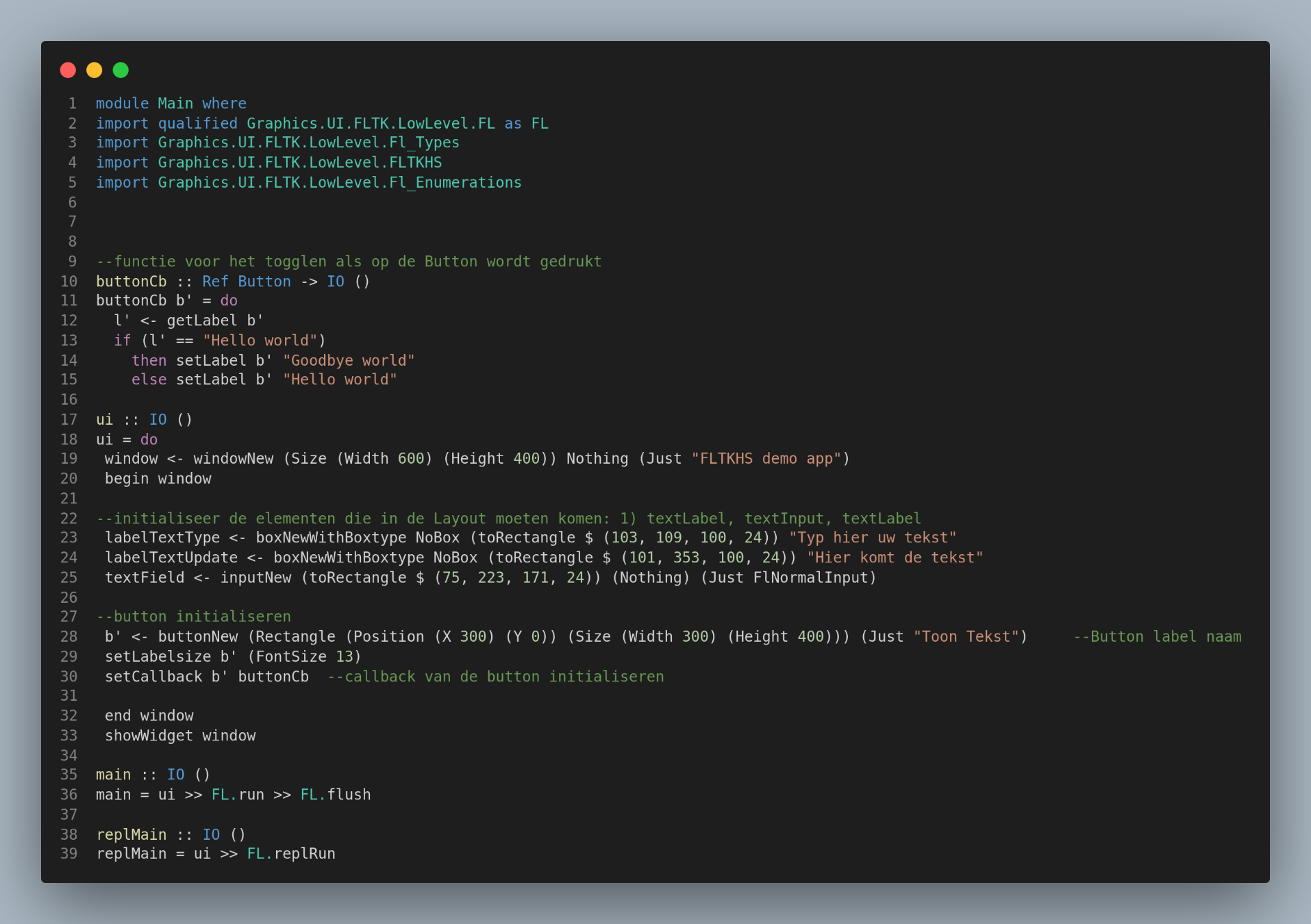
Tabel 3 toont de tegenhanger van iedere JavaFx widget in gi-gtk. Het opvallende is dat de implementatie van een *Layout* (*AnchorPane* in JavaFX) vervangen is door een *box*. Dit werd gedaan omdat het programma niet compileerde met de *Layout* constructor van gi-gtk wat ervoor zorgt dat de widgets niet meer vrij geplaatst kunnen worden.

De gi-gtk kent grotendeels dezelfde voordelen als de GTK2Hs bibliotheek zoals de native-look van widgets, vaste structuur in de code en veel directe tegenhangers van JavaFX widgets. De gi-gtk bibliotheek heeft het voordeel boven de GTK2Hs bibliotheek, dat er meer directe bindings voorzien zijn dan GTK2Hs. Bijna iedere functie en widget in GTK+ is terug te vinden in gi-gtk. Toch werkte niet alle bindings naar behoren aangezien het aanmaken van een Layout niet mogelijk is. Dit is een groot nadeel aangezien de Layout-widget, een directe cruciale tegenhanger van de JavaFX Anchorpane, niet direct bruikbaar is. Ten slotte heeft de gi-gtk bibliotheek het nadeel dat de attributen van de widgets aparte geïnitialiseerd moet worden wat in meer lijnen code resulteert.



Figuur 6: uiteindelijke demo-applicatie in gi-gtk

1. **FLTKHS**



1

2

3

4

Figuur 7: code van de implementatie demo-applicatie in FLTKHS

Figuur 7 toont de implementatie van de demo-applicatie in FLTKHS. Ten eerste is FLTKHS-bibliotheek meer low level vergeleken met de andere bibliotheken. Zo bestaan de widgets uit meer primitieve types die een vorm toegekend krijgen met de bijhorende parameters zoals lengte, breedte en coördinaten. Hierdoor wordt de hiërarchie ook moeilijk te definiëren wat ertoe heeft geleid dat widgets op een manuele manier werden geplaatst met coördinaten. Bovendien kunnen *callback* functies moeilijk in het hoofd-block (*ui*-block) gedefinieerd worden. Daarnaast zijn de *callback* functies zodanig gemaakt dat ze alleen de widget zelf kunnen aanpassen die de *callback* functie toegewezen heeft gekregen. Daarom is het in de implementatie niet mogelijk om direct vanuit de *callback* functie van de *Button* de tekst elders aan te passen. Ten slotte heeft het een programma bij FLTKHS de volgende structuur:

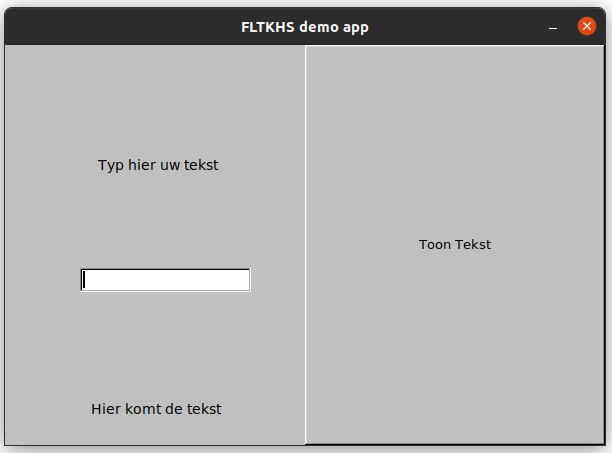
1. Definities van de *callback* functies
2. Ui-block waar alle widgets en windows worden gedefinieerd
3. Het ui-block samen met de *FL.flush* functie die ervoor zorgt dat het display up-to-date blijft en ervoor zorgt dat alle I/O buffers worden ververst [31]
4. Het ui-block samen met de *FL.ReplRun* functie die voor de uiteindelijke GUI zorgt [32]

Tabel 4: tegenhanger van de JavaFX widgets in FLTKHS

|  |  |
| --- | --- |
| **JavaFX** | **fltkhs** |
| GridPane | / |
| Anchorpane | / |
| Text | Box (zonder tekst) |
| TextField | Input |
| Button | Button |

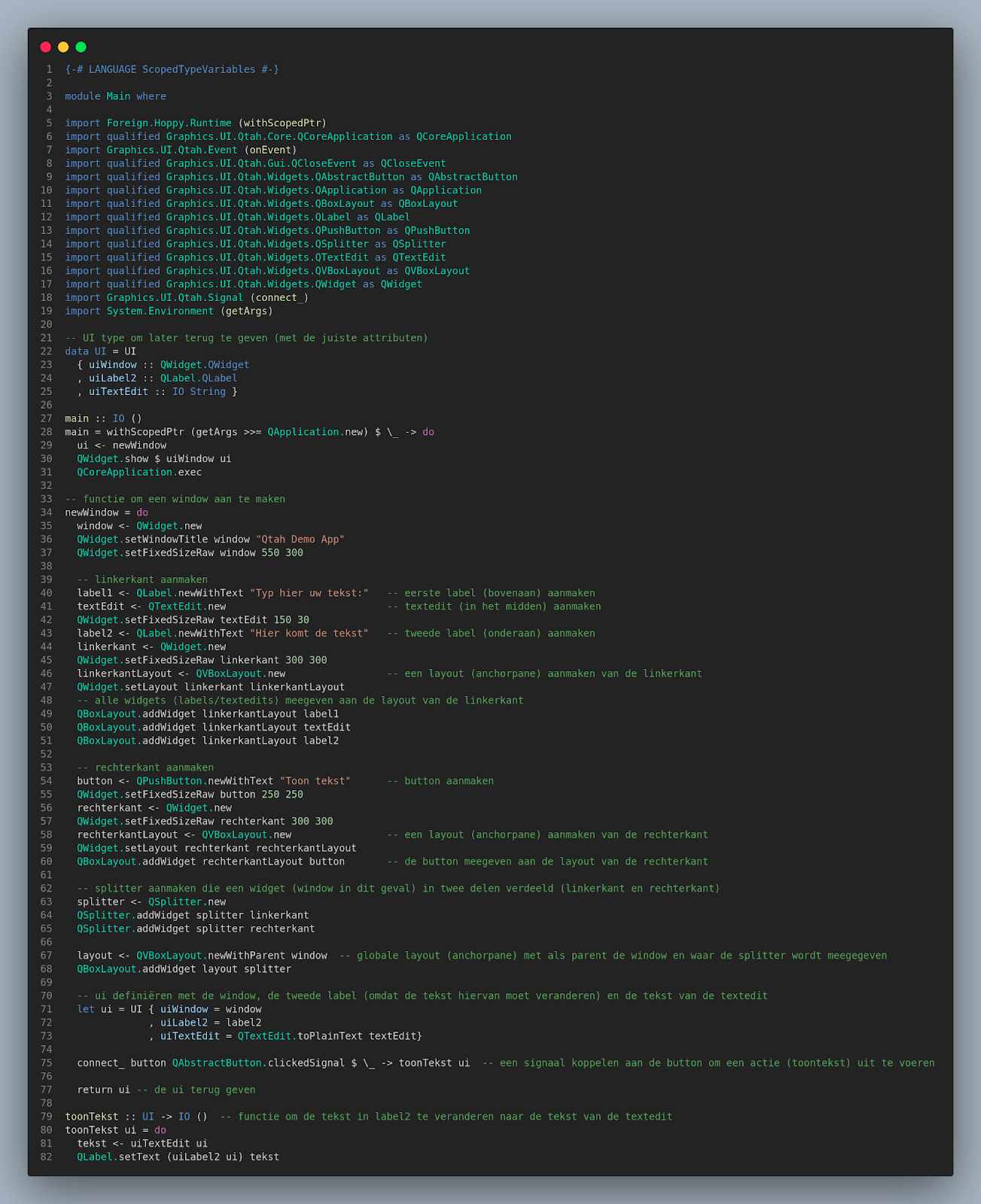
Tabel 4 toont de tegenhanger van iedere JavaFx widget in FLTKHS. Zoals eerder vermeld zijn de widgets primitief in FLTKHS. Er bestaan voor vele JavaFX widgets geen directe tegenhanger in FLTKHS. Zo wordt bijvoorbeeld een Label uit JavaFX geïmplementeerd als een primitieve Box widget zonder randen in FLTKHS.

De FLTKHS-bibliotheek toont meer voordelen dan nadelen vergeleken met de andere bibliotheken. Ten eerste zijn door het low level karakter van de bibliotheek bestaande JavaFX widgets moeilijker te implementeren aangezien deze widgets opnieuw zelf moeten worden gemaakt. Het low level karakter van FLTKHS zorgt er wel voor dat custom widgets maken toegankelijker is dan bij andere bibliotheken. Echter is dit een nadeel aangezien het zelf maken van de widgets meer tijd en lijnen code vereist. Daarnaast zijn de *callback* functies moeilijker te implementeren in FLTKHS en is het bewerken van andere widgets in de *callback* functie zelf niet evident. Ten slotte heeft de FLTKS geen native look zoals is te zien in fig 8.



Figuur 8: uiteindelijke demo-applicatie in fltkhs

1. **Qtah**



1

2

3

Figuur 9: code van de implementatie demo-applicatie in Qtah

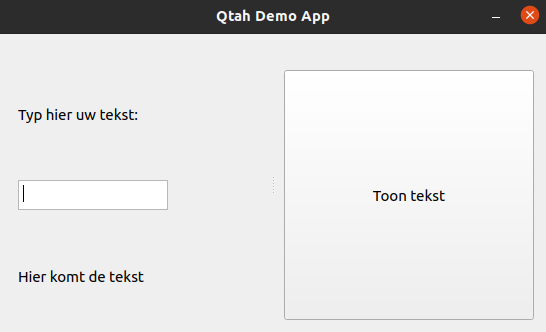
Figuur 9 toont de code van de demo applicatie in Qtah. Hetgeen wat direct opvalt, is dat er veel meer regels code is dan de vorige bibliotheken. Om te beginnen komt dit omdat elke widget die gebruikt wordt, individueel moet worden geïmporteerd (regel 5 tot en met regel 19). Daarnaast is het bij Qtah ook niet mogelijk om een widget al direct alle juiste attributen (zoals de hoogte en breedte bijvoorbeeld) mee te geven. Vaak kan enkel de tekstwaarde worden meegegeven bij het aanmaken van een widget. Regel 40 toont dit principe, waarbij een label wordt aangemaakt met als tekstwaarde “Typ hier uw tekst:”. Hierdoor zullen dus enkele regels code gebruikt moeten worden om de attributen van de widgets nog juist in te stellen. Verder heeft Qtah een vaste structuur dat er als volgt uitziet:

1. Aanmaken van de applicatie, waarin een nieuw venster (window) wordt gecreëerd.
2. Window aanmaken met alle widgets die erin horen en deze ook al op de juiste plaats neerzetten met de juiste attributen.
3. Widgets koppelen aan bepaalde signalen (Events) met een *callback* functie.

|  |  |
| --- | --- |
| **JavaFX** | **Qtah** |
| GridPane | ≈ QSplitter |
| Anchorpane | ≈ Q(V)BoxLayout |
| Text | QLabel |
| TextField | QTextEdit |
| Button | QPushButton |

Tabel 5: tegenhanger van de JavaFX widgets in Qtah

Tabel 5 toont de tegenhanger van elke JavaFX widget in Qtah. Voor elke widget bestaat er ongeveer een soortgelijke tegenhanger, al zijn de grotere widgets (zoals de Panes) niet exact hetzelfde. Dit brengt uiteraard moeilijkheden met zich mee omdat er geen duidelijke directe manier is voor een tegenhanger van een JavaFX widget in Qtah te implementeren.

De Qtah bibliotheek heeft enkele voordelen. Om te beginnen bestaat de bibliotheek uit een groot aantal widgets en GUI-elementen. Dit zorgt voor veel mogelijkheden om widgets en GUI-elementen van JavaFX om te kunnen zetten (met behulp van de codegenerator) naar Haskell code. Een ander voordeel is dat de bibliotheek uitbreidmogelijkheden heeft waardoor er nieuwe widgets kunnen toegevoegd worden indien nodig. Toch zijn er ook enkele nadelen. Doordat er zoveel widgets en GUI-elementen zijn, is het om moeilijk om de juiste widget te vinden die overeenkomt met deze in JavaFX. Ook zijn de meeste widgets niet exact hetzelfde waardoor deze moeilijk zijn om te omvormen zoals de *GridPane* of *AnchorPane* in het voorbeeldprogramma. Een ander nadeel van Qtah is dat het zeer moeilijk is om coördinaten mee te geven aan bepaalde widgets. Dit zorgt ervoor dat er moet gewerkt worden met verschillende vormen van layouts in combinatie met een vaste waarde voor de breedte en hoogte om de verschillende widgets op de juiste plaats te krijgen. Deze manier van werken is uiteraard niet ideaal. Ten slotte toon figuur 10 de demo applicatie van Qtah.

Figuur 10: uiteindelijke demo-applicatie in Qtah

**Conclusie bibliotheekkeuze**

Tabel 6: gedetailleerd overzicht voor en nadelen van de onderzochte bibliotheken

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | GTK2Hs | gi-gtk | fltkhs | Qtah |
| Aantal regels code (≈) | 33 | 34 | 29 | 60 |
| Voordelen | + Directe tegenhangers JavaFX widgets beschikbaar  +*callback* functies gemakkelijk te integreren  +set-functie beschikbaar | + Grotere subset bindings dan GTK2Hs | / | + Grote subset aan widgets beschikbaar |
| Nadelen | / | - Belangrijke widget niet implementeerbaar (*Layout*)  - Geen *set*-functie voor attributen van widget in te stellen | - Veel te low level waardoor widgets handgemaakt moeten worden  - *callback* functies moeilijk te implementeren | - Geen directe aanhanger of meerdere aanhangers mogelijk voor bepaalde JavaFX widget  - Niet evident om widgets met coördinaten te plaatsen  - Veel imports |

Het zijn de meeste bibliotheken gelukt om het voorbeeldprogramma na te maken in JavaFX. Echter waren bepaalde bibliotheken beter dan andere bibliotheken. Zo toont tabel 6 de gedetailleerde voor en nadelen van de geteste bibliotheken.

De fltkhs-bibliotheek heeft de grootste gebreken uit alle bibliotheken. Ten eerste is deze bibliotheek veel te low-level waardoor veel widgets uit JavaFX zelf gemaakt moeten worden. Dit is niet evident aangezien dit veel regels code en moeite vergt. Bovendien waren de *callback* functies in fltkhs moeilijk te implementeren en kunnen ze alleen widgets aanpassen waarnaar de *callback* functie zelf is aangewezen. Hoewel deze bibliotheek het minst aantal lijnen code nodig had, moet er rekening mee gehouden worden dat het niet mogelijk was om de functionaliteit van het voorbeeldprogramma correct te implementeren.

De Qtah bibliotheek had het grote voordeel dat er geen grote subset aan widgets aanwezig waren. Echter waren er voor bepaalde JavaFX widgets zoals de AnchorPane geen directe tegenhanger, wat ook betekende dat werken met coördinaten voor widgets niet evident was. De Qtah bibliotheek had ook het meeste lijnen code nodig.

De GTK2Hs en gi-gtk bibliotheken leken sterk op elkaar. Bijna iedere *binding* en *widget* hadden dezelfde naam in beide bibliotheken en beide bibliotheken hadden veel directe tegenhangers van de JavaFX widgets. Het werken met de *callback* functies was in beide bibliotheken ook probleemloos toepasbaar. Toch is de GTK2Hs bibliotheek een betere keuze aangezien de Layout-widget niet direct implementeerbaar is in gi-gtk. Daarnaast zorgde GTK2hs er ook voor dat de attributen van een bepaalde widget op een eenvoudige manier geïnitialiseerd kon worden met de *set*-functie, die bij gi-gtk niet aanwezig is.

De GTK2Hs bibliotheek is uit alle bibliotheken de bibliotheek die dus het grootst aantal voordelen toont met de minste gebreken. Daarom zal voor de codegenerator gebruik worden gemaakt van de GTK2Hs bibliotheek

# Bibliografie

|  |  |
| --- | --- |
| [1] | R. G. Flatscher en G. Müller, „Employing Portable JavaFX GUIs with Scripting Languages,” *Proceedings of the Central European Conference on Information and Intelligent Systems,* vol. XXXII, pp. 333-341, 2021. |
| [2] | J. Jenkov, „JavaFX Overview,” Jenkov.com, 9 maart 2021. [Online]. Available: http://tutorials.jenkov.com/javafx/overview.html. [Geopend 26 februari 2022]. |
| [3] | L. Prechelt, „An Empirical Comparison of Seven Programming Languages,” *Computing Practices,* pp. 23-29, 2000. |
| [4] | L. G. Lima, F. Soares-Neto, P. Lieuthier, F. Castor, G. Melfe en F. J. Paulo, „On Haskell and energy efficiency,” *The Journal of Systems and Software,* vol. CXLIX, pp. 554-580, 2019. |
| [5] | S. Marlow, L. Brandy, J. Coens en J. Purdy, „There is no Fork: an Abstraction for Efficient, Concurrent, and Concise Data Access,” *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming,* pp. 325-337, 2014. |
| [6] | A. Hejlsberg en M. Torgersen, „Overview of C# 3.0,” Microsoft, March 2007. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/bb308966(v=msdn.10)?redirectedfrom=MSDN. [Geopend 11 December 2021]. |
| [7] | A. S. Mena, „A Real World Guide to Programming,” *Practical Haskell,* vol. II, pp. 1-595, 2019. |
| [8] | S. Nanz en C. A. Furia, „A Comparative Study of Programming Languages in Rosetta Code,” *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering,* pp. 778-788, 2015. |
| [9] | V. Tirronen, S. Uusi-Mäkelä en V. Issomöttönen, „Understanding beginners’ mistakes with Haskell,” *Journal of Functional Programming,* pp. 1-30, 2015. |
| [10] | D. Leijsen, „wxHaskell - A Portable and Concise GUI Library for Haskel,” *Proceedings of the ACM SIGPLAN workshop on Haskell - Haskell '04,* pp. 57-68, 2004. |
| [11] | C. Antony en E. Conal, „Genuinely Functional User Interfaces,” in *ACM SIGPLAN Haskell Workshop*, Firenze, 2001. |
| [12] | P. Alan J., „EPIGRAMS IN PROGRAMMING,” *SIGPLAN Notices,* vol. 17, nr. 9, pp. 7-13, 1982. |
| [13] | E. Conal en J. Paul, „Haskell UI framework,” Stack Overflow, 26 augustus 2012. [Online]. Available: https://stackoverflow.com/questions/2860988/haskell-ui-framework?noredirect=1&lq=1. [Geopend 27 februari 2022]. |
| [14] | M. Sage, „FranTk – A Declarative GUI Language for Haskell,” *ACM SIGPLAN Notices,* pp. 106-117, 2000. |
| [15] | P. Hal en H. Michael, „GUI Event-Driven Programming,” 25 februari 2022. [Online]. Available: https://courses.cs.washington.edu/courses/cse331/22wi/. [Geopend 27 februari 2022]. |
| [16] | J. Smart, K. Hock en S. Csomor, Cross-platform GUI Programming with wxWidgets, United States of America, 2006, pp. 1-700. |
| [17] | M. M. Schrage, A. van IJzendoorn en L. C. van der Gaag, „Haskell Ready to Dazzle the Real World,” *Proceedings of the 2005 ACM SIGPLAN workshop on Haskell - Haskell '05,* pp. 17-26, 2005. |
| [18] | A. Siram, „Proceedings of the 2005 ACM SIGPLAN workshop on Haskell - Haskell '05,” Hask Anything!, 2016. [Online]. Available: https://haskanything.com/content/presentation/presentation--fltkhs-easy-native-guis-in-haskell-today.html. [Geopend 11 December 2021]. |
| [19] | K. Aerts, „VISTO: A DECLARATIVE METHODOLOGY FOR GRAPHICAL USER INTERFACES, BASED ON HASKELL,” *Computer-Aided Design of User Interfaces II,* pp. 1-294, 2001. |
| [20] | K. Hoste, „An Introduction to Gtk2Hs, a Haskell GUI Library,” pp. 1-15. |
| [21] | W. Swierstra, „The Real World,” 2007. |
| [22] | J. Bouwmans, „Managing consistency between dependent objects,” *Center for Software Technology Department of Information and Computing Sciences,* pp. 1-46, 2010. |
| [23] | S. E. Panitz en H. Rheinmain, „A Client for the Z21 Model Railway Control,” pp. 1-71. |
| [24] | D. Wagner, „Threading and Gtk2Hs,” [Online]. Available: http://dmwit.com/gtk2hs/. [Geopend 11 December 2021]. |
| [25] | V. Zavialov, „Introduction to GUI programming in Haskell,” Serokell, 2 oktober 2020. [Online]. Available: https://www.youtube.com/watch?v=k1aq8ikO-8Q&t=1105s. [Geopend 26 februari 2022]. |
| [26] | „haskell-gi,” haskell-gi, januari 2022. [Online]. Available: https://github.com/haskell-gi/haskell-gi. [Geopend 26 februari 2020]. |
| [27] | T. Hallgren en M. Carlsson, „Programming with Fudgets,” *Computing Science,* pp. 3-49, 1995. |
| [28] | B. Gardiner, „Qtah - Qt Bindings for Haskell,” khumba.net, 2021. [Online]. Available: http://khumba.net/projects/qtah. [Geopend 12 Maart 2022]. |
| [29] | B. Gardiner, „Hoppy - C++ FFI Generator for Haskell,” khumba.net, 2021. [Online]. Available: http://khumba.net/projects/hoppy. [Geopend 12 Maart 2022]. |
| [30] | „Class AnchorPane,” Oracle, 2015. [Online]. Available: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/layout/AnchorPane.html. [Geopend 9 maart 2022]. |
| [31] | S. Michael, E. Craig P. en S. Bill, „FLTK 1.0.10 Programming Manual,” The FLTK Team, 7 mei 2001. [Online]. Available: https://www.fltk.org/doc-1.0/. [Geopend 11 maart 2022]. |
| [32] | S. Aditya, „fltkhs: FLTK bindings,” Hackage, 21 februari 2020. [Online]. Available: https://hackage.haskell.org/package/fltkhs. [Geopend 10 maart 2022]. |
| [33] | G. Tony, M. Ian en v. T. Hans, „Gtk2Hs Tutorial,” The GTK Team, 2008. [Online]. Available: https://www.muitovar.com/gtk2hs/index.html. [Geopend 28 februari 2022]. |