**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 getest worden om uiteindelijk een keuze te kunnen maken voor de bibliotheek die gebruikt zal worden voor de codegenerator.

**Verschillende kandidaat 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 aparte uitgebreide documentatie over FLTKHS, 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, wat 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 gtk3 wordt 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**

[moet nog gedaan worden]

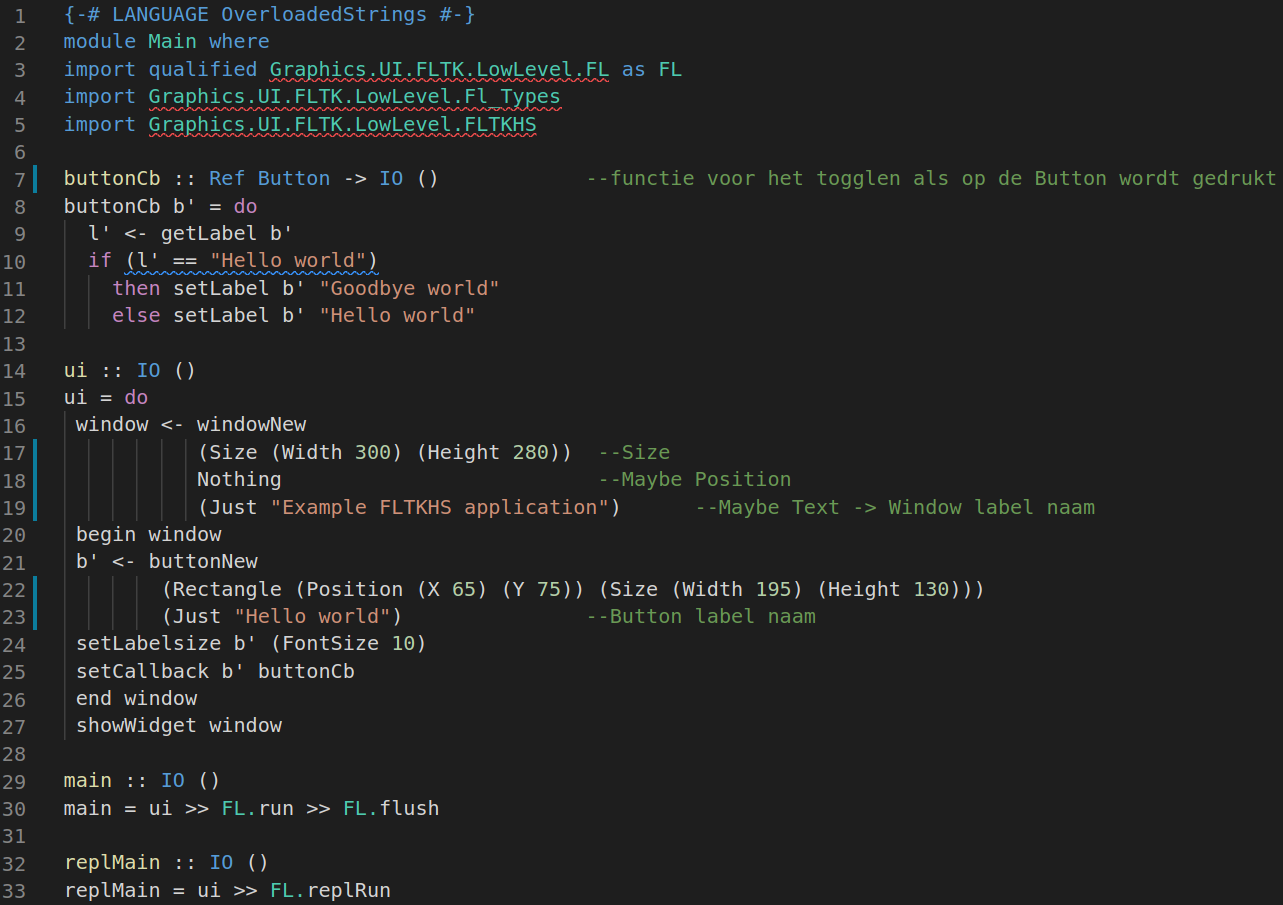
**Vergelijking verschillende bibliotheken**

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) |
| **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 | / |
| **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  -Lange compileertijd voor het programma de eerste keer te starten |

**Enkele kandidaat-bibliotheken uitgewerkt**

1. **FLTKHS**

Figuur 1: implementatie demo-applicatie in FLTKHS

1

2

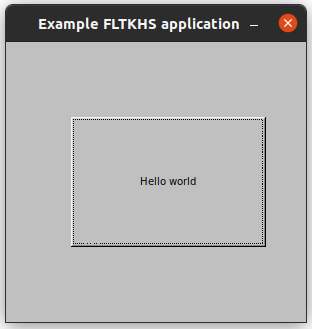
3

Figuur 1 toont de implementatie in Haskell van een demo-applicatie waarbij de FLTKHS-bibliotheek wordt gebruikt. Deze demo-applicatie, getoond in fig. 2 beschrijft een simpel venster met een *button* (knop) die een van tekst veranderd wanneer erop wordt gedrukt.

Er vallen enkele zaken op over het gebruik van de bibliotheek. Het eerste wat opvalt is de stijl van de *medium-low-level* bibliotheek waarbij bepaalde elementen imperatief worden opgeroepen, maar op een declaratieve stijl worden geïmplementeerd. Zo wordt het updatenvan de *button* op een meer imperatieve manier geïmplementeerd door eerst de naam van de *button* op te vragen en het vervolgens verder af te handelen door *setLabel* op te roepen. Verder is er een main functie/blok (*ui*) waar er een duidelijke imperatieve structuur is te zien die ook op fig. 1 is te zien:

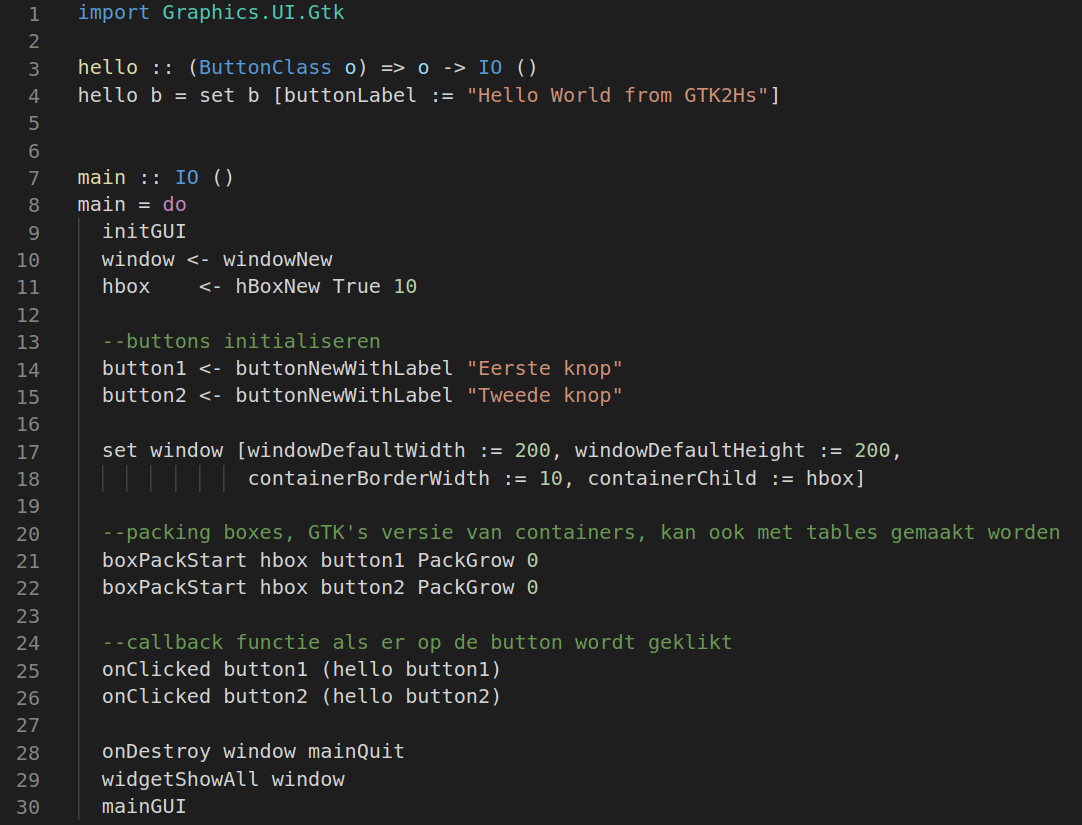
1) *window* initialiseren 2) widgets initialiseren en *callbacks* koppelen 3) afsluiten met *end window*

Daarnaast valt ook te zeggen dat hiërarchie en groeperingen van widgets in FLTKHS geïmplementeerd kan worden door *groups.* Echter was er geen directe duidelijk tutorial/instructie te vinden om dit op een eenvoudige manier toe te passen zal de implementatie van hiërarchie verder moeten worden opgezocht in de documentatie. Ten slotte heeft de FLKTHS bibliotheek geen *native look* vergeleken met andere GUI-bibliotheken.



Figuur 2: demo-applicatie in FLTKHS

1. **GTK2Hs**

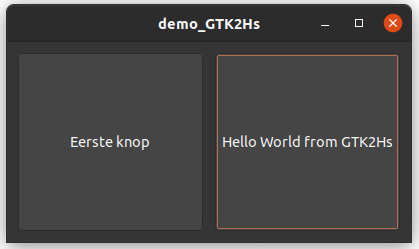
Figuur 3: implementatie demo-applicatie in GTK2Hs

1

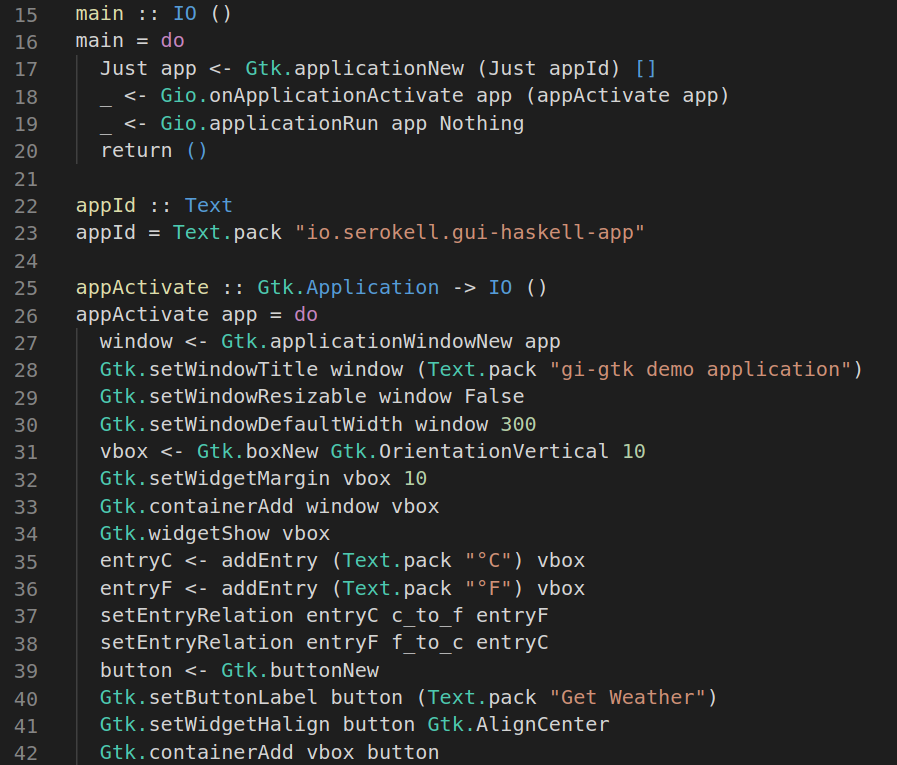
2

3

Figuur 3 toont de implementatie van een demo-applicatie waarbij de GTK2Hs-bibliotheek wordt gebruikt. Deze demo-applicatie, getoond in fig. 4 beschrijft een simpel venster met twee *buttons*  die een van tekst veranderen wanneer erop wordt gedrukt.

Ook bij de implementatie van de demo-applicatie in GTK2Hs vielen er enkele zaken op. Ten eerste heeft deze bibliotheek net als FLKTHS dezelfde stijl (*medium-level*) waarbij bepaalde elementen imperatief en declaratief worden afgehandeld. Ten tweede is de hiërarchie en groeperingen van widgets op eerste zicht beter uit te werken dan in FLTKS. Het principe om widgets te groeperen wordt in de tutorial van [28] beschreven als *packing*. Het principe van *packing* kan in GTK2Hs geïmplementeerd worden door *boxes* en *tables*. Bij *boxes* zijn de widgets niet even groot en bij *tables* zijn de widgets wel even groot zijn de widgets als in ware in een tabel geïmplementeerd [28]. Ten slotte is er net als in FLTKHS een vaste structuur aanwezig in om een applicatie op te starten die ook op fig. 3 is te zien:  
  
1) GUI en *window* initialiseren (*initGUI*) 2) *widgets* en *callbacks* initialiseren 3) afsluiten met *mainGUI.*

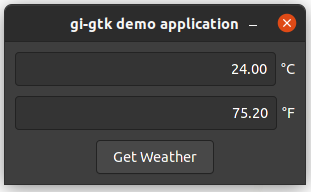
Figuur 4: demo-applicatie in FLTKHS

1. **gi-gtk**

Figuur 5: deel van de implementatie van een demo-applicatie in gi-gtk [25]

Figuur 5 toont de implementatie in Haskell van een demo-applicatie, gemaakt door [25], waarbij de gi-gtk-bibliotheek wordt gebruikt. Deze demo-applicatie, getoond in fig. 6 beschrijft complexer programma die de huidige temperatuur kan ophalen en omzetten in graden Celsius en Fahrenheit.

Gi-gtk kan worden gezien als een uitbreiding van GTK2Hs, maar toch met enkele verschillen en opmerkingen. Net als bij GTK2Hs en FLKTHS is er ten eerste een vaste structuur om een applicatie op te starten. Echter is de functionaliteit op eerste zicht meer uitgebreider dan in GTK2Hs. Zo kan bijvoorbeeld direct de vensternaam worden aangepast met *Gtk.setWindowTitle* wat in GTK2Hs niet direct op deze manier mogelijk is.

****Figuur 6: demo-applicatie in FLTKHS

# 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] | 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]. |