**Introductie**

[moet nog gedaan worden]

**Aanleiding en achtergrond**

Informatica en programmeren wordt traditioneel geassocieerd met professioneel Informatie- en Communicatietechnologie-gerelateerd (ICT) werk op de werkvloer. Maar met het huidige tempo van digitalisering, softwareontwikkeling en technologische evolutie speelt informatica een grotere rol op bijna alle gebieden van samenleving. Meer en meer curriculum beginnen een breder scala aan ICT-relateerde vakken aan te bieden om studenten de nodige problem-solving- en “thinking”-skills aan te leren [1].

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. Hoewel JavaFX een krachtige tool is om applicaties te ontwikkelen, kunnen andere programmeer-paradigma’s, programmeertalen en platformen voordelen bieden die later worden besproken.

**JavaFX en SceneBuilder**

JavaFX is een software platform voor (desktop)applicaties met een GUI te ontwikkelen die ondersteund worden door verschillende besturingssystemen waaronder Windows, Linux, MacOS, iOS en Android [2].

Een GUI wordt in JavaFX gedefinieerd als een boomstructuur van JavaFX nodes die de GUI vertegenwoordigen. De JavaFX nodes kunnen gedefinieerd worden in een XML gemarkeerd tekstbestand van het type FXML. FXML-bestanden kunnen zijn gemaakt en onderhouden met een GUI-tool, genaamd *SceneBuilder* waarbij studenten eenvoudig hun GUI-design kunnen beschrijven waarbij automatisch een FXML gegenereerd kan worden die gebruikt kan worden voor een JavaFX applicatie te ontwikkelen [2].

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 voor een alternatief platform en 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 meebepaald de programmeertaal Haskell.

Grote bedrijven die zich zorgen maken over energieverbruik, zoals Facebook, gebruiken functionele talen zoals Haskell 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’s 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 plus punt is vergeleken met imperatieve programmeertalen [4], [7].

Recursie is de norm van programmeerstijl voor Haskell omdat reguliere iteratieve lussen toestandsmutaties van variabelen vereisen waardoor complexe code aanzienlijk korter kan worden beschreven [4]. Gecombineerd met het feit dat Haskell functies kan gebruiken zonder bijwerkingen zorgt kan een robuust programma gerealiseerd 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 memory efficiënt 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 gedeprecieerd zijn en er voor andere programmeertalen GUI’s ontwikkelen populairder is.

**Verschillende Haskell GUI bibliotheken.**

1. **WxHaskell**

WxHaskell is gebaseerd op WxWidgets, een cross-platform GUI bibliotheek geschreven in C++ waarbij ongeveer 75% van WxWidgets functionaliteit is overgenomen. 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 [11], [10]. 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 native widgets van het besturingssysteem gaat gebruiken [10], [11]. Ten slotte kan WxHaskell met relatief weinig lijnen code geschreven worden en biedt het goede prestatie, korte compile time en weinig memory leaks [10], [12], [13].

WxHaskell kent ook nadelen. WxHaskell is in contrast tot WxWidgets minder onderhouden waardoor niet alle functionaliteit van WxWidgets *up to date* zijn. Zoals eerder vermeld wordt ook niet de volledige WxWidgets functionaliteit ondersteund. Hoewel de compile time klein is, zorgt WxHaskell soms wel voor grote binaries 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 [13].

FLTKHS kent als eerste voordeel dat er weinig dependencies zijn. Het is ook een van de gemakkelijkste bibliotheken te installeren op Windows. Bovendien is *Fluid*, een GUI generator die oorspronkelijk bedoeld was voor FLTK-code te genereren sinds kort beschikbaar voor FLTKHS [13]. De Haskell en C++ callbacks lijken volgens [13] ook sterk op elkaar waardoor ontwikkeling vlotter kan verlopen.

Echter kent FLTKHS ook nadelen. Zo zijn volgens [13] de compile en link tijden lang en niet optimaal. Hoewel FLTKHS als low level wordt beschreven is volgens [13] 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], [15]. 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 Action Event Listeners te implementeren zoals in Java. Daarnaast wordt FranTK geïmplementeerd in een tookit 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 beperken [14].

1. **GTK2Hs**

GTK2Hs is gebaseerd op GTK+, een multi-platform toolkit voor GUI’s te ontwikkelen, met een eerder medium tot low level interface [16], [17].

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 [18] 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 [19]. Ten slot 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 [16].

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 [18]. Hoewel er volgens [18] 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 [20] die snelle oplossingen biedt voor multithreading in GTK2Hs.

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, p46]. Een fudget is tegelijktijdig ook een proces die kan communiceren met andere fudgets en met de buitenwereld [21].

Volgens [21] 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**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Stijl** | **Multi platform** | **Onderhouden** | **Voordelen** | **Nadelen** |
| **WxHaskell** | Imperatief, met toch een declaratieve smaak (Medium-level) | Ja | Ja, maar 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 |
| **FLTKHS** | Low level | Ja | Ja | +Haskell en C++ callbacks lijken op elkaar  +Gemakkelijk te installeren op Windows | -Geen apparte uitgebreide documentatie, veel baseren op FLTK |
| **FranTK** | Declaratief, high-level | Ja, alleen Linux en Windows | Nee | +Listeners geparametriseerd  +High level | -Efficiency en performantie niet gegarandeerd door high-level |
| **GTK2Hs** | Medium-low-level | Ja | Ja | +Actieve development  +Rijkere interface elementen vergeleken met WxHaskell  +API beschikbaar (voor GTK+) | -Geen officiële support voor Multithreading |
| **Fudgets** | High level | Alleen Unix-like systemen. Werkt het beste op GNU/Linux. | Ja, maar laatste major release in 2000. | +Duidelijke documentatie  +Gemakkelijk hiërarchisch op te bouwen met fudgets |  |
| **Qtah** | Medium level | Ja | Ja | +Gebaseerd op Qt  +Actieve development | -Geen duidelijke en uitgebreide tutorials/demo’s, veel baseren op Qt  -Relatief jong  -Niet de volledige Qt bindings |

[1] Development of computational thinking, digital competence and 21st century skills when learning programming in K-9

[2] Employing Portable JavaFX GUIs with Scripting

Languages

[3]An Empirical Comparison of Seven Programming Language

[4] On Haskell and energy efficiency

[5] There is no Fork: an Abstraction for Efficient,

Concurrent, and Concise Data Access

[6] <https://docs.microsoft.com/en-us/previous-versions/dotnet/articles/bb308966(v=msdn.10)?redirectedfrom=MSDN>

[7] Practical Haskell A Real World Guide to Programming Second Edition

Alejandro Serrano

[8] A Comparative Study of Programming Languages in Rosetta Code

[9] Understanding beginners’ mistakes with Haskell

[10] wxHaskell - A Portable and Concise GUI Library for Haskell

[11] Cross-Platform GUI Programming with wxWidgets

[12] Haskell Ready to Dazzle the Real World

[13] <https://haskanything.com/content/presentation/presentation--fltkhs-easy-native-guis-in-haskell-today.html>

[14] FranTk – A Declarative GUI Language for Haskell  
[15] Doctoraat Aertsboy

[16] An Introduction to Gtk2Hs, a Haskell GUI Library

[17] The Real World - Wouter Swierstra - AFP 15-03-07

[18] Managing consistency between dependent objects

[19] A Client for the Z21 Model Railway Control

[20] [Threading and Gtk2Hs (dmwit.com)](http://dmwit.com/gtk2hs/)

[21]