Vrije Universiteit Brussel

Faculteit ingenieurswetenschappen
Vakgroep ingenieurswetenschappen

Implementatie van een elektronische circuit-simulator op basis van emotie-gebaseerde modellen

Proefschrift ingediend met het oog op het behalen van de graad van
Master of Science in de Industriele Ingenieurswetenschappen

Steven Vanden Branden

Promotor: Prof. Abdellah Touhafi

AUG 2014

Vrije Universiteit Brussel

Faculty of Engineering Sciences
Department of Science

Implementation of an electronic circuit simulator based on emotions-based models

Graduation thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Engineering

Steven Vanden Branden

Promotor: Prof. Abdellah Touhafi

AUG 2014

Abstract

This thesis describes the implementation of emotional models for electronic components in the developed circuit simulator. In this work the reasons for choosing to program in the chosen programming environment(Vala, GTK3 and Autotools) are explained. In this thesis a modified NGSpice simulator is used so that it is possible to attach emotions to components. These models for emotions of the components are described in this thesis.

The purpose of adding emotions to the components of the simulation is to clearly indicate to the student wether or not he has used the correct component. The student will need to take in the account the extra parameters of the components. The user of the developed software can easely adjust the used parameters and read the simulation data and the components emotions.

Samenvatting

Dit eindwerk beschrijft de implementatie van emotionele modellen voor elektronische componenten in de ontwikkelde elektronisch circuit simulator. Hierbij wordt de gebruikte programmeeromgeving (Vala, GTK3 en Autotools) beschreven en de redenen waarom deze software is gekozen. In dit eindwerk wordt gebruik gemaakt van een gemodificeerde NGSpice simulator die het mogelijk maakt om emoties toe te kennen aan de componenten. Deze ontwikkelde modellen voor de emoties van de componenten worden besproken in dit eindwerk.

Het doel van het toevoegen van emoties aan de componenten van de simulator is op een duidelijke en snelle manier aan de student duidelijk te maken of de juiste component in het circuit is gebruikt. De student zal hierbij rekening moeten houden met extra parameters (zoals het te dissiperen vermogen van een weerstand). De gebruiker van de ontwikkelde software kan eenvoudig de gebruikte parameters manipuleren waarna de simulatiegegevens en emoties meteen worden aangepast.

Résumé

Cette thèse décrit la mise en œuvre de modèles émotionnels pour les composants électroniques dans le simulateur de circuit développé. Dans ce travail les raisons sont expliqué pour choisir l'environnement de programmation choisi (Vala, GTK3 et Autotools). Dans cette thèse, un simulateur ngspice modifié est utilisé qui permet de fixer les émotions de composants. Ces modèles pour les émotions des composants sont décrits dans cette thèse.

Le but d'ajouter des émotions à des composants de la simulation est d'indiquer clairement à la étudiant il a utilisé l'appareil adéquat ou pas. L'étudiant devra prendre en compte les paramètres supplémentaires des composants. L'utilisateur du logiciel développé permet de facilement régler les paramètres utilisés et obtenir les données de simulation et d'émotions de composants.

Inhoudsopgave

Hoofdstuk 1

Inleiding

1.1 doelstellingen

H et doel van dit eindwerk is het ontwikkelen van een elektronische circuit simulator waarbij de gezondheidstoestand van de gebruikte componenten wordt voorgesteld aan de hand van emoties. Door het gebruik van emoticons voor het weergeven van de emoties van de componenten kan de gebruiker op een efficiënte manier de gezondheidstoestand van alle componenten in het circuit bepalen.

E en tweede doelstelling is de gebruikers (bijvoorbeeld studenten) bij gebruik van het programma er attent op worden gemaakt dat bij fysieke implementatie van een berekende schakeling er extra parameters zijn waarmee men rekening dient te houden. Deze parameters zorgen er in de praktijk voor dat schakelingen defect gaan of niet correct functioneren.

E en derde doelstelling is het ontwerpen van een uitbreidbare en dynamische simulator. Hiermee wordt bedoeld dat de verschillende delen (en klassen) van de implementatie niet sterk afhankelijk zijn van elkaar. Dit zorgt ervoor dat in de toekomst andere implementaties van verscheidene onderdelen van het programma kunnen worden ontwikkeld.

E en bijkomende doelstelling is het gebruik van vrij beschikbare technologie zodat studenten in staat zijn om elk onderdeel van de implementatie te onderzoeken en hieraan aanpassingen te doen. Hieronder vallen niet enkel de programma's gebruikt in de implementatie van de simulator maar ook de benodigde programma's voor het compileren van de onderdelen.

D e onderzoeksvraag kan samengevat worden als:

Onderzoek en implementatie van een vrij beschikbare en uitbreidbare simulator van elektronische circuits waarbij de gezondheidstoestand van de componenten wordt weergegeven aan de hand van emoties en bijhorende emoticons.

1.2 stappenplan en verloop

Een Gant-tabel van de planning is bijgevoegd in bijlage. De Gant-shart is als html bestand toegevoegd aan de digitale versie.

T eneinde aan alle doelstellingen te kunnen voldoen ben ik begonnen met een literatuurstudie. Deze literatuurstudie onderzocht de verscheidene programmeeromgevingen die beschikbaar waren op het Linux besturingssysteem. Ik heb gekozen om te werken op dit besturingssysteem omdat ik ermee vertrouwd ben en omdat (bijna) alle broncode voor dit besturingssysteem vrij beschikbaar is alsook het besturingssysteem zelf. Dit onderzoek is samengevat in sectie 1.3.

D e keuze voor de simulatie backend van de implementatie is gevallen op de NGSpice simulator. Dit is een voortzetting van de Berkeley Spice3f5 simulator gecombineerd met CIDER. Dit zorgt ervoor dat de simulator zowel analoog als digitale circuits kan simuleren, de combinatie is ook mogelijk.

I nitieel werd de simulator opgeroepen waarna de invoer en de uitvoer via Linux-pipes werd in- en uitgevoerd. Nadien werd deze vervangen door het rechtstreeks gebruiken van de gedeelde NGSpice bibliotheek. Aangezien de NGSpice simulator geschreven is in C code en de ontwikkelde simulator in Vala code heb ik een VAPI bestand geschreven. Dit bestand zorgt ervoor dat bij compilatie de Vala functies worden omgezet naar de overeenkomstige C functies. Dit zorgt ervoor dat de C compiler rechtstreeks de gedeelde bibliotheek kan aanspreken.

I nitieel was de conclusie gemaakt voor het gebruik van de QT toolkit voor het schrijven van het programma en de interface. Aangezien zoals beschreven in sectie 1.3.3 de code kan worden gecompileerd naar JavaScript via de Emscripten compiler zou dit ervoor zorgen dat het programma volledig in de browser kan draaien. Na enig onderzoek bleek echter dat hiervoor een grondig herschreven versie van de QT toolkit moet gebruikt worden. Deze versie

is reeds sterk verouderd en ondersteund onder andere geen netwerkfunctionaliteit. Dit zorgde ervoor dat de keuze voor de toolkit werd verworpen ten voordele van de GTK3 toolkit.

H et gedeeltelijk programma is omgezet naar een programma geschreven in Vala code die gebruik maakt van de GTK3 toolkit. Deze programmeertaal lijkt in verband met de syntax sterk op C#. Het heeft echter als grote voordeel dat de code van Vala broncode naar C broncode wordt gecompileerd. Dit zorgt ervoor dat er geen extra framework (zoals .NET) nodig is.

D e simulator zelf is modulair opgebouwd zodat er later gemakkelijk nieuwe componenten (of andere klassen) kunnen worden toegevoegd. Bij het schrijven van het programma werd echter duidelijk dat er geen standaard grafiek widget beschikbaar was in de GTK3-toolkit. Eerst heb ik overwogen om de GOffice bibliotheek te gebruiken die een grafiek widget bevat. Deze bibliotheek is echter geschreven in C taal en ik zou dus net zoals bij de NGSpice simulator een VAPI bestand moeten schrijven. Daarom leek het mij beter om zelf een grafiek widget te schrijven om zo de afhankelijkheden van het programma te verminderen.

B ij implementatie van het emotioneel weerstandsmodel in de NGSpice simulator heb ik dit eerst trachten te doen met de XSpice extensie van NGSpice. Hierbij kan men bij het draaien van de simulator nieuwe componenten toevoegen waardoor het niet nodig is om de gedeelde bibliotheek te hercompileren voor het toevoegen van componenten. Deze uitbreiding van NGSpice is echter niet volledig geïntegreerd met de simulator interface en het was dus onmogelijk om de gewenste data tijdens elke stap van de simulatie op te vragen.

T eneinde dit probleem te overkomen zijn de modellen rechtstreeks geschreven in de C code van de NGSpice simulator. Dit zorgt ervoor dat het toevoegen van de modellen een hercompilatie van de gedeelde bibliotheek vereist. De modellen bevinden zich in de gedeelde NGSpice bibliotheek zodat ook andere programma's of implementaties hiervan gebruikt kunnen maken.

D e emotionele modellen voor de condensator en de inductor zijn opgesteld en geïntegreerd in de gedeelde NGSpice bibliotheek.

T eneinde het programma te kunnen gebruiken op alle platformen die een HTML5 browser en websockets ondersteunen heb ik gebruik gemaakt van de

GTK3 Broadway backend. Dit zorgt ervoor dat de applicatie kan gedraaid worden op de server en dat men via de browser het programma kan besturen en bekijken. Hiervoor heb ik een webserver opgezet die de gebruiker inlogt, de Broadway server start en de ElektroSim applicatie hierop laat draaien.

1.3 Opties programmeeromgevingen

M ijn doel is een programma te schrijven dat op zoveel mogelijk platformen in te zetten is als educatief hulpmiddel. Dit vereist dat het programma zowel op computers(Windows/Linux) als op mobiele apparaten dient te functioneren.

A angezien het gebruik van het programma op een desktop waarschijnlijk handiger is wens ik de ontwikkeling vooral op het desktop platform te richten. Aangezien ik mijn doelgroep echter niet wil beperken tot de desktop moet het programma ook te gebruiken zijn in de webbrowsers van mobiele apparaten en systemen waarbij men niets kan/mag installeren. Hierbij is het wijselijk om het gebruik van (browser)plug-ins te vermijden zodat de gebruiker hiervoor niet eerst een plug-in moet downloaden en installeren.

1.3.1 Adobe Flash

D eze programmeeromgeving werkt met een tijdlijn die onderverdeeld is in verschillende frames. Het programma of script doorloopt deze frames en afhankelijk van de interactie van de gebruiker of de opbouw van het script wordt er geschakeld tussen de verschillende frames. Hierbij wordt gebruik gemaakt van een grafisch interface. Het maken van flash applicaties is veel minder gericht op het schrijven van code zoals programmeertalen C en C++.

E en nadeel van flash is dat de browsers steeds moeten voorzien worden van een plug-in om deze programma's weer te geven in de browsers. Hierbij is het belangrijk op te merken dat flash niet (meer) wordt ondersteund op Linux. De enige browser op Linux die wel nog ondersteund wordt is Google Chrome. Dit zorgt ervoor dat de ondersteuning voor dit platform niet toereikend is. Het gebruik van de plug-in voor websites op Android is vanaf versie 4.1 ongedaan gemaakt en wordt niet verder ondersteund. Hierbij valt wel op te merken dat applicaties gemaakt in flash kunnen draaien met behulp van AIR , hierbij draait het programma als een applicatie op het mobiel systeem. Het uitvoeren van Java in de browser is hierbij echter niet meer mogelijk en Adobe geeft dan ook de voorkeur aan HTML om die plaats in te nemen.

Op Apple platformen waaronder de Iphone heeft nooit Flash gedraaid in de browser, mobiele applicaties en AIR worden echter wel ondersteund.

1.3.2 HTML5 , CSS en javascript

HTML5 bestaat uit 3 componenten namelijk html of hypertekst markup Language, CSS en javascript. Hypertext Markup Language is een taal die definieert hoe webpagina's zijn opgebouwd. Hierbij staat Hypertext voor de verschillende links die de pagina's met elkaar kunnen verbinden. Markup wijst erop dat alle informatie die getoond wordt zich binnen tags bevindt die bepalen welke soort data deze bevat. HTML5 is een update van deze taal met enkele belangrijke verbeteringen: Heeft betere ondersteuning voor Semantic markup. Dit houdt in dat alle informatie metadata meedraagt die door onder andere zoekrobots wordt gebruikt om een webpagina te begrijpen. Dit maakt het mogelijk dat alle inhoud (zoals video, audio en foto's) steeds de nodige data meedraagt. Er is ondersteuning voor het maken van interactieve webapplicaties. Hierbij zijn de toevoeging van een Audio en Video tag zeer belangrijk omdat het nu mogelijk is (zonder plug-ins) om video en geluidsmateriaal toe te voegen aan een website. Ook is er ondersteuning voor een canvas waarop kan getekend worden en ondersteuning voor drag and drop.

C SS of Cascading Style Sheets is een taal die instaat voor de lay-out en vormgeving van een webpagina. Deze taal staat in voor het stylen van de verschillende elementen aan de hand van tags. Javascript is een scripttaal die gebaseerd is op C. Deze scripttaal maakt het mogelijk om cliënt-side interactie met de webpagina te verwezenlijken alsook server-site.

1.3.3 QT

Dit is een programmeeromgeving in C++ waarbij een programma eenmaal kan geschreven worden en daarna kan gebruikt worden op verschillende platformen. Deze platformen zijn onder andere Linux, Windows desktop besturingssystemen alsook mobiele systemen waaronder Symbian(Nokia), Android en IOS. Het is ook (beperkt) mogelijk om Qt code om te zetten naar javascript voor het gebruik in een webbrowser. Hiervoor kan er gebruik gemaakt worden van de Emscripten compiler alsook de Clang compiler. De LLVM/Clang compiler[25] is een C, C++ en objectieve-C compiler die duidelijk foutmeldingen geeft en die in staat is om C++ (en andere talen) om te zetten in de LLVM intermediaire representatie of ook LLVM-bitcode genoemd.

E mscripten is een LLVM naar javascript compiler die LLVM-bitcode kan omzetten naar javascript. Eigenlijk spreken we hier over een subset van javascript namelijk ASM.Js [1]. Dit is een subset die geoptimaliseerd is voor het gebruik met compilers. Bij het gebruiken van deze subset in javascript (en een browser die het ondersteund) kan de javascript code geoptimaliseerd verwerkt worden door de javascript verwerker. Dit wordt mogelijk gemaakt door de mogelijkheid om van te voren (voor de code echt gebruikt wordt) te compileren. Hierbij moet er geen rekening worden gehouden met onder andere garbage collection en bailouts. Dit komt omdat het geheugen manueel wordt beheerd door functies soortgelijk aan malloc and free, en het geheugen bestaat uit een grote HEAP getypeerde array.

D e ASM.js draait momenteel native op alle webbrowsers omdat het een subset is van javascript. Momenteel is enkel Firefox bezig met een geoptimaliseerde engine te schrijven die in staat is om deze subset te versnellen. Dit maakt het mogelijk om Qt-applicaties te draaien in een webbrowser. Hiervan vindt u enkele voorbeelden in bijgevoegde link [7].

E r zijn echter enkele beperkingen waaronder het gebruik van threads (Qthreads) niet wordt ondersteund omdat Java geen ingebouwde thread ondersteuning heeft. Een andere belangrijke opmerking is dat locale event loops niet kunnen worden gebruikt. Dit komt erop neer dat de Main functie moet afsluiten voordat men interactie kan maken met het programma. Synchrone versies van functies werken niet en dus zal men steeds asynchrone versies moeten gebruiken. Ook is de netwerkinterface niet compatibel met het draaien in de browser.

A angezien de Qt toolkit ook grotendeels dient aangepast of herschreven moet worden om deze functionaliteit werkend te houden is er na enig experimenteren met deze technologie toch gekozen om de technologie in de volgende paragraaf toe te passen voor deze toepassing.

1.3.4 GTK3 en de broadway daemon

D e GTK3-toolkit is een grafische interface toolkit die oorspronkelijk ontwikkeld is voor GIMP. GIMP is een beeldverwerker die mogelijkheden biedt die vergelijkbaar zijn met Adobe Photoshop.

T he GTK3-toolkit is een object georiënteerde toolkit die geschreven is in de C programmeertaal. De objectgeoriënteerde implementatie draagt de naam

Glib object systeem en zorgt ervoor dat men objectgeoriënteerd kan werken met de GTK3-toolkit. Deze toolkit wordt voornamelijk gebruikt op Linux op de Xorg displayserver. Door gebruik te maken van de GDK bibliotheek kan deze toolkit echter ook gebruikt worden op Windows, BSD, Unix, en Mac OS.

A angezien het schrijven van een programma in C veel boilerplate code vereist heb ik besloten om het programma te schrijven in de Vala programmeertaal. Dit is een source naar source compiler die gebruikt maakt van het Glib object systeem. Dit zorgt ervoor dat programma geschreven in Vala eerst worden gecompileerd naar overeenkomstige C code waarna een C compiler (zoals GCC) deze kan compileren naar machinecode.

H et voordeel van deze programmeertaal ten opzichte van bijvoorbeeld C# is dat deze taal wordt gecompileerd naar C code. Dit zorgt ervoor dat geen nieuw applicatie framework (zoals .NET) hoeft geïnstalleerd te worden en dat de C code en dus ook het programma kan gebruikt worden op systemen die niet beschikken over de Vala compiler. Ook zorgen de VAPI bestanden ervoor dat alle op Gobject gebaseerde bibliotheken (zoals gstreamer, Gio en vele anderen) gemakkelijk kunnen aangeroepen worden vanuit de Vala code. Elke niet op Gobject gebaseerde bibliotheek kan ook gebruikt worden in Vala maar men zal handmatig het correcte VAPI bestand moeten schrijven.

S inds versie 3.2 van deze toolkit is deze ook in staat om via de zogenaamde Broadway GDK backend applicaties te laten draaien in een HTML5 browser. Dit maakt het mogelijk om GTK3 applicaties op een server te draaien en op elk toestel met een webbrowser hiermee interacties uit voeren.

1.4 simulatoren

1.4.1 Falstad Circuit Simulator Applet

Deze simulator is een Java webbrowser toepassing die elektronische circuits kan simuleren. Hierbij is de simulator volledig geschreven in Java en maakt geen gebruik van een webservice voor het simuleren van de circuits. De simulator kan zowel digitale als analoge schakelingen simuleren en toont hierbij de stroom en spanning over de componenten. De simulator kan ook als Java applicatie draaien op de desktop.

A angezien deze simulator gebruik maakt van Java dient een Java virtuele machine te zijn op de computer van de gebruiker. Verder dient ook de Java webbrowser plugin voor de gebruikte webbrowser geïnstalleerd te zijn. Dit zorgt ervoor dat deze keuze niet voldoet aan de vereisten.

1.4.2 GeckoCIRCUITS

Deze simulatie software heeft als doel het simuleren van vermogen elektronica. Hierbij beschikt het ook over geïntegreerde thermische simulatiemodellen. De software kan ook geïntegreerd worden met Matlab(Simulink) en bestaat in een basis versie die vrij te gebruiken is. De professionele versie is betalend en bied ondersteuningscontracten aan. De software wordt ontwikkeld door het bedrijf Gecko Simulations [13](een spinoff van de ETH universiteit te Zurich).

A angezien deze software ook geschreven is in Java voldoet deze ook niet aan de vereisten.

1.4.3 Gnucap

Gnucap is een analoge en digitaal gemixte circuit simulator geschreven in C++. Deze simulator is niet gebaseerd op de Spice simulator en is dus (momenteel) niet in staat om alle Spice simulaties uit te voeren. De simulator ondersteunt verder Verilog en Spectre als simulatie talen.

H et grote verschil met andere simulators is dat deze sterk gebaseerd is op het gebruiken van plugins. Dit zorgt ervoor dat dit systeem zeer flexibel is omdat er gemakkelijk nieuwe componenten kunnen worden toegevoegd. Zo zijn onder andere modellen, commando's, simulatie algoritmen en interfaces met andere software programma's geschreven als plugins voor de simulator. De plugins worden momenteel vooral in C++ geschreven maar men is ook bezig met het schrijven van comptabiliteitslagen voor het ondersteunen van plugins in andere programmeertalen.

A angezien deze software (nog) niet alle Spice simulaties (volledig) kan uitvoeren is de keuze niet op deze simulator gevallen. Indien deze simulator in staat wordt om deze simulaties uit te voeren is het een sterk alternatief voor de gekozen NGSpice simulator door het gebruik van het flexibele plugin systeem.

1.4.4 NGSpice

D eze simulator is een voortzetting van de Berkeley Spice3f5 simulator waarbij extra functionaliteit is toegevoegd. Deze simulator ondersteunt verder ook de oudere Spice2 syntax. De simulator is geschreven in C en is de meest robuuste open-source voortzetting van de Spice Simulator. Deze simulator is ook beschikbaar op de meeste Linux platformen. De simulator staat beschreven in het hoofdstuk NGSpice.

Hoofdstuk 2

NGSpice

De NGSpice simulator [23] is een circuit simulator voor non-lineaire en lineaire simulaties. Hierbij bestaan er modellen voor onder andere weerstanden, capaciteiten, spanningsbronnen en stroombronnen. Ook zijn er modellen voor transmissielijnen, schakelaars, diodes, BJT's, JFET's, MESFET's en MOSFET's.

D eze simulator is opgebouwd uit 3 delen. Het belangrijkste deel is een voortzetting van de Spice3f5 simulator. Hierbij worden bestaande fouten opgelost en worden nieuwe features toegevoegd. Deze simulator is de laatste Berkeley uitgave van de Spice3 simulator.

T en opzichte van de Spice3f5 simulator kan de NGSpice simulator ook overweg met mixed-level simulatie. Dit houdt in dat zowel analoge als digitale (bv SOC's)componenten kunnen worden gesimuleerd. Dit wordt gerealiseerd door gebruik te maken van 2 uitbreidingen CIDER en GSS. CIDER is een mixed-level circuit en device simulator die geïntegreerd is in NGSpice. De naam komt van de uitbreiding die reeds bestond voor Spice3. De tweede extensie is GSS TCAD. Dit is een 2 dimensionale simulator die onafhankelijk van NGSpice wordt ontwikkeld maar waarvoor NGSpice een socket-interface voorziet.

V oor het gebruik van mixed-level simulatie wordt er gebruik gemaakt van XSPICE software waarbij men modellen kan toevoegen door de SPICE code-model interface of door de ADMS interface gebaseerd op Verilog-A en XML.

2.1 Simulatie Algoritmen

NGSpice beschikt over 7 circuit analyses:
— gelijkstroom analyse (werkpunt en sweep)
— wisselstroom klein-signaal model analyse
— transiënt analyse
— nulpunt analyse
— klein-signaal afwijking analyse
— gevoeligheidsalyse
— storingsanalyse

H ierbij kunnen analoge schakelingen gebruik maken van alle simulatiemethoden (met uitzondering van het code model subsysteem dat enkel de eerste 3 ondersteunt). Digitale schakelingen en modellen kunnen enkel gebruik maken van gelijkspanningsanalyse en transiënt simulatie.

2.1.1 gelijkstroom(DC) analyse

B ij gelijkstroomanalyse wordt het werkpunt van de schakeling bepaald. Dit houdt in dat condensators een open circuit worden en dat spoelen worden kortgesloten. Indien men gebruik maakt van zowel analoge als digitale modellen zal de simulatie het circuit indelen in een analoge en digitale simulatie waarbij er geïtereerd wordt over de verbindingen tussen de simulaties zodat de waarden consistent worden. Hiervoor gebruikt men de controleregel .op .

D eze simulatie kan indien nodig ook het klein-signaal model van de transfer functie berekenen alsook de ingangsweerstand en uitgangsweerstand.

O ok kan men hiermee sweeps uitvoeren over een gespecificeerde onafhankelijke spannings— of stroombron, weerstand of temperatuur. Waarna men de invloed op de spanning over de component kan testen. Deze functie over weerstanden en temperaturen is niet beschikbaar in de SPICE3 simulator. Hierbij gebruikt men de .dc controleregel waarbij men 4 argumenten meegeeft waarvan het eerste argument de component of temperatuur is waarvan men een sweep wilt uitvoeren. Het tweede en derde argument bepalen respectievelijk de start- en eindwaarden van de component. Het vierde argument bepaalt de stap waarmee de sweep wordt uitgevoerd.

2.1.2 wisselstroom klein-signaal model analyse

D eze analyse maakt gebruik van het klein-signaal model voor het simuleren van het circuit. Hiervoor wordt er eerst een DC analyse uitgevoerd waarbij men de klein-signaal equivalenten berekent voor de gebruikte analoge componenten. Dit circuit wordt dan op de door de gebruiker bepaalde frequenties geanalyseerd. Hierbij is het meestal de bedoeling om de frequentie antwoord van het systeem te bepalen. Hiervoor gebruikt men de .ac controleregel met 4 argumenten. Het eerste argument bepaalt het interval (bv per decade (dec)). Het tweede argument bepaalt het aantal punten per decade. het derde en vierde argument bepalen respectievelijk de begin- en eindfrequentie van de simulatie.

2.1.3 transiënt analyse

H ierbij wordt een DC analyse gemaakt waarbij de tijd een extra dimensie is. Hiervoor wordt eerst het werkpunt bepaalt door een DC analyse waarna de tijdsafhankelijke componenten terug worden toegevoegd. Indien er componenten zijn die niet veranderen in de tijd zoals vaste spannings- of stroombronnen wordt de DC waarde voor deze componenten gebruikt.

H et instellen van deze analyse gebeurt met de .TRAN controleregel. Het eerste argument is de maximale tijdstap die toegestaan is in deze simulatie. Het tweede argument geeft aan hoeveel seconden de transiënt analyse moet worden berekend.

2.1.4 polen-nulpunten analyse

Hierbij wordt eerst een DC analyse gedaan zodat men het werkpunt en het equivalent klein-signaal model van de niet lineaire componenten kan bepalen.

D eze simulatie werkt voor alle soorten componenten met uitzondering van transmissielijnen. Ook maakt het gebruik van een numeriek algoritme dat niet altijd een correcte oplossing berekent. Ofwel vindt het geen nulpunten/-polen ofwel net heel veel.

2.2 Netlist

O m de indeling van het circuit duidelijk te maken aan de simulator is het nodig dat deze wordt beschreven in een netlist bestand. De eerste regel is

steeds de titel van het circuit die men vrij mag kiezen, de laatste regel bevat enkel .end . Van alle andere regels mag men de volgorde vrij bepalen (enkel indien deze niet van voorgaande regels afhankelijk zijn). Verder worden spaties aan het begin van een lijn genegeerd alsook lege regels.

V oor het toevoegen van commentaar aan de netlist moet men de commentaarlijn voorafgaan door * teken. Ook kan men aan het einde van de regel '//' of ';' typen waarna alle tekst erachter op dezelfde regel als commentaar wordt beschouwd.

2.2.1 Elementen

Elke component in een circuit wordt beschreven met volgende syntax:

[naam component] [verbindingspunt 1] [verbindingspunt 2] [parameters voor model]

D e naam van de component begint steeds met een hoofdletter, die aangeeft van welk type model deze component is. De naam kan verder worden uitgebreid met getallen en letters zodat elke component uniek kan beschreven worden. De 2 verbindingspunten beschrijven de verbindingsnetten en worden voorgesteld door cijfers waarbij 0 gereserveerd is voor de grond. De overige parameters zijn afhankelijk van het model. De volgende lijst beschrijft een selectie van de NGSpice modellen. Voor de volledige lijst zie referentie [23].

A: XSPICE code model

C: Capaciteit

D: Diode

I: stroombron

L: spoel

M: MOSFET

R: weerstand

V: spanningsbron

X: subcircuit

2.2.2 modellen

. MODEL zorgt ervoor dat men bij een complexe component(met vele argumenten) die meermaals voorkomt in de beschrijving van het circuit niet

alle argumenten steeds opnieuw hoeft te typen. Hierbij worden de nieuwe modellen beschreven aan de hand van bestaande NGSpice modellen met vaststaande parameters.
.model MODR100 R (R=100)
Het eerste argument is de naam voor het model dat kan gebruikt worden in de netlist. Het tweede argument bepaalt welk component model gebruikt wordt en de argumenten tussen de haakjes beschrijven welke waarden sommige parameters van het model zullen gebruiken.

2.2.3 subcircuits

S ubcircuits kunnen gebruikt worden in een netlist om elementen samen te voegen die later opnieuw kunnen gebruikt worden en dus maar 1 keer beschreven moeten worden. Dit zorgt ervoor dat men hiërarchische structuren kan maken. De NGSpice simulator zal deze subcircuits echter gewoon invoegen (de beschrijving kopiëren) op plaatsen waar men het subcircuit invoert.

O m een subcircuit te beschrijven gebruikt men de .subckt controleregel. De regel wordt gevolgd door de naam die begint met een X om aan te duiden dat dit een subcircuit is. Hierna volgen de verbindingspunten. Om de beschrijving van het subcircuit af te sluiten gebruikt men .ends Xnaam. Om deze component ergens in te voegen gebruikt men gewoon de naam gevolgd door de verbindingspunten.

2.3 XSPICE

X SPICE is een uitbreiding van de SPICE3 simulator die het mogelijk maakt om gemakkelijk eigen modellen toe te voegen die gebruik maken van een functiebeschrijving in C code en een interfacebeschrijving in tekst formaat. Dit als een alternatief voor het gebruik van subcircuits en modellen (SPICE3 modellen). Deze modellen hebben als voordeel dat ze als gedeelde bibliotheek kunnen worden ingeladen als het programma start en dus niet samen met NGSpice moeten worden aangeleverd. Dit maakt het mogelijk om nieuwe modellen toe te voegen aan een bestaande installatie. Deze modellen worden steeds voorafgegaan met een 'a' om aan te duiden dat het om een gecodeerd model gaat met het XSPICE code-model.

D eze uitbreiding voegt aan SPICE 3 toe:
- — zelf te kiezen aantal in en uit poorten
- — zowel scalaire als vector poorten en parameters

— differentiële en aan de grond gerefereerde poorten
— poorten die zowel ingang als uitgang zijn
— 12 reeds beschikbare poort types (12 state digitale poort, string,etc..)
— poorten die meerdere types kunnen ontvangen
— zelfgegenereerde poorttypes
— logische inversie op poorten
— gebruik van null om niet gebruikte poort aan te duiden
— parameters van types real,integer, complex, string en boolean
— standaard waarden voor parameters en controle op grenzen

2.3.1 XSPICE code modellen

D eze modellen bestaan uit 2 bestanden die het model beschrijven. Het eerste bestand is het ifspec.ifs bestand dat de interface beschrijft. Het tweede bestand is het cfunc.mod bestand die het model beschrijft waarbij een C-functie de implementatie van het model beschrijft.

2.3.2 eigen model toevoegen

E en eigen model toevoegen aan XSPICE [6] gebeurt in 4 stappen.

D e eerste stap is het toevoegen van een map met de naam van de nieuwe component in NGSpice/src/xspice/icm/*/

D e tweede stap is het kopiëren van de 2 bestanden die het model beschrijven van een soortgelijk model of van de voorbeelden. Hierna moet men deze 2 bestanden aanpassen om ervoor te zorgen dat het overeenkomt met het model dat men wilt beschrijven.

D e derde stap is het aanpassen van het modpath.lst zodat de naam van de component zich in het bestand bevindt. Dit zorgt ervoor dat de component zal toegevoegd worden in de gedeelde bibliotheek. Hierna compileert men de code met make.

2.3.3 het interface specificatie bestand

D it bestand beschrijft het model waarbij wordt aangeduid welke poorten , parameters en variabelen gebruikt worden in het model. Dit gebeurt door middel van 4 soorten tabellen.

D e eerste tabel is de 'NAME_TABLE' tabel. Deze tabel heeft 3 parameters:

C_Function_Name: hier geeft men de naam in van de c functie die beschreven is in het bijhorende cfunc.mod bestand. Om te vermijden dat er naam conflicten zouden optreden wordt de naam voorafgegaan door ucm_, wat staat voor user code model.

SPICE_Model_Name: Dit is de naam van het model (naam van de map waarin dit bestand zich bevindt).

Description: beschrijving van het model.

D e tweede tabel is de 'PORT_TABLE' tabel. Deze beschrijft de verschillende poorten van het model. Hiervoor zijn er 8 argumenten:

Port_Name: Dit is de naam van de poort die moet voldoen aan de eigenschappen van een geldige SPICE identificator.

Description: beschrijving van de functie van de poort

Direction: De richting waarin de data (spanning,stroom,etc.) door de poort gaat. Hierbij wordt in onderstaande tabel aangeduid wat de mogelijkheden zijn afhankelijk van het type poort.

type poort	beschrijving	geldige richtingen
d	digitale poort	in of out
g	geleiding(VCCS)	inout
gd	differentiële geleiding(VCCS)	inout
h	weerstand(CCVS)	inout
hd	differentiële weerstand(CCVS)	inout
i	stroom	in of out
id	differentiële stroom	in of out
v	spanning	in of out
vd	differentiële spanning	in of out
zelf gedefinieerd	zelf gedefinieerd type	in of out

Default_Type: Het standaard type van de poort (zie bovenstaande tabel)

Allowed_Types: de toegelaten types van poorten in syntax:[type 1 type 2]

Vector: definieert of de poort een bus of een enkele poort voorstelt. De waarden YES of TRUE zorgen ervoor dat de poort als bus beschouwd wordt. NO en FALSE zorgen ervoor dat geen vector wordt gebruikt.

Vector_Bounds: beschrijft (indien de poort een vector is) de minimum en maximum grootte van de vector door volgende syntax: [ondergrens bovengrens]. Indien de poort geen vector is is de waarde gelijk aan '-'. Deze waarde kan ook als boven- of ondergrens worden gebruikt om aan te duiden dat de vector geen boven- of/en ondergrens heeft.

Null_Allowed: Indien YES of TRUE zal de simulator geen foutmelding geven als de poort niet verbonden is. Bij NO of FALSE is dit wel het geval.

D e derde tabel is de 'PARAMETER_TABLE' tabel. Deze beschrijft de verschillende parameters van het model. Hiervoor zijn er 8 argumenten:

Parameter_Name: Dit is de naam van de parameter die moet voldoen aan de eigenschappen van een geldige SPICE identificator.

Description: beschrijving van de parameter

Data_Type: Het type van parameter, mogelijkheden zijn boolean,complex,int,real of string.

Default_Value: standaard waarde voor parameter indien niet gespecificeerd in de simulator.

Limits: beschrijft bij integers of real type variabelen welke waarden toegelaten zijn. De syntax is gelijkaardig aan vector_bounds.

Vector: definieert of de parameter een vector of een enkele waarde voorstelt. De waarden YES of TRUE zorgen ervoor dat de parameter als vector beschouwd wordt. NO en FALSE zorgen ervoor dat geen vector wordt gebruikt.

Vector_Bounds: beschrijft (indien de parameter een vector is) de minimum en maximum grootte van de vector door volgende syntax: [ondergrens bovengrens]. Indien de parameter geen vector is is de waarde gelijk aan '-'. Deze waarde kan ook als boven- of ondergrens worden gebruikt om aan te duiden dat de vector geen boven- of/en ondergrens heeft. Ook is het mogelijk om hier een andere parameter of poort in te vullen. Hierbij zullen de grenzen worden overgenomen.

Null_Allowed: Indien YES of TRUE moet bij het beschrijven van de component in de simulator geen waarde worden meegegeven (optioneel). Hierbij zal de standaardwaarde worden gebruikt. Bij NO of FALSE zal de simulator een foutmelding geven indien deze niet is ingevuld.

D e vierde optionele tabel is de 'STATIC_VAR_TABLE' tabel. Deze beschrijft statische variabelen die behouden worden bij het opnieuw aanroepen van het model. Hiervoor zijn er 3 argumenten:

Name: Dit is de naam van de statische variabele die een geldige C identificator moet zijn.

Description: beschrijving van de statische variabele

Data_Type: Het type van statische variabele, mogelijkheden zijn boolean,complex,int,real,st of pointer.

2.3.4 het model definitie bestand

O m toegang te krijgen tot de parameters,poorten en statische variabelen die beschreven zijn in het interface specificatie bestand wordt gebruik gemaakt van MACRO's. Deze Macro's aangemaakt bij het compileren en kunnen dus meteen gebruikt worden in de C-code functie.

circuit gegevens

Dit zijn de macro's die toegang bieden dat informatie over de toestand van het circuit waarvan dit model een onderdeel is.

A RGS is een argument dat moet worden meegegeven als argument voor alle model definiërende functies in C-code. Dit maakt het mogelijk om te refereren naar alle andere macro's.

C ALL_TYPE geeft EVENT of ANALOG terug als waarden. De EVENT waarde wordt teruggegeven als de aanroepende simulator en digitale simulator is en ANALOG indien het de analoge SPICE simulator betreft. Dit is handig indien men een model van een component beschrijft die de overgang van een analoog naar een digitaal signaal moet maken.

I NIT duidt aan of dit de eerste aanroep is van dit model. Dit is handig indien men geheugen dient te alloceren voor statische variabelen.

A NALYSIS heeft de waarde DC,AC of TRANSIENT en geeft aan welk type simulatie er uitgevoerd is.

F IRST_TIMEPOINT geeft aan of dit de eerste oproep van het model is in de huidige analyse stap. Dit is handig voor het verwerken van een verloop of sweep.

T IME geeft de analyse tijd weer bij een transiënt analyse.

R AD_FREQ geeft de huidige frequentie in een AC-analyse weer in radialen per seconde.

T EMPERATURE geeft de huidige temperatuur van de simulatie weer.

parameter gegevens

Dit zijn de macro's waarmee men de gedefinieerde parameters kan aanspreken. Hierbij wordt p als voorbeeld parameter gebruikt.

P ARAM(p) geeft de waarde van de parameter terug, indien deze een string is wordt een pointer terug gegeven. Men kan ook de waarden van een vector opvragen met bijvoorbeeld p[1].

P ARAM_SIZE(p) geeft een integer terug die de grote van de vector weergeeft. Bij scalaire parameters is deze waarde niet gedefinieerd.

P ARAM_NULL(p) geeft 0 weer indien de waarde gespecificeerd was en 1 indien deze niet gespecificeerd was en dus de standaard waarde heeft aangenomen.

poort gegevens

Dit zijn de macro's waarmee men de poorten kan aanspreken. Hierbij wordt p als voorbeeld poort gebruikt.

P ORT_SIZE(p) geeft de grootte van de bus weer. Indien de poort geen bus is, is deze waarde onbepaald.

P ORT_NULL geeft 0 weer indien de waarde gespecificeerd was en 1 indien deze niet gespecificeerd was en dus de standaard waarde heeft aangenomen.

L OAD(p) wordt gebruikt bij een digitaal model om een capacitieve lading op een poort te plaatsen.

T OTAL_LOAD is de som van all LOAD(p) die op de poort geplaatst zijn.

2.4 SPICE modellen in c

V oordat de XSPICE extensie bestond werden de nieuwe modellen slechts op 2 manieren toegevoegd. Door het gebruiken van subcircuits en de .include regel in de netlist. Een tweede manier was het schrijven (of aanpassen) van een model in de c-code van NGSpice. Deze tweede methode zal in dit hoofdstuk worden beschreven (zie ook [21]).

Z odoende een model te kunnen toevoegen aan de bestaande implementatie moet men enkele functies toevoegen aan het programma. Hieronder worden enkel belangrijke functies en tabellen aangehaald en wat de werking is van deze structuren en functies.

2.4.1 model en instantie beschrijving

E en eerste belangrijk paar structuren zijn de instantie en model structuren. Deze structuren beschrijven de parameters van het model of zijn instantie. Een model beschrijft de parameters die gedeeld kunnen worden tussen verschillende instanties van een model waarbij het model de gemeenschappelijke parameters bevat. Hierna volgt een voorbeeld van een instantiebeschrijving en modelbeschrijving.

```
//definieert een instantie voor een weerstand
        typedef struct sRESinstance {
        //pointer naar het model van deze instantie
    struct sRESmodel *RESmodPtr;
    //pointer naar volgende instantie
    struct sRESinstance *RESnextInstance;

    IFuid RESname;  //pointer naar de naam van deze
       instantie
    int RESstate;   //staat van de instatie
    int RESposNode; //aantal positieve connectiepunten
    int RESnegNode; //aantal negatieve connectiepunten

    /* enkele parameters van de instantie

    double REStemp;
    double RESdtemp;
    double RESconduct;
```

```
    double RESresist;
    double REScurrent;
    double RESacResist;
    double RESacConduct;
                ...

    //pointers naar matrix diagonalen

    double *RESposPosptr;
    double *RESnegNegptr;
    double *RESposNegptr;
    double *RESnegPosptr;

    //vlaggen die aanduiden of een parameter is
       aangeleverd of niet

    unsigned RESresGiven    : 1;
    unsigned RESwidthGiven  : 1;
    unsigned RESlengthGiven : 1;

} RESinstance ;

/* instantie parameters die genummerd worden en
   gebruikt worden om de data op te vragen*/
#define RES_RESIST 1
#define RES_WIDTH 2
#define RES_LENGTH 3
#define RES_CONDUCT 4
...
//weerstandsmodel
typedef struct sRESmodel {
    int RESmodType;
       // index van model type
    struct sRESmodel *RESnextModel; // pointer naar
       volgend model
    RESinstance * RESinstances;         // pointer naar
       lijst van
    IFuid RESmodName;                   // pointer naar
       naam van model


        //model parameters
    double REStnom;
```

```
    double REStempCoeff1;
    double REStempCoeff2;

    // vlaggen die aanduiden of parameters
       gespecificeerd zijn
    unsigned REStnomGiven       :1;
    unsigned REStc1Given        :1;
    unsigned REStc2Given        :1;

} RESmodel;
// model parameters die genummerd worden en gebruikt
   worden om de data op te vragen
#define RES_MOD_TC1 101
#define RES_MOD_TC2 102
#define RES_MOD_RSH 103
#define RES_MOD_DEFWIDTH 104
```

2.4.2 parameter tabel structuren

Deze tabellen worden aan de NGSpice simulator doorgegeven zodat deze weet welke parameters toegelaten zijn en welke data hij kan opvragen uit het model. Ook geeft het hierbij de afkorting, een beschrijving, een getal dat de parameter voorstelt(zie model en instantiebeschrijving) en een type mee. Deze tabellen worden ingevuld met macro's die gedefinieerd zijn in het devdefs.c bestand.

```
IFparm RESpTable[] = { /* parameters */
    IOPP(  "resistance",        RES_RESIST,
       IF_REAL,    "Resistance"),
...
    OP(    "i",             RES_CURRENT,
       IF_REAL,    "Current"),
    OP(    "p",             RES_POWER,
       IF_REAL,    "Power")
};

//devdefs.c bestand
IOP( )          ingave of uitleesparameter
IOPP( )         hoofdparameter van een model
IOPA( )         parameter gebruikt in AC-analyses
IOPAP( )        hoofdparameter gebruikt in AC-analyse
IOPAA( )        parameter enkel gebruikt in AC-analyses
```

```
IOPAAP( )       hoofdparameter enkel gebruikt in
   AC-analyses
IOPN( )         hoofdparameter enkel gebruikt in
   ruisanalyses
IOPR( )         maakt het mogelijk om een parameter 2
   namen te geven (2 regels)
....U( )        wordt niet getoond in het show commando
 // gebruikt voor gevoeligheidsanalyse
IOPX( )         parameter die niet gebruikt wordt bij
   gevoeligheidsanalyse
IOPQ( )         Indien deze parameter niet 0 is zal
   hieropvolgende parameter IOPZ gebruikt worden in
   gevoeligheidsanalyse
IOPZ( )         vorige IOPQ parameter moet 0 zijn voor
   gevoeligheidsanalyse
```

2.4.3 parameter ingave functies

O m te voorkomen dat de frontend(NGSpice simulatie) de interne structuren van de onderliggende modellen dient te kennen wordt gebruik gemaakt van 2 functies die de opgegeven parameters doorgeven aan het model of de instantie. Hieronder volgt een voorbeeld van de structuur van een instantie ingave functie. De model ingave functie is analoog aan de instantie maar maakt gebruik van parameters met syntax XX_MOD_PARAMETER.

```
int RESparam(int param, IFvalue *value, GENinstance
   *inst, IFvalue *select)
// Hierbij wordt select gebruikt voor het invoegen van
   data in een vector
{
    double scale;

    RESinstance *here = (RESinstance *)inst;

    NG_IGNORE(select);

        //zet de schaalfactor voor de parameter
           (k,meg,etc...)
    if (!cp_getvar("scale", CP_REAL, &scale))
        scale = 1;

        //afhankelijk van welke parameter wordt
           aangeleverd wordt deze weggeschreven naar de
```

```
            instantie.
    switch(param) {
    case RES_TEMP:
        here->REStemp = value->rValue + CONSTCtoK;
        here->REStempGiven = TRUE;
        break;
    case RES_DTEMP:
        here->RESdtemp = value->rValue;
        here->RESdtempGiven = TRUE;
        break;
...

    default:
        return(E_BADPARM);
    }
    return(OK);
}
```

2.4.4 parameter uitlees functies

Om nogmaals te vermijden dat de NGSpice simulator de interne structuur van modellen en instanties moet kennen wordt gebruik gemaakt van functies die de waarden van de parameters en variabelen kan uitlezen zodat de simulator deze kan gebruiken in zijn berekeningen.

2 gelijkaardige functies stellen de simulator in staat om data op te vragen uit de instantie of zijn model. Hierbij moet de code rekening houden met het feit dat alle data niet op elk moment beschikbaar kan zijn. Indien deze niet beschikbaar is moet men aanduiden dat een slechte parameter is opgevraagd.De CKTcircuit parameter geeft een pointer naar de circuitdata zodat men deze data kan gebruiken. Hieronder een voorbeeld van een uitleesfunctie voor een instantie.

```
int
RESask(CKTcircuit *ckt, GENinstance *inst, int which,
   IFvalue *value,
       IFvalue *select)
{
    RESinstance *fast = (RESinstance *)inst;
    double vr;
    double vi;
```

```
    double sr;
    double si;
    double vm;
    double workingzone;
    static char *msg = "Current␣and␣power␣not␣available␣
       for␣ac␣analysis";

    switch(which) {
    case RES_TEMP:
        value->rValue = fast->REStemp - CONSTCtoK;
        return(OK);
...
    case RES_QUEST_SENS_MAG:
        if (ckt->CKTsenInfo) {
            vr = *(ckt->CKTrhsOld + select->iValue + 1);
            vi = *(ckt->CKTirhsOld + select->iValue +
               1);
            vm = sqrt(vr*vr + vi*vi);
            if (vm == 0) {
                value->rValue = 0;
                return(OK);
            }
            sr =
               *(ckt->CKTsenInfo->SEN_RHS[select->iValue
               + 1] +
                   fast->RESsenParmNo);
            si =
               *(ckt->CKTsenInfo->SEN_iRHS[select->iValue
               + 1] +
                   fast->RESsenParmNo);
            value->rValue = (vr * sr + vi * si) / vm;
        }
        return(OK);
...
    default:
        return(E_BADPARM);
    }
    /* NOTREACHED */
}
```

2.4.5 opruim functies

D eze functies zijn niet strikt noodzakelijk voor de werking van de simulator maar zorgen ervoor dat indien modellen of instanties niet meer worden gebruikt deze hun geheugen vrij geven zodat het systeem niet met een tekort aan geheugen wordt gesteld.

E en eerste functie is de destroy functie die wordt aangeroepen indien het hele circuit wordt uitgeladen uit de simulator. Deze functie zal over alle modellen en instanties itereren en gebruikt geheugen terug beschikbaar stellen aan het besturingssysteem. Hierbij wordt niet gelet op afhankelijkheden tussen de verschillende instanties en modellen omdat deze functies wordt uitgevoerd om het volledige circuit te verwijderen. Hieronder een voorbeeld van een destroy functie.

```
void RESdestroy(GENmodel **inModel)
{
    RESmodel **model = (RESmodel **)inModel;
    RESinstance *here;
    RESinstance *prev = NULL;
    RESmodel *mod = *model;
    RESmodel *oldmod = NULL;

    for( ; mod ; mod = mod->RESnextModel) {
        if(oldmod) FREE(oldmod);
        oldmod = mod;
        prev = NULL;
        for(here = mod->RESinstances ; here ; here =
           here->RESnextInstance) {
            if(prev) FREE(prev);
            prev = here;
        }
        if(prev) FREE(prev);
    }
    if(oldmod) FREE(oldmod);
    *model = NULL;
}
```

E en tweede paar functies zijn de delete functies. Deze functies verwijderen respectievelijk een instantie of een model (met al zijn instanties) uit een circuitsimulatie en zorgen ervoor dat het geheugen wordt vrijgegeven. De

functie die een instantie verwijdert zal deze uit de gelinkte lijst verwijderen en ervoor zorgen dat de 2 omringende instanties aan elkaar gekoppeld worden. De model delete functie wordt hieronder weergegeven:

```
int RESmDelete(GENmodel **inModel, IFuid modname,
   GENmodel *kill)
{
    RESmodel **model = (RESmodel **)inModel;
    RESmodel *modfast = (RESmodel *)kill;
    RESinstance *here;
    RESinstance *prev = NULL;
    RESmodel **oldmod;
    oldmod = model;
    for( ; *model ; model = &((*model)->RESnextModel)) {
        if( (*model)->RESmodName == modname ||
                (modfast && *model == modfast) ) goto
                   delgot;
        oldmod = model;
    }
    return(E_NOMOD);

delgot:
    *oldmod = (*model)->RESnextModel; /* cut deleted
       device out of list */
    for(here = (*model)->RESinstances ; here ; here =
       here->RESnextInstance) {
        if(prev) FREE(prev);
        prev = here;
    }
    if(prev) FREE(prev);
    FREE(*model);
    return(OK);

}
```

2.4.6 simulatie functies

V olgende functies worden gebruikt door de simulator om het bepaalde component te simuleren. Deze functies beschrijven hoe de component zich gedraagt en hoe het moet worden opgezet.

setup

Deze functie zet de component klaar voor simulatie. Deze functie wordt uitgevoerd nadat de component zijn verbindingspunten verbonden zijn met de andere componenten en zijn parameters zijn ingesteld. In deze functie kunnen standaard waarden worden ingesteld die niet afhankelijk zijn van andere parameters of modellen. Ook wordt er geheugen gereserveerd voor de deelmatrixen zodat de simulatie kan worden versneld.

```
int RESsetup(SMPmatrix *matrix, GENmodel *inModel,
   CKTcircuit*ckt, int *state)
{
    RESmodel *model = (RESmodel *)inModel;
    RESinstance *here;

    NG_IGNORE(state);
    NG_IGNORE(ckt);

    for( ; model != NULL; model = model->RESnextModel )
       {

        if(!model->RESbv_maxGiven)
            model->RESbv_max = 1e99;

        for (here = model->RESinstances; here != NULL ;
                here=here->RESnextInstance) {

            if(!here->RESmGiven)
                here->RESm = 1.0;

            TSTALLOC(RESposPosptr, RESposNode,
               RESposNode);
            TSTALLOC(RESnegNegptr, RESnegNode,
               RESnegNode);
            TSTALLOC(RESposNegptr, RESposNode,
               RESnegNode);
            TSTALLOC(RESnegPosptr, RESnegNode,
               RESposNode);
        }
    }
    return(OK);
}
```

temperature

Deze functie wordt opgeroepen na de setup functie. Deze functie wordt verder opgeroepen bij elke wijziging aan een parameter van deze component alsook bij elke temperatuurwijziging in het circuit. Indien waarden van parameters niet zijn meegegeven moet de functie hiermee rekening houden. Hieronder een voorbeeld van de temperature functie:

```
int REStemp(GENmodel *inModel, CKTcircuit *ckt)

{
    RESmodel *model =  (RESmodel *)inModel;
    RESinstance *here;
    double factor;

    for( ; model != NULL; model = model->RESnextModel )
       {

        if(!model->REStnomGiven) model->REStnom
           = ckt->CKTnomTemp;
        if(!model->RESsheetResGiven) model->RESsheetRes
           = 0.0;
...
        for (here = model->RESinstances; here != NULL ;
                here=here->RESnextInstance) {

           if(!here->REStempGiven) {
                here->REStemp   = ckt->CKTtemp;
                if(!here->RESdtempGiven)
                   here->RESdtemp  = 0.0;
            } else { /* REStempGiven */
                here->RESdtemp = 0.0;
                if (here->RESdtempGiven)
                    printf("%s: Instance temperature
                       specified, dtemp ignored\n",
                       here->RESname);
            }

            if(!here->RESwidthGiven)   here->RESwidth
               = model->RESdefWidth;
   ...

            difference = (here->REStemp +
               here->RESdtemp) - model->REStnom;
```

```
    ...
              here -> RESconduct = (1.0/(here->RESresist
                 * factor * here->RESscale));
          }
      }
      return(OK);
}
```

load

Deze functie wordt bij elke iteratie opgeroepen en is verantwoordelijk voor het evalueren van elke instantie in AC of DC simulatie. Hierbij wordt gecontroleerd of er conversie optreedt of niet. In NGSpice wordt deze functie opgesplitst in een AC en een DC load functie.

```
int RESload(GENmodel *inModel, CKTcircuit *ckt)
{
    RESmodel *model = (RESmodel *)inModel;
    double m;

    for( ; model != NULL; model = model->RESnextModel )
       {
        RESinstance *here;
        for (here = model->RESinstances; here != NULL ;
                here = here->RESnextInstance) {

            here->REScurrent =
               (*(ckt->CKTrhsOld+here->RESposNode) -
                                *(ckt->CKTrhsOld+here->RESnegNode))
                   * here->RESconduct;

            m = (here->RESm);
            *(here->RESposPosptr) += m *
               here->RESconduct;
...             }
    }
    return(OK);
}

int RESacload(GENmodel *inModel, CKTcircuit *ckt)
{
    RESmodel *model = (RESmodel *)inModel;
    double m;
```

```
    NG_IGNORE(ckt);
    for( ; model != NULL; model = model->RESnextModel )
       {
        RESinstance *here;
        for (here = model->RESinstances; here != NULL ;
             here = here->RESnextInstance) {

            m = (here->RESm);
            if(here->RESacresGiven) {
                *(here->RESposPosptr) += m *
                   here->RESacConduct;
...
            } else {
                *(here->RESposPosptr) += m *
                   here->RESconduct;
...                     }
        }
    }
    return(OK);
}
```

Hoofdstuk 3

GTK3

D e GTK3-toolkit [19] is een grafische interface toolkit die oorspronkelijk ontwikkeld is voor GIMP. GIMP is een beeldverwerker die mogelijkheden biedt die vergelijkbaar zijn met Adobe Photoshop.

D e GTK3-toolkit is een object georiënteerde toolkit die geschreven is in de C programmeertaal. De objectgeoriënteerde implementatie draagt de naam Glib object systeem en zorgt ervoor dat men objectgeoriënteerd kan werken met de GTK3-toolkit. Deze toolkit wordt voornamelijk gebruikt op Linux op de Xorg displayserver. Door gebruik te maken van de GDK bibliotheek kan deze toolkit echter ook gebruikt worden op Windows, BSD, Unix, en Mac OS.

S inds versie 3.2 van deze toolkit is deze ook in staat om via de zogenaamde Broadway GDK backend applicaties te laten draaien in een HTML5 browser. Dit maakt het mogelijk om GTK3 applicaties op een server te draaien en op elke toestel met een webbrowser hiermee interacties uit te voeren. In volgende secties worden de verschillende nodige bibliotheken voor de GTK3 toolkit besproken.

3.1 GLib

D e GLib is een algemene bibliotheek die niet enkel beperkt is tot het gebruik met grafische toolkits. De GLib bibliotheek implementeert onder andere volgende onderdelen:

- object en type systeem (hoofdlus)
- dynamisch laden van modules of plug-ins
- draad ondersteuning en gedraade queues

— timer ondersteuning
— geheugen beheerder
— collecties : lijsten, hash tafels, reeksen, strings en tekenset
— XML parser

3.2 GObject

D eze bibliotheek voorziet GTK3 van een generiek en dynamisch type systeem genaamd Gtype. Ook voorziet de bibliotheek in enkele fundamentele types zoals een leeg type (G_TYPE_NONE), types verwijzend naar long,int,char,bool,float,string,pointers en enumeraties (alsook vlaggen).

V erder voorziet de bibliotheek in enkele object-georiënteerde fundamentele types zoals het GObject dat de wortel is van de klassen-hiërarchie in GLib. Ook voorziet het in een basis interface type en een type voor metadata voor object eigenschappen.

V oor inter-object communicatie voorziet het GObject systeem in Closures (callback) en signalen.

3.3 Gio

D e Gio bibliotheek en bijhorende interface (API) maakt het mogelijk om te communiceren met het GVFS (Gnome virtueel bestandssysteem). Toevoegend wordt het ook gebruikt voor netwerk en DBus communicatie. Deze bibliotheek is een onderdeel van de GLib bibliotheek.

G VFS is een virtueel bestandssysteem dat de verschillende bestandssystemen in kaart brengt door middel van de GVFS-daemon. Deze is verantwoordelijk voor het correct koppelen en ontkoppelen van bestandssystemen (externe harde schijf,usb, cd-rom,etc). Deze bibliotheek gebruikt intern de tweede versie van udisks. GVFS heeft verscheidene backends die elk een type van bestandssysteem ondersteunen zoals:

trash: deze backend zorgt voor de werking van een prullenbak. Dit maakt het mogelijk om bestanden te herstellen na het verwijderen of om bestanden tijdelijk te verwijderen om later te beslissen om ze volledig te verwijderen

afc: biedt ondersteuning voor verwijderbare media zoals usb sticks en camera's.

smb: maakt het mogelijk om gedeelde Windows mappen en printers aan te spreken.

afp: ondersteuning voor het Apple Filing Protocol.

goa: ondersteuning voor de integratie met Gnome Online Accounts. Dit zorgt ervoor dat men 1 maal voor elk account de gegevens ingeeft en hierna elke applicatie gebruik kan maken van deze webservices.

mtp: ondersteuning voor het Media Transfer Protocol. Dit is een protocol waarbij men werkt op bestandsniveau voor het oversturen van bestanden. Dit gebeurt in atomische operaties.

fuse: ondersteuning voor het verbinden met bestandssystemen aangemaakt door applicaties of als niet root gebruiker, door middel van de Filesystem In Userspace bibliotheek.

3.4 Pango

P ango is een bibliotheek voor het behandelen van internationaliseren van de applicatie. Dit zorgt ervoor dat alle tekstvelden en labels kunnen vertaald worden zonder dat het programma herschreven dient te worden.

3.5 ATK

A TK staat voor de Accessibility Toolkit. Dit is een bibliotheek die sinds GTK3 ingebouwd zit in alle standaard widgets. Deze bibliotheek maakt het mogelijk om applicaties toegankelijk te maken voor mensen met een beperking. Zo kan een schermvoorlezer de tekst voorlezen uit een GTK3 interface zodat de gebruiker hierop kan reageren of naar de krant kan luisteren.

3.6 GdkPixbuf

D it is een kleine bibliotheek die het mogelijk maakt om GDK pixel buffer objecten aan te maken vanuit een foto-bestand. Dit maakt het mogelijk om foto's in de laden in een GTK3 programma.

3.7 GDK en Cairo

G DK of GIMP Drawing Kit is een abstractielaag die ervoor zorgt dat de GTK3 toolkit op verschillende vensterbeheer software kan draaien. In

Linux is GDK gelegen tussen de X-server en de GTK3 toolkit en verzorgt de verwerking van elementaire rendering zoals het tekenen van primitieven, bitmaps, cursors en lettertypen. Ook voorziet de GDK in drag en drop functionaliteit.

C airo [4] is een grafische bibliotheek voor het werken met 2-dimensionale composiet afbeeldingen alsook vector afbeeldingen. Deze bibliotheek maakt het mogelijk om met dezelfde code op verschillende systemen en naar verschillende bestandsformaten dezelfde afbeelding te bekomen. Dit wordt mogelijk gemaakt omdat de Cairo bibliotheek bestaat uit vele verschillende backends (dit zijn de surfaces), namelijk:

image: deze backend is gericht op het werken met een afbeelding buffer in het geheugen. Deze buffer kan dan opgeslagen worden in een bestand of kan aangereikt worden aan een ander grafisch systeem waarvoor Cairo geen ingebouwde backend heeft.

Cairo-gl: deze backend maakt gebruikt van hardwarematige versnelling (grafische kaart) voor het renderen en tonen van de afbeeldingen via OpenGL. Hierbij worden GLX, WGL en EGL ondersteund.

ps: backend voor het aanmaken van een PostScript bestand voor hoge kwaliteit- afdrukken (printers).

pdf: backend voor het aanmaken gevectorizeerde pdf voor hoge kwaliteit-afdrukken (printers).

xlib: deze backend wordt gebruikt voor de communicatie met het Xorg vensterbeheer systeem. Deze backend kan hierbij gericht worden op vensters of op afbeeldingen binnen de Xorg-server.

xcb: gelijkaardig aan bovenstaande maar gebruikt de XCB interface in plaats van de xlib interface.

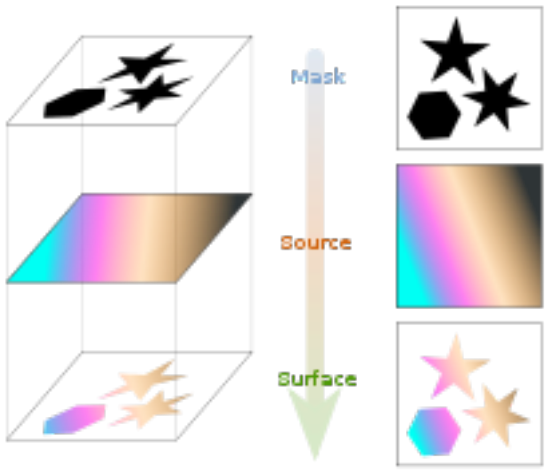
quartz: Mac OS/X is mogelijk door deze backend.

win32: backend voor Microsoft Windows die gebruik maakt van de Windows grafische apparaat interface (GDI). Hierbij communiceren de applicaties via GDI met de hardwaredrivers.

svg: backend voor het aanmaken van Scalable Vector Graphics bestanden.

beos: BeOS/Zeta backend. Zeta is een opensource besturingssysteem en de opvolger van het gesloten BeOS.

D e Cairo bibliotheek heeft een andere manier van renderen dan bijvoorbeeld SVG. Hierbij wordt er eerst een afbeelding als bron gekozen. Nu zal men een masker definiëren over deze bron. Dit masker wordt dan door de bron gedrukt waarna men het resultaat bekomt op het afbeeldingsoppervlak. Voor het drukken kan men zowel het hele masker vullen (oppervlakken) of het masker met een lijndikte doordrukken (bv tekst). Het resultaat hiervan wordt dan overgebracht naar de juist backend die het dan toont/afdrukt [31].



Figuur 3.1 – Cairo teken methode

D e GTK3 bibliotheek gebruikt voor het tekenen van de widgets de Cairo bibliotheek. Hierbij wordt er geen extra laag rond de Cairo API gebruikt maar zorgt GTK3 ervoor dat Cairo contexts gebruikt kunnen worden voor het tekenen op GDK.

G DK zorgt er dus (in samenwerking met Cairo) voor dat GTK3-applicaties kunnen gedraaid worden op verschillende platformen door het gebruik van verschillende GDK backends. Er is een Xlib backend voor het gebruik in Linux, een win32 backend voor het gebruik in Windows en sinds versie 3.2 een Broadway backend die het mogelijk maakt om de applicatie te draaien in een HTML 5 canvas. Ook is er een quartz backend voor Apple Os en een wayland backend voor de opvolger van de Xorg-Server, Wayland.

3.8 Broadway GDK backend

D e Broadway backend [18] zorgt ervoor dat alle applicaties die gebruik maken van GTK3+ versie 3.2 of hoger kunnen worden weergegeven en bestuurd worden in een browser die HTML 5 canvas en websockets ondersteund. Dit wordt mogelijk gemaakt doordat de Broadway daemon een webserver aanmaakt die een HTML-pagina toont die enkel een canvas bevat. Verder bevat het ook een javascript node Broadway.js die verbinding maakt met de Broadway backend die op de server draait via een websocket. De invoer wordt via deze socket naar de backend gestuurd en de gedeelten van het canvas die hertekend moeten worden, worden verstuurd van de backend naar de website.

V oor het gebruik van een applicatie in een webbrowser zet men eerst een Broadway server daemon op waarna men het gewenste programma start.

```
broadwayd :5
BROADWAY_DISPLAY=:5 gtk3-demo
```

D aarna kan men via de webbrowser naar het volgende adres surfen.

`http://127.0.0.1:8084`

D e poort kan men instellen bij het laden van de Broadwayd en is standaard 8080+(dispay nummer)-1. Men kan sinds versie 3.10 deze server ook beveiligen door middel van de hash waarde van het paswoord weg te schrijven naar een configuratie bestand met volgend commando.

```
openssl passwd -1  > ~/.config/Broadway.passwd
```

Hoofdstuk 4

Vala

D eze programmeertaal lijkt zeer sterk op C Sharp en heeft gelijkwaardige functionaliteiten. Deze taal heeft echter geen nieuwe runtime omgeving nodig zoals C Sharp de .net omgeving nodig heeft. Dit komt omdat de Vala compiler de Vala code compileert naar C code die door een C compiler kan worden gecompileerd.

E en groot voordeel van deze taal is dat men geen rekening moet houden met geheugenbeheer (free en malloc in C) omdat door middel van reference counting de objecten automatisch worden vrijgegeven indien ze niet meer vereist zijn voor het programma. Deze taal laat ook toe om bibliotheken in C te gebruiken door middel van VAPI bestanden. Deze zorgen voor een interface tussen Vala en de C bibliotheken. Vala is net zoals C Sharp hoofdlettergevoelig. De Vala code wordt opgeslagen in bestanden met de .Vala extensie waarna de Vala compiler deze omzet naar C code. Vala kan niet meer dan mogelijk is met de C taal en Glib maar zorgt ervoor dat het schrijven van code veel overzichtelijker en gemakkelijker wordt.

D e compiler voor Vala wordt valac genoemd.

```
valac source1.Vala source2.Vala --pkg gtk+-3.0 -o
   program
```

H ierbij worden 2 source bestanden gecompileerd en wordt de GTK 3.0 bibliotheek gebruikt.

4.1 Vergelijking met C#

I n deze sectie worden de verschillen en gelijkenissen tussen c# en Vala uitgelegd [27].

4.1.1 Methoden en constructors

V ala kent in tegenstelling tot C# geen overladen van methodes , in Vala dient men andere namen te gebruiken of gebruik te maken van standaard waarden die steeds op het einde van de argumentenlijst moeten staan.

```
class Demo : Object {

    public void draw_text (string text) {
    }

    public void draw_shape (Shape shape) {
    }
    void f (int x, string s = "hello", double z = 0.5) {
    }
}
```

N et zoals bij methoden moeten constructeurs in Vala steeds een andere naam hebben indien ze andere argumenten aannemen. Voor het aanroepen van de constructeur van een superklasse maakt men gebruik van de syntax base(arg,...). Dit kan men uitbreiden naar base.NAAM (arg,...) waarbij de specifieke constructor wordt opgeroepen. Men kan binnen dezelfde klasse andere constructors aanroepen door this te gebruiken.

```
class Foo : Bar {
    public Foo () {
        base (42);
        // ...
    }
}
```

E en statisch constructor wordt aangeroepen wanneer de eerste klasse of een subklasse ervan wordt uitgevoerd.

```
class Foo : Object {
    static construct {
        stdout.printf ("Static␣constructor␣invoked.\n");
    }
```

```
}
```

I ndien men echter een constructor wilt gebruiken die bij het uitvoeren van de eerste klasse , alsook bij het elke eerste subklasse, wordt uitgevoerd kan men gebruik maken van "class construct"blok.

4.1.2 Signalisatie

H iervoor maakt Vala gebruik van signalen en connectors voor deze signalen.

```
class Foo {
    public signal void some_event (int i);
}

class Demo {
    static void on_some_event (Foo sender, int i) {
        stdout.printf ("Handler␣A:␣%d\n", i);
    }

    static void main () {
        var foo = new Foo ();
        foo.some_event.connect (on_some_event);
        foo.some_event (42);
        foo.some_event.disconnect (on_some_event);
    }
}
```

De stappen voor signalisatie zijn:
— Definieer een signaal in een klasse
— Maak een methode die zal worden uitgevoerd
— Verbindt het signaal met de methode door connect
— Verstuur een signaal
— Verbreek de verbinding

D e teruggeefwaarde en de argumenten van de methode dienen overeen te komen met deze van het signaal. Dit zorgt ervoor dat men methodes van een klasse kan uitvoeren zonder dat men een verwijzing naar die klasse moet bijhouden.

4.1.3 Interfaces

```
interface IfaceA : Object {
    public abstract void method_a ();
}

interface IfaceB : Object, IfaceA {
    public abstract void method_b ();
}

class Demo : Object, IfaceA, IfaceB {
    public void method_a () { }
    public void method_b () { }
}
```

B ij Vala interfaces is het nodig te specificeren dat het om publieke en abstracte methoden gaat zodat men kan spreken over een interface. Dit komt omdat Vala ook toelaat dat interfaces methodes bevatten die een implementatie bevatten en zelf privé methoden. In Vala kan een interface niet gebaseerd zijn op een interface maar een interface kan wel vermelden dat deze enkel kan geïmplementeerd worden indien ook de andere interface wordt geïmplementeerd.

4.1.4 Enumeraties en structuren

```
enum Season {
    SPRING, SUMMER, AUTUMN, WINTER;

    public bool is_hot () {
        return this == SUMMER;
    }
}

var p1 = Point ();
var p2 = Point () { x = 2, y = 3 };
Point p3 = { 2, 3 };
```

I n Vala kunnen enumeraties zelf methoden bevatten. Structuren worden geïnitialiseerd zonder de new operator.

4.1.5 null en properties

A lle referenties naar objecten kunnen niet null bevatten aangezien ze worden omgezet naar C code. Door een speciale syntax wordt aangeduid dat deze mogelijk null worden.

```
Foo? method (Foo? foo, Bar bar) {
    return null;
}
```

B ij properties is het mogelijk de volgende syntax te gebruiken om meteen een default waarde mee te geven aan de constructor.

```
class Person : Object {
    public string name { get; set; default = "Default␣
       Name"; }
}
```

W anneer een property van een Object (of afgeleide klasse) verandert zal deze een notify signaal uitsturen , deze kan worden verbonden met een een methode die op basis hiervan een bepaalde actie onderneemt.

```
person.notify["name"].connect (Methode)
person.notify.connect (Methode)
```

I ndien de property '_' bevat moeten deze vervangen worden door '-'. Men kan ook vermijden dat een property een signaal verzendt door deze vooraf te gaan met een Ccode attribuut.

```
    [CCode (notify = false)]
    public int without_notification { get; set; }
```

4.1.6 Exceptions

E xceptions in Vala zijn niet klasse gebaseerd maar maken gebruik van een errordomain struct. Een methode die mogelijk een error genereert moet deze struct opgooien.

```
errordomain MyError {
    FOO,
    BAR
}

void method () throws MyError {
    // error domain, error code, error message
    throw new MyError.FOO ("not␣enough␣foo");
}
```

```
try {
    method ();
} catch (MyError e) {
    stderr.printf ("Error: %s\n", e.message);
}
```

D e compiler kan zonder try catch Blok compileren maar zal hierover waarschuwingen geven. Dit maakt prototypen mogelijk en zorgt ervoor dat lege (of writeline) in een catch blok minder vaak voorkomen.

4.1.7 Asynchrone oproepen

```
class AsyncDemo {

    static async int adder (int a, int b) {
        return a + b;
    }

    static async void start () {
        int sum = yield adder (4, 5);
        stdout.printf ("Addition completed\n");
        stdout.printf ("Result was: %d\n", sum);
    }

    static void main () {
        start ();

        /* wait */
        var loop = new MainLoop (null, false);
        loop.run ();
    }
}
```

O ndersteuning voor asynchrone methoden is ingebouwd in Vala indien bibliotheek gio-2.0 aan de compiler wordt toegevoegd.

4.1.8 Verzamelingen en indexering

V erzamelingen zijn geïmplementeerd in de Gee bibliotheek en men dient voor het gebruik het volgende argument mee te geven aan de compiler --pkg gee-0.8". Vala ondersteund indexers waarbij de methoden moeten gedefini-

eerd worden. Vala ondersteund ook de in operator voor verzamelingen en strings.

```
class SampleCollection<T> {

    private T[] arr = new T[100];

    public T get (int i) {
        return arr[i];
    }

    public void set (int i, T item) {
        arr[i] = item;
    }
}

void main (string[] args) {
    var string_collection = new
       SampleCollection<string> ();

    string_collection[0] = "Hello,World";
    stdout.printf ("%s\n", string_collection[0]);
}
```

4.1.9 Varia

I n tegenstelling tot C# mag de main methode zich ook buiten een klasse bevinden. Verder wordt in Vala steeds de Glib bibliotheek gebruikt.

Documentatie en console invoer en uitvoer is hetzelfde als in C#.

D e standaard types zoals int en long zijn afhankelijk van de architectuur. Verder heeft Vala ook (u)int{8,26,32,64} types die onafhankelijk van de architectuur zijn. Vala kent echter geen byte, sbyte of decimalen.

V ala ondersteund ook verbatim strings. Verbatim strings zijn strings waarvan de inhoud wordt beschouwd als de exacte inhoud (hierbij moeten speciale tekens niet worden ontlopen).

```
"""verbatimstring"""
```

K lassen , structuren, en gedelegeerde types worden in CamelCase geschreven„ methoden, eigenschappen, signalen, lokale variabelen , velden worden

in lower_case geschreven. De syntax voor constanten en waarden van enumeraties is UPPER_CASE.

4.2 VAPI bestanden

Een VAPI bestand staat voor een vala publieke interface en beschrijft de API van de bibliotheek. Bij het compileren van aan vala-bibliotheek wordt dit bestand gegenereerd uit de gecompileerde op Gobject gebaseerde C-bibliotheek.

D it bestand wordt gebruikt zodat het mogelijk is voor de Vala compiler om rechtstreeks de vala code om te zetten in overeenkomstige C-code. De VAPI bestanden worden dus niet gebruikt tijdens het draaien van het programma maar enkel bij het compileren ervan.

D e eerste stap bij het genereren van dit bestand is de vala-gen-introspect tool die de metadata uit de c-code bestanden opvraagt. Dit gebeurt door geannoteerde comment in de c-code van de bibliotheek. Deze tool maakt hiermee een GObject introspectie bestand aan. GObject introspectie is een tussenlaag tussen C bibliotheken en bindingen met verscheidene programmeertalen waaronder JavaScript, Java en Python. Dit zorgt ervoor dat men het gi bestand kan aanmaken bij compilatie van de C bibliotheek.

D eze eerste stap wordt momenteel vervangen door het integreren van Gobject introspectie in autotools. Hiervoor dient deze software op het systeem geïnstalleerd te zijn en dient men in de Makefile.am en de Configure.Ac 1 regel toe te voegen. Dit zorgt ervoor dat gecontroleerd wordt of GObject introspectie op het systeem aanwezig is en geeft de –enable-introspection vlag mee.Dit bestand wordt dan een gir bestand genoemd.

E en tweede optionele stap bestaat eruit om eventuele ontbrekende metadata toe te voegen aan het .gi(r) bestand. Ook kan men indien de bibliotheek niet voldoet aan de standaard interface die gewenst is hierbij verschillende wijzigingen aanbrengen. Dit dient echter vermeden worden indien mogelijk door het voorzien van een correcte interface met annotaties in de GObject gebaseerde bibliotheek.

D e laatste stap is de generatie van het VAPI bestand vertrekkende van het gi(r) bestand. Dit gebeurt met de vapigen tool.

A angezien de gebruikte NGSpice bibliotheek echter niet gebaseerd is op het GObject type systeem is de enige optie voor het gebruik ervan in vala om het VAPI bestand manueel te schrijven.

O mdat Vala meer semantisch expressief is en het in C niet steeds duidelijk is wat de variabelen of functies juist betekenen is het schrijven van een VAPI bestand enkel mogelijk indien men kan achterhalen wat de intentie van de gebruikte code is.
Zo kan bijvoorbeeld char* in C programmeertaal verscheidene betekenissen hebben zoals pointer, array, string. Ook weet men hierbij niet of deze variabele zal aangepast of teruggegeven worden door een functie.

H ierbij is het op te merken dat het gebruik van pointers in Vala wordt afgeraden omdat hierop geen type controle kan worden gedaan bij compilatie.

D e opmaak en syntax van het VAPI bestand worden besproken bij de bespreking van het NGSPICE VAPI bestand.

Hoofdstuk 5

Toolchain

In dit hoofdstuk worden de verschillende programma's en tools uitgelegd die het mogelijk maken om dit programma te schrijven, te compileren en te debuggen.

5.1 Valama IDE

I nitieel is het project ontwikkeld in de Anjuta IDE. Dit is de ontwikkelomgeving van Gnome waarbij alle programmeertalen en bijhorende tools geïntegreerd zijn met als doel 1 complete IDE voor het Gnome platform te realiseren.
Deze IDE gebruikt autotools[5] maar maakt hierbij gebruik van recursieve makefile(.am) bestanden. Zoals aangegeven in paper [28] lijd het gebruik van recursieve makefile bestanden tot complexe onderhouds- en debugomstandigheden. Omdat de Valama IDE hier geen gebruikt maakt van recursieve makefile bestanden maar alles onderbrengt in 1 makefile bestand ben ik overgeschakeld op deze IDE.

D e valama IDE is een ontwikkelomgeving geschreven in Vala voor het ontwikkelen van projecten geschreven in de Vala programmeertaal. Deze IDE beschikt over autocomplete door middel van de Guanako bibliotheek. Verder kan men in deze IDE zowel projecten maken met de Cmake tool alsook met de GNU autotools. Deze IDE is nog volop in ontwikkeling maar voorziet in alle basisbehoeften van een IDE. Als toevoeging hierop heb ik de Nemiver debugger gebruikt voor het debuggen van de code. Dit is een grafische interface die gebruik maakt van de GDB.

5.2 Autotools

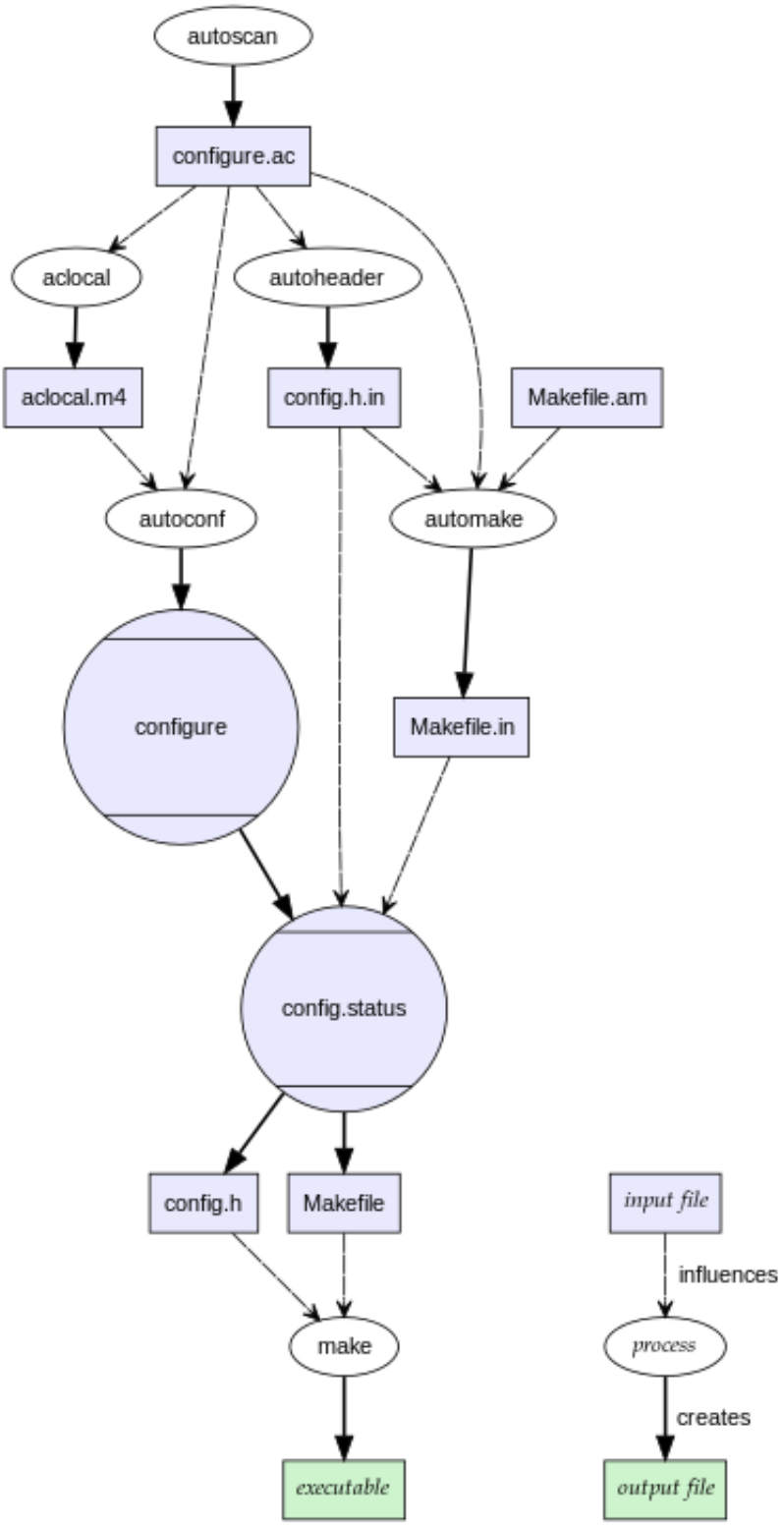
het GNU build system [4] of Autotools is een pakket van programma's waarmee men vele verschillende soorten programma's en bibliotheken in vele programmeertalen kan laten compileren. De grootste moeilijkheid hierin is de mogelijk te ondersteunen om het programma of de bibliotheek op vele verschillende platformen vertrekkende vanuit de broncode te kunnen compileren. De 3 belangrijkste componenten van dit pakket zijn autoconf, automake en libtool.

H et programma is zo ontwikkeld dat een gebruiker van het programma (die het wil compileren en installeren) geen beschikking hoeft te hebben tot Autotools. Dit zorgt ervoor dat men shell script moet kunnen draaien en een make variant dient te hebben op het systeem van de eindgebruiker. Voor de eindgebruiker volstaat het dan om de bronbestanden uit te pakken en het configure script uit te voeren. Hierdoor wordt dan een makefile aangemaakt waarna de gebruiker de gekende make commando kan gebruiken voor het installeren van het programma of bibliotheek.

5.2.1 autoconf

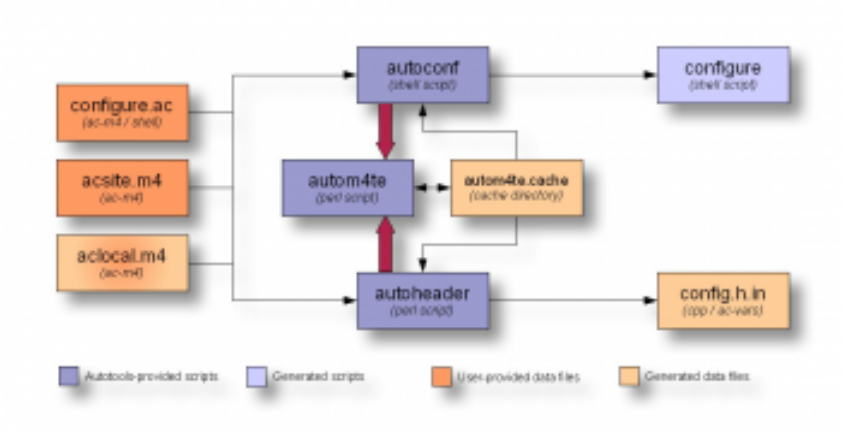
Autoconf is ontstaan in 1991 omdat de configuratie shellscripts (die er onder andere voor zorgden dat het programma op meerdere platformen kon worden gecompileerd) enorm groot en complex werden. Men heeft toen ook opgemerkt dat men in dit script slechts enkele variabelen moest specificeren. Dit heeft geleid tot het ontwikkelen van een softwarepakket dat in staat is vanuit klein configuratiebestand dit complexe shellscript te genereren. Dit zorgt ervoor dat men minder kans heeft op fouten en zorgt er ook voor dat men enkel het configuratiebestand dient te controleren. Het pakket bestaat verder uit enkele Perl scripts die gebruikt worden doorheen verschillende onderdelen van het Autotools software pakket.

D e werking van autoconf kan worden uitgelegd aan de hand van onderstaand schema. Voor het genereren van een configure script zal men eerst het configure.ac bestand configureren. Een grotendeels correct voorbeeld kan gegeneerd worden door het autoscan script. Hierna kan men het Autoconf script draaien en deze zal dan het vertrekkende van het configure.ac specificatiebestand het configure script genereren. Het autom4te script zorgt voor het aanmaken van een cache. Omdat het configure.ac bestand bestaat uit M4 macro's die bij verwerking door de verschillende scripts worden omgezet in vele lijnen shell code worden deze conversies opgeslagen in deze cache.



Figuur 5.1 – autoconf en automake

Figuur 5.2 – schema



Figuur 5.3 – autoconf en autoheader

De 2 M4 bestanden bevatten de M4 macro's die kunnen gebruikt worden in het configure.ac bestand.

5.2.2 automake

D eze toevoeging [2] zorgt ervoor dat men bij elke project de standaard make functies zoals install, all en vele anderen beschikbaar heeft nadat men het configure script heeft gedraaid. Dit wordt geïmplementeerd door het schrijven van een Makefile.am bestand. In dit bestand kan men ook gebruik maken van vele M4 macro's die het schrijven van de initiële makefile vergemakkelijken. Men hier echter ook een traditioneel makefile bestand invoegen. Deze wordt dan door automake doorgegeven aan de Makefile.in.

D oor het gebruik van macro's kan deze invoer voor automake veel compacter worden opgesteld. De uitvoer van dit script is dan het Makefile.in bestand dat door de configure stap wordt verwerkt tot een traditionele makefile.

5.2.3 libtool

A angezien bij vele platformen de naamgeving en functionaliteiten voor het inladen van bibliotheken verschillen dient men hiervoor een programma te gebruiken die ervoor zorgt dat deze verschillen geen problemen geven. Hiervoor voorziet de libtool enkele M4 macro's die gebruikt kunnen worden in de makefile. Libtoolize is een shellscript die een libtool script aanmaakt voor het project. Dit libtool script wordt dan gebruikt door automake gegenereerde makefiles en wordt ltmain.sh genaamd.

5.2.4 autoreconf

A utoreconf is een Perl script dat het gebruiken van voorgaande tool vergemakkelijkt. Deze tool zal namelijk aan de hand van aanpassingstijden van bestanden en de status van het project de juiste voorgaande programma's draaien zodat het project wordt klaargemaakt om te distributieheren. Hierbij maakt het het project klaar om met het configure en make tools het project te bouwen en te installeren.

5.2.5 configure en make

D e vorige tools (met autoreconf als binder) zorgen ervoor dat met het configure script en het Makefile.inf bestand worden aangemaakt. Bij het uitvoeren van het configure script maakt deze een config.status shell script aan aan de hand van de platform specifieke karakteristieken en de beschikbare tools en softwarepakketten. Dit nieuw gegenereerde shell script bevat alle informatie voor het aanmaken van de vereiste bestanden en programma's (ook deze gespecificeerd in het configure.ac bestand. Dit script gegeneerd bij het uitvoeren vertrekkende van de Makefile.in en de header file de uiteindelijke config en makefile voor het project.

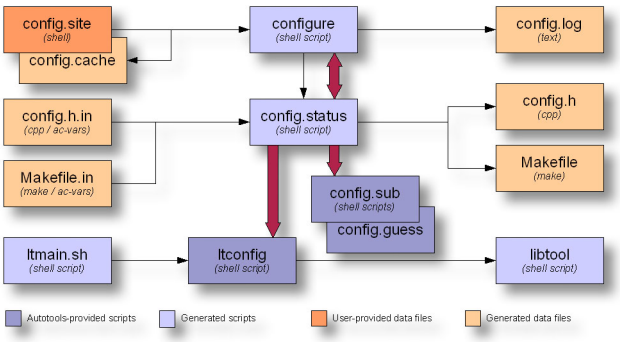
V erder wordt indien aangegeven in het configure.ac bestand ook het ltmain shell script verwerkt waaruit het libtool shell script wordt gevormd. Dit wordt later opgeroepen door de make tool [16] bij het verwerken van de Makefile.

E en ander voordeel is dat indien men de makefile of config header wilt regenereren met de configure opties die meegegeven zijn aan het configure script kan men simpelweg het config.status script aanroepen.

D e laatste stap voor het compileren en installeren van het project is het gebruik van de make tool. Hierbij worden de Makefile, de config.h en de bronbestanden gebruikt voor compilatie van het project.

5.3 configure.ac en makefile.am

In deze sectie worden de 2 configuratie bestanden voor autotools uitgelegd zoals gebruikt in het project.



Figuur 5.4 – het configure commando

5.3.1 configure.ac

In dit bestand wordt de configuratie opties voor autoconf en automake gespecificeerd. Deze voor autoconf beginnen met AC_ en die voor automake met AM_. De LT_ macros zijn voor het definiëren van de libtool macro's. Deze M4 macro's worden rechtstreeks omgezet tot grote stukken shell script. Dit is gelijkaardig aan definities bij de C-preprocessor.

```
# definieert de naam van het project en de versie, een
   derde argument kan worden toegevoegd zodat een
   e-mailadres kan worden toegevoegd.
AC_INIT([elektrosim], [0.0])

# zorgt ervoor dat gecontroleerd wordt of de header kan
   worden gecompileerd.
AC_CONFIG_HEADERS([config.h])

# dit voegt de M4 macros uit de M4 map toe
AC_CONFIG_MACRO_DIR([m4])

# dit initieert automake functionaliteit
# check-news zorgt ervoor dat make dist faalt als
   bovenaan het NEWS bestand geen huidig versienummer
   detecteert
# dist-bzip2 zorgt ervoor dat na make dist ook
   dist-bzip2 wordt gedraaid om het distributiepakket
   te comprimeren
# subdir-objects zorgt ervoor dat source code wordt
   gecompileerd naar dezelfde map als het bronbestand.
```

```
   Dit maakt het mogelijk om recursieve makefile.am te
   vermijden.
AM_INIT_AUTOMAKE([check-news dist-bzip2 subdir-objects])
m4_ifdef([AM_SILENT_RULES], [AM_SILENT_RULES([yes])])

#voegt ondersteuning toe voor archiveren door onder
   andere Microsoft bil
AM_PROG_AR

# initieert en activeert de libtool integratie
LT_INIT

# controle van beschikbare c compiler
AC_PROG_CC

# gebruik de vala compiler in plaats van de c compiler
AM_PROG_VALAC

# controleert of de pakketten nodig voor elektroSim
   beschikbaar zijn op het systeem via pkg-cfg
PKG_CHECK_MODULES(ELEKTROSIM, [gee-1.0 gio-2.0 glib-2.0
   gobject-2.0 librsvg-2.0 ngspice ])

#defineert een variabele die gebruikt kan worden in
   elke makefile.in
AC_SUBST(ELEKTROSIM_CFLAGS)
AC_SUBST(ELEKTROSIM_LIBS)

# definieert de te genereren makefile (of meerdere)
   toevoeging van .am is niet nodig.
AC_CONFIG_FILES([Makefile])

# zorgt dat automake de bestanden aanmaakt
AC_OUTPUT
```

D e macro AC_CONFIG_AUX_DIR([aux]) kan worden toegevoegd na de AC_INIT macro zodat de meeste hulp en gegenereerde bestanden in deze map worden opgeslagen. Dit zorgt ervoor dat de hoofdmap overzichtelijker blijft.

5.3.2 makefile.am

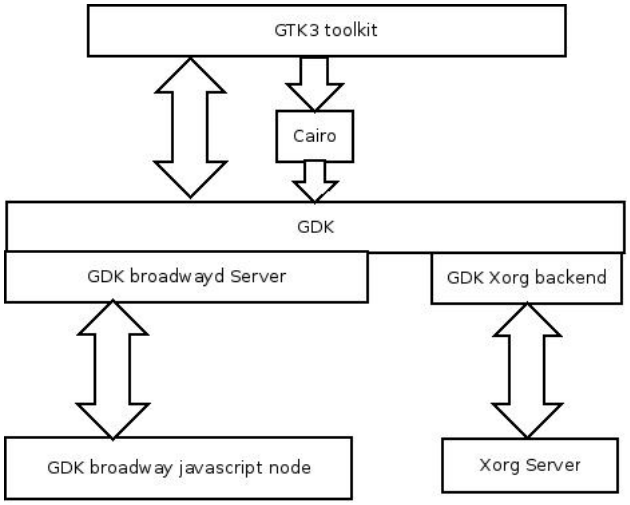
```
AM_CFLAGS = $(ELEKTROSIM_CFLAGS)

#het gedeelte voor de \_ definieert de installatie
   locatie (hier bin). Het gedeelte het tweede deel
   definieert het product type(hier programma). Hierbij
   wordt aangegeven dat het elektrosim uitvoerbaar
   programma zal worden geinstalleerd door make install.
bin_PROGRAMS = elektrosim

#alle volgende macro worden begonnen met elektrosim\_ ,
   dit wijst erop dat dit de opties zijn voor het
   bouwen van het elektrosim doel

# hierbij worden de bibliotheken aangegeven die door de
   linker moeten worden gebruikt
elektrosim_LDADD = $(ELEKTROSIM_LIBS)

# hier worden de bronbestanden definieert voor het
   elektrosim doel
elektrosim_SOURCES = src/Line.vala
   src/NGSpiceSimulator.vala src/XYGraph.vala
   src/capicitor.vala src/component.vala
   src/componentlist.vala src/elektrosim.vala
   src/ground.vala src/inductor.vala src/parameter.vala
   src/point.vala src/power-source.vala
   src/resistor.vala src/simulation.vala
   src/simulationArea.vala vapi/config.vapi

# hier worden de vlaggen voor de vala compiler
   definieert
elektrosim_VALAFLAGS = --pkg config --pkg gee-1.0 --pkg
   gio-2.0 --pkg glib-2.0 --pkg gobject-2.0 --pkg
   librsvg-2.0 --pkg ngspice

#voegt volgende bestanden toe aan de lijst voor make
   clean
CLEANFILES = *.c *.o elektrosim
```

Hoofdstuk 6

software architectuur

6.1 GTK3 en javascript interactie

D e software structuur die gebruikt is voor het weergeven en werken met de applicatie wordt hieronder weergegeven.



Figuur 6.1 – interactieschema GTK,cairo en GDK

D e GTK3 toolkit maakt gebruik van de Cairo bibliotheek voor het tekenen van het scherm/venster op een GDK tekenoppervlak (surface). De GDK bibliotheek zorgt er ook voor dat ingave en muisbewegingen kunnen worden teruggestuurd naar de verschillende klassen van de GTK3 bibliotheek. Ook kan de GTK3 bibliotheek communiceren met de GDK bibliotheek.

V oor het draaien van de applicatie op Linux via de Xorg server wordt er gebruik gemaakt van de GDK Xorg backend die via interproces communicatie en de Xlib bibliotheek met de Xorg server communiceert.

V oor het weergeven en communiceren met een HTML5 canvas biedt de GDK Broadway backend een serverdeel aan die via websockets communiceert met een javascript node die gedraaid wordt in de browser. Deze javascript node verzorgt de uitwisseling van ingave en muis bewegingen naar GDK en de uitvoer van de gewijzigde delen van het beeld naar het canvas.

6.2 ElektroSim

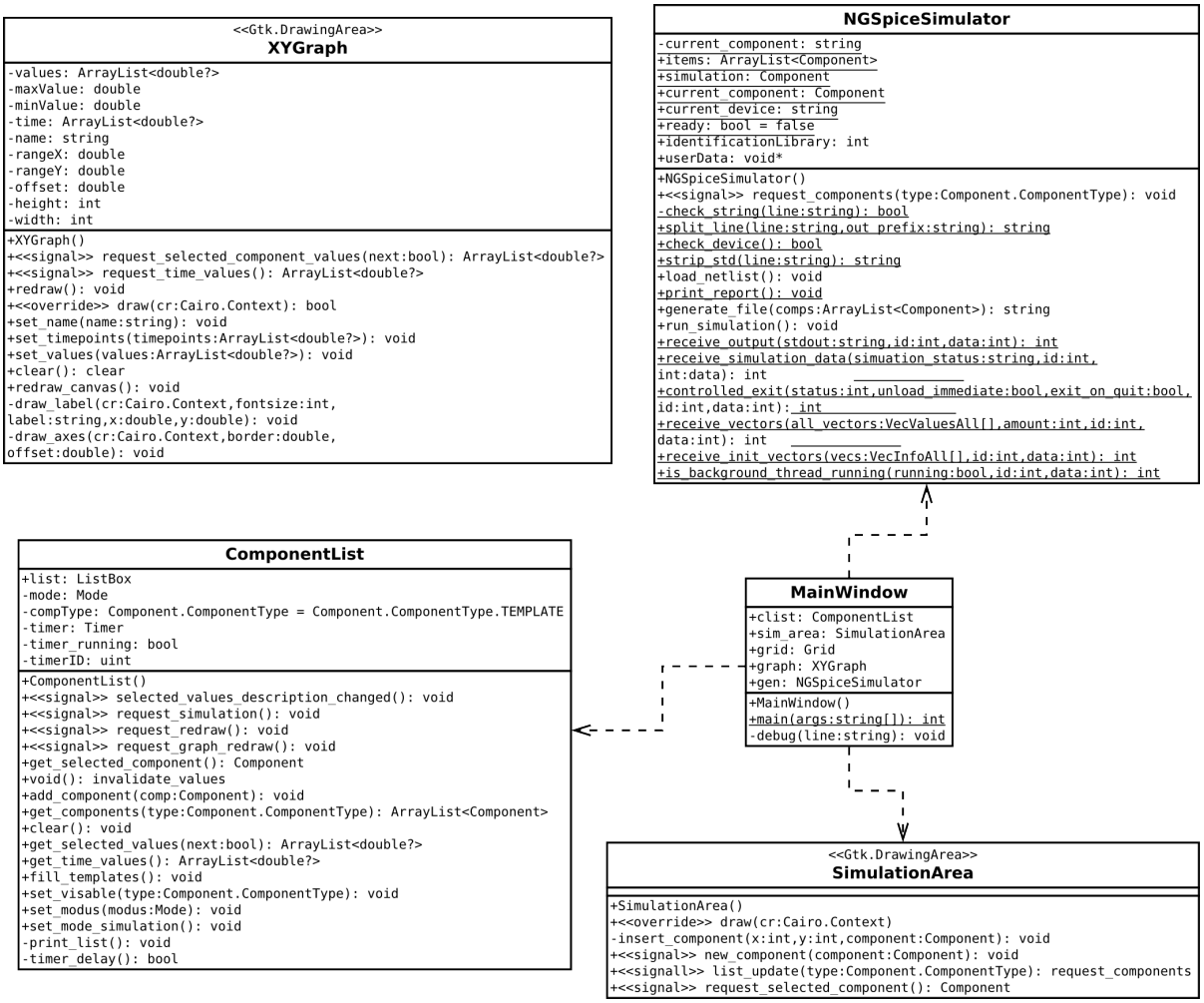
E lektroSim is de grafische werkomgeving ontworpen voor het aanmaken en simuleren van circuits. Hierbij is het mogelijk om de invloed van het wijzigen van de parameters te zien. Dit zowel in cijfers alsook aan de hand van emoticons, waarden en grafieken. De software is geschreven in Vala en maakt dus gebruik van de GTK3 toolkit. Verder wordt bij de compilatie gebruik gemaakt van het geschreven VAPI bestand voor de NGSpice bibliotheek. Dit zorgt ervoor dat de applicatie rechtstreeks met de ngspice simulator kan spreken en tussen elke stap van de berekening extra data kan opvragen.

D e volgende secties geven uitleg over de verschillende klassen waaruit het programma is opgebouwd. Hieronder kunt u het schema terugvinden van de hoofdcomponenten van het programma. De Component klasse die een bepaalde component beschrijft en de bijhorende klassen staan verder in dit hoofdstuk uitgelegd.

6.2.1 MainWindow

D eze klasse is de implementatie van het hoofdscherm. Deze klasse maakt een ComponentList, SimulationArea en XYGraph aan die door deze klasse in een Grid worden geplaatst. Verder maakt deze klasse ook een NGSpiceSimulator aan die zorgt voor de communicatie met de gedeelde bibliotheek.

D eze klasse voorziet ook in de main functie die nodig is voor het schrijven van een werkend programma. Deze functie initialiseert de GTK3 bibliotheek en maakt het hoofdvenster aan. Daarna voert deze de GTK3 hoofdlus uit. Deze lus is nodig omdat de GTK3 toolkit een event-driven toolkit is.



Figuur 6.2 – schema hoofdbestanddelen ElektroSim

H et hoofdscherm bestaat uit een Grid waarin links de lijst met elementen in een scrolwindow is geplaatst. Aan de rechterzijde wordt er een SimulationArea ingevoegd waar het circuit en de emoties van de componenten te zien zijn. Rechts onderaan wordt de grafiek ingevoegd indien men bij de simulatie de juiste component selecteert en daarna op de grafiek klikt. De headerbar bovenaan heeft enkele knoppen waarmee men schakelt tussen de verschillende werkingen van het programma. In de design-modus kan men een nieuw circuit aanmaken, dit is tevens het startscherm. In de simulatie-modus worden de emoticons op het circuit getoond (rechts) en worden links de gebruikte componenten getoond die men in realtime kan aanpassen. De clear-knop wist het hele circuit en brengt ons terug in de design-modus.

D eze klasse is ook verantwoordelijk voor het verbinden van de signalen van de andere klassen met methodes van deze andere klassen. Dit zorgt ervoor dat men enkele klassen zoals bijvoorbeeld de XYGraph, NGSpiceSimulator en SimulationArea desgewenst kan vervangen door andere implementaties. Ook zorgt dit ervoor dat de klassen onderling niet moeten doorgegeven worden aan elkaar.

6.2.2 SimulationArea

D eze klasse is een superklasse van de GTK3.DrawingArea en voorziet dus in het canvas waarop het circuit en de emoticons worden getekend. Deze klasse verwerkt ook de muisklik gebeurtenissen op het canvas.

B ij het aanmaken van de klasse zorgt een lambda methode ervoor dat het muisklik signaal van de canvas ervoor zorgt dat de functie insert_component wordt aangeroepen. Deze functie krijgt de locatie op het canvas mee en zal via het request_selected_component signaal van deze klasse ook de geselecteerde component in de ComponentList opvragen. De geselecteerde component (die van het type TEMPLATE is) zal via zijn clone methode worden gekloond in een nieuwe onafhankelijke component. Deze component wordt dan via zijn snap functie aan een nabijgelegen component gekoppeld of, indien geen nabijgelegen component beschikbaar is, vrij toegevoegd. Het new_component signaal geeft een de ComponentList door dat er een nieuwe component toegevoegd is.

H ierna wordt de redraw_canvas methode aangeroepen om het zichtbare gebied van de DrawingArea te invalideren, wat ervoor zorgt dat de hoofdlus van GTK dit gedeelte zal laten hertekenen. Dit zorgt ervoor dat de draw

methode aangeroepen wordt. Deze methode zal eerst de achtergrond kleuren waarna het via het request_components signaal alle componenten zal opvragen. Voor elke component wordt zijn draw functie aangeroepen.

6.2.3 XYGraph

A angezien de GTK3 toolkit geen ingebouwde klasse heeft voor het weergeven van grafieken heb ik deze zelf geïmplementeerd. Deze functie maakt ook gebruik van de Gtk.DrawingArea en voorziet in de nodige functies voor het tekenen van een 2 dimensionale grafiek.

B ij het aanmaken van de grafiek wordt het klikken op de grafiek gekoppeld aan een functie die door middel van signalen opvraagt welke gegevens er in de grafiek dienen worden weergegeven. Doordat deze klasse zelf de signalen uitstuurt kan men deze koppelen aan gelijk welke andere klasse voor het opvragen van een reeks waarden. Eerst wordt de functie set_values aangeroepen die door middel van het request_selected_components_values signaal de waarden(voor de y-as) inleest. Deze functie bepaalt ook de maximum waarde (maxValue) en de minimum waarde (minValue). Hieruit wordt dan het bereik over de y-as berekend (rangeY) alsook de offset van de x-as (offset) tot de minimale waarde.
Hierna wordt de set_timepoints methode aangeroepen die via het request_time_values signaal de tijdswaarden opvraagt. Ook wordt hierbij het bereik over de X-as berekend (rangeX).

T enslotte wordt de grafiek getekend in de draw functie. Deze functie voorziet eerst in een achtergrondkleur. Daarna wordt een label toegevoegd aan de bovenzijde van de grafiek die aangeeft welke parameter er wordt weergegeven. Eerst worden aan de hand van de gewenste rand en de offset de 2 assen getekend. Nadien wordt de oorsprong van de Cairo.Context verplaatst naar het snijpunt van de 2 assen. Hierna wordt de grafiek getekend door alle (geschaalde) waarden op de grafiek met elkaar te verbinden.

6.2.4 ComponentList

D eze klasse voorziet in een lijst van de componenten in de vorm van een ListBox. Deze functie voorziet de lijst van componenten in de grafische werkomgeving en houdt tegelijkertijd ook alle componenten en templates van componenten bij. Verder voorziet deze lijst in functies die de inhoud van de lijst kunnen filteren alsook de Componenten in deze lijst op verschillende manieren kan laten weergeven.

De methode add_component maakt het mogelijk om een component toe te voegen aan deze lijst. Deze component zal afhankelijk van de ComponenType van de lijst worden getoond of verborgen. Ook worden via de set_mode methode van de component meteen de juiste weergave ingeladen (zie Parameter). Ook het request_redraw en het request_graph_redraw signaal verbonden aan de overeenkomstige signalen van de component.

Indien het request_simulate signaal van een component wordt ontvangen zal er een timer beginnen lopen die na 3 seconden de timer_delay functie zal triggeren. Indien binnen deze 3 seconden nog een nieuw signaal binnenkomt zal de timer terug starten vanaf 3 seconden. De timer_delay functie zorgt ervoor dat alle waarden van de parameters worden geïnvalideerd. Dit zorgt ervoor dat bij de eerste nieuwe data van de simulatie de gegevens worden overschreven. Hierna zal er een signaal worden uitgezonden die een simulatie aanvraagt bij de NGSPiceSimulator klasse.

De methode fill_templates zorgt ervoor dat de lijst wordt gevuld met de templates van de verschillende subklassen van Componenten alsook met een simulatie component.

De mode waarin de lijst bevindt bepaalt hoe de componenten worden weergegeven in de lijst. Hierbij zijn momenteel 2 opties namelijk EDIT en SIMULATION. Deze modes refereren naar de 2 modes waarin een Parameter kan worden weergegeven. Per modus kan men bij een Paramater instellen welke layout van widgets er wordt gebruikt in de lijst.

De methodes get_time_values en get_selected_values geven respectievelijk de tijd waarden en de waarden van de geselecteerd parameter terug. Deze laatste methode heeft een argument waarmee men ofwel de huidige geselecteerde parameter kan opvragen of zijn opvolgende parameter.

6.2.5 NGSpiceSimulator

Deze klasse voorziet in de communicatie met de NGSpice simulator. Dit gebeurt door het toevoegen van de geschreven NGSpice.vapi file. Dit zorgt ervoor dat men functies en delegaties kan gebruiken in vala die bij het omzetten van vala naar c code mee worden vertaald.
De eerste stap is het aanmaken en starten van de ngpspice simulator. Bij deze init stap worden er 6 functies meegegeven die voldoen aan de modellen voor deze functies die beschreven zijn in her VAPI-bestand. Deze functies

worden hierna door de ngspice simulator aangeroepen en hebben volgende functionaliteit:

receive_output verwerkt de regels tekst die normaal naar de standaard uitvoer worden gestuurd.

receive_simulation_data ontvangt de huidige status van de simulatie (bij 'ready' wordt de gelijknamige bool op true gezet)

controlled_exit geeft de exit waarde terug en geeft aan of de ngspice simulator moet beëindigt worden bij het gebruik van exit commando.

receive_vectors wordt steeds opnieuw opgeroepen na elke stap in de simulatie en maakt het mogelijk om data op te vragen uit de simulatie vectoren (VecValuesAll)

receive_init_vectors geeft de initiële vectoren weer

is_background_thread_running geeft weer of de draad op de achtergrond draait

D e belangrijkste methode van deze klasse is de run_simulation methode. Deze methode vraagt eerste de simulatie component op via het request_components signaal en zal indien deze gevonden is de andere componenten ook opvragen via hetzelfde signaal. Hierna wordt de netlist ingeladen en wordt de gevraagde simulatie uitgevoerd.
Het inladen van de netlist gebeurt volgens de regels voor het opstellen van de nestlist waarbij de regel per component wordt ingeladen door zijn get_net_line methode. De shared spice interface voorziet een methode voor het opladen van een reeks van string die in het VAPI bestand is hernoemd tot upload_circuit.

N adat de simulatie is gestart zal na elke stap de receive_vectors methode worden aangeroepen. Hierbij wordt steeds voor elke component een show commando uitgevoerd zodat alle gegevens van de component kunnen worden ingelezen via de receive_output functie. Ook wordt de tijdsvector opgevraagd en de huidige waarde toegevoegd aan de simulatiegegevens van ElektroSim. Dit maakt het mogelijk om de grafiek en waarden in realtime aan te passen en bij te werken.
De receive_output functie zal de ontvangen tekst per regel analyseren waarbij eerst de prefix 'std****:' wordt verwijderd door de strip_std functie. Hierna controleert de check_string methode of de regel geldige waarden bevat. Deze functie zal indien de regel begint met 'device' de huidige component instellen, indien deze voorafgaat door 'Index' hoort de opvolgende informatie toe aan de

simulation component. Indien deze regel correct is en niet voldoet aan vorige voorwaarden geeft deze functie waar terug aan de oproepende functie. Deze functie zal nadien nog controleren of de gevraagde component bestaat en de waarden toevoegen aan de juiste parameter via de insert_simulation_data van de component klasse.

6.2.6 Component

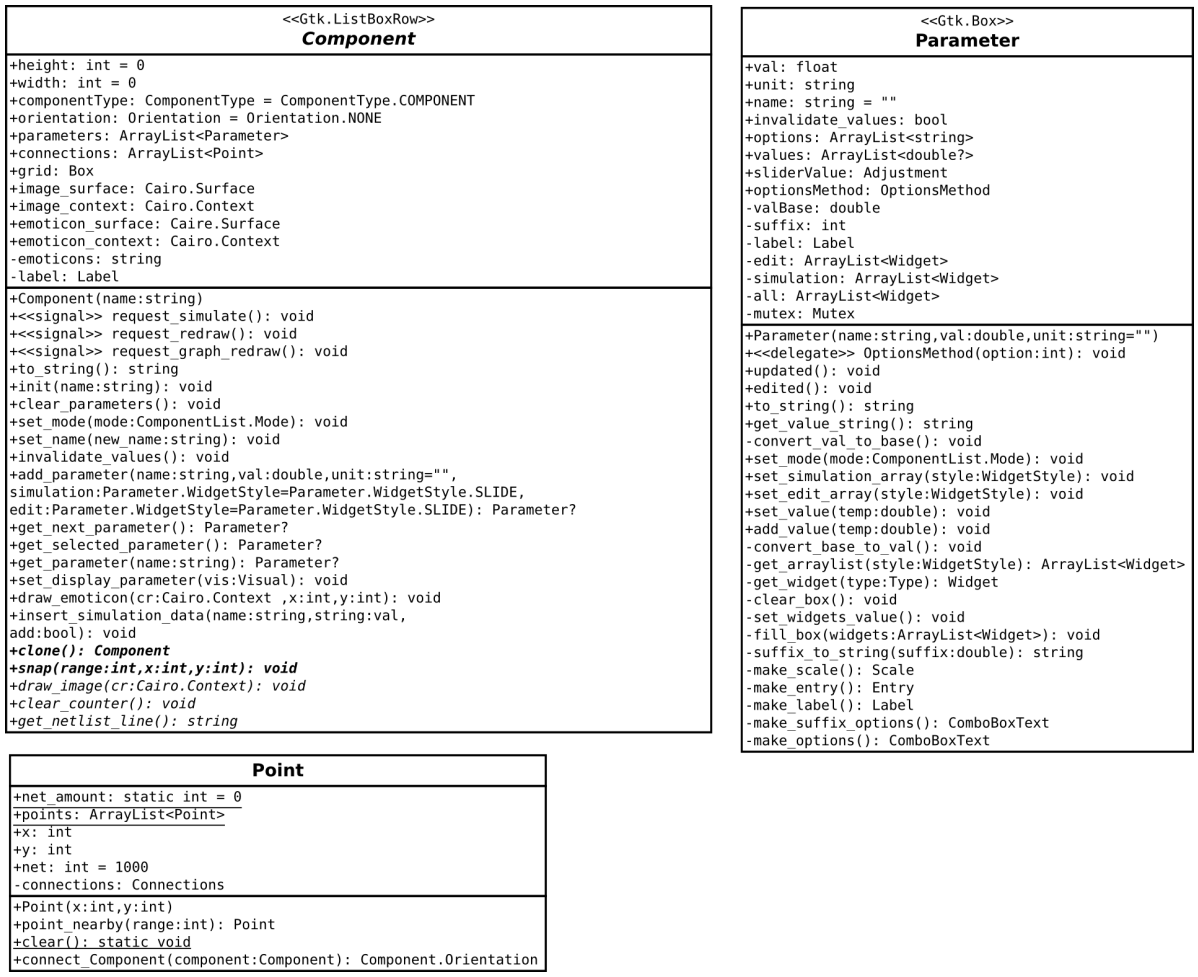
D e Component klasse implementeert een abstracte component van het circuit. In deze klasse worden alle gemeenschappelijke eigenschappen en methodes voor alle componenten geïmplementeerd. Dit zorgt ervoor dat nieuwe componenten gemakkelijk kunnen toegevoegd worden. Deze klasse is een superklasse van ListBoxRow zodat deze kan gebruikt worden in de lijst aan de linkerzijde.

D e Component heeft zowel een hoogte als een breedte als eigenschappen alsook een naam en een oriëntatie. Ook heeft de Component en ComponentType. De Component beschikt verder over een lijst met parameters en een lijst met punten (connections) die de verbindingspunten van de component bevatten. Ook zijn er voor zowel de afbeeldingen als het emoticon een Cairo.Surface en Cairo.Context aanwezig.

D e constructor van deze klasse maakt een GTK3.Box aan die alle parameters zal bevatten. Verder worden de stroom (i), het vermogen (p), de activiteit (activity) en de werkzone (work_zone) toegevoegd als gemeenschappelijke parameters. Ook wordt het updated signaal van de activity en de work_zone parameter gekoppeld aan het request_redraw signaal van de Component. Dit signaal zorgt voor het hertekenen bij wijziging van de emotie van de component.

D e methode add_parameter wordt gebruikt voor het toevoegen of bijwerken van een parameter aan de component. Hierbij kan men naast de naam en de waarde ook optioneel de eenheid en de WidgetStyle voor respectievelijk de simulatie en de edit mode meegeven. Deze 2 laatste waarden bepalen welke widgets er voor deze parameter worden weergegeven voor de 2 respectievelijke modi. De get_parameter methode zorgt ervoor dat een parameter bij naam kan worden opgevraagd.

D e functie draw_emoticon zal afhankelijk van de activiteit en de werkzone van de component het juiste emoticon tekenen op de locatie die als argument



Figuur 6.3 – schema Component en bijhorende klassen

wordt meegeven. Deze functie maakt gebruik van de librsvg-2.0 bibliotheek voor het inladen van de svg emoticons. Deze functie kan gebruikt worden bij de implementatie van Componenten met emoties.

D e functie get_netlist_line voorziet de NGSpiceSimulator van de regel die nodig is voor de component in het netlist bestand. Deze functie dient overschreven te worden indien er een regel nodig is. Het invoegen van simulatiegegevens in de component gebeurt door de insert_simulation_data methode.

D e methoden clone en snap moeten door de subklassen worden geïmplementeerd. De clone methode wordt gebruikt voor het aanmaken van de component vertrekkende van het model. Hierbij dient men bij genummerde componenten de statische parameter counter te verhogen zodat elke nieuwe component een unieke naam krijgt. De snap functie is de implementatie voor elke component die instaat voor het snappen van het eerste punt aan een naburig punt en afhankelijk van de oriëntatie de andere punten correct aan te maken.

D e methoden draw_image en clear_counter zijn niet verplicht om te implementeren maar vooral de draw_image functie wordt sterk aangeraden omdat men anders de component en zijn emotie niet kan visualiseren in het circuit. Deze functie vult namelijk de Cairo context op zodat er een afbeelding wordt getekend bij het hertekenen van de simulatieomgeving. De clear_counter methode moet geïmplementeerd worden voor componenten die genummerd dienen te worden en zorgt ervoor dat deze kunnen worden gereset.

T enslotte zijn de methoden get_next_parameter en get_selected_parameter geïmplementeerd zodat het mogelijk is om per oproep naar de functie de volgende en respectievelijke huidige parameter op te vragen zodat er door de parameters kan worden geschakeld. Dit zorgt ervoor dat de grafiek bij elke klik een andere parameter van de component kan tonen.

6.2.7 Point

D eze klasse is de implementatie van een punt (x,y) in de SimulationArea. Deze heeft verder een netnummer alsook een statische integer om het aantal verschillende subnetten bij te houden. Ook houdt deze klasse een statische lijst bij van alle punten die al zijn aangemaakt. Tenslotte bevat deze ook

een structuur Connections. Deze structuur slaagt voor elke richting (boven,onder,links,rechts) op welke component is aangekoppeld.

D e methode point_nearby zoekt in de lijst van Point naar een punt dat binnen het gevraagde bereik ligt (en een connectie vrij heeft). Indien deze is gevonden wordt dit punt als geretourneerde waarde gegeven, indien er echter geen naburig vrij punt is zal een nieuw punt worden aangemaakt op de coördinaten van het oorspronkelijke punt. Deze krijgt dan ook een opvolgende netnummer en wordt aan de statische lijst toegevoegd.

D e methode Connect_Component voegt de gegeven component toe aan het punt. Hierbij wordt gekeken of de oriëntatie reeds bekend is. Indien de oriëntatie van de toe te voegen component gekend is zal deze worden gekoppeld aan de overeenkomstige verwijzing in de structuur connections. Indien deze geen gekende oriëntatie heeft wordt deze component toegevoegd aan de eerste vrije richting (rechts,links,onder,boven) waarna de toegekende oriëntatie wordt teruggegeven.

D e methode clear verwijdert alle punten uit de statische lijst en zet het aantal subnetten op nul. Deze functie wordt gebruikt bij het verwijderen van het circuit.

6.2.8 Parameter

D eze klasse implementeert een parameter van de Component klasse. Deze klasse is een superklasse van een Gtk.Box container wat ervoor zorgt dat we hierin verschillend widgets kunnen plaatsen.

D e huidige (of bij emoties en work_zone maximale) waarde van de parameter wordt opgeslagen in de dubbele precisie variabele val. De functie convert_val_to_base zet de waarde van de parameter om in een fractie(valBase) en een exponent(suffix). Dit zorgt ervoor dat de waarde leesbaar is in labels en bij de sliders. De omgekeerde functie is ook toegevoegd zodat de waarden uit de sliders en eenheid opties kunnen worden omgezet naar de correcte waarde.

B ij het aanmaken van een nieuwe parameter wordt de naam, de waarde en de eenheid van de parameter ingeladen. Hierbij wordt ook de conversie functie gedraaid zodat ook deze waarden beschikbaar zijn. Hierna dient men de set_simulation_array en de set_edit_array methode te draaien. Deze

nemen als inkomend argument een WidgetStyle. Deze heeft volgende opties (enum):

NONE geef de parameter (naam en waarde) niet weer

LABEL geef de parameter (naam en waarde) weer in een label

ENTRY geef de parameter (naam in label) weer in aanpasbaar label

OPTIONS geef de parameter (naam in label) weer in een combobox

SLIDER geef de parameter (naam in label) weer met een slider en een combobox voor de eenheid

Deze functies roepen beide de functie get_arraylist aan. Deze zal afhankelijk van de WidgetStyle de juiste widgets toevoegen aan de edit of simulatie array. Dit gebeurt door het zoeken van de widgets in de ArrayList all. Indien deze hier niet gevonden is wordt deze aangemaakt met de verschillende make_* methoden. De methoden voor entry,scale,suffix_options en options sturen allen een edited signaal uit indien de waarde gewijzigd is. Dit zorgt ervoor dat de simulatie aangevraagd wordt. De options widget voert echter eerst de optionsMethod uit. Dit is een verwijzing(delegate) naar een methode die door de implementatie van de Component kan worden ingesteld en het mogelijk maakt om afhankelijk van de optie andere parameters aan de component toe te wijzen.

D e ArrayList values houdt de opeenvolgende waarden bij zodat deze kunnen worden getekend op de grafiek. De edit, simulation en all reeksen bestaan uit Widgets. De set_value methode past de parameter waarde aan in de alle widgets.

6.2.9 Simulation, Ground, Line, Resistor, PowerSource, Capacitor en Inductor

D eze klassen erven allemaal methoden en gemeenschappelijke eigenschappen van de Component klasse. Hierbij worden enkele unieke parameters toegevoegd en worden enkele functies geïmplementeerd zoals beschreven in de vorige sectie.

D e simulation klasse bepaald welke type simulatie er zal worden uitgevoerd. Deze klasse bevat ook de tijd parameter en ook een status. Bij start van elke simulatie wordt de status op 0 gezet, hierna wordt de status terug op 1 gezet bij einde van simulatie. Dit zorgt ervoor dat de emoticons worden aangepast en de grafiek wordt hertekend door het gebruik van signalen.

D e aarde heeft geen enkele parameter en de constructor verwijdert dus alle parameters uit de lijst. De snap functie zorgt er verder voor dat elk punt dat verbonden wordt aan dit net het netnummer 0 toegewezen krijgt. Het netnummer 0 staat voor het grondnet in de NGSpice simulator.

D e lijn component is een component die aangemaakt dient te worden door 2 maal te klikken en deze twee punten dan verbindt. Deze functionaliteit wordt verkregen door na ingave van het eerste punt de second_point_needed variabele op juist te zetten. Bij het toevoegen van een lijn zal de SimulationArea namelijk controleren of er nog een bestaande lijn component bestaat die nog een tweede punt nodig heeft en deze hieraan toevoegen. Verder worden ook hier de basisparameters verwijderd.

D e weerstand component implementeert de netlist, clone en clear_counter methode. De weerstand heeft als extra parameters de weerstand (R) en het maximale te dissiperen vermogen (Max Power). Ook heeft de weerstand 4 optionele parameters namelijk ac,dtemp,bv_max en noisy. De snap functie zal verder afhankelijk van de oriëntatie van de component het tweede verbindingspunt toevoegen.

D e spanningsbron component is de implementatie van een spanningsbron. Aangezien NGSpice verschillende soorten spanningsbronnen ondersteund zijn hierbij ook verschillende soorten geïmplementeerd. Deze component maakt gebruik van de ChangeType methode om bij keuze van een andere soort spanningsbron de correcte parameters op te vragen. Bij deze component wordt bij het dupliceren van de component uit het model een unieke naam aangemaakt. Dit zorgt ervoor dat deze klasse en counter variabele nodig heeft en een clear methode om deze terug op 0 te kunnen zetten. Hier wordt ook de netlist functie geïmplementeerd zodat de stroombron kan worden toegevoegd aan de netlist. Deze functie zal dan afhankelijk van de soort spanningsbron verschillend zijn.

D e condensator component is gelijkaardig aan de implementatie van de weerstand maar heeft als parameters de capaciteit (capacitance) en de werkspanning (rated voltage).

D e inductor component is gelijkaardig aan de implementatie van de weerstand maar heeft als parameters de inductiviteit (inductance) en de aangeraden maximale stroom (rated current).

6.3 Server

D e server applicatie bestaat uit 3 delen namelijk de webpagina, de databank en de Server applicatie.

D e webpagina is een eenvoudige webpagina waar men kan registreren en inloggen. Dit gebeurt door middel van een connectie met een MariaDB (een MySQL fork). Na het inloggen wordt er op de server de Server applicatie gestart. Bij het starten van de applicatie wordt het ID van de login meegegeven als argument. Hierna start de applicatie de Broadwayd server op het poortnummer 8080+ID. Tenslotte start deze applicatie op de Broadway server ook een ElektroSim applicatie op.Hierna stuurt de website u door naar de Broadwayd waar men met de applicatie kan werken.

6.4 het NGSpice VAPI bestand

D it bestand zorgt ervoor dat de vala code verbonden wordt met de c header file van de gedeelde NGSpice bibliotheek. Bij compilatie van het programma weet de vala compiler hierdoor welke c functies en variabelen door de vala code worden bedoeld.

6.4.1 init en delegates

D e init functie initieert de ngspice simulator en bij initiatie dient men verschillende callback functies mee te geven.

```
int  ngSpice_Init(SendChar* printfcn,
        SendStat* statfcn, ControlledExit* ngexit,
    SendData* sdata, SendInitData* sinitdata,
    BGThreadRunning* bgtrun, void* userData);
```

I n C worden deze functies meegegeven door middel van pointers van de functie types die gedefinieerd zijn per functie. De void pointer userData wijst naar data door de gebruiker gedefinieerd, dit wordt in het programma niet gebruikt.

```
[CCode (cname = "ngSpice_Init",simple_generics = true)]
                public static int
                   init<T>(SendOutput<T>? a,
                   SendSimulationStatus<T>? b,
```

```
ControlledExit<T>? c,
SendVectorData<T>? d,
SendInitializationData<T>? e,
IsBackgroundThreadRunning<T>? f,
T data);
```

De naam van de functie wordt veranderd in vala naar init omdat in plaats van elke functie vooraf te gaan door NGSpice hiervan een naamruimte wordt gebruikt die alle functies en variabelen omvat. Er wordt ook een generische functie van gemaakt zodat alle callback functies alsook de init functie zelf gebruik kunnen maken van een zelf gekozen type van userData. Dit zorgt ervoor dat men op type kan controleren in plaats van het gebruik van void pointers zoals in de C programmeertaal. Het vraagteken achter elke terugroepfunctie vertelt de vala compiler dat deze delegates ook de waarde null kunnen hebben.

delegates

D e 6 delegate functies zijn gelijkaardig geïmplementeerd. Hierbij worden functiedefinities (typedef teruggeefwaarde (functie)) in C omgezet naar gelijkwaardige delegate functies in Vala. Verder wordt bij elke functie ook een id teruggegeven die aanduidt welke instantie (indien er meerdere draaien) van de gedeelde bibliotheek de functie oproept.

SendOutput Deze functie wordt aangeroepen indien er een tekstregel naar de standaard terminal of naar de standaard foutterminal wordt gestuurd. Hierbij is een char* type omgezet in een string type.

SendSimulationStatus Deze functie wordt opgeroepen indien er informatie over de voortgang van de simulatie beschikbaar wordt in de vorm van een string bestaande uit de status (start,einde,etc) en de procentuele voortgang.

ControlledExit Deze functie wordt aangeroepen indien de NGSpice bibliotheek afgesloten wordt of wilt afsluiten. Dit laatste wordt aangegeven door een bool waarde die indien waar, ervoor zorgt dat de gedeelde bibliotheek meteen ontkoppeld wordt, en indien vals wordt deze pas ontkoppeld als de functie is doorlopen. Een tweede bool waarde geeft aan of de gedeelde bibliotheek afgesloten is door het quit commando of door een interne fout. Ook wordt aan de hand van een string de exit waarde teruggeven aan de functie.

SendVectorData Deze functie geeft de vector informatie terug aan de hand van een array van alle vectoren (VecValuesAll), ook geeft deze functie het aantal vectoren als int terug. Dit omdat men in C de

lengte van vectoren niet kan meegeven aan de array en deze dus als aparte variabele moet worden meegestuurd.

SendInitializationData Deze functie wordt opgeroepen na initialisatie van het circuit voor simulatie en geeft een VecinfoAll reeks terug die de vectoren beschrijft.

IsBackgroundThreadRunning Deze functie geeft aan of de achtergrond thread(die de simulatie uitvoert) wordt gedraaid aan de hand van een boolean waarde.

Hoofdstuk 7

De emotionele modellen

7.1 Emoties

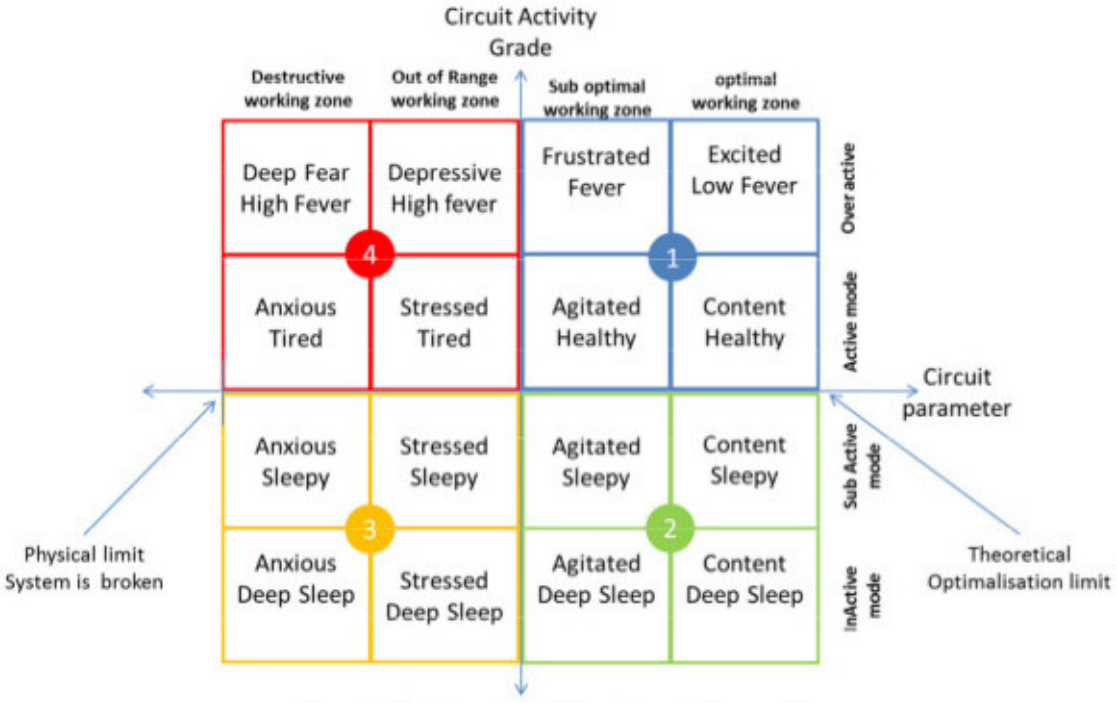
In paper [30] wordt beschreven hoe men componenten van een elektronisch circuit op een Hedonistisch manier kan beschrijven. Hiervoor maakt men gebruik van 2 parameters van de component, namelijk zijn werkingsgebied en de tijd die hij doorbrengt in elk gebied.

I ndien men enkel het werkingsgebied in beschouwing neemt kan men stellen dat er 4 gebieden zijn namelijk sub-optimaal, optimaal, buiten het bereik van de component en de destructieve zone. De tijd die de component in een zone doorbrengt kan men opdelen in 4 grote zones namelijk inactief, sub-actief, actief, overactief.

I ndien men de component wil beschrijven aan de hand van emoticons dan zal men elke status moeten weergeven door een emotie en de bijhorende emoticon. Indien de component niet overactief is worden de emoties angst, stress, nerveus en tevreden. Indien echter de component zich in de overactieve zone bevindt worden dit respectievelijk diepe angst, depressief, gefrustreerd en opgewonden.

D e component kan verder ook een gezondheidsstatus meekrijgen die grotendeels afhankelijk is van de activiteit van de component, inactieve en sub-actief krijgen respectievelijk diepe slaap en slaperig als gezondheidsstatus. Indien de component overactief is zal het afhankelijk van het werkgebied , zwakke koorts, koorts of sterke koorts zijn. Indien de component normaal actief is zal hij afhankelijk van het werkgebied gezond of vermoeid zijn.

A lles samen geeft dit de volgende matrix. Daaronder zijn de emoticons die gebruikt worden in het programma te zien. Hierbij is gekozen om de inactieve zone allemaal hetzelfde slapend emoticon te geven om aan te duiden dat deze component eigenlijk niet actief is in deze schakeling.

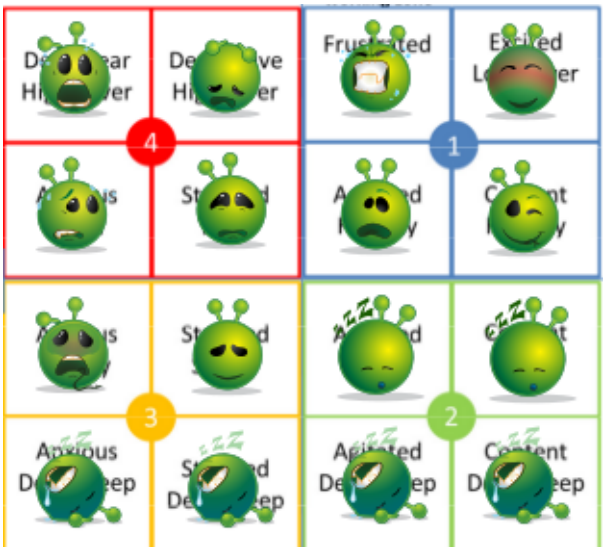


Figuur 7.1 – emotionele matrix

7.2 Het emotioneel weerstandsmodel

H et werkingsgebied waarin de weerstand kan functioneren is afhankelijk van de hoeveelheid energie die de weerstand kan dissiperen. Dit is het vermogen op een bepaald moment namelijk P=U.I . De reguliere weerstanden hebben een vermogendissipatie van 100,125 of 250 milliwatt. Vermogensweerstanden zijn meestal groter en vaak uitgerust met een koelelement dat ervoor zorgt dat er meer warmte kan worden afgevoerd. Het werkingsgebied kan dus gedefinieerd worden:

$$W_z(t) = \frac{P(t)}{Pmax}$$



Figuur 7.2 – voorgestelde emoticons (Tango emoticon set)

Afhankelijk van de waarde van de werkzone kunnen we de weerstand indelen in 4 zones:

Sub-optimale zone($W < 0.5$)**:** weerstand bevindt zich onder zijn werkgebied.

Optimale zone($0.5 < W <= 1$)**:** weerstand wordt gebruikt in zijn werkgebied.

Out Of Range zone($1 < W <= 1.1$)**:** weerstand werkt boven zijn specificaties.

Destructieve zone($W > 1.1$)**:** weerstand gaat defect.

D e activiteit A van een weerstand kunnen we definiëren door de volgende formule:

$$A(t) = \frac{\frac{\int_{t0}^{t} P(t)dt}{(t-t0)} + \frac{dE(t)}{dt}}{Pmax}$$

Z olang het gedissipeert vermogen binnen het werkgebied van de weerstand blijft zal de activiteit tussen 0 en 1 gesitueerd zijn. Indien het vermogen over de weerstand hoger wordt voor een voldoende lange tijd of het vermogen plotseling snel toeneemt zal de activiteit snel stijgen. De Activiteit kan ingedeeld worden in 4 zones:

non-actieve zone($0 <= A < 0.1$)**:** weerstand verbruikt geen vermogen

sub-actieve zone($0.1 <= A < 0.3$)**:** weerstand verbruikt zeer weinig vermogen

actieve zone($0.3 <= A <= 1$)**:** weerstand verbruikt gespecificeerd vermogen

destructieve zone($A > 1$)**:** weerstand verbruikt meer/sneller dan toegelaten

7.2.1 implementatie

I nitieel ben ik begonnen met het schrijven van een emotioneel weerstandsmodel met het XSPICE Code model. Aangezien na enig onderzoek bleek dat het in XSPICE niet zo eenvoudig was om de vereiste functionaliteit toe te voegen ben ik overgeschakeld op het c-model. Dit model bestaat reeds in de NGSpice simulator en is dus nu uitgebreid met berekeningen en parameters voor het toevoegen van emoties.

E en ander voordeel van deze methode is dat de parameters rechtstreeks kunnen worden weergegeven bij het show all commando. Dit maakt dat de code gemakkelijker te debuggen is en dat de integratie met de simulator veel sneller verloopt. Het grote nadeel is hierbij dat de modellen samen met het programma moeten gecompileerd worden.

I n volgende sectie worden de aanpassingen toegelicht worden ten opzichte van het basis weerstandsmodel van NGSpice.

res.patch

D eze patch zorgt ervoor dat het basismodel van de simulator (/src/libspice/devices/res) wordt aangepast en voorzien van de nodige extra parameters en functies zodat men de emoties van de component ook kan laten berekenen.

H iervoor zijn er 4 parameters (waarvan 3 zichtbaar in de simulator) toegevoegd. De max_power parameter is nodig voor het berekenen van de activity en working_zone parameters. Deze laatste 2 parameters geven de emotionele toestand van de weerstand aan. De vierde parameter die niet zichtbaar is in de simulator wordt gebruikt om de activity parameter te berekenen bij een transiënt analyse. Deze aanpassingen bevinden zicht in het res.c bestand.

```
IFparm RESpTable[] = {
        ...
    IOPP( "max_power",          RES_MAX_POWER,
       IF_REAL,    "max␣power␣value"),
    IOP(   "activity",          RES_ACTIVITY,
       IF_STRING,    "activity␣emotion"),
    IOP(   "work_zone",         RES_WORKING_ZONE,
            IF_STRING,    "working␣zone␣emotion"),
    IOPU(   "power_sum",        RES_POWER_SUM,
       IF_REAL,    "activity␣power␣sum")
}
```

B ij de instantiestructuur is de activiteit en werkzone parameter toegevoegd van het type Array van char. De som van het reeds opgenomen vermogen(RESpowerSum) en het maximaal te dissiperen vermogen van de weerstand worden als double toegevoegd. Verder worden er ook een aantal defines voor deze parameters gegeven zodat de simulator de verschillende variabelen kan opvragen per nummer. De 'Given' parameters worden gebruikt bij het inlezen van de parameters om aan te duiden welke parameters door de gebruiker zijn meegegeven. Deze toevoeging gebeurt in het resdefs header bestand.

```
typedef struct sRESinstance {
        ...
        unsigned RESmaxPowerGiven     :1;
        unsigned RESactivityGiven     :1;
        unsigned RESworkingZoneGiven  :1;

        double RESpowerSum;
    double RESmaxPower;
    char *RESactivity;
    char *RESworkingZone;
}
...
/*emotions*/
#define RES_MAX_POWER 19
#define RES_ACTIVITY 20
#define RES_WORKING_ZONE 21
#define RES_POWER_SUM 22
```

I n het resask bestand worden de 3 extra parameters toegevoegd. Deze functies worden aangeroepen als men de gegevens over de component op-

vraagt via het show of show all commando. Het maximale te dissiperen vermogen van de weerstand geeft de ingegeven parameter terug. De 2 andere parameters geven de activiteit en de werkzone terug terug zoals aangegeven in dit hoofdstuk. Hierbij is fast de instantie van de component en stelt ckt de circuit parameters voor.

```
case RES_WORKING_ZONE:
    if (ckt->CKTrhsOld) {

    workingZone = (*(ckt->CKTrhsOld + fast->RESposNode)
       - *(ckt->CKTrhsOld + fast->RESnegNode)) *
              fast->RESconduct *
     (*(ckt->CKTrhsOld + fast->RESposNode) -
          *(ckt->CKTrhsOld + fast->RESnegNode));
           if(workingZone<0){
                workingZone*=-1;
                }
                workingZone=workingZone/fast->RESmaxPower;
            workingZone *= fast->RESm;
            if(workingZone<0.5){
            workingZoneString = TMALLOC(char,
               strlen("Sub-optimal") + 10);
            sprintf(workingZoneString,
               "suboptimal:%f",workingZone);
            value->sValue=workingZoneString;
            }else ...

case RES_ACTIVITY:
    if(ckt->CKTrhsOld){
        power = (*(ckt->CKTrhsOld + fast->RESposNode) -
    *(ckt->CKTrhsOld + fast->RESnegNode))
       *fast->RESconduct *
    (*(ckt->CKTrhsOld + fast->RESposNode) -
       *(ckt->CKTrhsOld + fast->RESnegNode));

        power *= fast->RESm;
   if(ckt->CKTcurrentAnalysis &(DOING_AC | DOING_TRAN)){
     activity=fast->RESpowerSum/ckt->CKTtimePoints[ckt->CKTtimeIndex];
        // mean dissipated power
     activity+=power/ckt->CKTdeltaList[ckt->CKTtimeIndex];
        //divide by time lapse
activity/=(fast->RESmaxPower*fast->RESm); // normalize
    }else{ //dc or op
                        activity=power/(fast->RESmaxPower*fast->RESm);
```

```
            }
    if(activity<0.1){
             activityString = TMALLOC(char,
                strlen("Inactive") + 10);
            sprintf(activityString,
               "inactive:%f",activity);
            value->sValue=activityString;
    }else...
```

In het resload bestand is enkel het berekenen van de som van het vermogen toegevoegd. Dit zorgt ervoor dat bij elke stap de som van het reeds opgenomen vermogen wordt bijgehouden.

```
here->RESpowerSum+=(here->REScurrent*here->REScurrent*here->RESresist)
                        *ckt->CKTdeltaList[ckt->CKTtimeIndex];
```

In het bestand dat de parameters beschrijft (resparameter.c) is de maxPower parameter toegevoegd. Dit zorgt ervoor dat het mogelijk is de weerstand aan maximaal te dissiperen vermogen mee te geven. Bij toevoeging van deze parameter wordt via de RESmaxPowerGiven boolean aangegeven dat deze is toegevoegd.

```
    case RES_MAX_POWER:
           here->RESmaxPower = value->rValue;
           here->RESmaxPowerGiven=TRUE;
        break;
```

7.3 Het emotioneel condensatormodel

Het werkingsgebied waarin de condensator kan functioneren is afhankelijk van de hoeveelheid energie die de condensator kan opnemen. Hierbij wordt de energiedensiteit van de capacitor gegeven door de energie die er per volume kan worden opgeslagen. Aangezien men in de specificaties niet veel aandacht besteed aan het exacte volume van de condensator en het basismodel in ngspice enkel als ingangsparameter de capaciteit vereist heb ik de energie die opgeslagen kan worden anders berekend.
Doordat men meegeeft wat de maximale werkspanning van de condensator is kunnen we deze waarde gebruiken om de maximale energie die de condensator kan opnemen te bereken. Dit zorgt ervoor dat we ook de huidige opgeslagen energie kunnen berekenen.
$E_d(t) = \frac{C*U(t)^2}{2}$

Nu kunnen we de functionele werkzone van de condensator berekenen ten opzicht van de maximale op te slagen energie in deze condensator.

$W_z(t) = \frac{C*U(t)^2}{E_d*2}$

Afhankelijk van de waarde van de werkzone kunnen we de condensator indelen in 4 zones:

Sub-optimale zone($W < 0.5$)**:** condensator bevindt zich onder zijn werkgebied.

Optimale zone($0.5 < W <= 1$)**:** condensator wordt gebruikt in zijn werkgebied.

Out Of Range zone($1 < W <= 1.1$)**:** condensator werkt boven zijn specificaties.

Destructieve zone($W > 1.1$)**:** condensator neemt teveel energie op en gaat defect.

D e activiteit A van de condensator kunnen we definiëren door de volgende formule:

$A(t) = \frac{\frac{\int_{t0}^{t} E_d(t)dt}{(t-t0)} + \frac{dE(t)}{dt}}{E_d}$

Z olang de energie opgenomen door de condensator binnen het werkgebied van de condensator blijft zal de activiteit tussen 0 en 1 gesitueerd zijn. Indien de spanning over de condensator plotseling snel toeneemt zal de activiteit snel stijgen. Ook zal hierdoor het de opgenomen energie nog sneller stijgen. Indien de opgenomen energie de maximale op te nemen energie overstijgt zal de condensator beginnen te geleiden en doorbranden of stuk gaan. De Activiteit kan ingedeeld worden in 4 zones:

non-actieve zone($0 <= A < 0.1$)**:** condensator neemt (bijna) geen energie op en is (bijna) niet geladen

sub-actieve zone($0.1 <= A < 0.3$)**:** condensator heeft een lading ver onder de maximale waarde en/of geeft of neemt minder dan de toegelaten energie op of af.

actieve zone($0.3 <= A <= 1$)**:** condensator neemt aangeraden energie op

destructieve zone($A > 1$)**:** condensator laadt te veel of sneller op dan toegelaten

7.3.1 implementatie

I n volgende sectie worden de aanpassingen toegelicht ten opzichte van het basis condensatormodel van NGSpice.

cap.patch

D eze patch zorgt ervoor dat het basismodel van de simulator (/src/libspice/devices/cap) wordt aangepast en voorzien van de nodige extra parameters en functies zodat men de emoties van de component ook kan laten berekenen.

H iervoor zijn er 5 parameters (waarvan 4 zichtbaar in de simulator) toegevoegd. De rated_voltage parameter is nodig voor het berekenen van de energie die is opgenomen en bepaalt dus de activity en working_zone parameters. Deze laatste 2 parameters geven de emotionele toestand van de weerstand aan. De vijfde parameter die niet zichtbaar is in de simulator wordt gebruikt om de activity parameter te bereken bij een transiënt analyse. Deze aanpassingen bevinden zicht in het cap.c bestand.

```
IFparm RESpTable[] = {
...
IOPP( "rated_voltage",CAP_RATED_VOLTAGE,IF_REAL,"rated␣
   voltage"),
IOP(   "activity",CAP_ACTIVITY,IF_STRING, "activity␣
   emotion"),
IOP(   "work_zone",CAP_WORKING_ZONE,IF_STRING, "working␣
   zone␣emotion"),
IOPU(   "energy_sum",CAP_ENERGY_SUM,IF_REAL, "activity␣
   energy␣sum"),
IOP( "energy",CAP_ENERGY,IF_REAL,"max␣energy␣density")
}
```

B ij de instantiestructuur is de activiteit en werkzone parameter toegevoegd van het type Array van char. De som van de reeds opgenomen/afgestane energie(CAPenergy) en de standaard spanning worden als variabelen toegevoegd. Verder worden er ook een aantal defines voor deze parameters gegevens zodat de simulator de verschillende variabelen kan opvragen per nummer. De 'Given' parameters worden gebruikt bij het inlezen van de parameters om

aan te duiden welke parameters door de gebruiker zijn meegegeven. Deze toevoeging gebeurt in het capdefs header bestand.

```
typedef struct sRESinstance {
        ...
    unsigned CAPratedVoltageGiven     :1;
        unsigned CAPactivityGiven     :1;
        unsigned CAPworkingZoneGiven      :1;
    unsigned CAPenergyGiven     :1;

        double CAPenergySum;
    double CAPenergy;
    double CAPratedVoltage;
    char *CAPactivity;
    char *CAPworkingZone;
}
...
/*emotions*/
#define RES_MAX_POWER 19
#define RES_ACTIVITY 20
#define RES_WORKING_ZONE 21
#define RES_POWER_SUM 22
```

I n het capask bestand worden de 4 extra parameters toegevoegd. Deze functies worden aangeroepen als men de gegevens over de component opvraagt via het show of show all commando. De maximale spanning over de condensator wordt hier gewoon uitgelezen en de energie die zich in de condensator bevindt wordt berekend. De functies voor het weergeven van de werkzone alsook de activiteit zijn gelijkaardig aan deze voor de weerstand.

```
    case CAP_ENERGY:
        value->rValue = (here->CAPratedVoltage
        *here->CAPratedVoltage*here->CAPcapac)*1/2;
        value->rValue *= here->CAPm;
        return(OK);
        case CAP_RATED_VOLTAGE:
        value->rValue = here->CAPratedVoltage;
        return(OK);



case CAP_WORKING_ZONE:
```

```
if (ckt->CKTrhsOld) {

workingZone = (*(ckt->CKTrhsOld + here->CAPposNode) -
   *(ckt->CKTrhsOld + here->CAPnegNode)) *
 here->CAPcapac* (1/2) * (*(ckt->CKTrhsOld +
    here->CAPposNode) -
 *(ckt->CKTrhsOld + here->CAPnegNode));

if(workingZone<0){
  workingZone*=-1;
}
workingZone=workingZone/here->CAPenergy;
workingZone *= here->CAPm;
if(workingZone<0.5){...

case CAP_ACTIVITY:

if(ckt->CKTrhsOld){
energy = (*(ckt->CKTrhsOld + here->CAPposNode) -
     *(ckt->CKTrhsOld + here->CAPnegNode)) *
    here->CAPcapac * (1/2) * (*(ckt->CKTrhsOld +
       here->CAPposNode) -
    *(ckt->CKTrhsOld + here->CAPnegNode));

energy *= here->CAPm;
if(ckt->CKTcurrentAnalysis &  (DOING_AC | DOING_TRAN)){
   activity=here->CAPenergySum/ckt->CKTtimePoints[ckt->CKTtimeIndex];
      // energy in cap
   activity+=energy/ckt->CKTdeltaList[ckt->CKTtimeIndex];
      //divide by time lapse
   activity/=(here->CAPenergy*here->CAPm); // normalize
   }else{ //dc or op
      activity=energy/(here->CAPenergy*here->CAPm);
   }
   if(activity<0.1){...
```

I n het capload bestand is enkel het berekenen van de som van het opgenomen en afgestaan vermogen toegevoegd. Dit zorgt ervoor dat bij elke stap de huidige lading in de condensator wordt bijgehouden.

```
here->CAPenergySum+=vcap*vcap*here->CAPcapac*1/2*
        ckt->CKTdeltaList[ckt->CKTtimeIndex];
```

In het bestand dat de parameters beschrijft (capparameter.c) is de rated voltage parameter toegevoegd. Dit zorgt ervoor dat het mogelijk is de condensator de maximale spanning mee te geven. Bij toevoeging van deze parameter wordt meteen de corresponderende maximaal op te nemen energie berekend.

```
case CAP_RATED_VOLTAGE:
 here->CAPratedVoltage = value->rValue;
 here->CAPenergy =
    (here->CAPratedVoltage*here->CAPratedVoltage*here->CAPcapac)*1/2;
 here->CAPratedVoltageGiven=TRUE;
 break;
```

7.4 Het emotioneel inductormodel

Een inductor bestaat uit een gewikkelde draad en eventueel een kernmateriaal. Een inductor is de tegenhanger van een condensator en zal dus bij toevoer van de stroom een tegenwerkende spanning genereren waarvan de energie omvat zit in een elektrisch veld [9].

De grootste oorzaak van falende condensatoren is het doorbranden ervan doordat er een te grote stroom door de inductor loopt. De maximale stroom waarop de condensator correct functioneert wordt in de datasheets aangegeven met de 'maximum rated current' parameter van de component. Indien men boven deze waarde komt bevindt de inductor zich buiten zijn werkgebied en kan dus stukgaan. Deze parameter wordt bepaald door het meten van de opwarming van de component bij deze stroom. Indien de saturatie stroom (waarbij de afwijking van de inductantie groter wordt dan 10 procent) kleiner is dan de maximum rated current dient mijn deze waarde te gebruiken. Bij het te snel omschakelen van de inductor zal deze steeds minder functioneren als inductor en komt de capacitieve werking (tussen de windingen) steeds sterker naar voor. Aangezien het model van de NGSpice simulator uitgaat van een ideale inductor treedt er geen verlies door weerstand op (die groter wordt met de frequentie).

Bij zeer hoge spanning is er een mogelijkheid tot kortsluiting door het ontstaan van spanningsbruggen door de lucht of de isolator. Deze maximale spanning is echter zeer sterk variabel van inductor tot inductor en wordt hierdoor niet meegegeven in de de datasheet.

H et vermogen opgenomen in het magnetisch veld van de inductor is gerelateerd aan de inductantie in Henri en de stroom door deze inductor. Aangezien de maximale stroom gekent is kan men gemakkelijk het maximaal op te nemen vermogen door de inductor berekenen.
$E_m(t) = \frac{L*I_m(t)^2}{2}$
Indien we de huidige opgenomen energie door de inductor delen door de maximale op te nemen energie kunnen we hieruit de werkzone berekenen.
$W_z(t) = \frac{\frac{L*I(t)^2}{2}}{E_m} == \frac{L*I(t)^2}{E_m*2}$

Afhankelijk van de waarde van de werkzone kunnen we de inductor indelen in 4 zones:

Sub-optimale zone($W < 0.5$)**:** inductor bevindt zich onderaan zijn werkgebied.

Optimale zone($0.5 < W <= 1$)**:** inductor wordt gebruikt in zijn werkgebied.

Out Of Range zone($1 < W <= 1.1$)**:** inductor werkt boven zijn specificaties.

Destructieve zone($W > 1.1$)**:** inductor gaat defect of werkt buiten zijn werkgebied.

D e activiteit A van een inductor kunnen we definiëren door de volgende formule:

$$A(t) = \frac{\frac{\int_{t0}^{t} E_d(t)dt}{(t-t0)} + \frac{dE(t)}{dt}}{E_m}$$

Z olang de energie, opgenomen door de inductor, binnen het werkgebied van de inductor blijft zal de activiteit tussen 0 en 1 gesitueerd zijn. Indien de stroom door de inductor plotseling snel toeneemt zal de activiteit snel stijgen. Ook zal hierdoor het de opgenomen energie stijgen. De spanning opgewekt door de inductor zal tegengesteld zijn aan de stroomwijziging. Indien de opgenomen energie de maximale op te nemen energie overstijgt zal de inductor buiten zijn werkgebied belanden. De Activiteit kan ingedeeld worden in 4 zones:

non-actieve zone($0 <= A < 0.1$)**:** inductor heeft toegelaten energie opgenomen en laadt/ontlaadt niet sneller dan toegelaten.

sub-actieve zone($0.1 <= A < 0.3$)**:** inductor heeft toegelaten opgenomen energie en/of geeft of neemt minder dan de toegelaten energie op of af.

actieve zone($0.3 <= A <= 1$)**:** inductor neemt aangeraden energie op en/of bezit deze.

destructieve zone($A > 1$)**:** inductor neemt teveel energie op

7.4.1 implementatie

I n volgende sectie worden de aanpassingen toegelicht ten opzichte van het basis inductormodel van NGSpice.

ind.patch

D eze patch zorgt ervoor dat het basismodel van de simulator (/src/libspice/devices/ind) wordt aangepast en voorzien van de nodige extra parameters en functies zodat men de emoties van de component ook kan laten berekenen.

H iervoor zijn er 5 parameters (waarvan 4 zichtbaar in de simulator) toegevoegd. De rated_current parameter is nodig voor het berekenen van maximaal op te nemen energie en bepaalt dus de activity en working_zone parameters. Deze laatste 2 parameters geven de emotionele toestand van de inductor aan. De vijfde parameter die niet zichtbaar is in de simulator wordt gebruikt om de activity parameter te bereken bij een transiënt analyse. Deze aanpassingen bevinden zicht in het ind.c bestand.

```
IFparm INDpTable[] = {
...
IOPP( "rated_current",          IND_RATED_CURRENT,
        IF_REAL,    "rated␣current"),
IOP(   "activity",      IND_ACTIVITY,        IF_STRING,
      "activity␣emotion"),
IOP(   "work_zone",     IND_WORKING_ZONE,
   IF_STRING,    "working␣zone␣emotion"),
IOPU(   "energy_sum",           IND_ENERGY_SUM,
   IF_REAL,    "activity␣energy␣sum"),
IOPU( "energy",         IND_ENERGY,          IF_REAL,
      "max␣energy␣density")
}
```

B ij de instantiestructuur is de activiteit en werkzone parameter toegevoegd van het type Array van char. De som van de reeds opgenomen/afgestane energie(INDenergy) en de standaard stroom worden als variabelen toegevoegd. Verder worden er ook een aantal defines voor deze parameters gegevens zodat de simulator de verschillende variabelen kan opvragen per nummer. De 'Given' parameters worden gebruikt bij het inlezen van de parameters om aan te duiden welke parameters door de gebruiker zijn meegegeven. Deze toevoeging gebeurt in het inddefs header bestand.

```
typedef struct sRESinstance {
        ...
    unsigned INDratedCurrentGiven      :1;
    unsigned INDactivityGiven      :1;
    unsigned INDworkingZoneGiven      :1;
    unsigned INDenergyGiven      :1;


    double INDenergySum;
    double INDenergy;
    double INDratedCurrent;
    char *INDactivity;
    char *INDworkingZone;
}
...
/*emotions*/
#define IND_ENERGY 15
#define IND_ACTIVITY 16
#define IND_WORKING_ZONE 17
#define IND_ENERGY_SUM 18
#define IND_RATED_CURRENT 19
```

I n het indask bestand worden de 4 extra parameters toegevoegd. Deze functies worden aangeroepen als men de gegevens over de component opvraagt via het show of show all commando. De maximale stroom door de inductor wordt hier gewoon uitgelezen en de energie die zich in de condensator bevindt wordt berekend. De functies voor het weergeven van de werkzone alsook de activiteit zijn gelijkaardig aan deze voor de weerstand.

```
    case IND_ENERGY:
        value->rValue = (here->INDratedCurrent
        *here->INDratedCurrent*here->INDinduct)/2;
        return(OK);
```

```
    case IND_RATED_CURRENT:
         value->rValue = here->INDratedCurrent;
         return(OK);

case IND_WORKING_ZONE:
if (ckt->CKTrhsOld) {

workingZone = *(ckt->CKTrhsOld + here->INDbrEq) *
     here->INDinduct* (1/2) *
     ( *(ckt->CKTrhsOld + here->INDbrEq));
if(workingZone<0){
  workingZone*=-1;
}
workingZone=workingZone/here->INDenergy;
workingZone *= here->INDm;
if(workingZone<0.5){...

case IND_ACTIVITY:

if(ckt->CKTrhsOld){
 energy = ( *(ckt->CKTrhsOld + here->INDbrEq)) *
here->INDinduct * (1/2) *
 ( *(ckt->CKTrhsOld + here->INDbrEq));
                            energy *= here->INDm;

if(ckt->CKTcurrentAnalysis &  (DOING_AC | DOING_TRAN)){
   activity=here->INDenergySum/ckt->CKTtimePoints[ckt->CKTtimeIndex];
   activity+=energy/ckt->CKTdeltaList[ckt->CKTtimeIndex];
   activity/=(here->INDenergy*here->INDm);// normalize
   }else{ //dc or op
      activity=energy/(here->INDenergy*here->INDm);
   }
   if(activity<0.1){...
```

I n het indload bestand is enkel het berekenen van de som van het opgenomen en afgestaan vermogen toegevoegd. Dit zorgt ervoor dat bij elke stap de huidige lading in de inductor wordt bijgehouden.

```
if(*(ckt->CKTrhsOld + here->INDbrEq)<0){
            here->INDenergySum-=*(ckt->CKTrhsOld +
            here->INDbrEq)* *(ckt->CKTrhsOld +
            here->INDbrEq)*here->INDinduct/m *1/2
            *ckt->CKTdeltaList[ckt->CKTtimeIndex];
```

```
            }else{
                  here->INDenergySum+=*(ckt->CKTrhsOld +
                  here->INDbrEq)* *(ckt->CKTrhsOld +
                  here->INDbrEq)*here->INDinduct/m *1/2
                  *ckt->CKTdeltaList[ckt->CKTtimeIndex];
}
```

In het bestand dat de parameters beschrijft (indparameter.c) is de rated current parameter toegevoegd. Dit zorgt ervoor dat het mogelijk is de inductor de maximale stroom mee te geven. Bij toevoeging van deze parameter wordt meteen de corresponderende maximaal op te nemen energie berekend.

```
case IND_RATED_CURRENT:
 here->INDratedCurrent = value->rValue;
 here->INDenergy = (here->INDratedCurrent*
 here->INDratedCurrent*here->INDinduct)*1/2;
 here->INDratedCurrentGiven=TRUE;
break;
```

Hoofdstuk 8

Voorbeeld

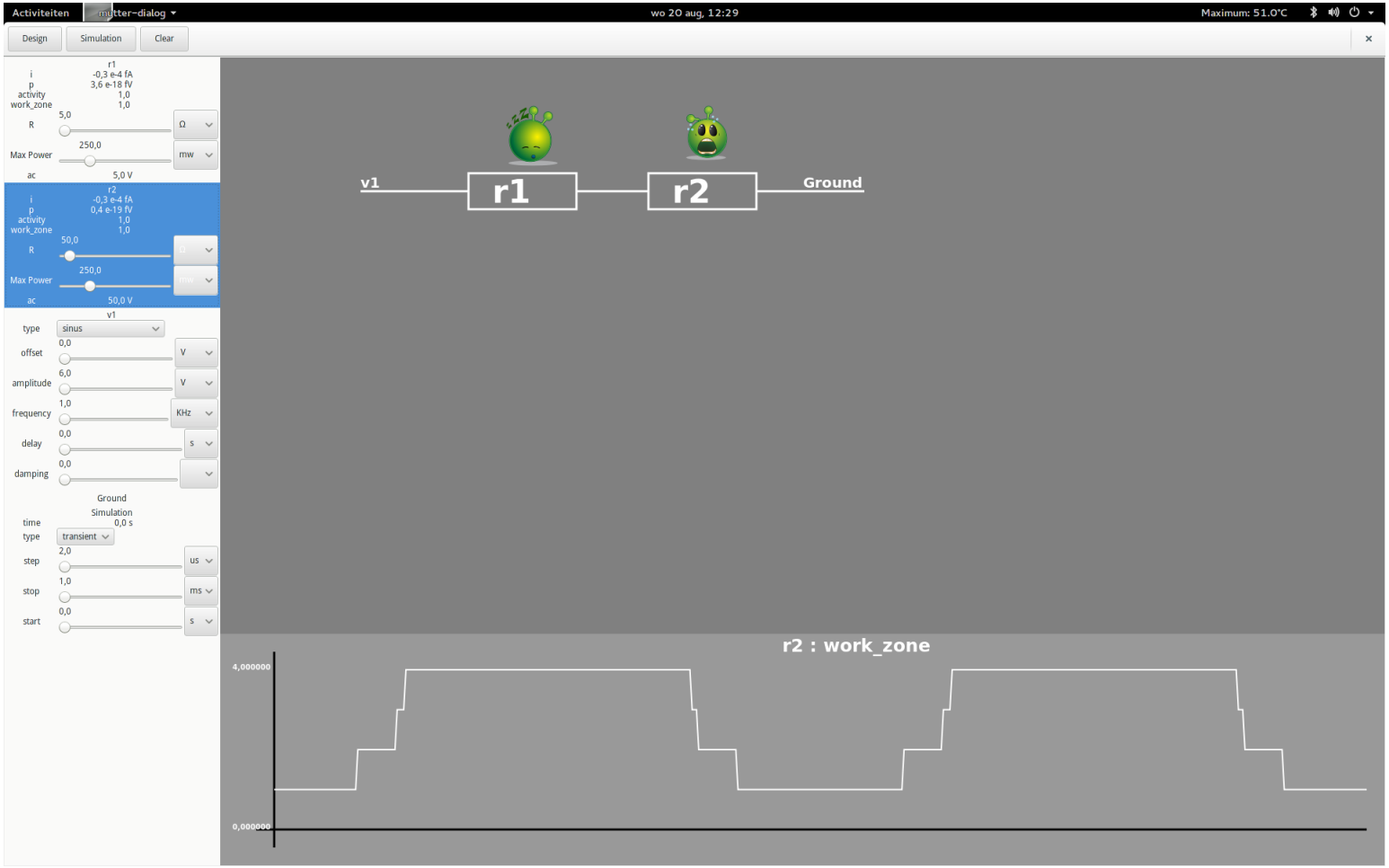
In dit hoofdstuk word een voorbeeldschakeling in de simulator gesimuleerd en beschreven.

8.1 spanningsdeler

I n deze simulatie simuleren we een spanningsdeler die wordt gevoed door een sinusvormige spanningsbron. We maken eerst het schema aan door de juiste componenten met elkaar te verbinden, ook voeren we de juiste simulatiemethode in zoals te zien is in volgende afbeelding.

D e bedoeling van het circuit is om het voltage uit de spanningsbron (sinus met amplitude 6 volt) te doen dalen tot 90 procent van de ingangsspanning van de spanningsbron. Hiervoor zou men een weerstand van 10 ohm en een weerstand van 90 ohm kunnen gebruiken.
$U_2 = U * \frac{R_2}{R_1+R_2}$ Hieruit blijkt dat de schakeling in staat is om het voltage correct te verlagen. Indien we echter de simulatie uitvoeren blijkt dat de tweede weerstand echter zeer angstig is wat erop wijst dat deze component zal falen. Dit komt omdat er maximaal 324 mW aan energie dient gedissipeert te worden door de weerstand. Deze weerstand is echter in staat om maximaal 250 mW aan warmte te dissiperen.

O m deze situatie op te lossen kan de student dan de weerstanden verhogen naar bijvoorbeeld 10 KOhm en 90 KOhm. Dit zorgt ervoor dat de 2 componenten een emotie krijgen van diepe slaap. Dit zorgt ervoor dat deze componenten haast niet belast worden en weinig vermogen consumeren wat aan te raden is bij een spanningsdeler.



Figuur 8.1 – voorbeeld spanningsdeler

Hoofdstuk 9

Conclusies

H et schrijven van emotionele modellen in de XSpice extensie voor NGSpice gaf na integraties niet de gewenste gegevens terug die nodig waren voor het gebruik door de software. Hierbij was vooral het gebrek aan het kunnen uitlezen van de parameters tijdens elke stap een probleem. Teneinde dit probleem op te lossen is er gekozen voor het implementeren van de emotionele modellen in de C code van de(voortgezette) Spice simulator van NGSpice. Dit bleek na enig onderzoekwerk beter haalbaar dan gebruik te maken van de XSpice extensie. De integratie van de emoties in de kern van de NGSpice simulator leiden ertoe dat de gewenste informatie op elke ogenblik kan worden opgevraagd. Het nadeel bij deze methode is dat de source code dient gepatcht te worden voor compilatie door het gegeneerde diff bestand. De keuze om de NGSpice simulator te gebruiken ten nadele van de Gnucap simulator was gebaseerd op het feit dat de Xspice extensie dynamisch toevoegen van modellen ondersteunt en dat deze simulator beter in staat was om alle Spice3 en Spice2 simulaties uit te voeren. Aangezien door de beperkte functionaliteit van de Xspice extensie deze niet meer gebruikt kan worden in dit project was het mogelijk beter geweest om de Gnucap simulator te integreren in het project.

H et gebruik van de NGSpice simulator zorgt ervoor dat alle functies die beschikbaar zijn in de simulatiesoftware nog niet beschikbaar zijn in de grafische interface en software.

D e keuze voor het ontwikkelen van de gebruikersinterface software is vooral gestuurd door de keuze om het project zowel als desktop applicatie als webapplicatie beschikbaar te kunnen maken. De initiële keuze was hierbij gericht op de Qt toolkit waarbij zoals beschreven in de inleiding de Emscripten compiler

kon gebruikt worden voor het compileren van de software naar javascript zodat het in de webbrowser kan worden uitgevoerd. Hiervoor bleek echter een zwaar gemodificeerde versie van de QT toolkit benodigd te zijn. Aangezien dit een fork van de toolkit betreft zal het onderhouden en bijwerken van deze versie zeer veel tijd en werkuren omvatten. Zodoende is besloten om deze optie als niet haalbaar op langere termijn te bestempelen.

D e uiteindelijke software maakt voor de grafische gebruikersomgeving gebruik van de (originele) GTK3 toolkit. Dit zorgt ervoor dat er geen fork moet onderhouden worden. Ook beschikt deze toolkit over de Broadway Daemon die het mogelijk maakt om de grafische gebruikersomgeving te gebruiken in een webbrowser.

D e keuze voor Vala als programmeertaal is er omdat deze allereerst compileert naar C code. Dit zorgt ervoor dat er geen extra framework of virtuele machine dient geïnstalleerd te worden voor het draaien van het programma. Ook is de Vala programmeertaal in gebruik en syntax gelijkaardig aan de C# programmeertaal wat ervoor zorgt dat er geen zwaar leerproces is indien men kan programmeren in C#. Ook is het een high-level programmeertaal en zorgt dit ervoor dat de code zelf overzichtelijker en duidelijker is. De taal heeft als bijkomend voordeel dat het in tegenstelling tot bijvoorbeeld Javascript gebruikt maakt van sterke type controles bij compilatie. Het gebruik van de vele C bibliotheken door middel van VAPI bestanden die enkel nodig zijn bij compilatie zorgt ervoor dat deze bibliotheken native kunnen gebruikt worden.

B ij de integratie van de emoties in de modellen van de simulator is gebleken dat de XSpice extensie niet dezelfde functionaliteiten biedt als het schrijven van de modules in C code. De integratie van de modellen in de C code zorgen ervoor dat deze moeten worden toegevoegd worden bij compilatie van de gedeelte NGSpice bibliotheek. Een voordeel van de NGSpice simulator is echter dat er een C interface is voorzien waardoor het programma directe toegang krijgt tot de simulator. Dit zorgt ervoor dat we bij elke stap van de simulatie extra informatie kunnen opvragen. Aangezien deze simulator niet gebaseerd is op het GLib Objectsysteem was het nodig om een VAPI bestand handmatig te schrijven zodat deze kon worden gekoppeld aan de Vala code.

D e ontwikkelde Elektrosim software maakt het mogelijk om een circuit aan te maken en deze te simuleren. Hierbij kan men de parameters aanpassen en (bijna) meteen de invloed van het aanpassen van de parameters op het circuit

bekijken. Het toevoegen van emoties aan de componenten maakt het mogelijk voor de gebruiker om snel te detecteren of een component beschadigd wordt of deze binnen zijn bereik werkt. Dit zorgt ervoor dat de gebruiker het circuit en de componenten kan aanpassen om ervoor te zorgen dat deze allen binnen hun bereik werken.

H et programma is modulair opgebouwd zodat de verschillende delen kunnen worden vervangen of in andere programma's kunnen worden gebruikt. Aangezien de GTK toolkit niet voorziet in een grafiek component moest deze echter zelf worden toegevoegd. Het toevoegen van extra componenten gebeurt door eenvoudig over te erven van de basis Component klasse. Door invulling van 4 methoden kan men eenvoudig een component toevoegen aan het programma.

H et gebruik van autotools zorgt ervoor dat het programma kan worden gecompileerd op elke systeem dat beschikt over een Bourne Shell en de make software. Het gebruik van de GTK toolkit zorgt er verder voor dat het programma (indien nodig) kan draaien op zowel Linux, Mac en Windows. Door gebruik te maken van de Broadway backend voor GDK kan men de software gebruiken in elke HTML5 compatibele browser op elk platform.

Hoofdstuk 10

Toekomstig werk

H et programma dient nog uitgebreid te worden met de componenten die beschikbaar zijn in NGSpice bibliotheek. Ook zou het aanmaken van nieuwe componenten in het programma een handige toevoeging zijn. De gebruikersomgeving zou verder configureerbaar dienen te zijn zodat ook het gebruik van het programma op een tablet handiger wordt. Momenteel bestaat de gebruikersinterface uit 1 groot scherm waarin zowel de componenten, het circuit als de grafiek zijn geïntegreerd. Voor het gebruik op apparaten met een kleinere oppervlakte zou het beter zijn dat men kan schakelen tussen circuit en bijvoorbeeld de grafiek.
Het weergeven van meerdere parameters op eenzelfde grafiek dient nog geïmplementeerd te worden.

H et programma zou later kunnen uitgebreid worden om gebruik te maken van echte meetapparatuur en een echt circuit. Zo zou een fysiek circuit eerst kunnen gesimuleerd worden waarna deze ook effectief kan worden getest. Hierbij kunnen dan de meetgegevens ingevoerd worden naast de simulatiegegevens. Dit zou het ook mogelijk maken om een circuit te testen vanuit de thuisomgeving zonder beschikking te hebben over geavanceerde meetapparatuur en componenten.

h et toevoegen van de Gnucap simulatie software door een interface te maken als plugin voor Gnucap. Dit zou het mogelijk maken om dynamisch componenten toe te voegen aan de simulator (zoals initieel gepland was voor NGSpice). Er is ook sprake van de ontwikkeling van een Apache plugin voor de Gnucap simulator zodat deze kan worden aangeroepen vanuit webpagina's.

Bibliografie

[1] *asm.js: A Low Level, Highly Optimizable Subset of JavaScript for Compilers | Badass JavaScript*. URL: http://badassjs.com/post/43420901994/asm-js-a-low-level-highly-optimizable-subset-of (bezocht op 23-08-2014).

[2] *automake: amhello's configure.ac Setup Explained*. URL: http://www.gnu.org/software/automake/manual/html_node/amhello_0027s-configure_002eac-Setup-Explained.html (bezocht op 15-08-2014).

[3] A. Braeken e.a. « E-Learning Platform with SPICE web service ». In: *World Academy of Science, Engineering and Technology* 65 (2012), p. 454–458. URL: http://www.etro.vub.ac.be/Publications/ShowPub.asp?PubID=P7000000000216195 (bezocht op 23-08-2014).

[4] *cairographics.org*. URL: http://cairographics.org/ (bezocht op 19-08-2014).

[5] *Chapter 1: A brief introduction to the GNU Autotools*. URL: http://www.freesoftwaremagazine.com/articles/brief_introduction_to_gnu_autotools (bezocht op 14-08-2014).

[6] F.L Cox e.a. *XSpice CodeModel SubsysInterface Design Document*. URL: http://users.ece.gatech.edu/mrichard/Xspice/XSpice_CodeModelSubsysInterfaceDesign.pdf (bezocht op 19-08-2014).

[7] *Demos - emscripten-qt - Redmine*. URL: http://vps2.etotheipiplusone.com:30176/redmine/projects/emscripten-qt/wiki/Demos.

[8] *EasyEDA - Web-Based EDA, schematic capture, spice circuit simulation and PCB layout Online*. URL: http://easyeda.com/ (bezocht op 19-08-2014).

[9] *Energy Stored in an Inductor*. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/electric/indeng.html (bezocht op 20-08-2014).

[10] *Falling Cubes | Demo Studio | MDN*. URL: https://developer.mozilla.org/en-US/demos/detail/falling-cubes (bezocht op 23-08-2014).

[11] *Falstad Circuit Simulator Applet*. URL: http://www.falstad.com/circuit/ (bezocht op 19-08-2014).

[12] *Filesystem Hierarchy Standard*. URL: http://www.pathname.com/fhs/ (bezocht op 14-08-2014).

[13] *GeckoCIRCUITS*. URL: http://www.gecko-simulations.com/geckocircuits.html (bezocht op 19-08-2014).

[14] *GNU build system*. In: *Wikipedia, the free encyclopedia*. Page Version ID: 604437922. 22 jul 2014. URL: http://en.wikipedia.org/w/index.php?title=GNU_build_system&oldid=604437922 (bezocht op 14-08-2014).

[15] *GNU coding standards - GNU Project - Free Software Foundation*. URL: http://www.gnu.org/prep/standards/ (bezocht op 14-08-2014).

[16] *GNU Make Manual - GNU Project - Free Software Foundation (FSF)*. URL: http://www.gnu.org/software/make/manual/ (bezocht op 14-08-2014).

[17] *GSoap: SOAP and XML Web services for Qt apps - Nokia Developer Wiki*. URL: http://developer.nokia.com/community/wiki/GSoap:_SOAP_and_XML_Web_services_for_Qt_apps (bezocht op 23-08-2014).

[18] *GTK+ 3 Reference Manual: broadwayd*. URL: https://developer.gnome.org/gtk3/3.10/broadwayd.html (bezocht op 19-08-2014).

[19] *GTK+ 3 Reference Manual: Part GTK+ Overview*. URL: https://developer.gnome.org/gtk3/stable/gtk.html (bezocht op 19-08-2014).

[20] *Gtk+ Kick-Start Tutorial for Vala on Vimeo*. URL: http://vimeo.com/9617309 (bezocht op 19-08-2014).

[21] Thomas L.Quarles. « Analysis of Performance and Convergence Issues for Circuit Simulation ». In: (1989), p. 1–142. URL: http://diyhpl.us/~bryan/papers2/Analysisofperformanceandconvergence.pdf.

[22] *Magnetic fields and inductance : Inductors - Electronics Textbook*. URL: http://www.allaboutcircuits.com/vol_1/chpt_15/1.html (bezocht op 20-08-2014).

[23] Peter Miller. « Recursive make considered harmful ». In: *AUUGN Journal of AUUG Inc* 19.1 (1998), p. 14. URL: http://www.unix-ag.uni-kl.de/svn/kbibtex/kbibtex/tags/release-0.1/admin/unsermake/doc/auug97.pdf (bezocht op 23-08-2014).

[24] Paolo Nenzi en Holger Vogt. « Ngspice Users Manual Version 26 (Describes ngspice-26 release version) ». In: (2014). URL: http://distfiles.gentoo.org/distfiles/ngspice-26-manual.pdf (bezocht op 23-08-2014).

[25] *NGSPICE Online simulator*. URL: http://www.ngspice.com/ (bezocht op 19-08-2014).

[26] *Overview LLVM 3.6 documentation*. URL: http://llvm.org/docs/ (bezocht op 23-08-2014).

[27] *Projects/GObjectIntrospection - GNOME Wiki!* URL: https://wiki.gnome.org/Projects/GObjectIntrospection (bezocht op 19-08-2014).

[28] *Projects/Vala/ValaForCSharpProgrammers - GNOME Wiki!* URL: https://wiki.gnome.org/Projects/Vala/ValaForCSharpProgrammers (bezocht op 19-08-2014).

[29] *The GTK+ Project*. URL: http://www.gtk.org/ (bezocht op 19-08-2014).

[30] Abdellah Touhafi. « A Novel Emoticons-based Model to Describe electronic circuits behavior in a hedonics way ». In: *International Conference On Engineering Education and Research* (2013), p. 1–5.

[31] Carl Worth. *cairo_ddc2005*. 2005. URL: http://cworth.org/~cworth/papers/cairo_ddc2005/cairo_ddc2005.pdf (bezocht op 23-08-2014).

Hoofdstuk 11

Bijlagen

11.1 Gant-tabel

WBS	Naam	Start	Finish	Werk
1	**Literatuurstudie**	**Oct 1**	**Oct 28**	**14d**
1.1	Emscripten-Qt	Oct 1	Oct 7	3d
1.2	flash en Air	Oct 8	Oct 14	3d
1.3	spice simulator	Oct 15	Oct 16	2d
1.4	Gsoap qt en webservice spice	Oct 21	Oct 21	1d
1.5	spice simulatie +testen	Oct 22	Oct 22	1d
1.6	emoticons	Oct 23	Oct 28	2d
1.7	beschrijven qt architectuur	Oct 23	Oct 28	2d
1.8	rapport 1	Oct 28	Oct 28	
2	**Qt simulator**	**Oct 29**	**Nov 20**	**11d**
2.1	opzetten git-repos+project	Oct 29	Oct 30	2d
2.2	Initieele GUI	Nov 4	Nov 6	3d
2.3	Component+ drag and drop	Nov 11	Nov 13	3d
2.4	onderzoek omvomen Qt naar c+onderzoek alternatieve toolkit	Nov 18	Nov 20	3d
3	**port van Qt naar GTK**	**Nov 25**	**Jan 22**	**15d**
3.1	broadway backend test	Nov 25	Nov 27	3d
3.2	schrijven rapport GTK3	Dec 2	Dec 4	3d
3.3	rapport 2	Dec 4	Dec 4	
3.4	ontwikkelomgeving kiezen+opzetten project	Dec 9	Dec 11	3d
3.5	testen van proefprogrammas en volgen tutorials	Dec 16	Dec 18	3d
3.6	porten van project van qt naar gtk	Jan 20	Jan 22	3d

4	**GTK3 implementatie**	**Jan 27**	**May 14**	**100d 6h**
4.1	opzetten project	Feb 3	Feb 3	1d
4.2	Opstellen layout user interface	Feb 4	Feb 5	2d
4.3	toevoegen simulatie weergave	Feb 10	Mar 11	15d
4.4	Toevoegen basiscomponentklasse	Mar 12	Mar 12	1d
4.5	Toevoegen parameters	Mar 17	Apr 1	8d
4.6	Toevoegen weerstand	Apr 2	Apr 15	5d 6h
4.7	interactie met ngspice bibliotheek	Apr 15	Apr 23	4d
4.8	toevoegen condensator	Apr 23	Apr 29	2d
4.9	toevoegen inductor	Apr 29	May 5	2d
4.10	toevoegen ground en spanningsbron	Apr 15	Apr 28	5d
4.11	toevoegen grafiek	Apr 29	May 12	5d
4.12	debuggen en testen	Jan 27	May 14	50d
5	**Webinterface**	**May 19**	**Jun 10**	**11d**
5.1	webpagina en registratie databank	May 19	May 27	5d
5.2	verbinding met elektrosim via websockets	May 28	Jun 10	6d
6	**NGSpice Modellen**	**Apr 23**	**Jun 9**	**19d**
6.1	implementatie weerstandsmodel	Apr 23	May 12	7d
6.2	uitwerken capaciteitsmodel	May 12	May 19	3d
6.3	implementatie capaciteitsmodel	May 19	May 26	3d
6.4	uitwerken inductormodel	May 26	Jun 2	3d
6.5	implementatie inductormodel	Jun 2	Jun 9	3d
7	**thesis**	**May 19**	**Aug 25**	**40d**
7.1	schrijven thesis	May 19	Aug 18	40d
7.2	indienen titelblad	Aug 18	Aug 18	
7.3	indienen thesistekst	Aug 25	Aug 25	