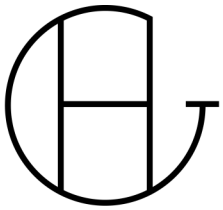


HoGent

Academiejaar 2012–2013

Geassocieerde faculteit Toegepaste Ingenieurswetenschappen
Valentin Vaerwyckweg 1 – 9000 Gent

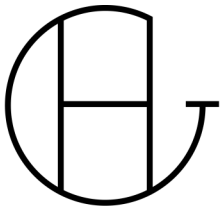
Migratiestrategieën van C++ naar C#

Masterproef voorgedragen tot het behalen van het diploma van
Master in de industriële wetenschappen: informatica

Martijn SAELENS

Promotoren: *Leen BROUNS*
Joris VAN MALDEGHEM (ICORDA)

Begeleider: *Tom EERAERTS (ICORDA)*



HoGent

Academiejaar 2012–2013

Geassocieerde faculteit Toegepaste Ingenieurswetenschappen
Valentin Vaerwyckweg 1 – 9000 Gent

Migratiestrategieën van C++ naar C#

Masterproef voorgedragen tot het behalen van het diploma van
Master in de industriële wetenschappen: informatica

Martijn SAELENS

Promotoren: *Leen BROUNS*
Joris VAN MALDEGHEM (ICORDA)

Begeleider: *Tom EERAERTS (ICORDA)*

Woord vooraf

De informaticawereld is een zichzelf steeds vernieuwende stroom van technologieën. Altijd doen er nieuwe technologieën hun intrede en raken andere verouderd. Eén van deze ondertussen verouderde technologieën is de programmeertaal Visual C++ 6.0. Tegenwoordig wordt Visual C++ 6.0 nog zelden gebruikt en geven de meeste bedrijven de voorkeur aan de meer recente technologieën zoals Visual C++ 2010 en Visual C# 2010.

De voornaamste reden is dat Visual C++ 6.0 niet meer door Microsoft wordt ondersteund. Via Visual C++ 6.0 kan er dus geen beroep gedaan worden op recente technologieën. Ook zijn er steeds minder werknemers vertrouwd met Visual C++ 6.0. Nieuwe technologieën kunnen wel aangesproken worden met Visual C++ 2010, één van de recente versies van Visual C++.

Visual C# geniet de voorkeur. Het is zeer recent, wordt constant geüpdatet en biedt meer functionaliteit aan. Visual C# code is duidelijker en heeft een minder steile leercurve dan Visual C++. 6.0. Zo wordt onderhoud van projecten eenvoudiger en minder foutgevoelig.

Voor het opwaarderen naar een nieuwe versie van Visual C++ en het omzetten van Visual C++ naar Visual C# bestaan er op dit moment bijna geen hulpmiddelen. Buiten upgradefuncties in Visual Studio, enkele simpele C++ naar C# vertalers en aanwijzingen op Microsoft Developer Network (MSDN) berust dit werk op manuele handelingen. Het doel van deze masterproef is om een zo generiek en geautomatiseerd mogelijke methode aan te bieden om projecten geschreven in Visual C++ 6.0 op te waarderen naar Visual C++ 2010 en daarna om te zetten naar Visual C#. Vaak is dit een vervelend, foutgevoelig werk van lange adem. Door het aanbieden van een generieke geautomatiseerde methode kan dit werk sterk verlicht worden.

Om deze methode op te stellen wordt vooral gesteund op opgedane ervaring bij het manueel opwaarderen van een Visual C++ 6.0 project en op de informatie beschikbaar in de MSDN.

Graag wil ik nog enkele personen bedanken die mij gesteund, geholpen en begeleid hebben bij het tot stand komen van deze masterproef. Als eerste wil ik graag mevr. Leen Brouns bedanken. Telkens als ik vragen of onduidelijkheden had, kon ik steeds bij haar terecht voor uitgebreide constructieve feedback. Daarnaast wil ik ook dhr. Joris Van Maldeghem bedanken voor het aanbieden van een stageplaats op ICORDA, voor de mogelijkheid om hun software te mogen gebruiken voor onderzoek en voor de steun bij het volbrengen van de opdracht. Ook een speciaal dankwoord aan dhr. Tom Eeraerts voor de begeleiding op ICORDA. Ik wil hem bedanken omdat hij me telkens opnieuw kennis liet maken met nieuwe technologieën, voor de hulp bij het uitspitten van talloze debug assertions en unhandled exceptions, en me leerde om efficiënt te debuggen.

Abstract

Veel bedrijven werken nog steeds met programma's geschreven in de verouderde programmeertaal Visual C++ 6.0. Omdat de verouderde versie niet meer wordt ondersteund, en zelf ook geen aansluiting toelaat op nieuwere technologieën, wordt er doorgaans uitgeweken naar Visual C++ 2010 en Visual C#. In deze masterproef wordt eerst het opwaarderen van Visual C++ 6.0 naar Visual C++ 2010 behandeld, daarna het converteren van Visual C++ 2010 naar Visual C#.

Aan de hand van ervaringen opgedaan bij het manueel opwaarderen van vijf proefprojecten werd er gezocht naar een generieke geautomatiseerde methode voor het opwaarderen van (de verschillende versies van) Visual C++ 6.0 naar Visual C++ 2010.

Voor het converteren van Visual C++ 2010 naar Visual C# werden er twee strategieën weerhouden door het opstellen van een proof of concept: enerzijds unmanaged Visual C++ projecten als managed instellen, anderzijds gebruik maken van aanwezige Component Object Model (COM) interfaces. Er werd ook een tool ontwikkeld om de Graphical User Interface (GUI) uit Visual C++ projecten om te zetten naar Visual C# Windows Forms bestanden.

Tot slot werd er nagegaan in hoeverre het ontwikkelen van een vertaler van Visual C++ naar Visual C# zou bijdragen aan de oplossing. Hierbij werd de parser generator ANother Tool for Language Recognition (ANTLR) ingezet.

Inhoudsopgave

Afkortingen

ANTLR ANother Tool for Language Recognition. v, 8, 11, 115–117, 120, 122, 124, 125, 162

API Application Programming Interface. 138

ASP Active Server Pages. 58

AST Abstract Syntax Tree. 115

ATL Active Template Library. 13, 87, 121, 133

CLI Common Language Interface. 58, 59, 61, 63, 66

CLR Common Language Runtime. 57–59

COM Component Object Model. v, 60, 64, 72–76, 79–81, 83–85, 124, 161

CPU Central Processing Unit. 57, 58

CRL C Runtime Library. 13

CRT C Run-Time Libraries. 148

CTS Common Type System. 59

GUI Graphical User Interface. v, 11, 15, 16, 34, 35, 38, 39, 41, 46, 87, 101, 114, 124

GUID Globally Unique Identifier. 74, 75, 138

IDE Integrated Development Environment. 25, 40, 54

IO Input/Output. 101

IPC Inter-Process Communication. 85

JIT Just In Time. 57, 60

LINQ Language Integrated Query. 67, 68, 70

MFC Microsoft Foundation Class. 13, 21, 70, 80, 87, 92, 100, 109, 121, 132, 133

MIDL Microsoft Interface Definition Language. 81

MSDN Microsoft Developer Network. iv, 13, 15, 23, 64, 117, 118, 125

MSIL Microsoft Intermediate Language. 57, 58, 60

MTS Microsoft Transaction Server. 138

RAM Random-Access Memory. 146

STL Standard Template Library. 13, 121, 137

TFS Team Foundation Server. 20, 21

TPL Task Parallel Library. 35

UUID Universal Unique Identifier. 81

WCF Windows Communication Foundation. 58

WPF Windows Presentation Foundation. 34, 58, 66, 77, 87, 101

XML Extensible Markup Language. 24, 26, 35–38, 88, 93, 94, 100, 103, 107, 124

YACC Yet Another Compiler-Compiler. 116

Indeling

De scriptie is ingedeeld in twee grote delen:

Opwaarderen van Visual C++ 6.0 naar Visual C++ 2010

Dit deel behandelt het opwaarderen van programma's geschreven in Visual C++ 6.0 naar Visual C++ 2010.

Hoofdstuk 1 geeft een korte inleiding op bestaande versies van Visual C++.

Hoofdstuk 2 overloopt de stadia van het onderzoek. Dit gaat hand in hand met de ontwikkeling van oplossingen.

Hoofdstuk 3 bundelt de moeilijkheden die de kop opstaken bij het opwaarderen.

In de laatste twee hoofdstukken van dit deel, komen de uiteindelijk gevonden hulpmiddelen bij opwaardering aan bod.

Hoofdstuk 4 beschrijft hoe Visual Studio ingezet kan worden om het opwaarderingsproces te starten. Hier worden de bewerkingen gebundeld die reeds voorzien zijn in gekende software.

Hoofdstuk 5 beschrijft een applicatie die werd opgesteld in kader van deze masterproef. Deze toepassing is een hulpmiddel om, na de bewerkingen uit hoofdstuk 4, de opwaardering te vervolledigen.

Van Visual C++ 2010 naar Visual C# 2010

Als een programma is opgewaardeerd kan het worden geconverteerd naar Visual C#.

Hoofdstuk 6 geeft een overzicht van de impact van de conversie en mogelijke manieren waarop de conversie georganiseerd kan worden, zoals incrementele of gehele conversie.

Hoofdstukken 7 en 8 beschrijven elk een mogelijke strategie voor incrementele conversie, samen met de implementatie en resultaten van een proof of concept dat werd gemaakt in de loop van de masterproef.

Hoofdstuk 9 beschrijft een applicatie die ook in het kader van deze masterproef werd ontwikkeld voor het extraheren van de GUI uit een Visual C++ 2010 programma en dit converteert naar Visual C# Windows Forms.

Tot slot behandelt hoofdstuk 10 het onderzoek dat gevoerd werd naar het opstellen van een vertaler van Visual C++ naar Visual C#, gebruik makend van de parser generator ANTLR.

Deel I

Opwaarderen van Visual C++ 6.0 naar Visual C++ 2010

Hoofdstuk 1

Inleiding

Visual C++ en C++ zijn geen synoniemen. Als men het heeft over C++, dan heeft men het over de programmeertaal C++ ontwikkeld door Bjarne Stroustrup. Visual C++ daarentegen is een product van Microsoft.

> "*C++ is a language, and Visual C++ is a product.*" (Gregory, 2004)

Volgens de MSDN van Microsoft (2010d) bevat het product Visual C++ volgende componenten:

- De Visual C++ compiler tools voor zowel x64 als x86 platformen.
- De Visual C++ Libraries. Deze bevatten de Active Template Library (ATL), Microsoft Foundation Class (MFC) en standaard libraries (met onder andere de Standard C++ Library, Standard Template Library (STL) en C Runtime Library (CRL)).
- De Visual C++ ontwikkelingsomgeving. Beter bekend als Visual Studio.

Op het moment van schrijven (2012) heeft Microsoft net Visual C++ 2012 uitgebracht. Om redenen van compatibiliteit met projecten in het stagebedrijf ICORDA beperkt deze scriptie zich tot Visual C++ 2010.

Visual C++ bestaat al een geruime tijd. De eerste versie dateert van 1993. Met elke versie komt ook een versie van Visual Studio overeen. Elke versie krijgt een versienummer en benaming. De versienummers zijn oplopende getallen. De benaming bevat meestal het jaartal dat de versie uitkwam.

In dit document worden enkel de versies van 6.0 tot 2010 beschouwd, behalve Visual C++ .NET 2002. Visual C++ .NET 2002 wordt vermeden, omdat Visual C++ .NET 2003 eigenlijk meer een update is dan een nieuwe versie. Dit is duidelijk te zien aan het versienummer (tabel 1.1 op p.14). In de benaming van versies 2002 en 2003 duidt 'de term ".NET" erop dat het .NET-framework bij deze versies werd ingevoerd.

Een overzicht van de in dit document gebruikte en besproken versies bevindt zich in tabel 1.1 op p.14 (Voor de duidelijkheid worden in de tabel ook de gegevens van Visual C++ .NET 2002 en Visual C++ 2012 getoond).

Het doel van dit onderdeel is het opwaarderen van Visual C++ 6.0 projecten naar Visual C++ 2010. Dit verloopt niet altijd van een leien dakje en kan soms enorm veel werk met zich meebrengen. In dit onderdeel wordt nagegaan wat de grootste valkuilen zijn en of er

Naam	Versienummer	Jaar van uitgave
Visual C++ 6.0	6.0	1998
Visual C++ .NET (2002)	7.0	2002
Visual C++ .NET 2003	7.1	2003
Visual C++ 2005	8.0	2005
Visual C++ 2008	9.0	2008
Visual C++ 2010	10.0	2010
Visual C++ 2012	11.0	2012

Tabel 1.1: De verschillende Visual C++ versies vanaf Visual C++ 6.0 t.e.m. Visual C++ 2012

een generieke geautomatiseerde manier kan worden opgesteld om dit werk te verlichten. Om deze manier op te stellen zijn er manueel enkele Visual C++ 6.0 projecten opgewaardeerd naar Visual C++ 2010. Met de opgedane ervaring heeft werden er dan tools en documenten ontwikkeld die de conversie voor een groot deel versnellen en veraangenamen. In deze scriptie wordt uitleg en informatie gebundeld die gebruikt kan worden voor de conversie van andere programma's.

Hoofdstuk 2

Onderzoek

2.1 Inleiding en strategie

Het opwaarderen van Visual C++ 6.0 applicaties is een lineair proces. Om het eindpunt te bereiken moeten er steeds stappen worden gezet om bepaalde tussenpunten te kunnen bereiken. Deze punten zijn altijd dezelfde, maar kunnen soms op verschillende manieren bereikt worden. Dit kan vergeleken worden met een wandeltocht die een aantal vaste checkpoints bevat: men moet alle checkpoints passeren, maar de route die men van de ene naar de andere checkpoint volgt, ligt niet vast.

Een eerste opzet van het onderzoek kan als volgt geformuleerd worden: "Bepaal de checkpoints voor de opwaardering, evenals de methodes die de ontwikkelaars zo snel en eenvoudig mogelijk naar een volgend checkpoint brengen."

Het opzoeken van informatie geeft aan dat er niet veel oplossingen zijn voor dit probleem. Bijna alle gevonden gevallen zijn uiteindelijk overgegaan op manuele conversie. Er is weinig documentatie beschikbaar over beproefde methodes om Visual C++ projecten op te waarderen en bijna alle gevonden informatie steunt op artikels gevonden op het internet. Hierbij is de grootste informatiebron de MSDN. Die wordt dan ook veelvuldig gebruikt doorheen de scriptie.

Om de nodige punten, stappen en hinderpalen te bepalen zijn er een aantal applicaties manueel geconverteerd. De hierbij opgedane ervaring werd gebruikt om alternatieven te zoeken voor bepaalde stappen. Als er geen bestaande alternatieve methodes bestonden buiten manueel werk, werd er gekeken of er tools konden worden ontwikkeld om het werk te vereenvoudigen.

De behandelde projecten waren GandaFact, iBoMat, iPlant, iRent en TxKassa en werden aangeboden door het bedrijf ICORDA. De meeste van deze applicaties deelden grote delen gelijke code en steunden op externe libraries zoals Poco[1], Stingray[2], ... De applicaties werden na elkaar behandeld[3]. Dit liet toe om opgedane ervaringen te gebruiken in volgende

[1] De Poco libraries zijn een verzameling van open source libraries voor het ontwikkelen van platformonafhankelijke en netwerkgerichte C++ applicaties.
(`http://pocoproject.org/`)

[2] De Stingray libraries bieden GUI-componenten aan.
(`http://www.roguewave.com/products/stingray.aspx`)

[3] Soms werd er tijdens de behandeling van een applicatie overgeschakeld naar een andere applicatie omdat er gewacht moest worden op informatie van externe bedrijven, zoals bij TxKassa en iRent (hoofdstuk 2.2.2 op p.19).

applicaties. Ook konden zo zelfontwikkelde tools getest worden op de nog te behandelen applicaties. Indien dit succesvol bleek, werd het aandeel manueel werk voor die applicaties een stuk minder.

Zoals eerder besproken liggen er tussen Visual C++ 6.0 en Visual C++ 2010 vier versies: Visual C++ .NET 2002, Visual C++ .NET 2003, Visual C++ 2005 en Visual C++ 2008. Tussen elke twee opeenvolgende versies zijn er een aantal breaking changes in de Visual C++ specificatie en ontwikkelingstools. Dit betekent dat bij een rechtstreekse opwaardering van Visual C++ 6.0 naar Visual C++ 2010 er waarschijnlijk een groot aantal fouten zullen te wijten zijn aan verschillende breaking changes tussen 6.0 en 2003, 2003 en 2005, 2005 en 2008 en uiteindelijk tussen 2008 en 2010. Om op een gestructureerde manier manueel op te waarderen werd elke applicatie opgewaardeerd naar respectievelijk Visual C++ 2003, 2005, 2008 en 2010. Sommige projecten waren al geconverteerd naar Visual Studio 2005. Alle ontdekte fouten tijdens het opwaarderen en hun oplossingen zijn gedocumenteerd in bijlage B.1 op p.130.

Wat nu volgt, is een verslag van de gevolgde zoektocht - en kan programmeurs met dezelfde opdracht de opgedane ervaringen doorgeven.

2.2 Het opwaarderingsproces

2.2.1 De spits afbijten met TxKassa

Als eerste programma werd TxKassa gekozen. Er werd meteen geprobeerd het te openen in Visual Studio 2005. Via de automatische upgradetools in Visual Studio[4] was dit geen probleem. Helaas kwam hier al het probleem van externe libraries naar boven. De Stingray libraries werden niet gevonden door Visual Studio. Het bleek dat de folders opgegeven bij de optie “Include Files” in de “VC++ Directories” niet meer recursief werden doorlopen. De locaties van de headers *secall.h* en *gxall.h* moesten daardoor manueel worden toegevoegd.

Bij het runnen komen er al meteen *break errors* tevoorschijn . Onder de term break errors worden er hier zowel *unhandled exceptions* (figuur 2.1 op p.17) als *debug assertions* (figuur 2.2 op p.18) bedoeld. Debug assertions zijn fouten opgeworpen door controles in de Microsoft libraries. Ze wijzen erop dat er iets niet klopt en dat dit er waarschijnlijk voor zal zorgen dat het programma crasht. Ze treden enkel op tijdens het debuggen van het programma. Unhandled exceptions zijn foutmeldingen die opkomen als het programma crasht. Deze treden niet alleen op bij het debuggen, maar ook tijdens het gewoon runnen van het programma. Omdat deze foutmeldingen vaak hand in hand gaan[5] en wijzen op ernstige problemen, worden ze hier beide onder de noemer break errors geplaatst.

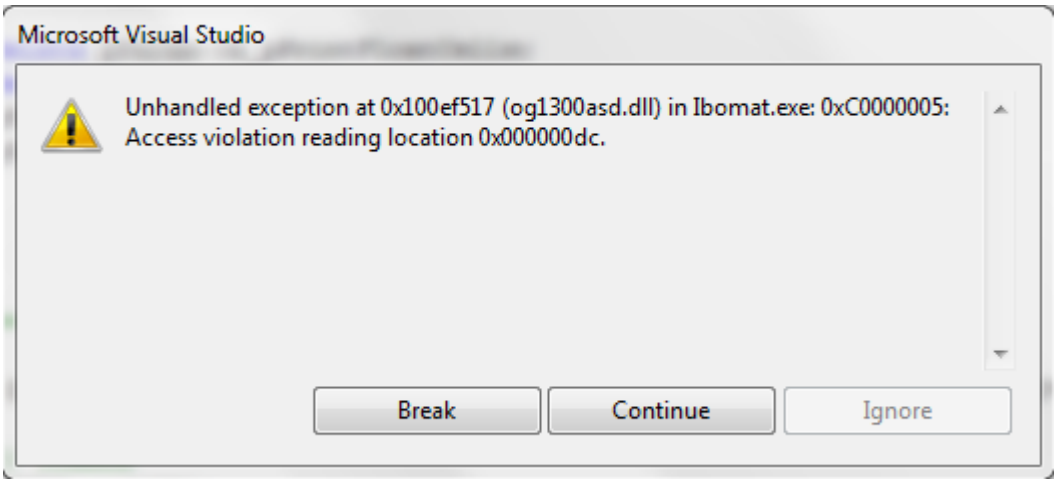
Het werd al snel duidelijk dat deze break errors te wijten waren aan de Stingray libraries. De break errors kwamen telkens boven als er een GUI-element zou worden getoond dat een Stringray Objective Grid component bevatte.

Omdat er een nieuwe versie van de Stingray libraries was uitgebracht, werd meteen besloten de nieuwe versie te implementeren[6]. Dit kon immers leiden tot de oplossing. De installatie

[4]Dit wordt uitgebreid besproken in hoofdstuk 4 op p.27

[5]In de meeste gevallen treden tijdens het debuggen eerst de debug assertions op gevolgd door de unhandled exception waarvoor de debug assertions waarschuwden.

[6]De huidige versie was 2006, de nieuwe 11. Tussen de twee versies is er bij een bepaald versie de overstap gemaakt van jaartallen naar versienummers om de versie aan te duiden.



Figuur 2.1: Een voorbeeld van het bericht van een unhandled exception.

van versie 11 was geautomatiseerd, in tegenstelling tot de 2006 versie. Hierdoor moesten de `include` statements aangepast worden, omdat de mappenstructuur was gewijzigd (listing 2.1 op p.17).

```
1 #include<gxall.h> // origineel
2 #include<secall.h> // origineel
3
4 #include<grid/gxall.h> // aangepast
5 #include<toolkit/secall.h> // aangepast
```

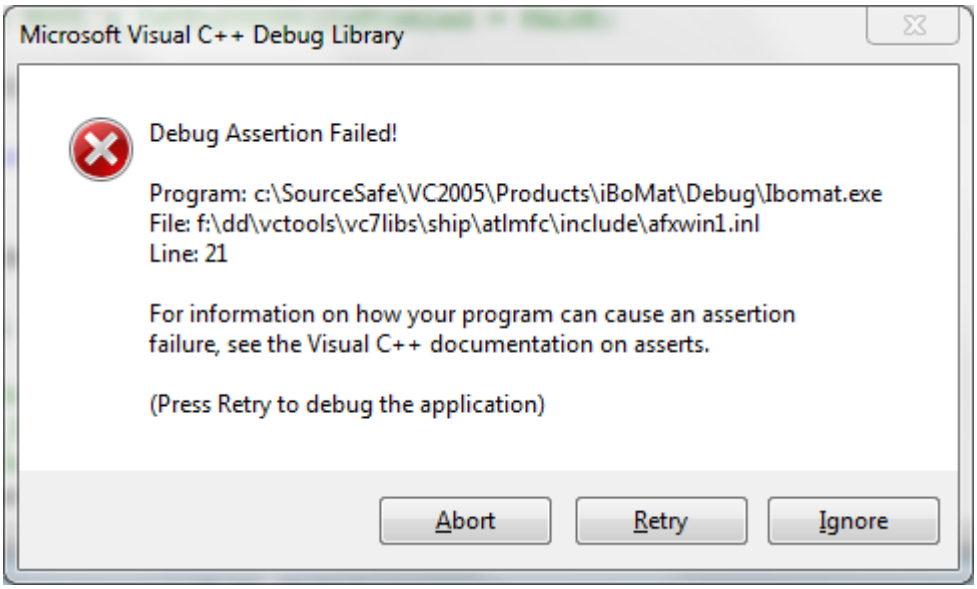
Listing 2.1: De wijziging in de code voor versie 2011 van de Stingray libraries.

Helaas bevatten de nieuwe licenties enkel de Stingray Objective Grid libraries en niet meer de Stingray Objective Toolkit libraries. Gelukkig beschikte ICORDA over een eigen library van klassen die dezelfde functionaliteit aanboden. Door deze bestanden op te nemen in de solution zelf en de `include` statements naar de header *secall.h* te wijzigen naar de header *icdall.h*, kon van deze library gebruik worden gemaakt[7]. De ICORDA libraries bevatten analoge structuren met een analoge naamgeving als die van Stingray. Om de overschakeling compleet te maken, moesten dan in de namen van alle gebruikte objecten en functies van Stingray de substring "SEC" vervangen worden door "ICD" (hoofdstuk B.1 op p.142).

Dit lostte de break errors niet op, maar de installatie van de nieuwe versie verduidelijkte wel hoe de libraries werden geleverd. De libraries moesten immers nog zelf gebuild worden. De oorzaak was dat er bij elke Visual C++ versie een versie van de Stingray libraries hoort. Om de juiste versies van deze libraries te bekomen moesten deze opnieuw gebuild worden voor de gewenste doelversies van Visual C++. De break errors onstonden dus door het proberen runnen van TxKassa versie 2005 met (reeds gebuilde versie 2006) Visual C++ 6.0 libraries. Door de (nog niet gebuilde versie 11) Visual C++ 2005 libraries te builden en includeren in de solution verdwenen de break errors.

Bij het opwaarderen van versie 2005 naar 2008 verschenen de build errors weer, ondanks het feit dat alle versies waren gebuild (dus ook versie 2008). Het werd nog vreemder toen bleek dat in 2010 deze build errors weer verdwenen. In de stap van 2005 naar 2008 was enkel de Visual Studio Conversion Wizard (hoofdstuk 4 op p.27) uitgevoerd en een enkele breaking

[7]Zoals te zien in listing 2.1 op p.17 behoort de header *secall.h* tot de Stingray Objective Toolkit library.



Figuur 2.2: Een voorbeeld van het bericht van een debug assertion.

change (hoofdstuk B.1 op p.148) opgelost. Uiteindelijk bleek dat Visual Studio telkens in de debug folders van de *solution*[8] op zoek ging naar de nodige dll's en daarna pas in de "Include" folder, ingesteld bij "VC++ Directories" als ze daar niet werden gevonden. Ondanks het feit dat de directories telkens ingesteld waren op de folder overeenstemmend met de Visual Studio versie, werd steeds de meest recente (2010) versie genomen[9]. Hierdoor kwamen de library versies nooit overeen met de solution versie, behalve in 2010, en waren deze inconsistenties steeds de oorzaak van de build errors. De oplossing was dan om de dll's overeenkomstig met de versie van de solution in de debugfolder van de solution te kopiëren. In Visual Studio 2010 kon gewoonweg gebruik gemaakt worden van de met Stingray meegeleverde *property sheets*[10]. In Visual Studio 2010 zijn de "VC++ Directories" instellingen opgenomen in property sheets en niet meer in de instellingen van Visual Studio zelf. Door het toevoegen van de Stingray property sheets aan de applicaties die op de Stingray libraries steunden werden de "VC++ Directories" instellingen ingesteld. Hierdoor wordt er gewezen naar de juiste locatie van de Visual C++ 2010 libraries.

Hierbij ging veel tijd verloren. Het heeft vooral veel tijd gekost om de fout te achterhalen. Daarbij werden de libraries talloze keren gebuild naar verschillende versies wat op zich veel tijd innam. Dit wijst meteen op de niet te onderschatten problematiek van externe libraries. Externe libraries moeten ook mee vernieuwen en dit kan enorm diverse problemen met zich meebrengen. Voor problemen met het upgraden van externe libraries is het zeer moeilijk om een generiek antwoord te formuleren omdat dit afhangt van ontzettend veel redenen zoals de solution, de libraries, financiën, beschikbare alternatieven, ... Het kan zijn dat er geen

[8]Een solution is een verzameling van projecten in Visual Studio die samen een enkele applicatie vormen. De projecten bevatten op hun beurt de codebestanden. Zie voor meer informatie hoofdstuk 3.2.1 op p.23.

[9]Vermoedelijk kwam dit doordat de automatische installatie van de Stingray libraries waarden toevoegde aan Visual Studio variabelen en systeemvariabalen.

[10]Property sheets (ook wel property pages genoemd) zijn Extensible Markup Language (XML)-bestanden, met als extensie ".props", die instellingen bevatten voor Visual Studio projecten. In Visual Studio 2008 en lager hebben deze de extensie ".vsprops" en is de inwendige XML-structuur anders.

nieuwere versies meer ontwikkeld zijn van de libraries of dat nieuwere libraries niet meer compatibel zijn met de code van de solution. Dit is zeer moeilijk op te lossen. Ofwel moet er dan worden uitgeweken naar alternatieven ofwel moet er zelf een library worden ontwikkeld. Dit kan zoveel werk met zich meebrengen dat er wordt afgezien van het opwaarderen van de solution. Bij het opwaarderen van externe libraries kunnen er ook financiële factoren meespelen. Vaak brengen nieuwe versies licensiekosten met zich mee.

Na het oplossen van de problemen met de libraries werd het bij het builden al snel duidelijk dat breaking changes een grote invloed hadden. Slechts twee projecten van de acht in de solution buildden zonder problemen en in totaal waren er 93 build errors. Vele fouten kwamen meerdere keren voor; de diversiteit van de fouten was dus klein. Na het manueel oplossen van deze fouten, kwamen er weer een kleine 100-tal andere fouten aan het licht. Dit is typisch gedrag voor een compiler.

> "*The simplest approach is for the parser to quit with an informative error message when it detects the first error. Additional errors are often uncovered if the parser can restore itself to a state where processing of the input can continue with reasonable hopes that the further processing will provide meaningful diagnostic information. If errors pile up, it's better for the compiler to give up after exceeding some error limit than to produce an annoying avalanche of 'spurious' error.*" (Aho et al., 2007)

Na het oplossen van de fouten was de solution compileerbaar. Er werd nog niet gekeken naar warnings. Warnings zijn waarschuwingen die wijzen op code die mogelijk voor onverwachte fouten kunnen zorgen. In tegenstelling tot fouten kan een programma met waarschuwingen wel gecompileerd worden. De code is dus conform met de Visual C++ specificatie, maar kan logische fouten bevatten in de applicatie. Voorbeelden hiervan zijn verlies van data, gebruik van deprecated functies, ... Alle tegengekomen warnings en hun oplossingen zijn gedocumenteerd in hoofdstuk B.2 op p.153.

Tijdens het compileren kwamen er ook een aantal meldingen aan het licht over ongebruikte geïncludeerde headerbestanden. Omdat deze geen invloed hadden op het opwaarderingsproces werd hier niet verder op ingegaan.

2.2.2 Het eerste succes met Irent

Omdat er door de problemen met de Stingray libraries gewacht moest worden op support van de RogueWave[11] helpdesk, werd er ondertussen ook begonnen aan de opwaardering van iRent. Door de ervaringen opgedaan bij TxKassa konden reeds tegengekomen problemen makkelijk opgelost worden.

In iRent kwamen er veel (first chance) COM-fouten (`_com_error`'s) naar boven. Door het gebruik van identieke code kwam dit ook bij andere applicaties van ICORDA voor. Deze werden uiteindelijk veroorzaakt door een methode waarbij de programmeur een '`if-else`'-structuur gebruikte op basis van exceptions, in plaats van zelf te controleren of een databankresultaat gelukt was of niet. Hierdoor werd ook duidelijk dat niet alle fouten te wijten zijn aan breaking changes. Er moet in het achterhoofd gehouden worden dat de geschreven code niet altijd correct is. Dit kan nagegaan worden door de applicatie te runnen in originele

[11] RogueWave is het bedrijf dat de Stingray libraries aanbiedt.

toestand en opgewaardeerde toestand en foutmeldingen te vergelijken. Indien de fouten in beide versies voorkomen ligt de fout bij de applicatie en niet bij een breaking change.

Er werden in iRent ook implementaties tegengekomen van zelfgeschreven functies die in latere versies van Visual C++ door Microsoft werden geïmplementeerd. Een voorbeeld hiervan was de functie `_atof`. Deze functie was in Visual C++ 6.0 nog niet aanwezig en werd dus door de programmeurs zelf geïmplementeerd. Later werd de functie door Microsoft toegevoegd en gaf dit conflicten. Na het lezen van documentatie werd duidelijk dat beide functies dezelfde bewerkingen uitvoerden en moest de zelfgeschreven implementatie uitgeschakeld worden om conflicten te verwijderen.

Buiten problemen met libraries en breaking changes in de code bleken er ook nog enkele veranderingen te zijn gebeurd in Visual Studio zelf. Zo zijn er bepaalde macro's van naam veranderd. De buildstep "`mc $(InputName)`" moest daardoor in Visual Studio 2005 veranderd worden naar "`mc $(InputFileName)`". In Visual Studio 2008 was er een analoog probleem met de macro `Filename` dat veranderd moest worden naar `Identity`. De oorzaak van sommige breaking changes ligt dus niet enkel aan Visual C++, maar soms ook aan Visual Studio. Soms ligt de oorzaak zelfs buiten het domein van Visual C++: omdat er vroeger geen spaties in bestandspaden waren toegelaten, waren de argumenten van de commando's in de buildsteps niet voorzien van aanhalingstekens. Omdat spaties nu wel voorkomen in padnamen (bijvoorbeeld "`C:\Program Files (x86)\`") zorgde dit ervoor dat de buildsteps niet meer correct waren en fouten veroorzaakten. Alle argumenten van commando's in buildsteps moesten dus ingesloten worden met aanhalingstekens.

Omdat de applicaties in ICORDA vaak in meerdere workspaces waren opgesplitst werden er bij iRent pogingen ondernomen om de workspaces te combineren. Maar door de omvang en foutgevoeligheid van het combineren bleek dit niet mogelijk. Elke poging veroorzaakte een grote hoeveelheid linker errors.

Toen iRent werd opgewaardeerd naar Visual C++ 2010 bleek alles te werken, behalve het weergeven van opgehaalde records uit de databank in een dialoogvenster. Er werd slechts één kolom weergegeven, terwijl het bij het debuggen duidelijk was dat alle gegevens correct uit de databank werden opgehaald. Uiteindelijk bleek dat het standaardgedrag van de component in het dialoogvenster, dat de records moest tonen, was veranderd. Om het oorspronkelijk gedrag te herstellen was er slechts één enkele lijn code nodig in het rc-bestand waar de component werd beschreven. Er werd eerst gedacht dat het lag aan de Poco libraries, maar uiteindelijk bleek dat de fout lag bij een project van iRent zelf. De reden waarom er zoveel tijd verloren ging aan het vinden van deze fout was dat er veel tijd verloren ging door het experimenteren met de Poco library. Deze moest bij het opwaarderen ook vernieuwen en daarbij moest iRent aan de nieuwe versie worden aangepast, wat complicaties met zich meebracht.

2.2.3 Na iRent ook succes met TxKassa en iBomat

Na het oplossen van dit probleem was iRent als eerste applicatie succesvol opgewaardeerd naar Visual C++ 2010. Door de opgedane ervaring van iRent en de gevonden oplossing voor de problemen met de Stingray libraries volgde hierna TxKassa met relatief weinig problemen. Daarna werd ook iBomat geconverteerd zonder bijkomende problemen. Omdat up-to-date solutions in ICORDA bewaard worden in Team Foundation Server (TFS) in plaats van Visual SourceSafe, waar de originele projecten zich bevonden, werden de opgewaardeerde Visual C++ 2010 applicaties aangepast aan de mappenstructuur in TFS. Ook nieuwe versies van Poco en Stingray moesten in TFS worden opgenomen. Hiervoor moesten hardgecodeerde

paden, property sheets en buildsteps worden aangepast. Zo werden bijvoorbeeld absolute paden vervangen door relatieve paden. Omdat TxKassa een afwijkende mappenstructuur heeft konden de (voor TFS aangepaste) Stingray property pages niet worden gebruikt en moesten alle instellingen manueel in de property pages van de projecten in TxKassa zelf worden toegevoegd.

2.2.4 Problemen met iPlant

De daaropvolgende opwaardering van iPlant verliep moeilijker dan aanvankelijk gedacht. Er is namelijk een conflict met meervoudige overerving door de aanwezigheid van een *diamond*structuur[12] en de omschakeling van gewone casts naar static casts in een macro. Na het oplossen van deze fout bleek ook dat er ergens een header in twee rc-bestanden tegelijk werd geïncludeerd. Dit zorgde voor conflicten omdat er hierdoor een lus aanwezig was in de *dependancy graph*[13]. Het toevoegen van *headerguards*[14] in het betrokken header-bestand had geen effect, dus was er geen andere optie dan zelf uit te zoeken welk header-bestand tweemaal werd geïncludeerd. Na het oplossen van deze problemen was ook iPlant succesvol opgewaardeerd en ingevoerd in TFS.

2.2.5 Afronden met GandaFact

GandaFact was de oudste applicatie van de behandelde applicaties en bracht een aantal extra moeilijkheden met zich mee. Het steunde op een aantal klassen in de Stingray Objective Toolkit libraries die niet meer aanwezig zijn in de nieuwe Stingray libraries. De nieuwe licenties omvatten immers enkel nog de Stingray Objective Grid libraries. Na nader onderzoek bleek dat de ontbrekende klassen `SECMarquee` en `SECRebar` konden verwijderd worden zonder verlies van functionaliteit. De ontbrekende klassen `SECBitmapDialogue`, `SECStatusbar` en `SECToolbar` konden op hun beurt vervangen worden door de klassen `CDialog`, `CStatusbar` en `ICDToolbar`[15].

Doordat de nieuwe Stingray libraries werkten met smartpointers in plaats van gewone pointers was de functionaliteit van GandaFact gebroken. Dit komt omdat smartpointers controleren op nullpointers: indien er een nullpointer aan een smartpointer wordt toegewezen genereert dit break errors. Omdat de functionaliteit van GandaFact berustte op het gebruik van nullpointers zou Gandafact herwerkt moeten worden om met de nieuwe versie van Stingray overweg te kunnen. Omdat dit een te omvangrijke en foutgevoelige opgave was werd er gebruik gemaakt van managed code. Dit wordt verder uitgelegd in hoofdstuk 7 op p.57. Uiteindelijk werd ook GandaFact, dankzij het gebruik van managed code, opgewaardeerd naar Visual C++ 2010 en ingeladen in TFS. Hiermee zijn alle applicaties succesvol opgewaardeerd naar Visual C++ 2010.

[12]Een diamondstructuur is een speciaal geval van meervoudige overerving. Een klasse erft over van twee of meer klassen, die op hun beurt allen overerven van dezelfde klasse.

[13]Een grafische voorstelling die de relaties tussen de headerbestanden (via `include` statements) in een project weergeeft.

[14]Headerguards zijn preprocessor statements die ervoor zorgen dat een header-bestand slechts éénmaal in het project wordt geïncludeerd.

[15]De klasse `ICDToolbar` is een klasse uit de library van ICORDA. De klassen `CDialog` en `CStatusbar` zijn MFC-klassen.

Hoofdstuk 3

Probleemstelling

Bij het opwaarderen van Visual C++ 6.0 projecten naar Visual C++ 2010 komt men voornamelijk in aanraking met twee problemen.

3.1 Visual C++ is geen standaard C++

Het eerste probleem houdt in dat Visual C++ de Microsoft interpretatie is van C++. Visual C++ komt dus niet overeen met standaard C++. Doorheen de jaren heeft men bij Microsoft beseft dat dit een belemmering is en poogt men dit verschil steeds te verkleinen. Elke nieuwe versie van Visual C++ dicht de kloof met standaard C++ conventies een beetje meer. Volgens Gregory (2004) kwam Visual C++ .NET 2003 voor 98% overeen met de C++ standaard opgesteld in de jaren '90. Op dit moment heeft is de C++ standaard bijna compleet bijgebeend.

> "*Visual C++ complies with the 2003 C++ Standard, with these major exceptions: two-stage name lookup, exception specifications, and export. Additionally, Visual C++ supports several C++0x features, for example, lambdas, auto, static_assert, rvalue references, and extern templates.*" (Microsoft, 2010e)

Bij het invoegen van *conformances*[1] worden volgens Rich (2006) meestal twee soorten conformances onderscheiden: positieve en negatieve. Positieve conformances zorgen ervoor dat structuren die vroeger een fout veroorzaakten nu door de compiler als geldig worden beschouwd. Dit is vooral handig bij gebruikers die werken met meerdere compilers. Als andere compilers een bepaalde structuur als geldig beschouwen en dezelfde code wordt ingevoerd in de Visual C++ compiler, dan is het gewenst dat deze ook die structuur als geldig beschouwt. Zoniet moet de code aangepast worden vanwege de Visual C++ compiler. Negatieve conformances zijn structuren die als geldig werden beschouwd, maar nu een fout opwerpen. Hier wordt vooral rekening gehouden met gebruikers die code ontwikkelen op de Visual C++ compiler. Als deze gebruikers hun code willen laten compileren door een andere compiler, dan blijkt het vol fouten te zitten omdat de Visual C++ compiler meer toelaat dan de andere compilers. Helaas heeft het invoegen van conformances invloed op alle reeds geschreven Visual C++ code. Vooral de negatieve conformances zijn hier de grote boosdoeners. Bestaande

[1] Conformances zijn aanpassingen van de specificatie van Visual C++ om de kloof met standaard C++ te verkleinen.

code blijkt niet meer compileerbaar in hogere versies van de compiler omdat bepaalde structuren vanaf nu als ongeldig worden beschouwd. Deze nieuwe fouten worden *breaking changes* genoemd. Sommige breaking changes zullen slechts waarschuwingen opwerpen (bv. voor negatieve conformances die geen kritieke fouten veroorzaken, maar best wel worden nagekeken). Dit komt omdat bij negatieve conformances de keuze moet worden gemaakt tussen het toelaten van een niet-conformante structuur of de structuur een fout moet laten opwerpen. Daartussen liggen er volgens Rich (2006) nog verschillende niveaus, zoals waarschuwingen en af te zetten foutmeldingen. Men baseert zich op grote hoeveelheden bestaande en nog steeds gebruikte code om te bepalen wat de geschikte reactie is voor elke negatieve conformance. Dit gebeurt aan de hand van factoren zoals de frequentie waarmee de breaking change voorkomt, hoe fout de originele code was en de moeilijkheidsgraad om de breaking change overal op te lossen. (Rich, 2006)

Er ontstaan dus bij elke nieuwe versie weer andere breaking changes, zodat code geschreven in oudere versies van Visual C++ in recentere ontwikkelomgevingen niet meer compileert. De breaking changes worden allemaal gedocumenteerd in de MSDN.

Tot slot bevat Visual C++ ook Microsoftspecifieke uitbreidingen (*Microsoft extensions*). Deze zijn niet compatibel met standaard C++. Men kan de compiler met de optie "/Za" zo instellen dat enkel standaard C++ wordt aanvaard (Microsoft, 2010f).

3.2 Verschillende projectstructuren

Het tweede probleem heeft te maken met de projectstructuur. Visual C++ projecten bevatten twee soorten bestanden: *solution*bestanden en *project*bestanden.

3.2.1 Solution- en projectbestanden

Als men spreekt over projecten heeft men het eigenlijk vaak over de solution. Een solution in Visual Studio is de overkoepelende structuur die een of meerdere projecten bevat. Het bevat alle instellingen en informatie om de projecten samen tot één enkele applicatie te compileren. De projecten zijn dan de kleinste compileerbare eenheden en bevatten de eigenlijke codebestanden. Ze bieden in een solution de mogelijkheid om soepel en ordelijk codebestanden te structureren en bestaande code in te voegen. Vanaf nu wordt in deze scriptie telkens gesproken over solutions en projecten analoog met de betekenis die deze hebben in Visual Studio.

De naamgeving en inhoud van solution- en projectbestanden hebben doorheen de versies enkele ingrijpende veranderingen ondergaan. In Visual C++ 6.0 waren solution- en projectbestanden respectievelijk dsw- en dsp-bestanden (*workspace*- en projectbestanden, workspace was de toenmalige naam voor een solution). Vanaf Visual C++ 2002 werden respectievelijk sln- en vcproj-bestanden gebruikt voor solution- en projectbestanden. Deze omwenteling werd teweeggebracht door de invoering van het .NET framework. Vanaf Visual C++ 2010 werden de projectbestanden vervangen door vcxproj-bestanden. Reden hiervoor is de omschakeling naar een ander buildsysteem (zie hoofdstuk 3.2.2 op p.24). Ondanks het feit dat voor meerdere versies sln- en vcproj-bestanden gebruikt worden zijn deze niet onderling compatibel. Om een sln- of vcproj-bestand met een lagere versie dan de versie van de gebruikte Visual Studio te openen, moet het opgewaardeerd worden. Omgekeerd kunnen in Visual Studio geen sln- of vcproj-bestanden van hogere versies geopend worden. Vanaf Visual Studio 2010 kan

dit wel met sln- en vcxproj-bestanden in versie 2010 of hoger dankzij het nieuwe buildsysteem. Een overzicht bevindt zich in tabel 3.1 op p.24.

Naam	Versienummer	Extensie solutionbestanden	Extensie projectbestanden
Visual C++ 6.0	6.0	dsw	dsp
Visual C++ .NET (2002)	7.0	sln	vcproj
Visual C++ .NET 2003	7.1	sln	vcproj
Visual C++ 2005	8.0	sln	vcproj
Visual C++ 2008	9.0	sln	vcproj
Visual C++ 2010	10.0	sln	vcxproj
Visual C++ 2012	11.0	sln	vcxproj

Tabel 3.1: De verschillende Visual C++ versies vanaf Visual C++ 6.0 met de extensies van hun solution- en projectbestanden

Het verschil tussen deze bestanden zit niet alleen in de inhoud, maar vooral in de structuur van die inhoud. Zoals reeds gezegd bevatten solution- en projectbestanden alle informatie nodig voor Visual Studio om het project of de applicatie te compileren. Het is vanzelfsprekend dat inhoudelijke verschillen slaan op toegevoegde of gewijzigde opties in Visual Studio. Wanneer men de structuur van naderbij bekijkt, ziet men dat men in de sln-, vcproj- en vcxproj-bestanden de inhoud opslaat in een glsxml-structuur volgens de XML 1.0 standaard (listing A.3, A.4 en A.5 in bijlage A op p.127), terwijl dit in dsw- en dsp-bestanden niet het geval is (listing A.1 en A.2 in bijlage A op p.127).

3.2.2 VCBuild en MSBuild

Bij de invoering van Visual C++ 2010 is er overgeschakeld op een ander buildsysteem. Het buildsysteem is het systeem dat aangeroepen wordt om solutions en projecten te builden. Het leest alle instellingen in, voert opgegeven buildtaken uit en spreekt de nodige toepassingen aan die samen het project builden.

Tot Visual Studio 2010 gebruikte werd voor Visual C++ het buildsysteem VCBuild gebruikt. Het werd meegeleverd bij elke versie van Visual Studio.

Vanaf Visual Studio 2005 werd samen met het .NET framework het MSBuild buildsysteem ingevoerd. MSbuild werd meegeleverd met het .NET framework in tegenstelling tot VCBuild dat telkens werd meegeleverd met Visual Studio. Met elke versie van Visual Studio komt er een frameworkversie overeen (tabel 3.2 op p.25). MSBuild is de applicatie die werd opgeroepen bij het builden van een solution of project in Visual Studio. Alhoewel MSBuild wordt meegeleverd met het .NET framework sinds versie 2.0 (Visual Studio 2005), was het enkel beschikbaar voor Visual C# en Visual Basic tot versie 4.0. Vanaf versie 4.0 (Visual Studio 2010) is MSBuild ook beschikbaar voor Visual C++. Dit verklaart de verandering van de extensies van projectbestanden. Daarvoor deed MSBuild beroep op VCBuild telkens het een Visual C++ project tegenkwam.

MSBuild heeft een aantal grote voordelen ten opzichte van VCBuild:

Multi-Targeting

MSBuild laat toe om vanuit Visual Studio 2010 applicaties te ontwikkelen tegen verschillende versies van het framework (managed multi-targeting) en verschillende versies

van toolsets (native multi-targeting)[2]. Een toolset bevat alle tools van een bepaalde Visual Studio versie, zoals de Visual C++ compiler *cl.exe*. Op deze manier wordt de Integrated Development Environment (IDE) losgekoppeld van het framework en toolsets. Zo kan in Visual Studio 2010 nu ook ontwikkeld worden tegen versie 3.5 van het framework en met de tools van Visual Studio 2008. De voorwaarde is wel dat de doeltoolsets en doelversies van het .NET framework aanwezig zijn op het systeem van de ontwikkelaar. Voor native multi-targeting is enkel de doeltoolset vereist; het framework wordt immers niet gebruikt. Managed multi-targeting vereist zowel de doeltoolset als de doelversie van het .NET framework. Om frameworkversie 3.5 te gebruiken, moet men ook Visual Studio 2008 Service Pack 1 installeren.

> "*First of all VS 2008 SP1 must be installed on your system (you cannot do without SP1). This is a mandatory requirement for targeting framework version 3.5. Installing only the .NET Framework 3.5 redist won't do. C++ tools like cl.exe in VS 2010 are only capable of targeting v4.0. So to target an earlier framework version, we need the tools from that earlier version. And these tools are shipped with VS and not with .NET Framework.*" (Adharapurapu, 2009)

Zo kan er met een enkele Visual Studio applicaties die verschillen in toolset- en frameworkversie onderhouden worden. Ontwikkelaars moeten niet constant meer verwisselen tussen verschillende Visual Studio versies en telkens hun gebruikersinstellingen, zoals shortcuts en kleuren in de editor, op elkaar afstemmen.

Deze feature komt zeker goed van pas bij het opwaarderen van projecten. Helaas is voor Visual C++, wegens het latere invoeren van MSBuild voor Visual C++, deze functionaliteit beperkt tot toolsets 2008 (vs90), 2010 (vs100), 2012 XP (vs110_xp), 2012 (vs110) en frameworkversies 3.5, 4.0 en 4.5. Om in Visual C++ projecten[3] gebruik te maken van multi-targeting, moeten ze dus eerst worden opgewaardeerd naar Visual C++ 2010. (Shao, 2009)

Naam	frameworkversie	Jaar van uitgave
Visual Studio .NET (2002)	1.0	2002
Visual Studio .NET 2003	1.1	2003
Visual Studio 2005	2.0 & 3.0	2005
Visual Studio 2008	3.5	2008
Visual Studio 2010	4.0	2010
Visual Studio 2012	4.5	2012

Tabel 3.2: De verschillende frameworkversies en de overeenkomstige Visual Studio versies waarmee ze werden geïntroduceerd (Microsoft, 2012m).

Gedetailleerde output

De *verbosity*, de hoeveelheid output en het detail ervan, van de logger in MSBuild kan op verschillende niveaus worden ingesteld. Er kan dus bepaald worden hoeveel

[2]Het verschil tussen native en managed applicaties wordt besproken in hoofdstuk 7 op p.57.

[3]Native multi-targeting (`Properties > Configuration Properties > General > Platform Toolset`) en Managed multi-targeting (`Properties > Target framework`) wordt ingesteld per project.

output er geproduceerd moet worden bij het builden van projecten en solutions. In tegenstelling tot VCBuild wordt er veel meer informatie weergegeven en kunnen ook externe programma's tijdens het buildproces output toevoegen.

Flexibiliteit

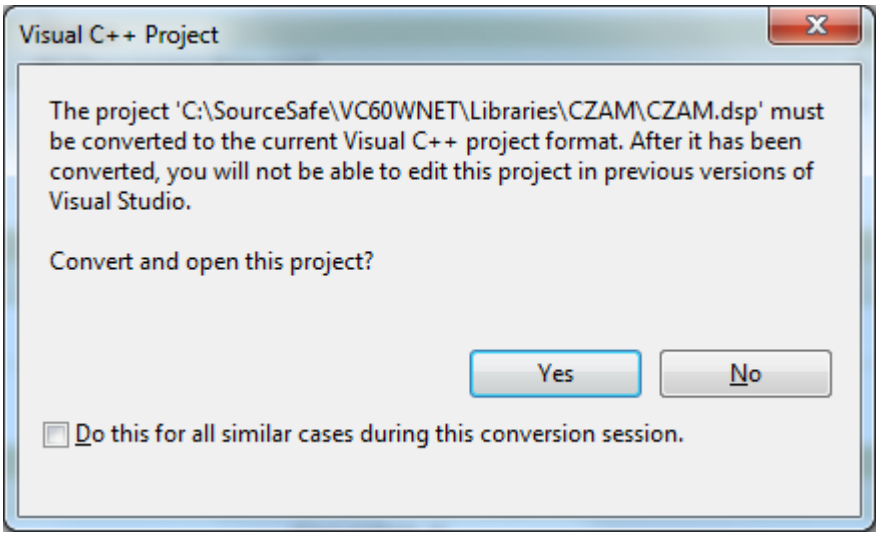
Het buildproces kan dankzij MSBuild zoveel aangepast en uitgebreid worden als gewenst. In VCBuild zit de buildlogica in de binaire code. MSbuild daarentegen werkt met *tasks*. Een task beschrijft een actie die door MSBuild moet uitgevoerd worden. In plaats van, zoals VCBuild rechtstreeks de tools aan te roepen, zal MSBuild kijken naar welke tasks het moet uitvoeren. Elke task beschrijft een opdracht zoals een aanroep van een programma, het kopieren van bestanden, ... Zo zijn er de tasks *CL.XML*, *Link.XML*, *Lib.XML*, *RC.XML*, *MIDL.XML*, *MT.XML*, *XSD.XML*, *XDCMake.xml* en *BSCmake.xml* voor het aanroepen van de respectievelijke programma's *cl.exe*, *link.exe*, *lib.exe*, *rc.exe*, *midl.exe*, *mt.exe*, *xsd.exe*, *xdcmake.exe* en *bscmake.exe*. In VCBuild zijn deze aanroepen vastgelegd in de binaire code. Door het gebruik van tasks kan men zelf tasks ontwikkelen en toevoegen aan een bepaald solution of project. Dit maakt het mogelijk MSBuild te gebruiken als buildengine voor zowel Visual Basic.NET, Visual C# als Visual C++. Om het gemakkelijk te maken om zelf aanpassingen aan te brengen, zijn alle bestanden die te maken hebben met het MSBuild opgesteld in een XML-structuur, de MSBuild syntax. (Shao, 2008)

Dit is slechts een beknopt overzicht van de voordelen van MSBuild. MSBuild wordt vanaf Visual Studio 2010 gebruikt als buildsysteem voor alle .NET talen.

Hoofdstuk 4

Conversie van de projectstructuur

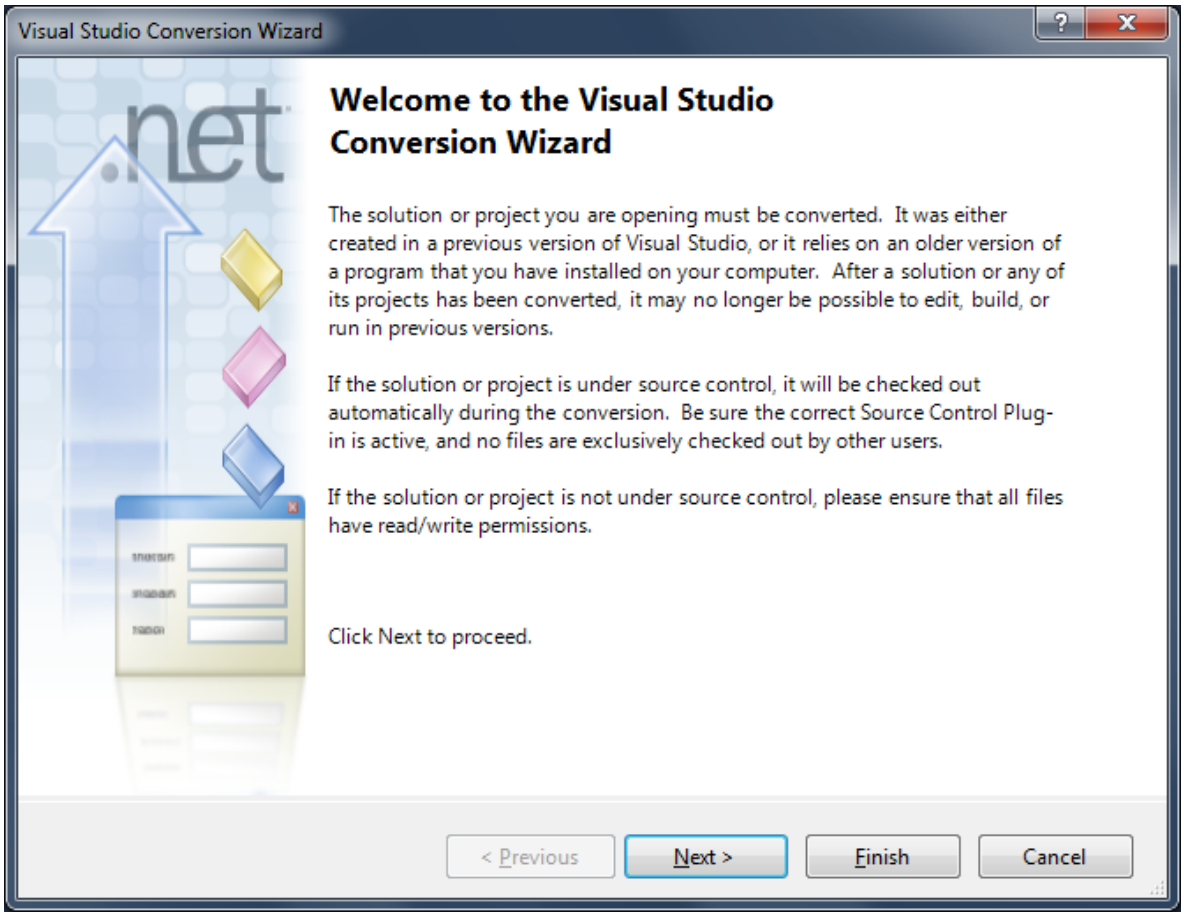
Het converteren van de projectstructuur is de eerste stap in het opwaarderen van een Visual C++ 6.0 project en tevens de eenvoudigste. Een solution of project kan eenvoudig worden opgewaardeerd door dit te openen met de Visual Studio versie waarnaar er opgewaardeerd moet worden. Afhankelijk van de versie van het project zal Visual Studio de gewenste stappen ondernemen om de projectstructuur te converteren. Indien het gaat om een Visual C++ 6.0 project zal Visual Studio vragen om elk project op te waarderen. Men ziet dan het venster in figuur 4.1 op p.27.



Figuur 4.1: Opwaarderen van de projectstructuur van een Visual C++ 6.0 project met behulp van Visual Studio.

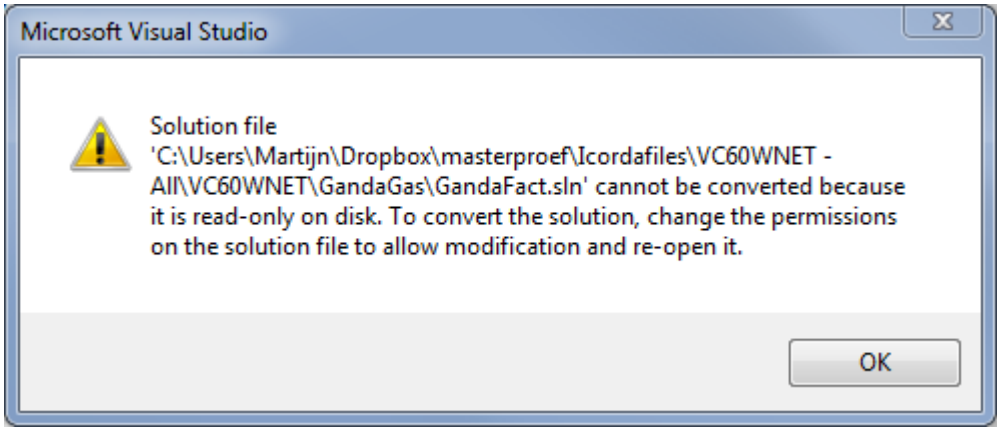
Indien het gaat om een Visual C++ 2002 of hoger project zal Visual Studio de Visual Studio Conversion Wizard opstarten (zie figuur 4.2 op p.28). Deze wizard begeleidt de gebruiker doorheen het proces en laat ook toe om backups aan te maken. Achteraf biedt de wizard een overzicht aan met de resultaten van het conversieproces. Vaak wordt er ook een log bijgehouden. De logs bevatten verslagen van alle bewerkingen die Visual Studio heeft uitgevoerd bij het converteren evenals fouten en waarschuwingen die het detecteerde.

Er moet wel voor gezorg worden dat voor het openen van de solution- en projectbestanden door Visual Studio, de eigenschap 'read-only' op die bestanden uitgeschakeld staat. Anders



Figuur 4.2: Opwaarderen van de projectstructuur van een Visual C++ 2002 of hoger project met behulp van Visual Studio.

kan Visual Studio de solution- en projectbestanden niet aanpassen en zal de conversie falen (figuur 4.3 op p.29).



Figuur 4.3: Solution- en projectbestanden die op 'read-only' ingesteld staan, kunnen niet worden opgewaardeerd door Visual Studio.

Hoofdstuk 5

iSE

Na het oplossen van het probleem met de projectstructuren is men klaar voor de volgende stap: het oplossen van de breaking changes. Omdat (vooral bij grote projecten) dit veel repetitief en ingrijpend werk aan de code met zich meebrengt, is er een applicatie ontwikkeld om het werk te verlichten. Dit gebeurde in het kader van deze masterproef en op basis van ervaringen met het manueel opwaarderen van Visual C++ programma's. Deze applicatie noemt iSE.

iSE staat voor "I See Everything". Het duidt aan waar in de solution er zich fouten bevinden veroorzaakt door breaking changes. Voor elke gevonden fout wordt er ook een mogelijke oplossing voorgesteld en kan men iSE deze oplossing meteen laten toepassen.

5.1 Gebruikswijze

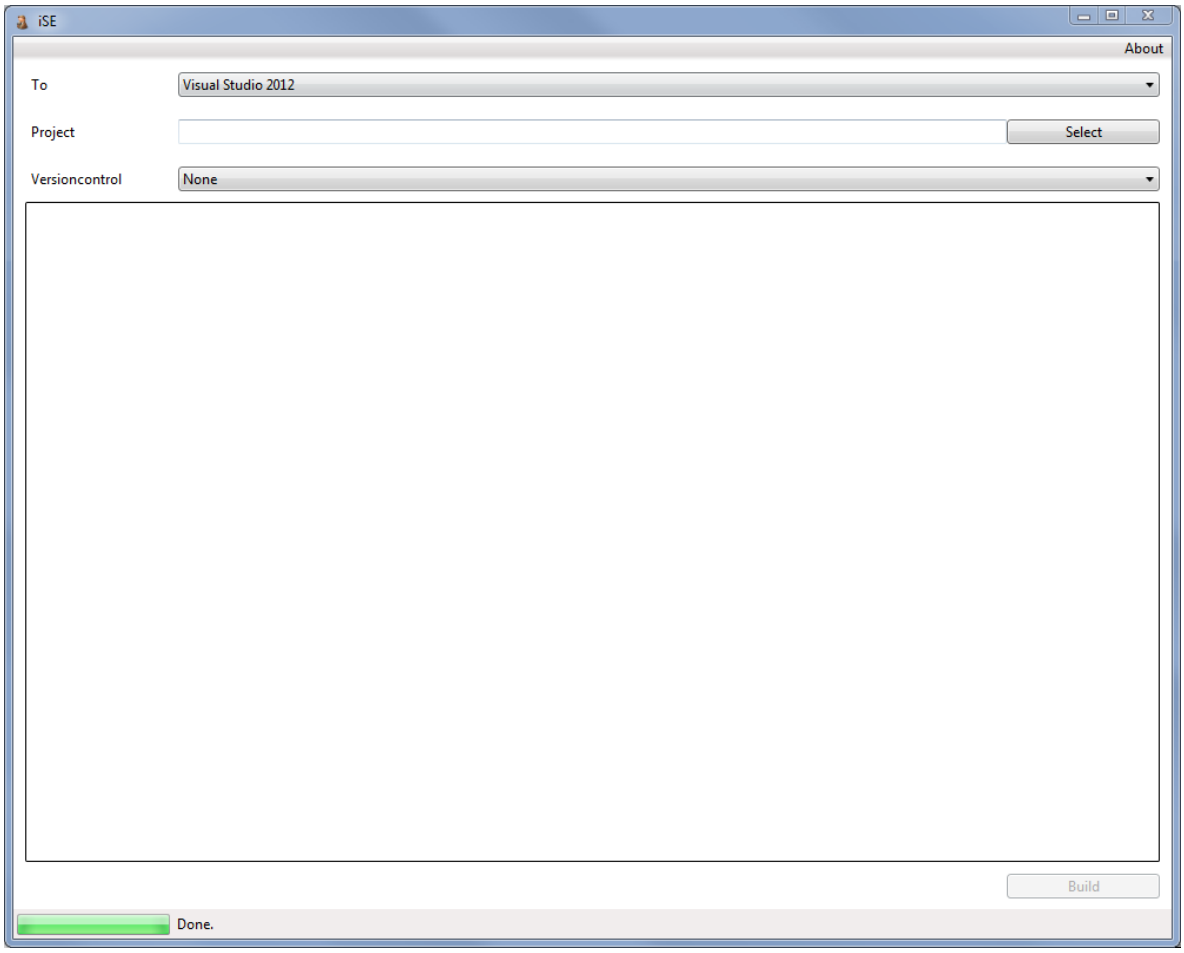
Voordat de achterliggende structuur en werking van naderbij wordt bekeken, wordt eerst uitgelegd hoe iSE gebruikt wordt.

Bij het opstarten van iSE verschijnt meteen het hoofdvenster (figuur 5.1 op p.31). De belangrijkste controls in het hoofdscherm zijn de volgende:

To
: Hier kan gekozen worden uit de geïnstalleerde versies van Visual Studio op het systeem die worden ondersteund door iSE. Deze versie komt best overeen met de versie van de solution die men wil behandelen. Als de versie van de solution hoger ligt, zal Visual Studio dit niet willen aanvaarden. Andersom zal het proberen om het project te configureren. Dit laatste kan onherstelbare wijzingen aanbrengen aan de solution.

Project
: Door op "Select" te klikken komt er een dialoogvenster tevoorschijn waarbij een dsw- of sln-bestand geselecteerd kan worden (dit hangt af van de geselecteerde Visual Studio versie).

Versioncontrol
: Er is een rudimentaire ondersteuning aanwezig voor versiebeheerprogramma's. Indien de solution zich onder de hoede van dergelijk programma bevindt, is het vaak zo dat dit bestand niet gewijzigd kan worden. Om ervoor te zorgen dat men niet telkens bij



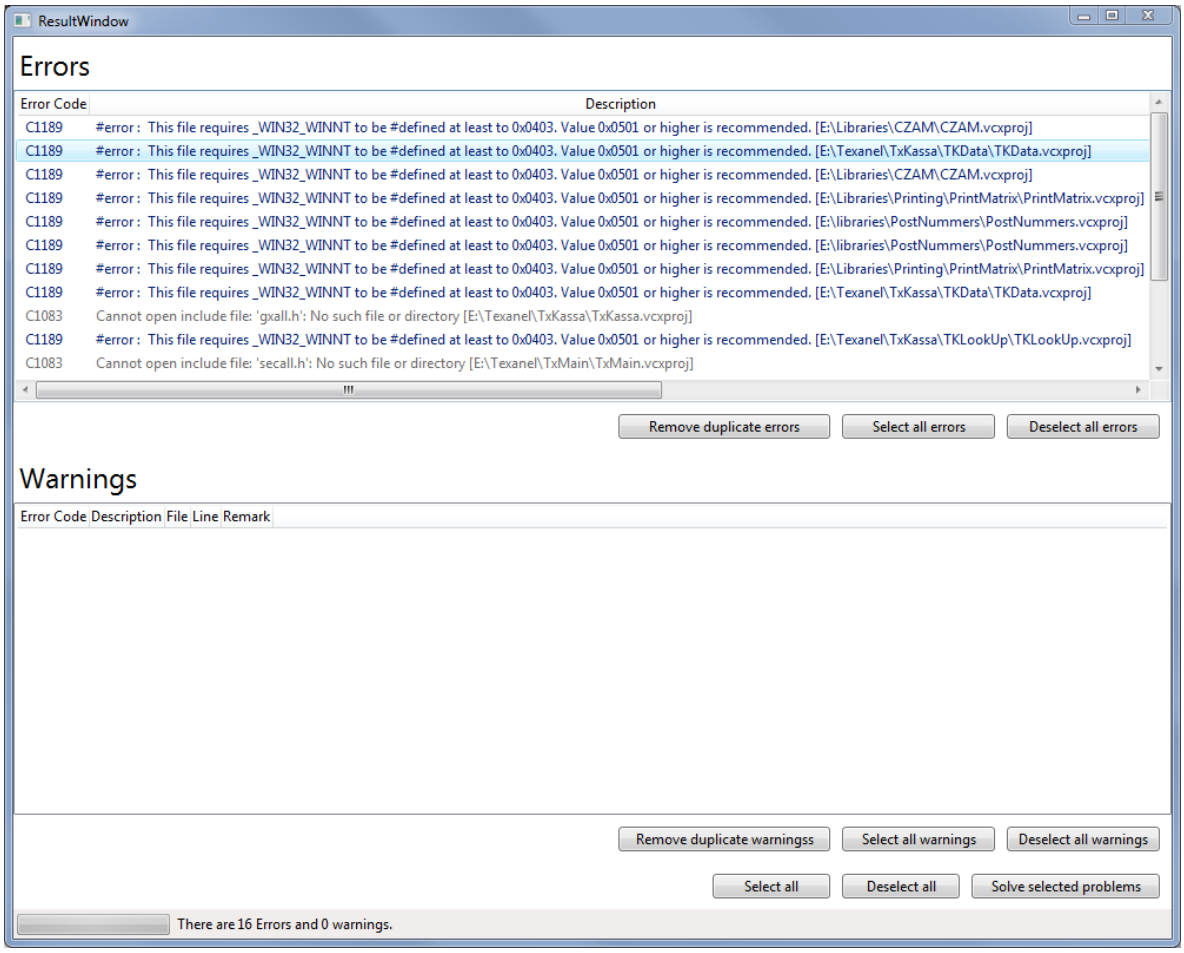
Figuur 5.1: Het hoofdscherm van iSE.

alle bestanden de optie 'read-only' moet uitschakelen, kan iSE de bestanden die gewijzigd worden uitchecken. Net als bij "To" kunnen hier enkel versiebeheerprogramma's geselecteerd worden die geïnstalleerd staan op het systeem en ondersteund worden door iSE.

Het is van belang dat alle instellingen goed worden ingesteld. Indien er bijvoorbeeld een versiebeheerprogramma wordt geselecteerd waar de solution niet mee verbonden is, kan dit foutmeldingen veroorzaken en iSE hinderen.

Als alle instellingen zijn ingesteld wordt de knop "Build" beschikbaar. Zodra hierop geklikt wordt zal iSE de solution proberen builden. Voor het builden spreekt iSE de commandline compiler van de Visual C++ doelversie aan. Het geeft enkele parameters mee waaronder de locatie van het sln-bestand en produceert output op de commandline. Deze output verschijnt dan in het centrale tekstkader. Via reguliere expressies wordt de output geparset. Indien het gaat om fouten veroorzaakt door breaking changes wordt er een oplossingsmethode aan de fout gekoppeld. Dit kan op elk moment met behulp van de ondertussen verschenen "Cancel"-knop geannuleerd worden, zodat het hoofdscherm weer beschikbaar wordt.

Nadat alle fouten geparset en aan een oplossing gekoppeld zijn, verschijnt het resultatenscherm (figuur 5.2 op p.32)



Figuur 5.2: Het resultatenscherm van iSE.

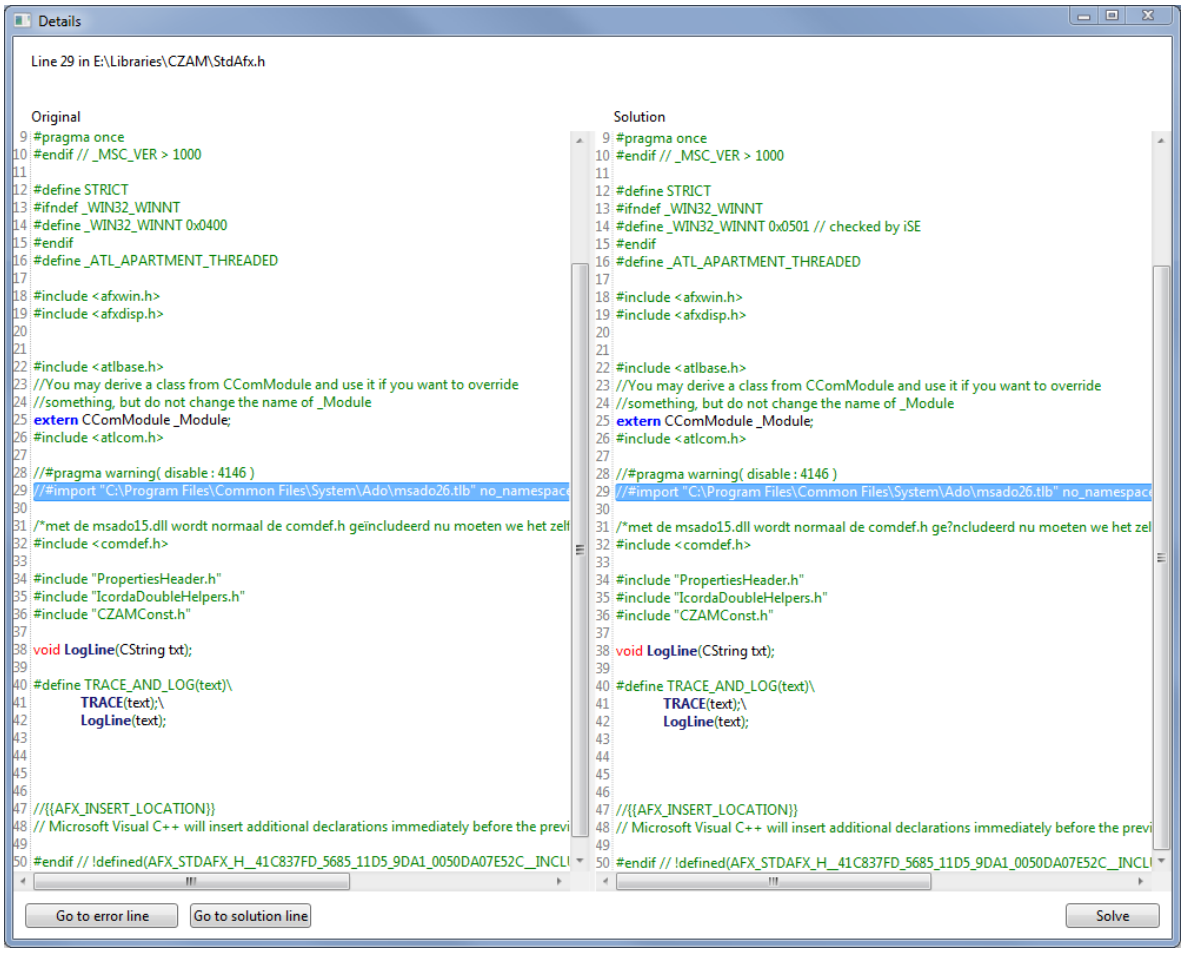
Het resultaatvenster is opgedeeld in twee delen: bovenaan staan alle compileerfouten en onderaan alle waarschuwingen. Het verschil tussen compileerfouten en waarschuwingen is net hetzelfde als in Visual Studio: als er compileerfouten zijn kan er niet gebuild worden, als er enkel waarschuwingen zijn wel. Om de uitleg verder eenvoudig te houden duidt de term "fout" op zowel compileerfouten als waarschuwingen.

Indien een fout geen oplossing heeft wordt de fout uitgegrijsd in de lijst weergegeven en kan die niet worden aangeklikt of geselecteerd.

Door op een fout te dubbelklikken wordt het detailvenster opgeroepen (figuur 5.3 op p.33). Het detailvenster geeft de broncode weer van het bestand waarin de fout voorkomt. Links staat de oorspronkelijke code (met de fout) en rechts de code zoals die er zou uitzien als na het toepassen van de verbonden oplossingsmethode. Er kan hier al gekozen worden om de fout op te lossen.

In het resultatenscherm kunnen fouten geselecteerd worden om op te lossen met behulp van de aanwezige knoppen of door één enkele keer op de fout te klikken. De fout zal dan blauw oplichten (zoals de tweede fout in figuur 5.2 op p.32). Als er dan op de "Solve selected problems"-knop gedrukt wordt worden alle geselecteerde fouten opgelost.

Er wordt sterk aangeraden om alle door iSE gegenereerde oplossingen te controleren. De



Figuur 5.3: Het resultatenscherm van iSE

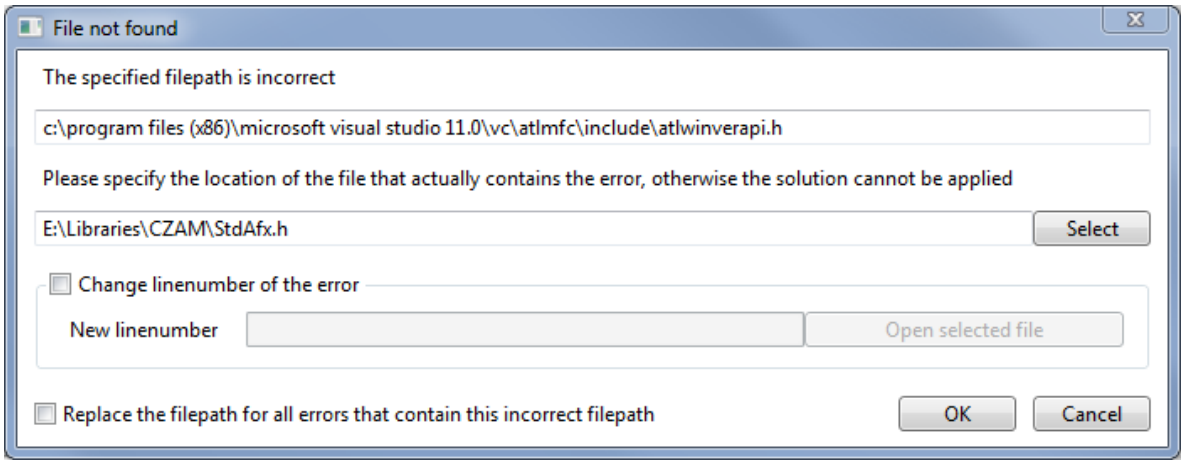
oplossingsmethoden van iSE zijn nooit 100% veilig. Het is altijd mogelijk dat er een situatie voorkomt die iSE nog niet kent. In dat geval kan de gebruiker ofwel de bestaande oplossingen aanvullen ofwel zelf oplossingen en tools ontwikkelen en die toevoegen aan iSE[1].

In het resultatenscherm kunnen ook alle duplicate fouten verwijderd worden. Indien er veel fouten worden geselecteerd (bijvoorbeeld met behulp van de selectieknoppen) is het aangeraden om dit te doen. Vooral voor Visual Studio 2010 is dit een vereiste. De command line compiler van Visual Studio 2010 heeft de neiging om na het builden een samenvatting van fouten naar de output te zenden. Dit zorgt ervoor dat fouten meerdere keren voorkomen in het resultatenscherm. Dit kan als gevolg hebben dat iSE meerdere keren tracht dezelfde fout op te lossen. iSE is op deze manier in staat om zijn opgeloste fouten te 'vernielen'. iSE verwijdert niet automatisch duplicate fouten, omdat er op dezelfde plaats in een broncodebestand meerdere (verschillende) fouten kunnen voorkomen. Het wordt aan de gebruiker van dit programma overgelaten om duplicate fouten te verwijderen of niet.

Soms worden fouten weergegeven met een verkeerde bestandslocatie. Het komt soms voor dat het weergegeven pad een relatief pad is. Het komt ook voor dat er gewezen wordt naar een Microsoft bestand omdat daar de fout werd gedetecteerd (maar niet veroorzaakt), in dat

[1]Zie hoofdstuk 5.2.4 op p.50 voor een kort overzicht voor het toevoegen van functionaliteit aan iSE.

geval is vaak ook het lijnnummer verkeerd. Het is dan mogelijk om met de rechtermuisknop te klikken op de fout en in het verschenen menu de optie "Change file to process by the solution of this error..." aan te klikken. Hierop verschijnt er een venster waarin het mogelijk is om het eigenlijke foutbestand te selecteren en eventueel het lijnnummer van de fout aan te passen (figuur 5.4 op p.34). Door de checkbox "Replace the filepath for all errors that contain this incorrect filepath" aan te vinken worden alle andere build errors[2] met dezelfde foutieve bestandslocatie ook aangepast.



Figuur 5.4: Het dialoogvenster van iSE om de bestandslocatie aan te passen.

Terugkeren naar het hoofdscherm kan door het resultatenscherm af te sluiten. De "Show results"-knop kan dan gebruikt worden om het laatste resultatenscherm terug weer te geven.

5.2 Achter de schermen

5.2.1 Overzicht

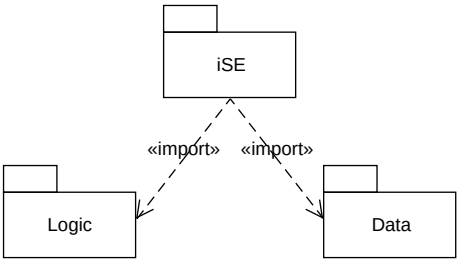
iSE is geschreven in Visual C# (.NET framework 4) en bestaat uit 3 grote delen. Elk deel is een project in de iSE solution:

iSE
: Dit is een Windows Presentation Foundation (WPF)-project. Het bevat de GUI en verbindt de Logic en Data projecten. Er werd eerst geopteerd voor Windows Forms, maar door betere performantie en een groter aanbod van functionaliteit werd dan toch voor WPF gekozen.

Logic
: Dit project bevat de logica zelf. Hier gebeurt al het werk: het aanspreken van de command line compiler, het koppelen van fouten en hun oplossingen, het oplossen van fouten, ...

Data
: Om de logica te kunnen toepassen, moet de broncode- en configuratie bestanden ingelezen en geschreven kunnen worden. Dit project bezit de daarvoor nodige functies.

[2]Of alle andere warnings als de fout waarop is geklikt met de rechtermuisknop een warning is.



Figuur 5.5: De verschillende projecten in iSE.

iSE steunt zowel op Logic als op Data (figuur 5.5 op p.35). Het vraagt input aan de gebruiker, gebruikt Data om de nodige bestanden in te lezen en roept Logic op om het eigenlijke werk zelf uit te voeren. iSE is multithreaded opgesteld: de GUI-thread is steeds gescheiden van de thread die het werk uitvoert. Op deze manier loopt de GUI niet vast en kan er in realtime output worden weergegeven. Toen iSE nog singlethreaded was opgebouwd tijdens de ontwikkeling had men vaak de indruk dat het vastliep omdat men geen informatie kreeg over wat iSE uitvoerde. Dit was vooral het geval als iSE een solution of project liet verwerken door de command line compiler. Dit kon voor sommige projecten gemakkelijk een tiental minuten duren. Daardoor werd het onduidelijk of iSE ergens bleef hangen of niet. Het is ook altijd voor gebruikers (en ontwikkelaars) aangenamer als men kan volgen wat er allemaal gebeurt en of alles goed verloopt. Door meerdere threads te gebruiken kan er informatie van de command line compiler doorgeven worden aan de GUI en is de gebruiker meteen op de hoogte als er fouten optreden of wil weten hoeveel de command line compiler heeft verwerkt.

De scheiding tussen GUI, logica en databeheer werd oorspronkelijk geïmplementeerd door namespaces en folders. Dit betekende dat de gehele solution uit één enkel project bestond. Bij de ontwikkeling van rc2form werd er een alternatieve aanpak gevolgd en elk deel in zijn eigen project geplaatst. Op deze manier kunnen GUI, logica en data met elkaar samenwerken en is er een duidelijkere en betere modulariteit. Dit bleek vooral handig bij het debuggen.

5.2.2 Uitgebreide workflow

Opstarten en inlezen van configuratiebestanden

Zodra iSE opstart wordt het hoofdscherm getoond en begint het inladen van configuratiegegevens. Om er voor te zorgen dat de GUI niet vastloopt werd er multithreading geïmplementeerd via `Task`-instanties uit de Task Parallel Library (TPL). De gebruiker kan ondertussen het doen en laten van iSE volgen via de statusbalk.

Het inladen van de configuratiegegevens gebeurt met behulp van deserialisatie. Objecten worden geëxtraheerd uit XML-bestanden met behulp van de `XmlSerializer`-klasse in het .NET framework. iSE bevat twee configuratiebestanden: *VisualStudios.xml* en *SourceControls.xml*. Deze bestanden bevatten informatie om aanwezige Visual Studio versies en versiebeheerprogramma's te detecteren op het systeem van de gebruiker.

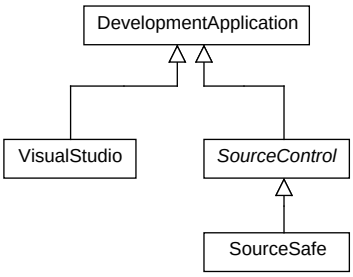
De locatie van de bestanden ligt vast in de *App.config* bestand van het iSE project (listing 5.1 op p.35).

```
1 <?xml version="1.0" encoding="utf-8" ?>
```

```
<configuration>
  <appSettings>
    <add key="VisualStudioXmlPath" value="ConfigurationFiles\visualstudios.xml"/>
    <add key="SourceControlXmlPath" value="ConfigurationFiles\sourcecontrols.xml"/>
  </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

Listing 5.1: Het *App.config* bestand van het project iSE.

Voor het deserializeren van de XML-structuren moeten deze voldoen aan een specifieke structuur. Deze structuur is gebaseerd op de klassestructuur van de klassen waarvan de instanties Visual Studio versies en versiebeheerprogramma's voorstellen (figuur 5.6 op p.36). Een voorbeeld van de XML-structuur van *VisualStudios.xml* en *SourceControls.xml* bevindt zich respectievelijk in de listings 5.2 en 5.3.



Figuur 5.6: De klassestructuur van de congiguratieklassen in iSE.

```
<?xml version="1.0"?>
<ArrayOfVisualStudio xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <VisualStudio>
        <Name>Visual Studio 6.0</Name>
        <Number>6</Number>
        <RegValue>InstallDir</RegValue>
        <!-- RegValue must come before Regkeys! -->
        <Regkey32>SOFTWARE\Microsoft\VisualStudio\6.0</Regkey32>
        <Regkey64>SOFTWARE\Wow6432Node\Microsoft\VisualStudio\6.0</Regkey64>
    </VisualStudio>
</ArrayOfVisualStudio>
```

Listing 5.2: VisualStudios.xml

```
<?xml version="1.0"?>
<ArrayOfSourceControl xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SourceControl xsi:type="SourceSafe">
        <Name>Visual SourceSafe</Name>
        <RegValue>SCCServerPath</RegValue>
        <!-- RegValue must come before Regkeys! -->
        <Regkey32>SOFTWARE\Microsoft\SourceSafe</Regkey32>
        <Regkey64>SOFTWARE\Wow6432Node\Microsoft\SourceSafe</Regkey64>
    </SourceControl>
</ArrayOfSourceControl>
```

Listing 5.3: SourceControls.xml

De configuratiebestanden hebben beide een analoge structuur. Het grote verschil zit in het feit dat in elke `SourceControl`-tag een type moet worden meegegeven. Dit type bepaalt het klassetype van het object dat gedeserializeerd wordt op basis van die specifieke `SourceControl`-tag. Dit komt omdat er voor elk versiebeheerprogramma een specifieke `CheckOut`-methode moet geïmplementeerd worden. Als oplossing is er een abstracte `SourceControl`-klasse met een abstracte `CheckOut`-methode. Voor elke implementatie van een versiebeheerprogramma moet er een klasse worden aangemaakt die overerft van `SourceControl` en de functie `CheckOut` overschrijft met een eigen implementatie. Zo hoeft men zich via polymorphisme bij het uitchecken van bestanden niets aan te trekken van de implementatie van de `CheckOut`-methode (zie hoofdstuk 5.2.2 op p.49). In *VisualStudios.xml* komt dit niet voor en wordt alles in instanties van de `VisualStudio`-klasse geladen.

Voor de rest zijn de tags (buiten de `ArrayOf`-tags) identiek gelijk:

`Name`
: De naam van de Visual Studio versie of het versiebeheerprogramma.

`Number`
: Het versienummer van de Visual Studio versie. Deze tag komt niet voor in *SourceControls.xml*.

`RegValue`
: Dit is de registerwaarde waaruit het pad van de command line compiler kan berekend worden.

`Regkey32`
: De registertak waar op 32-bits computers de waarde van `RegValue` kan teruggevonden worden. De registertak wordt relatief bepaald t.o.v. `HKEY_CLASSES_ROOT`. Om de tak "`HKEY_CLASSES_ROOT\X\Y`" op te geven, hoeft enkel "`X\Y`" tussen de tags worden gezet.

`Regkey64`
: De registertak waar op 64-bits computers de waarde van `RegValue` kan teruggevonden worden. Net als bij `Regkey32` wordt de registertak relatief bepaald t.o.v. `HKEY_CLASSES_ROOT`.

Het deserialiseren gebeurt met de `Read`-methode uit de `XmlIO`-klasse in het Data project. Er wordt een `FileStream` gegenereerd van het XML-bestand en meegegeven aan een `XmlSerializer`. Dit is een statische methode die gebruikt kan worden om zowel een container van `VisualStudio`-objecten als een container van `SourceControl`-objecten te genereren. Het is belangrijk dat de `RegValue`-tag wordt gedefinieerd voor de `Regkey32`- en `Regkey64`-tags. Van zodra de `Regkey32`- en `Regkey64`-properties in de bovenliggende `DevelopmentApplication`-klasse worden ingevuld, wordt meteen nagegaan of het programma, dat de instantie voorstelt, aanwezig is. Als de volgorde toch is omgewisseld, zal de `RegValue`-property niet ingevuld zijn en zullen dus de nodige waarden, om te detecteren of het programma is geïnstalleerd, niet gevonden worden. Het programma wordt dan dus niet als geïnstalleerd beschouwd ongeacht of het op het systeem van de gebruiker is geïnstalleerd of niet.

Het bepalen of een programma is geïnstalleerd gebeurt dus via het register. Er wordt geprobeerd de waarde op te halen in de registertak beschreven in de `Regkey32`- en `Regkey64`-tags

(iSE bepaalt zelf of het uitgevoerd wordt in een 32 of 64 bitomgeving en kiest de overeenkomstige registertak). In deze tak wordt de waarde opgehaald van de registervariabele in de `RegValue`-tag. Als dit niet blijkt te lukken, of als de opgehaalde waarde geen inhoud blijkt te hebben, wordt het programma als niet geïnstalleerd beschouwd. De opgehaalde gegevens worden (in de meeste gevallen) nog gebruikt voor het vinden van de locatie van het programma (zie hoofdstuk 5.2.2 op p.38).

Het zetten van de `Regkey32`- en `Regkey64`-properties is gekoppeld aan de methodes die bepalen of het programma al dan niet is geïnstalleerd. Zo moeten er geen aparte methoden worden voorzien. Het booleaanse attribuut `isPresent` is eigenlijk een samengesteld attribuut op basis van `RegValue`, `Regkey32` en `Regkey64`. Er kunnen inconsistenties ontstaan als het bepalen van `isPresent` wordt losgekoppeld[3] van de setters, in de properties van `RegValue`, `Regkey32` en `Regkey64`. Het gevaar van dergelijke inconsistenties wordt groter geacht dan de hinder die mogelijks ondervonden kan worden door de koppeling.

De gevonden geïnstalleerde applicaties op het systeem van de gebruiker worden hierop ingeladen in de GUI. Dan pas krijgt de gebruiker controle over de controls op het hoofdscherm. Er werd voor deze aanpak gekozen omdat ze twee vliegen in één klap slaat: de gebruiker krijgt up-to-date informatie over zijn mogelijkheden voor conversie en iSE krijgt de nodige informatie voor verdere bewerkingen. Het inlezen van alle waarden in beide XML-bestanden is geen bedreiging voor de performantie. Er zijn op dit moment slechts een handvol versies van Visual Studio die kunnen aangesproken worden voor de conversie en ook voor versiebeheer worden er zelden meer dan één of twee applicaties in eenzelfde (bedrijfs)omgeving gebruikt. De XML-bestanden kunnen steeds naar wens aangepast worden indien men vindt dat er toch teveel (zinloos) werk wordt uitgevoerd. De flexibiliteit van de XML-bestanden is ook één van de grootste pluspunten van deze aanpak. Zie hoofdstuk 5.2.4 op p.50 voor meer informatie over de flexibiliteit van iSE.

Aanspreken van de command line buildsystemen

Als de gebruiker alle opties heeft geselecteerd, kan hij het project laten builden. Zodra er op de "Build"-knop wordt gedrukt, wordt er een instantie van de `ApplicationSolution`-klasse aangemaakt. Deze klasse stelt een Visual Studio solution voor en bevat alle nodige informatie en methodes om de solution te builden en het resultaat ervan te analyseren. Er werd gekozen voor de naam `ApplicationSolution` om aan te duiden dat het een Visual Studio solution voorstelt en geen oplossing voor een buildfout.

Meteen daarna wordt de GUI aan de events `compilerOutput`, `output` en `errorOutput` van de `ApplicationSolution`-instantie gekoppeld. Deze events zijn van het type `OutputDelegate` en aanvaarden een `string`-object (listing 5.4 op p.38).

```
1 public delegate void OutputDelegate(string output);
```

Listing 5.4: De delegate `OutputDelegate`.

Voor de eenvoud zijn deze publiek beschikbaar omdat dankzij het keyword `event` de compiler de nodige restricties oplegt op de delegates in het events.

> "*By adding the* `event` *keyword, you limit the operations to += and -= (...) The use of the* `event` *keyword is the one time where it's okay to make a field* `public`*,*

[3] Bijvoorbeeld door de bepaling zelf in een aparte functie te plaatsen die apart opgeroepen moet worden.

> *because the compiler narrows the use to safe operations.*" (Sells in: HejlsBerg et al., 2011)

Het builden van de solution gebeurt in een andere thread dan de GUI-thread. Hierdoor kan de GUI informatie van de delegates in realtime verwerken. Hiervoor moet wel in de eventhandlers gekoppeld aan de delegates nagegaan worden of de eventhandler wordt opgeroepen vanuit de eigen (GUI-)thread of een andere thread. De ene thread kan immers niet zomaar aangesproken worden vanuit de andere thread. Als blijkt dat de eventhandler wordt aangesproken vanuit een andere thread dan de GUI-thread, zal aan de GUI-thread gevraagd worden om de eventhandler op te roepen.

Tijdens de ontwikkeling werd er in eerste instantie een beroep gedaan op het doorgeven van delegates in de constructor i.p.v. het samenwerken van delegates en eventhandlers. Dit leek een goede oplossing, maar naarmate de complexiteit van het programma groeide bleek het steeds meer restricties met zich mee te brengen. Het was geen flexibele implementatie. De logica in Logic was nu verplicht rekening te houden met de GUI. De constructor moest immers een delegate krijgen en met deze delegate omgaan. Er waren ook problemen met uitbreidbaarheid. Men wou verschillende verwerkingen voor verschillende boodschappen zoals gewone berichten, foutmeldingen en waarschuwingen. Om deze verwerkingen te implementeren moesten ofwel meerdere delegates aan de constructor worden meegegeven en de logica worden aangepast, ofwel een extra parameter geïmplementeerd worden in de delegate die de aard van de boodschap aanduidt. Deze extra parameter moest dan ook nog verwerkt worden in de eventhandler. Een absurdere oplossing was de invoer string testen op basis van reguliere expressies en op basis daarvan op een bepaalde manier te verwerken. Geen enkele van deze oplossingen bood een mooie scheiding van logica en GUI en dus werd er uiteindelijk de stap gezet naar delegates en events. Nu biedt de logica mogelijkheden aan en staat de GUI vrij om er iets mee te aan te vangen.

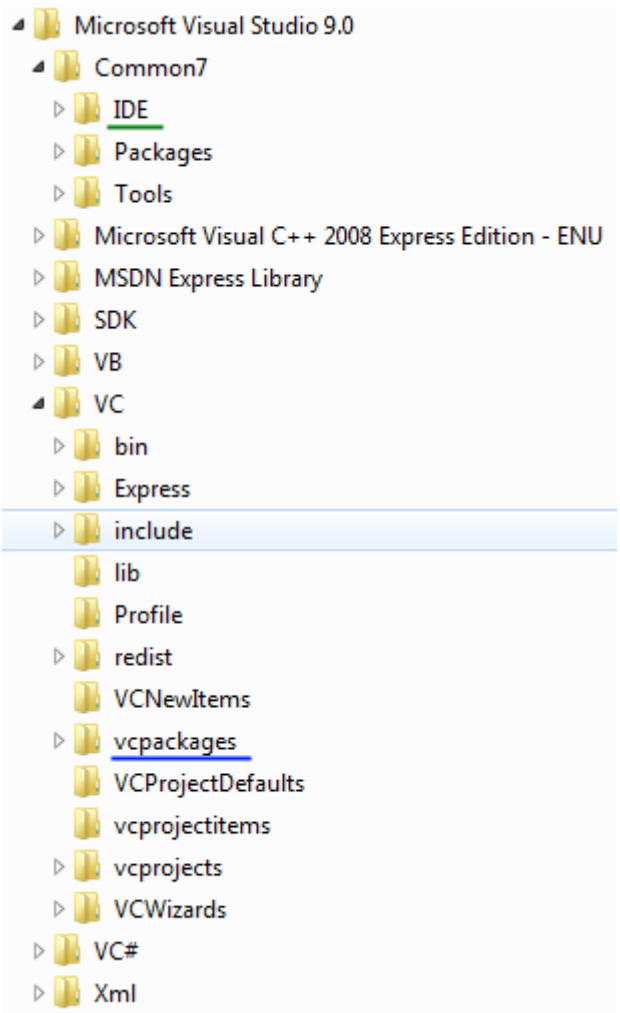
In de nieuwe thread worden achtereenvolgens de `Build-` en `ProcessResults`-methode aangeroepen om het project respectievelijk te builden en daarna de resultaten van de build te verwerken.

Om te builden wordt er een command line proces aangemaakt met de `Process`-klasse uit het .NET framework. Afhankelijk van de versie van de geselecteerde Visual Studio kan dit proces kiezen uit de buildsystemen besproken in hoofdstuk 3.2.2 op p.24: VCBuild en MSBuild. De keuze wordt bepaald door het versienummer van de geselecteerde Visual Studio. Als het versienummer kleiner is dan 10 wordt VCBuild gebruikt, anders MSBuild. Om de buildsystemen aan te spreken moeten de locaties van de buildsystemen bepaald worden. Dit gebeurd voor elke buildsysteem op een andere manier:

VCBuild

VCBuild wordt meegeleverd met Visual Studio. Dit wil zeggen dat *vcbuild.exe* zich bevindt in de mappenstructuur van Visual Studio. De locatie van VCBuild is "`.\Microsoft Visual Studio <versienummer>\VC\vcpackages\vcbuild.exe`". Het pad dat wordt ingelezen bij het deserialiseren van *VisualStudios.xml* is "`.\Microsoft Visual Studio <versienummer>\Common7\IDE\`". Vanaf versienummer 9 worden gehele versienummers in de naam van de Visual Studio map gevolgd met de postfix ".0" (bv. "Microsoft Visual Studio 8", "Microsoft Visual Studio 9.0", "Microsoft Visual Studio 10.0", ...). De locatie van *vcbuild.exe* kan dus ook geschreven worden als "`.\Microsoft Visual Studio <versienummer>\Common7\IDE\..\..\VC\`

`vcpackages\vcbuild.exe`". Op deze manier worden de ingelezen registerwaarden gebruikt om de locatie van *vcbuild.exe* te achterhalen. Een overzicht van de mappenstructuur van Visual Studio bevindt zich in figuur 5.7 op p.40.



Figuur 5.7: De mappenstructuur van Visual Studio 2008. De folderlocatie opgehaald uit het register is onderlijnd in groen, de locatie van *vcbuild.exe* is onderlijnd in blauw.

MSBuild

De folder van *MSBuild.exe* kan simpelweg opgevraagd worden aan het .NET framewerk met de methode `RuntimeEnvironment.GetRuntimeDirectory()`. Het volledige pad wordt dan "`RuntimeEnvironment.GetRuntimeDirectory() + "MSBuild.exe"`"[4]. Bij elke versie van het .NET framework wordt er een *MSBuild.exe* meegeleverd. Er zijn vaak verschillende versies op een systeem aanwezig: op het systeem waarop deze scriptie werd geschreven werden er alleen al negen *MSBuild.exe* bestanden gevonden (figuur 5.8 op p.41). Ook is er geen verwijzing in de registertak van Visual Studio naar de locatie van MSBuild. Dit komt door de scheiding van IDE en buildsysteem besproken

[4]Ondanks het feit dat in de command prompt de suffix ".exe" niet moet vermeld worden, bleek dit wel nodig te zijn in het `Process`-object voor MSBuild in iSE.

in hoofdstuk 3.2.2 op p.24. Het gebruik van deze methode omzeilt al deze problemen. Deze methode zal het *MSBuild.exe*-bestand nemen overeenkomstig met frameworkversie 4.0 (Visual Studio 2010) omdat iSE zelf ingesteld staat op frameworkversie 4.0. Omdat deze masterproef enkel reikt tot Visual Studio 2010 is er nog geen ondersteuning geïmplementeerd om zelf andere frameworkversies in te stellen.

Name	Path
MSBuild.exe	C:\Windows\Microsoft.NET\Framework64\v4.0.30319
MSBuild.exe	C:\Windows\Microsoft.NET\Framework\v4.0.30319
MSBuild.exe	C:\Windows\winsxs\amd64_msbuild_b03f5f7f11d50a3a_3.5.7600.16385_none_e85b8cfa614685bd
MSBuild.exe	C:\Windows\winsxs\amd64_msbuild_b03f5f7f11d50a3a_3.5.7601.17514_none_ea8ca0c25e350957
MSBuild.exe	C:\Windows\winsxs\amd64_msbuild_b03f5f7f11d50a3a_6.1.7600.16385_none_0e0d302b5908105b
MSBuild.exe	C:\Windows\winsxs\amd64_msbuild_b03f5f7f11d50a3a_6.1.7601.17514_none_0de23daf595f5711
MSBuild.exe	C:\Windows\winsxs\x86_msbuild_b03f5f7f11d50a3a_3.5.7600.16385_none_8c3cf176a8e91487
MSBuild.exe	C:\Windows\winsxs\x86_msbuild_b03f5f7f11d50a3a_6.1.7600.16385_none_55ba67026d843961
MSBuild.exe	C:\Windows\winsxs\x86_msbuild_b03f5f7f11d50a3a_6.1.7601.17514_none_558f74866ddb8017

Figuur 5.8: Verschillende versies van *MSBuild.exe* op een enkel systeem.

Om ervoor te zorgen dat alle code wordt bekeken worden de solutions gerebuild in debugmode. Dit wordt ingesteld door de command line parameters "`/rebuild "<path naar sln-bestand>" "Debug|Win32"`" voor VCBuild en "`/t:rebuild "<path naar sln-bestand>" /p:Configuration=Debug`" voor MSBuild.

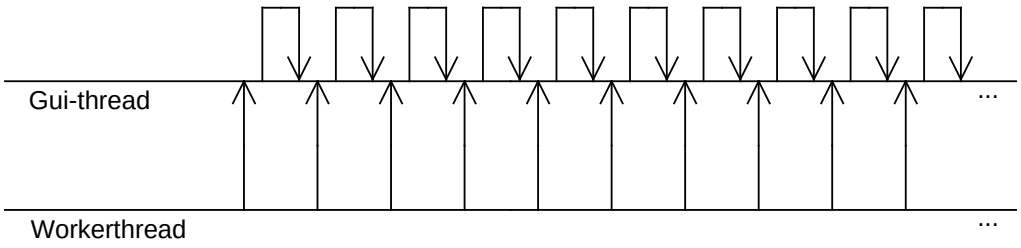
Aan het buildprocess, voorgesteld door een `Process`-instantie, zijn er twee eventhandlers verbonden: `OutputHandler` en `ErrorHandler`. Zij geven de output van het buildsysteem lijn per lijn terug voor respectievelijk standaard output en output voor foutmeldingen. De output wordt dan zelf weer doorgegeven aan de eventhandlers verbonden met de `ApplicationSolution`. Op deze manier wordt rechtstreeks in het hoofdscherm weergegeven wat het buildsysteem doet en of er problemen zijn opgetreden.

Men vroeg zich tijdens de ontwikkeling van iSE af of het uitschrijven van alle output zou zorgen voor performantieproblemen. Elke lijn wordt immers opgevangen in een eventhandler verbonden met de `Process`-instantie en daarna weer verder opgegooid naar een eventhandler van de GUI. Daar wordt dan nagekeken of de eventhandler wordt opgeroepen vanuit de GUI-thread. Indien dit niet het geval is moet er aan de GUI-thread gevraagd worden om de eventhandler uit te voeren met de ontvangen parameters uit de andere thread. Dit leidt tot het schema in figuur 5.9 op p.42.

Het is dankzij deze output dat foutmeldingen onderschept kunnen worden. De eventhandler `OutputHandler` is naast het doorgeven van output ook verantwoordelijk voor een oppervlakkige scheiding tussen gewone output en output met een grote kans dat het om een foutmelding gaat. Het voert deze scheiding uit met de simpele reguliere expressie in listing 5.2.2 op p.41.

```
1 /(error|warning) \w+\d+/i
```

Uitvoerlijnen die aan deze reguliere expressie voldoen worden opgeslagen in een List van string-objecten. Elke string stelt een lijn voor. In de voorbeelden van typische foutmeldingen in listing 5.2.2 op p.41 worden hiermee de delen "`error C4430`" en "`warning C4996`" gedetecteerd.



Figuur 5.9: Interactie tussen de Gui-thread en de werkthread bij het uitvoeren van het Build-Systeem.

```
c:\sourcesafe\vc60wnet\libraries\czam\czam_vic107data.cpp(41) : error C4430: missing
    type specifier - int assumed. Note: C++ does not support default-int

1>c:\sourcesafe\vc60wnet\libraries\czam\czamreaderwriter.cpp(174) : warning C4996:
    'strcpy': This function or variable may be unsafe. Consider using strcpy_s
    instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for
    details.
```

Deze kleine reguliere expressie is niet voldoende voor de verwerking van de foutmelding, enkel voor een ruwe scheiding van nuttige en niet nuttige informatie. Er kan geen gedetailleerde informatie uit de foutmelding geëxtraheerd worden en er is een grote kans op valse positieven. Buildsteps en andere programma's kunnen ook output genereren. Als bijvoorbeeld de solution bestanden heeft staan in een map "messages for warning 1 to 50" (paden met spaties zijn toegestaan in Windows, maar worden niet aangeraden.) en het buildsysteem overloopt alle bestanden in de map, worden alle lijnen als foutmelding geregistreerd telkens de mapnaam ook in de uitvoer voorkomt.

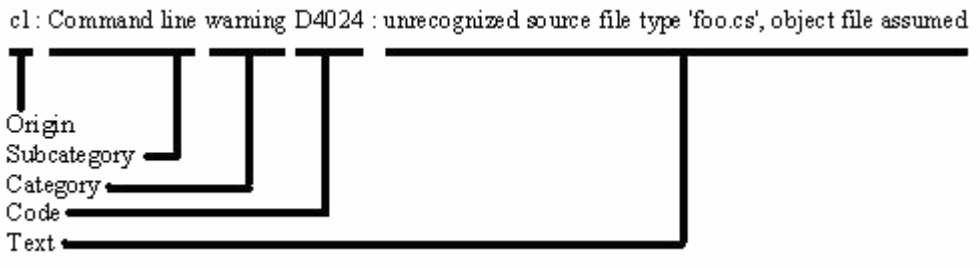
Om de gefilterde uitvoer verder te verwerken wordt de `ProcessResults()`-methode van de `ApplicationSolution`-instantie opgeroepen. In deze methode worden alle mogelijke foutmeldingen overlopen en vergeleken met een geavanceerde reguliere expressie. Indien de mogelijke foutmelding niet voldoet, wordt ze verworpen. Als de foutmelding wel voldoet, dan is meteen ook alle mogelijke informatie opgeslagen.

De structuur van de reguliere expressie bevindt zich in listing 5.2.2 op p.42. Ze is gebaseerd op de standaardstructuur van foutmeldingen gegenereerd door VCBuild en MSBuild.

```
/
(
    (
        (\d+>)?
        (?<fileorprocess>[^/*?"<>]*?)
            (\(
                (?<linenumber>\d+)
                (,\s?
                    (?<column>\d+)
                )?\)
            )?
        )\s?:\s
    )?\s?
    (?<fullerrorcode>(
        (\w*?\s)*
    )
```

```
17      (fatal\s)?
18      (?<warningorerror>warning|error)\s
19      (?<errorcode>\w+\d+)
20  )\s?:\s
21  (?<description>.*)
22  /xgo
```

Deze structuur bestaat volgens Fourie (2006) uit vijf grote delen (figuur 5.10 op p.43):



Figuur 5.10: Structuur van foutboodschappen in MSBuild. (Fourie, 2006)

Origin
Dit geeft de oorsprong aan van de fout. Dit kan de naam zijn van een toepassing of een pad naar een bestand (het pad mag zowel absoluut als relatief zijn). Indien het om een bestand gaat wordt dit gevolgd door informatie over de lijn of kolom waarin de fout voorkwam in een van de vormen '(lijn)', '(lijn-lijn)', '(lijn-kolom)', '(lijn,kolom-kolom)' of '(lijn,kolom,lijn,kolom)'[5]. Lijn- en kolomnummers worden telkens geteld vanaf 1.

Subcategory (optioneel)
Hier kan er meer informatie gegeven worden over het type van de fout.

Category
Er wordt hier aangeduid of het gaat om een fout of een waarschuwing met de termen "`warning`" of "`error`". Hoofdletters of kleine letters maakt niet uit.

Code
Dit is een code die de fout identificeert en mag geen spaties bevatten.

text (optioneel)
Een beschrijving van de fout.

(Fourie, 2006)

In de reguliere expressie zitten er *named capture groups*[6]. Deze groepen bevatten de geëxtraheerde informatie uit de foutmelding. De named capture groups zijn `fileorprocess`, `linenumber`, `column`, `fullerrorcode`, `warningorerror`, `errorcode` en `description`. Tabel 5.1 op p.44 bevat de resultaten als deze worden toegepast op de foutlijn in figuur 5.10 op p.43.

[5] "lijn" en "kolom" stellen hier respectievelijk de lijn- en kolomnummers voor.

[6] Named capture groeps zijn onderdelen in een reguliere expressie die informatie uit de waarde, die aan de reguliere expressie wordt getoetst, halen. Deze groepen hebben als syntax '`(?<naam>...)`' en kunnen aan de hand van hun naam aangesproken worden.

group	waarde
fileorprocess	cl
fullerrorcode	Command line warning D4024
warningorerror	warning
errorcode	D4024
description	unrecognized source file type 'foo.cs', object file assumed

Tabel 5.1: De resultaat van de reguliere expressie in listing 5.2.2 op p.42 toegepast op het voorbeeld in figuur 5.10 op p.43.

Indien de uitvoerlijn voldoet aan de reguliere expressie wordt alle informatie uit de foutmelding opgeslagen in een `BuildAlert`-object. Per solution wordt er zowel voor waarschuwingen als buildfouten een `List` van `BuildAlerts` bijgehouden. Tijdens de ontwikkeling van iSE was dit eerst geïmplementeerd met een enkele `List`. Het bleek uiteindelijk makkelijker, performanter en vooral duidelijker om te werken met twee `List`-objecten. Na het aanmaken van een overeenkomstig `BuildAlert`-object wordt er ook geprobeerd een oplossing te koppelen aan dit `BuildAlert`-object. Dit wordt verder besproken in hoofdstuk 5.2.2 op p.45.

Het filteren zelf werd in twee stappen gedaan om de performantie te verbeteren. De uitgebreide reguliere expressie gebruiken voor elke lijn output bleek veel werkkracht te vragen van het systeem. Op gegeven momenten werd er zelfs gedacht dat iSE bleef hangen, maar bleek het een uitvoerlijn te zijn die een kwartier nodig had om verwerkt te worden door de reguliere expressie. Dankzij het tweestap-systeem wordt door middel van een zeer korte en snelle reguliere expressie het kaf van het koren gescheiden en hoeft de geavanceerde reguliere expressie pas uitgevoerd te worden in gevallen waarbij het bijna altijd om een foutmelding gaat. Omdat dit soms nog voor problemen kan zorgen werd de notatie "`.*`" zoveel mogelijk vermeden. Een voorbeeld hiervan is "`(?<fileorprocess>[^/*?"<>]*?)`". Dit was oorspronkelijk geïmplementeerd als "`(?<fileorprocess>.*?)`". Dankzij deze truc verbeterde de performantie significant.

Deze reguliere expressie wordt ook slechts eenmaal gecompileerd dankzij de optie `RegexOptions.Compiled`. Dit gaat ten koste van een onmerkbare vertraging bij het opstarten van iSE.

Voor de tweede stap in het tweestap process wordt er gebruik gemaakt van een multithreaded `for`-lus. Hierdoor wordt de `List` van `string`-objecten opgedeeld naar in het aantal beschikbare processoren. Deze delen worden dan tegelijkertijd verwerkt. Om ervoor te zorgen dat de threads niet tegelijkertijd `BuildAlert`-objecten toevoegen aan de lijsten voor fouten en waarschuwingen van het project, wordt er gebruik gemaakt van het `lock`-mechanisme (listing 5.5 op p.44). Dit zorgt ervoor dat als er een thread naar een `BuildAlert`-lijst aan het schrijven is, de andere thread hun beurt moeten afwachten. Zo worden er conflicten tussen threads vermeden.

```
1  object lockObjectErrors = new object();
2  object lockObjectWarnings = new object();
3  // processOutput is een List<string> en
4  // wordt meegegeven als parameter.
5  // Het bevat alle foutmeldingen gefilterd in de eerste stap.
6  Parallel.ForEach<string>(processOutput, (item) =>
7      {
8          //...
9          lock (lockObjectWarnings) { warnings.Add(warning); }
10         //...
```

```
11          lock (lockObjectErrors) { errors.Add(error); }
12          //...
13      });
```

Listing 5.5: Het `lock`-mechanisme.

Koppelen van oplossingen aan fouten

Het koppelen van oplossingen aan fouten gebeurt door de `SolutionFactory`-klasse. De klasse bevat één enkele methode `LinkBuildAlert` en een reeks reguliere expressies. Een extract van de `LinkBuildAlert`-methode bevindt zich in listing 5.2.2 op p.45.

Voor elke oplossing is er een reguliere expressie aanwezig. Via '`if-else if-else`'-structuren wordt er gekeken of het om een build error of een warning gaat en aan welke reguliere expressie wordt voldaan. Het resultaat van de methode is een boolean die aangeeft of er een oplossing is gevonden voor de `BuildAlert`.

Het is belangrijk om na te gaan of foutmeldingen niet door meerdere expressies tegelijk kunnen worden aanvaard. Indien dit het geval zou zijn zal de eerste reguliere expressie waaraan de foutmelding voldoet worden verkozen. Dit kan ook gebruikt worden om algemene fouten en meer specifieke fouten van dezelfde soort van elkaar te onderscheiden. Dan moet ervoor gezorgd worden dat de expressies voor de algemene foutmelding vóór die van de specifiekere foutmeldingen komen in de '`if-else if-else`'-structuren.

```
1   private const string cexceptionExpr =
2   "C2259: 'CException' : cannot instantiate abstract class";
3   // rest van de reguliere expressies
4
5   public static bool LinkBuildAlert(BuildAlert ba)
6   {
7       if (!ba.IsWarning) // errors
8       {
9           if (Regex.IsMatch(ba.ErrorCode + ": " + ba.Description, cexceptionExpr))
10          {
11              ba.Solution = new CException();
12          }
13          else if (Regex.IsMatch(ba.ErrorCode + ": " + ba.Description,
                missingTypeSpecifierExpr))
14          {
15              ba.Solution = new MissingTypeSpecifier();
16          }
17          // ...
18          else // no solution found for this error
19          {
20              ba.Solution = null;
21              return false;
22          }
23          return true;
24      }
25      else // warnings
26      {
27          if (Regex.IsMatch(ba.ErrorCode + ": " + ba.Description, lossOfDataExpr))
28          {
29              Match m = Regex.Match(ba.ErrorCode + ": " + ba.Description,
                    lossOfDataExpr);
30              ba.Solution = new LossOfData(m.Groups[1].Value, m.Groups[2].Value,
                    m.Groups[3].Value);
31          }
32          else if (Regex.IsMatch(ba.ErrorCode + ": " + ba.Description,
                unsafeFunctionExpr))
33          {
```

```
34              Match m = Regex.Match(ba.ErrorCode + ": " + ba.Description,
                    unsafeFunctionExpr);
35              ba.Solution = new UnsafeFunction(m.Groups[1].Value, m.Groups[2].Value);
36          }
37          // ...
38          else // no solution found for this warning
39          {
40              ba.Solution = null;
41              return false;
42          }
43          return true;
44      }
45  }
```

Er werd voor deze structuur gekozen om een grote flexibiliteit te kunnen aanbieden. Elke oplossing heeft zijn eigen specifieke informatie nodig uit de foutmelding van de `BuildAlert`. Dankzij deze structuur moeten niet alle constructors uniek zijn en kan er voor elke oplossing aparte logica geïmplementeerd worden om informatie uit de foutmelding te halen. Zie hoofdstuk 5.2.4 op p.50 voor meer informatie over het toevoegen en verwijderen van oplossingen.

Er werd aanvankelijk gedacht om eventueel te werken met *runtime compilation*. Runtime compilation houdt in dat er broncodebestanden worden gecompileerd terwijl het programma loopt. Voor elke oplossing zouden er dan op een bepaalde plaats de nodige broncodebestanden moeten staan. Op deze manier zou iedereen oplossingen kunnen toevoegen zonder dat er in de broncode van iSE gewerkt zou moeten worden. iSE zou dan alle oplossingsbestanden inlezen en compileren. Bij nader inzien bleek dit toch geen goed idee. iSE zou elke keer als het start de bestanden moeten compileren. Het ontwikkelen van oplossingen is niet gemakkelijk en vereist veel debuggen. Ook zouden de methodes van ingebouwde tools (zoals de klasse `ScopDetector`) in iSE niet zichtbaar zijn met *intellisense*[7]. Uiteindelijk zou dan toch de solution van iSE in Visual Studio gebruikt moeten worden en dan kunnen de oplossingen evengoed samen met iSE gecompileerd worden.

Toepassen van oplossingen

Wanneer er een fout wordt opgelost, wordt de gekoppelde oplossingsklasse opgeroepen. Deze klasse erft over van de `ISolution` interface en bevat de `Solve`-methode (listing 5.6 op p.46).

```
1  using System.Collections.Generic;
2
3  namespace iSE.Logic
4  {
5      public interface ISolution
6      {
7          List<string> Solve(List<string> code,
8                             int linenumber,
9                             out List<int> insertedLines,
10                            out List<int> changedLines);
11     }
12 }
```

Listing 5.6: De interface ISolution.

De `Solve`-methode bevat volgende parameters:

`List<string> code`
: Een lijst van strings, waarbij elke string een lijn bevat van het sourcecode-bestand dat de fout bevat.

[7] intellisense is een hulpmiddel in Visual Studio dat zorgt voor gekleurde syntax en auto-aanvulling.

`int linenumber`
: Het nummer van de lijn waar de fout zich bevindt, geteld vanaf 1.

`List<int> insertedLines`
: Dit is een uitvoerparameter. Elke keer als er een lijn door de oplossing wordt toegevoegd, wordt het lijnnummer van deze lijn opgeslagen in een `List` van `integer`s. Dit laat toe om de GUI up te daten, samen met lijnnummers van fouten in hetzelfde sourcecodebestand wiens lijnnummer door het invoegen van lijnen wordt aangetast.

`List<int> changedLines`
: Analoog aan insertedLines, maar hier worden enkel de lijnnummers opgeslagen van lijnen die door de oplossing worden gewijzigd.

De moeilijkheid van de parameters `insertedLines` en `changedLines` ligt in de consistentie. Nummers van gewijzigde en ingevoegde lijnen moeten na het invoegen (of verwijderen) van een lijn worden aangepast[8]. Na het uitvoeren van de oplossing moeten deze parameters naar de correcte gewijzigde en ingevoerde lijnen wijzen. Ook op hoger niveau moet hiermee rekening gehouden worden. Het invoegen van lijnen tast ook de lijnnummers van fouten in hetzelfde bestand aan die na de ingevoegde lijnen komen (of er tussen). Omdat het steeds herbuilden van de solution na elke oplossing niet aanvaardbaar is wegens performantie (Dit wel manueel in iSE) wordt na elke oplossing de volledige lijst van build errors en waarschuwingen nagekeken en de lijnnummers indien nodig aangepast.

Een andere moeilijkheid is het feit dat de parameter `List<string> code` een *by reference*[9] parameter is. Dit wil zeggen dat als deze parameter wordt aangepast, er geen mogelijkheid is om de originele code terug te krijgen. Er wordt aangeraden om in het begin van de oplossing een kopie te maken van de parameter `List<string> code`. Ook moet er rekening gehouden worden met het invoegen van lijnnummers tijdens de oplossing, zodat de lijnnummer van de fout ook constant in de oplossing zelf wordt geüpdatet.

Als er tijdens het oplossing van een fout een exceptie optreedt, wordt een foutmelding weergegeven. De op te lossen fout wordt dan niet opgelost. Ook als er door middel van selectie in het resultaatvenster meerdere fouten in één keer worden opgelost, zal er bij oplossingen waar er excepties optreden een melding verschijnen en het oplossen van de fout worden overgeslagen. Aan het bestand dat de op te lossen fout bevat wordt er dan niets veranderd. Het is dus mogelijk om in oplossingen gebruik te maken van de klasse `Exception` (en de daarvan afgeleide klassen). Het attribuut `Message` van de klasse `Exception` wordt dan weergegeven in de foutmelding als de exceptie optreedt.

Op dit moment zijn er al twaalf oplossingen geïmplementeerd:

- CannotCast.cs
- CException.cs
- ExportedSymbolOrdinal.cs

[8] Alleen lijnnummers van gewijzigde en ingevoegde lijnen boven de ingevoegde lijn in het broncodebestand moeten worden aangepast.

[9] Een by reference variabele is een variabele die een wijzer bevat naar de waarde van de variabele. In C++ zijn dit objecten die worden aangesproken met pointers. In Visual C# wordt dit achter de schermen geregeld. De tegenhanger van by reference variabelen zijn de by value variabelen. Hier bevat de variabele de waarde zelf, in plaats van een wijzer naar de waarde.

- FunctionCallMissing.cs
- GenericException.cs
- LossOfData.cs
- MissingTypeSpecifier.cs
- NoLongerNeeded.cs
- SignedUnsignedMismatch.cs
- UndeclaredIdentifier.cs
- UnsafeFunction.cs
- Win32Winnt.cs

Ook zijn er enkele tools die aangewend kunnen worden in de oplossingen:

- ScopeDetector.cs

Fouten met een foute bestandslocatie

Sommige fouten worden op andere plaatsen gedetecteerd dan waar ze voorkomen. Een voorbeeld van dergelijke fout is de fout C1189 in hoofdstuk B.1 op p.148 (listing 5.7 op p.48).

```
1 c:\program files (x86)\microsoft visual studio
    11.0\vc\atlmfc\include\atlwinverapi.h(29): fatal error C1189: #error :  This file
    requires _WIN32_WINNT to be #defined at least to 0x0403. Value 0x0501 or higher
    is recommended. [C:\Users\Martijn\Desktop\TRR2\TRR2LookUp\TRR2LookUp.vcxproj]
```

Listing 5.7: De build error C1189.

In deze fout wordt verwezen naar het bestand "`c:\program files (x86)\microsoft visual studio 11.0\vc\atlmfc\include\atlwinverapi.h`". Dit bestand bevat geen foute code, maar de fout werd pas daar opgeworpen. Dit bestand mag zelfs niet gewijzigd worden, want het behoort tot het Visual C++ pakket. De bestandslocatie moet dus gewijzigd worden naar het bestand dat de fout bevat.

In iSE wordt de fout herkend en de juiste oplossing eraan gekoppeld dankzij de reguliere expressie die de fout identificeert. Foute locaties kunnen niet gedetecteerd worden[10]. Soms wordt er in het resultatenvenster bij "Remark" aangegeven dat er manueel een andere locatie geselecteerd moet worden, maar hiervan mag de gebruiker niet uitgaan. De gebruiker moet zich dus beroepen op de documentatie van de oplossing en kijken welk bestand hij moet selecteren. Voor fout C1189 moet bijvoorbeeld het bestand *StdAfx.h* in de overeenkomstige projectfolder worden opgegeven. Ook mag niet vergeten worden om het lijnnummer aan te passen. Hiervoor moet de documentatie geraadpleegd worden om een idee te krijgen hoe de lijn met de fout er zou kunnen uitzien. Voor sommige fouten kan het uitzicht van deze lijn zeer divers zijn.

Een ander vaak voorkomend probleem is dat sommige fouten relatieve paden weergeven. Dit wordt bij het oplossen van een fout of het aanroepen van het detailvenster gecontroleerd

[10]Dit zou wel mogelijk zijn indien er een tabel van verboden bestandslocaties wordt bijgehouden of waarden uit het register worden gebruikt (bijvoorbeeld om te detecteren of bestanden in de bestandsstructuur zitten van Visual Studio. Dit zijn wel lompe en ook foutgevoelige oplossingen).

door simpelweg na te gaan of het opgegeven bestand bestaat. Omdat het een relatief pad betreft wordt het bestand niet gevonden[11]. In dat geval wordt er automatisch een dialoogvenster getoond waar dan manueel het bestand met de fout en het lijnnummer opgegeven kan worden (figuur 5.4 op p.34). Ook hier moet de documentatie geraadpleegd worden om een idee te krijgen hoe de lijn met de fout er zou kunnen uitzien.

Er word telkens ook gecontroleerd bij het oplossen van een bestand of het leesbaar is. Als het ingesteld staat op 'read-only' wordt er in een dialoogvenster gevraagd om het leesbaar te maken. Indien dit niet wordt gedaan wordt de fout overgeslagen.

Versiebeheer

Er werd geëxperimenteerd met het automatisch uitchecken van bestanden. Inchecken wordt niet ondersteund omdat dit best wordt overgelaten aan de gebruiker. Op deze manier wordt de gebruiker verplicht om na te kijken wat iSE heeft aangepast en of dit correct is gebeurd. De kans op mislukte oplossingen is te groot om iSE de macht van het inchecken te geven.

Alvorens een fout op te lossen wordt nagekeken of er een versiebeheerprogramma is geselecteerd. De controle gebeurt na het controleren op foutieve bestandslocaties.

Zoals eerder besproken in hoofdstuk 5.2.2 op p.35 is er voor elk versiebeheerprogramma een klasse die overerft van de abstracte `SourceControl`-klasse (figuur 5.6 op p.36). De manier waarop vanuit iSE een bestand kan uitgecheckt worden door een bepaald versiebeheerprogramma is voor elk versiebeheerprogramma anders. Om aan deze diversiteit tegemoet te komen werd deze klassenstructuur geïmplementeerd. Op deze manier is er voldoende vrijheid om elk versiebeheerprogramma aan te spreken. Een interface is hier geen optie omdat de `SourceControl`-klasse moet overerven van de klasse `DevelopmentApplication`. Op die manier wordt redundante code het best vermeden.

De abstracte klasse `SourceControl` bevat de abstracte methode `CheckOut` (listing 5.8 op p.49).

```
1 public abstract void CheckOut(BuildAlert ba);
```

Listing 5.8: De methode `Checkout` in de abstracte klasse `SourceControl`.

Als *proof of concept*[12] werd er een klasse voor Visual SourceSafe 2005 geïmplementeerd. Er waren problemen opgetreden om deze via de command line aan te spreken. Er kwam steeds een foutmelding dat het *srcsafe.ini* bestand niet werd gevonden. De commando's `set` en `CheckOut` moeten allemaal in dezelfde sessie (dus hetzelfde *cmd.exe* proces) worden uitgevoerd. Dit was niet moeilijk te simuleren met de `Process`-klasse uit het .NET framework. Daarop werd besloten om een bat-bestand aan te maken dat de nodige bewerkingen bevat en dit aan te roepen met de benodigde parameters listing 5.9 op p.49. Het commando `cd` zorgt ervoor dat als het bestand wordt uitgecheckt, dat het op de juiste plaats terechtkomt. Bij het uitchecken van een bestand wordt het immers naar de working directory gekopiëerd.

```
1 :: parameters;
2 :: %1 <srcsafe.ini folder>
3 :: %2 <file w/ error folder>
4 :: %3 <sourcesafe ss.exe location>
5 :: %4 <file w/ error location>
6
7 set SSDIR=%1
```

[11] Als het relatief pad klopt ten opzichte van de locatie van iSE wordt het niet beschouwd als een foute bestandslocatie. iSE kan dan immers aan het bestand.

[12] Een proof of concept is een uitgewerkte test om aan te tonen dat iets mogelijk is.

```
8 cd %2
9 %3 Checkout %4
```

Listing 5.9: Het *sourcesafe.bat* bestand.

5.2.3 Vereisten

iSE heeft slechts twee vereisten:

- De solution moet minstens versie 7.0 (2002) of hoger bedragen. Deze eis wordt onrechtstreeks opgelegd omdat het builden van dsw- en dsp- bestanden niet meer wordt ondersteund door VCBuild en MSBuild. Ook het builden van vcproj-bestanden wordt door MSBuild niet meer ondersteund. Het is ook meestal geen goed idee om projecten op zich te builden als ze deel uitmaken van een solution met meerdere projecten. Het buildsysteem heeft dan vaak niet alle noodzakelijke informatie, wat dan bij het builden buildfouten veroorzaakt. Om deze reden is het best om de bestanden up te graden naar de juiste versie met Visual Studio (zie hoofdstuk 4 op p.27) en dan het sln-bestand te gebruiken in iSE.
- Op het systeem van de gebruiker moet de Visual Studio voor de gewenste Visual C++ geïnstalleerd staan.
- Indien er gebruik gemaakt zal worden van versiebeheerondersteuning in iSE moet het versiebeheerprogramma op het systeem aanwezig zijn.

5.2.4 Uitbreidbaarheid

iSE is zo flexibel mogelijk opgesteld. Er wordt ervan uitgegaan dat er nog veel oplossingen en ook ondersteuning voor Visual Studio versies en versiebeheerprogramma's zullen worden toegevoegd. Hier wordt een overzicht gegeven van de bewerkingen voor de verscheidene toevoegingen. De uitgebreide details bevinden zich in het voorgaande hoofdstuk 5.2.2 op p.35.

Toevoegen van Visual Studio versies

Om Visual Studio's toe te voegen moet er enkel een `VisualStudio`-tag worden toegevoegd aan de `ArrayOfVisualStudio`-structuur in *VisualStudios.xml*. Er wordt ervan uitgegaan dat er niets fundamenteel zal veranderen aan de manier waarop Visual Studio in de toekomst geïnstalleerd zal worden. Het implementeren van ondersteuning voor Visual Studio 2012 steunt deze veronderstelling. Voor het toevoegen van Visual Studio ondersteuning zijn er geen aanpassingen in de code nodig[13].

Er is geen ondersteuning geïmplementeerd voor het instellen van een andere framework-versie of toolset. De reden hiervoor is dat dit in het kader van de masterproef niet nodig was, er werd immers telkens gebuild met de overeenkomstige Visual Studio versie. Native en managed multi-targeting zijn pas aanwezig vanaf Visual Studio 2010, de doelversie in deze masterproef.

[13]Tenzij er natuurlijk significante wijzigingen zijn t.o.v. de huidige structuur van Visual Studio, zoals bijvoorbeeld een nieuw buildsysteem of command line interface

Toevoegen van versiebeheerondersteuning

Het toevoegen van ondersteuning voor versiebeheerprogramma's vereist wel aanpassingen in de code:

1. Er moet een klasse aangemaakt worden voor het versiebeheerprogramma in de namespace `Logic`. De klasse moet de klasse `SourceControl` overerven en de `ChechOut`-methode overloaden.

2. In het bestand *SourceControls.xml* moet er een `SourceControl`-tag worden toegevoegd in de `ArrayOfSourceControl`-tag. Hierbij moet de naam van de klasse uit stap 1 meegegeven worden bij het attribuut `xsi:type` in de `SourceControl`-tag.

Toevoegen van oplossingen.

Het toevoegen van oplossingen is de meest fundamentele uitbreiding. Het zijn de belangrijkste elementen van iSE. Om oplossingen toe te voegen moet er enkel code geschreven worden, er zijn geen configuratiebestanden bij betrokken[14].

1. Er moet een klasse geschreven worden voor de namespace `Logic.Solutions` die de interface `ISolution` implementeert. In deze klasse wordt de `Solve`-methode geïmplementeerd.

2. Het broncodebestand dat die klasse bevat wordt in de "Solutions"-folder in het project Logic geplaatst.

3. In de `SolutionFactory` wordt er een '`private static string`'-object toegevoegd dat een reguliere expressie bevat die beantwoordt aan de foutmelding.

4. In de `SolutionFactory` wordt er ook een '`else if`'-clausule toegevoegd in de methode `LinkBuildalert`. Daarin wordt de foutmelding vergeleken met de reguliere expressie in stap 3 en de klasse uit stap 1 geïnitialiseerd. De klasse wordt daarop meegegeven aan de `BuildAlert` (via diens `Solution`-property).

5.3 Opmerkingen

De ontwikkeling van iSE is pas op zijn eind als alle Visual C++ 6.0 solutions ter wereld zijn opgewaardeerd naar Visual C++ 2010. Voor de ontwikkeling van iSE werd er beroep gedaan op enkele projecten van ICORDA. Deze solutions zijn allen grotendeels analoog aan elkaar en bevatten vaak grote identieke stukken code. Dit zorgt ervoor dat de voorgekomen fouten slechts een beperkt deel zijn van alle mogelijke fouten die zouden kunnen optreden bij het opwaarderen. Helaas weet men pas het bestaan van een dergelijke breaking change als men die de eerste keer tegenkomt bij het opwaarderen. Pas dan kan er begonnen worden met de ontwikkeling van een oplossing voor deze fout en die toevoegen aan iSE. Dit geldt niet enkel voor nieuwe onbekende fouten. Het kan best zijn dat er reeds een oplossing bestaat voor een bepaalde breaking change, maar dat die oplossing niet op alle mogelijke gevallen is voorzien. Ook in dergelijke gevallen moet er een gedeelte of desnoods zelfs de gehele oplossingsmethode

[14] Tenzij er voor de oplossing zel configuratiebestanden moeten worden opgesteld.

herschreven worden. Het is pas zeker dat iSE alle mogelijke fouten aan kan en alle nodige oplossingen bevat, als alle mogelijke Visual C++ 6.0 solutions zijn opgewaardeerd. Helaas (en tegelijkertijd ironisch) is dat ook het punt dat iSE zijn nut verliest. De ontwikkeling van iSE stopt op hetzelfde moment dat iSE overbodig wordt.

Dit betekent uiteraard niet dat de ontwikkeling van iSE verspilde tijd is. In het begin zullen er veel onbekende fouten opduiken en zal er veel tijd gespendeerd worden aan het ontwikkelen van oplossingen. Hoe meer solutions er opgewaardeerd worden, hoe meer fouten er telkens opgelost kunnen worden en hoe zeldzamer nieuwe fouten (en het daarmee verbonden werk) zullen opduiken. De efficiëntie van iSE is dus recht evenredig met het gebruik en de ervaring (het aantal ontwikkelde oplossingsmethoden) van iSE. Dit is een typisch fenomeen dat zich bij elke tool, opgesteld voor deze masterproef, voordoet.

Deel II

Van Visual C++ 2010 naar Visual C# 2010

Hoofdstuk 6

Inleiding

In dit deel van de scriptie worden er verschillende strategieën bekeken voor het converteren van applicaties in Visual C++ 2010 naar Visual C#.

6.1 Waarom

Het converteren van een programma naar een nieuwe programmeertaal is geen lichte stap. De reden voor conversie is vaak het meegaan met de tijd. Nieuwe programmeertalen bieden nieuwe technologieën aan, zijn vaak duidelijker, performanter en gaan vaak hand in hand met up-to-date bijhorende programma's zoals IDE's. Meestal bieden nieuwere programmeertalen ook libraries aan die functionaliteit aanbieden die men anders nog zelf moet implementeren.

De keuze voor conversie komt eigenlijk neer op een kosten-baten analyse. Wegen de voordelen van de conversie op tegen de nadelen? Er zijn een aantal zaken die gelden voor eender welke conversie. Ten eerste is het converteren niet zonder gevaar. Conversie heeft als resultaat een nieuw (ongetest) systeem dat talloze nieuwe bugs kan bevatten.

> "*(...) Old code has been used. It has been tested. Lots of bugs have been found, and they've been fixed. There's nothing wrong with it. It doesn't acquire bugs just by sitting around on your hard drive. Au contraire, baby!*" (Spolksy, 2000)

Ten tweede eist de conversie ook mankracht en materiaal voor een veelal onbepaalde tijd. Tenzij de originele code ook tijdens de conversie wordt onderhouden, zullen er geen vernieuwingen of patches worden ontwikkeld voor gebruikers tijdens de conversieperiode. Als de gebruikers betalende klanten zijn kan dit zorgen voor een verlies van klanten. Deze zien immers gedurende deze periode geen vooruitgang.

> "*You are putting yourself in an extremely dangerous position where you will be shipping an old version of the code for several years, completely unable to make any strategic changes or react to new features that the market demands, because you don't have shippable code. You might as well just close for business for the duration.*" (Spolksy, 2000)

Ten slotte betekent converteren ook het weggooien van alle geïnvesteerde bronnen zoals kennis, tijd en geld. Externe bibliotheken moeten kunnen opgeroepen worden vanuit de

doeltaal of vervangen worden. Als dit niet mogelijk is moeten er alternatieven worden gezocht en dat kan voor bijkomende kosten zorgen.

Soms is de kosten-baten analyse heel simpel te bepalen: het kan zijn dat de taal waarin de applicatie werd geschreven niet meer wordt ondersteund of vernieuwd. Dit zijn serieuze beperkingen op de uitbreidbaarheid en ondersteuning van de applicatie. Een ander simpel geval voor de kosten-baten analyse te bepalen, is wanneer een aantal factoren zoals performantie een hoge prioriteit hebben. Het converteren naar een nieuwe taal kan dan een negatieve impact hebben op deze factoren.

Kosten en baten analyseren wordt minder duidelijk als men een applicatie heeft die nog steeds goed werkt. De redenen voor conversie kunnen dan zeer divers zijn. Het kan bijvoorbeeld zijn dat men technologieën of libraries van de doeltaal wil gebruiken: in de huidige taal moet men om een bepaalde functie te implementeren allerlei onduidelijke en vuile truken toepassen of ontzettend veel code schrijven voor iets dat in de doeltaal simpel, duidelijk en performant kan geïmplementeerd worden.

Economische factoren kunnen ook een rol spelen: er kunnen steeds minder programmeurs voor de originele programmeertaal op de markt zijn of men wil iedereen in dezelfde taal laten werken om de bedrijfsstructuur te vereenvoudigen.

Kortom, wel of niet converteren is een kwestie van het vergelijken van voor- en nadelen en die zijn voor elke situatie zeer specifiek.

6.2 Herschrijven of vertalen

Conversie staat altijd gelijk met herwerken. Men heeft nu de keus uit: het programma letterlijk vertalen naar de doeltaal of het programma in de doeltaal opnieuw ontwerpen en herschrijven. Het vertalen van de applicatie betekent de code vervangen door code in de doeltaal zodat de functionaliteit en het uitzicht van het programma gelijkaardig zijn aan het origineel. Bij het herschrijven wordt het origineel gebruikt als basis, maar is de functionaliteit en het uitzicht anders. Meestal zal een herschreven programma de originele functionaliteit bevatten aangevuld met ingrijpende vernieuwingen.

Het is belangrijk op voorhand te bepalen welke weg de conversie inslaat. Het is immers verleidelijk om bij het vertalen vernieuwingen te implementeren. Dit kan zeer nefaste gevolgen hebben voor het conversieproces: het implementeren van vernieuwingen heeft vaak een impact op meerdere onderdelen van de applicatie. De betrokken onderdelen kunnen dus niet meer vertaald worden en moeten worden herschreven. Deze vernieuwingen leiden dan ook vaak tot vertragingen en complicaties. Dit kan het conversieproces aanzienlijk vertragen en zelfs schade toebrengen.

6.3 Incrementeel of geheel

Er zijn twee verschillende aanpakken voor de conversie: incrementeel of geheel. Het onderscheid tussen de aanpakken komt grotendeels, maar niet volledig overeen met het onderscheid tussen herschrijven of vertalen in hoofdstuk 6.2 op p.55.

Incrementeel converteren betekent dat de applicatie deel per deel wordt geconverteerd. Het voordeel van deze aanpak is dat er geen lange periode aan de conversie moet worden besteed. Elk deel neemt slechts beperkte tijd in beslag en men kan kiezen wanneer men een bepaald deel verwerkt. Het onderhoud van de gehele applicatie hoeft er dus niet onder te

lijden (toch niet voor een langdurige periode). Bij de verwerking van een deel kunnen er al nieuwigheden worden geïmplementeerd. De gebruiker kan dan al van deze nieuwigheden gebruik maken terwijl de applicatie nog maar gedeeltelijk is geconverteerd. Daarom geniet dit model de voorkeur. Het nadeel daarentegen is dat deze methode slechts vernieuwingen toelaat in beperkte mate: de interface met nog niet geconverteerde delen moeten steeds behouden blijven. Men kan dus geen drastische vernieuwingen doorheen de hele applicatie aanbrengen zonder uit te wijken naar het geheel model. Het incrementele model is dus zowel geschikt voor vertalen als een enigszins beperkte vorm van herschrijven.

Het geheel model is zowel geschikt voor vertalen als herschrijven. Doordat de applicatie in zijn geheel onder de loep wordt genomen is het mogelijk de applicatie compleet te vernieuwen en aan te passen. Zoals eerder besproken betekent dit dat de applicatie voor een langdurige periode niet meer wordt onderhouden, tenzij de originele code ondertussen ook nog wordt onderhouden.

Hoofdstuk 7

Managed code

7.1 Managed versus unmanaged code

De solutions en diens projecten van ICORDA zijn unmanaged solutions. Dit houdt in dat ze geen gebruik maken van het .NET framework en rechtstreeks worden gecompileerd naar machinecode. Deze code wordt dan uitgevoerd door de processor. Als de code gecompileerd is in een Windowsomgeving is het uitvoerbaar bestand (meestal een exe-bestand) volgens Gregory (2004) niet overdraagbaar naar een ander besturingssysteem, ook al doet de broncode geen beroep op Windowsapplicaties en -onderdelen.

In het .NET framework worden applicaties niet rechtstreeks gecompileerd naar machinetaal voor de Central Processing Unit (CPU). Managed code wordt gecompileerd naar Microsoft Intermediate Language (MSIL). De MSIL-code wordt dan opgeslagen in een assembly, samen met metadata dat de klassen, methodes en attributen in de code beschrijft. Als de applicatie wordt uitgevoerd compileert de Common Language Runtime (CLR) delen van de MSIL-code naar machinecode vlak voor het wordt aangeroepen. Dit wordt Just In Time (JIT)-compileren genoemd, omdat enkel de delen van de MSIL-code die echt worden gebruikt gecompileerd worden. De gecompileerde delen worden dan bewaard voor het geval ze nogmaals moeten worden opgeroepen.

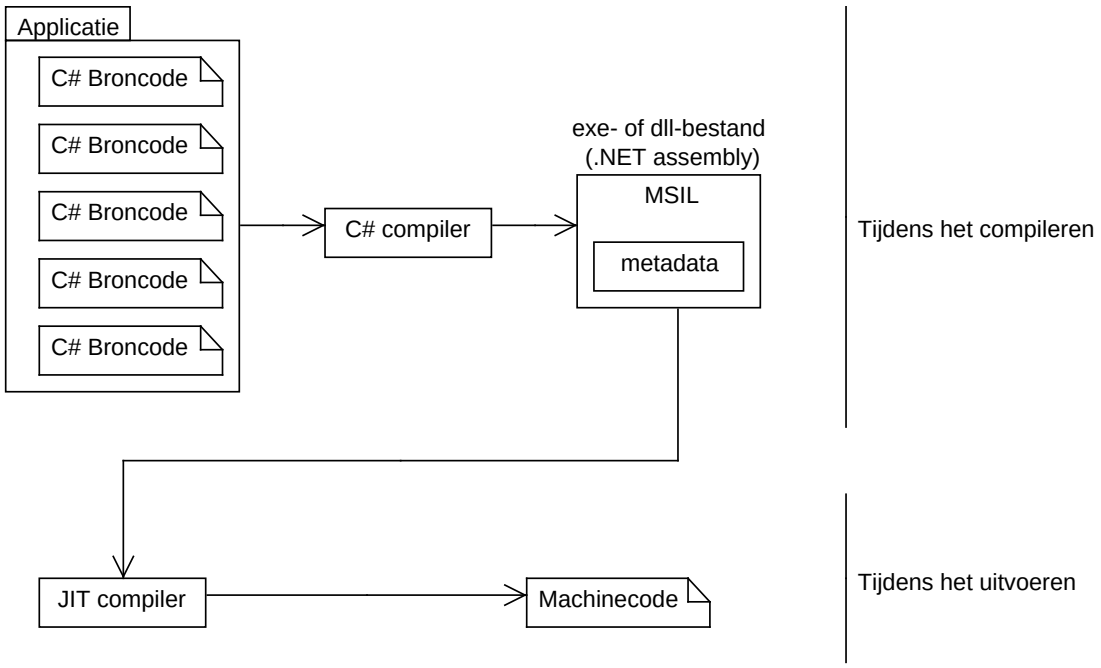
De CLR beheert dus eigenlijk de code, vandaar dat dergelijke code managed code wordt genoemd. Het beheert het geheugen, threads, de uitvoering en compilatie van code, controleert de veiligheid van code, ...

7.2 Voor- en nadelen

Managed code brengt veel voordelen, maar ook enkele nadelen met zich mee. Om te weten of het omzetten van solutions en projecten de moeite waard is, moeten de voor- en nadelen worden afgewogen. Hier wordt er een beperkte lijst met enkele belangrijke voor- en nadelen weergegeven.

7.2.1 Voordelen

- Het gebruik van de .NET Framework class library. Het .NET framework bevat naast de CLR ook een uitgebreide verzameling aan objectgeörienteerde klassen en functies die in managed code kunnen aangeroepen worden. Zo hoeft de programmeur geen functies



Figuur 7.1: De uitvoering van managed code.

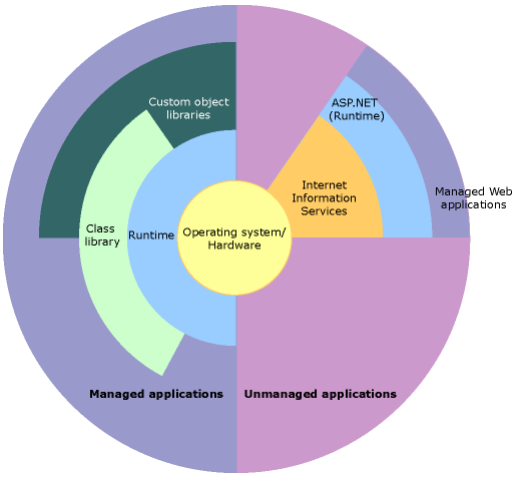
meer te schrijven voor bijvoorbeeld het werken met strings, databanken, bestanden, ... Externe bedrijven en programmeurs kunnen op basis van de class library zelf klassen en methodes schrijven die naadloos ineenvloeien met het .NET framework. De base class library biedt ook talloze structuren aan voor bijvoorbeeld het maken van vensters (Windows Forms, WPF, ...), webapplicaties (Active Server Pages (ASP).NET), Windows services, web services (Windows Communication Foundation (WCF)), ... (Microsoft, 2012o)

- Managed componenten hebben steeds een niveau van vertrouwen. Zo krijgen onbetrouwbare componenten geen toegang tot bestandsbeheer en dergelijke, zelfs al zit het samen met andere betrouwbare componenten in een applicatie. (Microsoft, 2012o)

- In een Visual Studio kunnen projecten geschreven in verschillende managed talen met elkaar interageren. Dit komt vooral in deze masterproef van pas om managed Visual C++ en Visual C#[1] projecten in één enkele solution te laten samenwerken. Dit komt omdat het .NET framework voldoet aan de Common Language Interface (CLI)-standaard, een standaard voor virtuele executie systemen[2].

[1] C# is een managed taal. In Visual Studio kunnen enkel Visual C++ applicaties unmanaged worden ontwikkeld.

[2] Het .NET framework is een virtueel executie systeem. Managed code wordt immers uitgevoerd als MSIL in de CLR en niet rechtstreeks als machinetaal op de CPU



Figuur 7.2: Het verschil tussen managed en unmanaged applicaties. (Microsoft, 2012o)

- Binnenin de CLI, hier het .NET framework, bevindt zich de Common Type System (CTS). Dit bevat de definitie van alle types beschikbaar voor talen compatibel met de CLI en definieert hoe deze worden gedeclareerd, gebruikt en beheerd. In het .NET framework zorgt dit ervoor dat alle managed talen beschikken over een set basistypes. Volgens Microsoft (2012d) zijn types in het .NET framework verdeeld in vijf categorieën: Classes, Structures, Enumerations, Interfaces en Delegates. Dit heeft als gevolg dat er gecontroleerd wordt hoe types met elkaar worden gebruikt (zoals bijvoobeeld bij casts). Code zal niet compileren als types niet juist worden gebruikt.

- Het geheugen wordt beheerd door de CLR. Dankzij garbage collection moet de programmeur niet zelf meer zorgen voor het toewijzen van geheugen aan objecten en het correct verwijderen ervan. Zo worden volgens Microsoft (2012o) de twee meest voorkomende programmeerfouten, memory leaks en foute referenties, voorkomen.

- Er is terugwaartse comptabiliteit tussen managed en unmanaged code. Dit wordt verder besproken in hoofdstuk 8 op p.72.

- Managed applicaties, ontwikkeld voor een bepaalde versie van het .NET framework, zullen op alle latere versies ook nog werken. Ook kunnen er meerdere versies van het .NET framework op hetzelfde systeem geïnstalleerd staan. Dit laat toe om applicaties te laten lopen op de versie van de .NET framework waarvoor ze zijn ontwikkeld. Microsoft (2012i)

7.2.2 Nadelen

- Om de managed applicaties te kunnen uitvoeren moet de juiste versie van .NET framework op de computer geïnstalleerd staan (of worden meegeleverd). Dit brengt ook bepaalde eisen naar hardware met zich mee (tabel 7.1 op p.60). Ook zijn er eisen qua besturingssystemen: Zo kan men bijvoorbeeld .NET Framework 4.5 slechts gebruiken op Windows Vista, Windows 7 en Windows 8. Er is geen ondersteuning meer voor Windows XP.

Hardware Requirements	Version 4.5	Version 4.0 Full	Version 4 Client	Version 3.5	Version 3.0	Version 2.0
Processor						
Minimum	1 GHz	1 GHz	1 GHz	400 MHz	400 MHz	400 MHz
Recommended	1 GHz	1 GHz	1 GHz	1 GHz	1 GHz	-
RAM						
Minimum	512 MB	512 MB	512 MB	96 MB	96 MB	96 MB
Recommended	512 MB	512 MB	512 MB	256 MB	256 MB	-
Disk space (minimum)						
32-bit	850 MB	850 MB	600 MB	280 MB	280 MB	280 MB
64-bit	2 GB	2 GB	1,5 GB	610 MB	610 MB	610 MB

Tabel 7.1: De hardware eisen voor het .NET framework. (Microsoft, 2012l; Microsoft, 2011).

- Vergeleken met unmanaged applicaties hebben managed applicaties een lichte overhead. Van code naar machinetaal is er immers een extra stap. De MSIL wordt gecontroleerd op onregelmatigheden en wordt nogmaals gecompileerd tijdens het uitvoeren. Het .NET framework probeert hieraan tegemoet te komen via JIT-compilatie en het cachen van de gecompileerde delen. Het geheugenbeheer zorgt er volgens Microsoft (2012o) voor dat het geheugen niet gefragmenteerd is en zorgt ervoor dat er zoveel mogelijk wordt tegemoet gekomen aan het lokaliteitsbeginsel[3]. Volgens Gregory (2004) is het verschil meestal niet echt juist te bepalen en kan het ook voorkomen dat managed code sneller is dan unmanaged code (bijvoorbeeld bij COM-operaties). Gregory (2004) raadt dan ook aan om de beslissing tussen managed en unmanaged niet te baseren op vooroordelen over snelheid.

- er zijn ook nog een aantal nadelen specifiek verbonden met managed Visual C++. Deze worden beschreven in hoofdstuk 7.3.2 op p.63.

7.3 Managed Visual C++ (C++/CLI)

Bij het opwaarderen van solutions of het converteren van solutions naar Visual C# kan het gebruik van managed code zeer interessant zijn. Bij het opwaarderen kunnen sommige breaking changes ervoor zorgen dat veel code moet worden aangepast of de breaking change niet makkelijk kan worden ontdekt. Sommige technologieën kunnen zodanig verouderd zijn dat ze niet meer worden ondersteund of up-to-date zijn. In dergelijke gevallen is het dan makkelijker om de solution als managed te compileren en de probleemcode te vervangen met code die steunt op .NET klassen en functies. Ook kunnen bepaalde technologieën die het ontwikkelen lastig maken achterwege gelaten worden, zoals COM. Voor de conversie is het managed maken van Visual C++ projecten interessant omdat die dan rechstreeks kunnen communiceren met de Visual C# projecten. Een incrementele conversie wordt zo heel wat makkelijker. Ook hier kunnen dan lastige technologieën zoals COM worden weggewerkt.

[3]Code of data dichtbij het punt van uitvoering in het geheugen heeft de grootste kans dat het weldra gebruikt zal worden.

7.3.1 Van unmanaged naar managed code

De overgang van een project van unmanaged naar managed op zich is redelijk eenvoudig. Er moet slechts een enkele compiler optie worden ingesteld: "/clr". Dit wordt per project ingesteld in `Properties` > `Configuration Properties` > `General` > `Common Language Runtime support`. De optie zorgt ervoor dat de code als managed wordt gebuild. Het is dan zelfs mogelijk om binnen een enkele codebestand bepaalde delen als managed en andere als unmanaged delen te markeren met behulp van `pragma` statements[4]. (listing 7.1 op p.61).

```
1 #pragma managed
2 #pragma unmanaged
3 #pragma managed([push,] on | off)
4 #pragma managed(pop)
```

Listing 7.1: De `pragma` statements voor het aangeven van managed en unmanaged codefragmenten. (Microsoft, 2012k)

Het instellen van de "/clr"-optie zorgt er niet voor dat ook data in het project als managed wordt gecompileerd. Het project is dus volgens Gregory (2004) nog altijd zelf verantwoordelijk voor het geheugenbeheer van zijn variabelen. Variabelen kunnen nog steeds zoals in unmanaged Visual C++ worden aangemaakt ofwel op de stack ofwel op de heap waar deze ook expliciete moeten worden verwijderd (door middel van het `delete` of `delete[]` statement). Om data ook als managed data te behandelen moet er gebruik gemaakt worden van een aantal keywords. Oorspronkelijk gebeurde dit met compilerspecifieke extensies (keywords die begonnen met een dubbel underscore, bv. `__gc`), beter gekend als managed extensions. Ondertussen zijn deze vervangen door keywords opgenomen in de Visual C++ specificatie. Microsoft (2008b) geeft weer dat deze overgang werd ingezet in Visual C++ 2005. Vanaf Visual C++ 2008 werden de managed extensions als deprecated gemarkeerd. De nieuwe Visual C++ specificatie wordt ook wel C++/CLI genoemd en is gestandardiseerd onder de Ecma C++/CLI-standaard (Microsoft, 2010c). Met behulp van de compileroptie "/clr:oldSyntax" kan er nog steeds met de deprecated managed extensions gewerkt worden. Voor het opwaarderen van managed extensions naar C++/CLI biedt Microsoft (2008b) een checklist.

Voor het gebruik van klassen en objecten uit de .NET Class Library moet er onderscheid worden gemaakt tussen by reference en by value types. By reference objecten steunen op mechanismen analoog aan pointers. Om in managed Visual C++ gebruik te maken van by reference objecten moeten ze gedeclareerd worden met de 'handle to object on managed heap'-operator (`^`). De initialisatie gebeurt dan met het `gcnew` keyword in plaats van `new`. Dit plaats het managed object op de managed heap en niet op de native heap. Hierdoor wordt duidelijk gemaakt dat het om een managed object gaat waarvan het geheugengebruik door het .NET framework wordt beheerd. Met andere woorden, het object moet niet meer door de programmeur uit het geheugen worden verwijderd na gebruik. Dit gebeurt nu automatisch dankzij het garbage collection mechanisme in het .NET framework. By value types kunnen zowel net als by reference objecten op de managed heap worden geplaatst[5] of op de stack (listing 7.4 op p.62)[6].

[4] Volgens Microsoft (2012k) kunnen deze niet binnenin functie-implementaties en voor `include` statements voorkomen.

[5] Dit gebeurt via boxing: de waarde wordt opgeslagen in een `System.Object` object en op de managed heap geplaatst.

[6] Een managed value type kan zelfs op de native heap worden geplaatst. Het moet dan wel ook met het `delete` of `delete[]` statement weer worden verwijderd.

```
using namespace System;

int main(){
    String ^ a = gcnew String ("test1");
    String ^ b = "test2"; // analoog aan gcnew String ("ctor");
    Object ^ c = gcnew String ("test3");
    return 0;
}
```

Listing 7.2: Declaratie en initialisatie van managed objecten in C++/CLI.

Om .NET klassen te gebruiken in managed code, moet er net als in Visual C# de juiste namespaces worden meegegeven (listing 7.2 op p.62). In Visual C# gebeurt dit met het keyword `using`, in managed Visual C++ met de keywords `using namespace`. Het is natuurlijk ook mogelijk om telkens de volledige naam van het type te gebruiken. Zo kan in listing 7.2 op p.62 de lijn "`using namespace System;`" weggelaten worden en de keywords `String` en `Object` vervangen worden door respectievelijk `System::String` en `System::Object`.

```
value class MyData
{
public:
    int Simple;
};

int main(){
    //Declaring the variable locally, the object is put on the stack.

    MyData d1;
    d1.Simple = 11;

    //Using the pointer syntax, the object can be put on the native heap:

    MyData* pd2 = new MyData();
    pd2->Simple = 22;
    delete pd2;

    //With the gcnew operator boxing and unboxing is done behind the scenes:

    MyData^ d3 = gcnew MyData();
    d3->Simple = 33;
    //delete d3;   // invokes Dispose (niet nodig)
    return 0;
}
```

Listing 7.3: De mogelijkheden van managed by value types in C++/CLI. (Nagel, 2005)

In het managed Visual C++ project kunnen dan ook de eigen klassen op dezelfde manier worden gebruikt. Bij de implementatie wordt voor het `class` of `struct` keyword aangegeven of het om een by reference of een by value managed type gaat, met de respectievelijke keywords `ref` en `value`. (listing 7.4 op p.62).

```
// mcppv2_ref_class2.cpp
// compile with: /clr
ref class MyClass {
public:
    int i;

    // nested class
    ref class MyClass2 {
    public:
        int i;
    };

    // nested interface
```

```
14      interface struct MyInterface {
15          void f();
16      };
17  };
18
19  ref class MyClass2 : public MyClass::MyInterface {
20  public:
21      virtual void f() {
22          System::Console::WriteLine("test");
23      }
24  };
25
26  public value struct MyStruct {
27      void f() {
28          System::Console::WriteLine("test");
29      }
30  };
31
32  int main() {
33      // instantiate ref type on garbage-collected heap
34      MyClass ^ p_MyClass = gcnew MyClass;
35      p_MyClass -> i = 4;
36
37      // instantiate value type on garbage-collected heap
38      MyStruct ^ p_MyStruct = gcnew MyStruct;
39      p_MyStruct -> f();
40
41      // instantiate value type on the stack
42      MyStruct p_MyStruct2;
43      p_MyStruct2.f();
44
45      // instantiate nested ref type on garbage-collected heap
46      MyClass::MyClass2 ^ p_MyClass2 = gcnew MyClass::MyClass2;
47      p_MyClass2 -> i = 5;
48  }
```

Listing 7.4: Het zelf definiëren van managed klassen en structs in C++/CLI. (Microsoft, 2012j)

Net als de 'handle to object on the heap'-operator (^) de managed variant is van de pointeroperator (*) is de 'tracking reference' (%) de managed variant van de referentie-operator (&).

7.3.2 Nadelen specifiek aan C++/CLI

Bij het gebruik van C++/CLI zijn ook een aantal nadelen verbonden.

- Er is geen Intellisense geïmplementeerd voor c++/CLI in Visual Studio 2010.

> "*We completely empathize with the need to work with C++/CLI and we think it's a great way to do interop between native code and managed code. (...) we had to make the difficult decision to reduce the scope to native C++ only for Intellisense. We still index symbols coming from C++/CLI code and you can browse them with Class View etc...*
> *While the lack of Intellisense for C++/CLI is unfortunate, we expect that it only represents a small portion of your source code that you don't need to edit nearly as often as the native code. Indeed, the only scenario we don't recommend is to use C++/CLI as a first-class .NET language. Instead, we think it's the ideal solution for interop.*" (Jabes, 2009)

Dit kan gelukkig omzeilt worden door Visual Studio 2012 te gebruiken met de Visual C++ 2010 toolset en 4.0 frameworkversion.

- Door het gebruik van de "/clr"-optie moeten sommige andere opties van de projecteigenschappen worden uitgeschakeld. In de MSDN wordt door Microsoft (2010a) een volledig overzicht aangeboden van alle restricties veroorzaakt door de "/clr"-optie. Hier wordt er een klein overzicht gegeven van de restricties waar men persoonlijk in aanraking mee kwam.

  /RTC
  Zorgt voor het detecteren van run-time fouten. Het moet worden ingesteld op "Default" om te voldoen aan de "/clr" restrictie.

  /Gm
  De "Minimal rebuild" optie zorgt ervoor dat enkel bronbestanden afhankelijk van klassedefinities in gewijzigde header-bestanden bij compilatie verwerkt moeten worden. Het afzetten van deze optie zorgt voor langere compilatietijden. Dit moet gewijzigd worden in "/Gm-" waardoor "Minimal Rebuild" wordt uitgeschakeld.

  /ZI
  Deze optie moet worden ingesteld naar "/Zi" waardoor de opgeslagen informatie van de debugger niet meer de functie "Edit and Continue" ondersteunt. Indien tijdens het debuggen de broncode wordt aangepast moet telkens het hele project opnieuw gecompileerd worden.

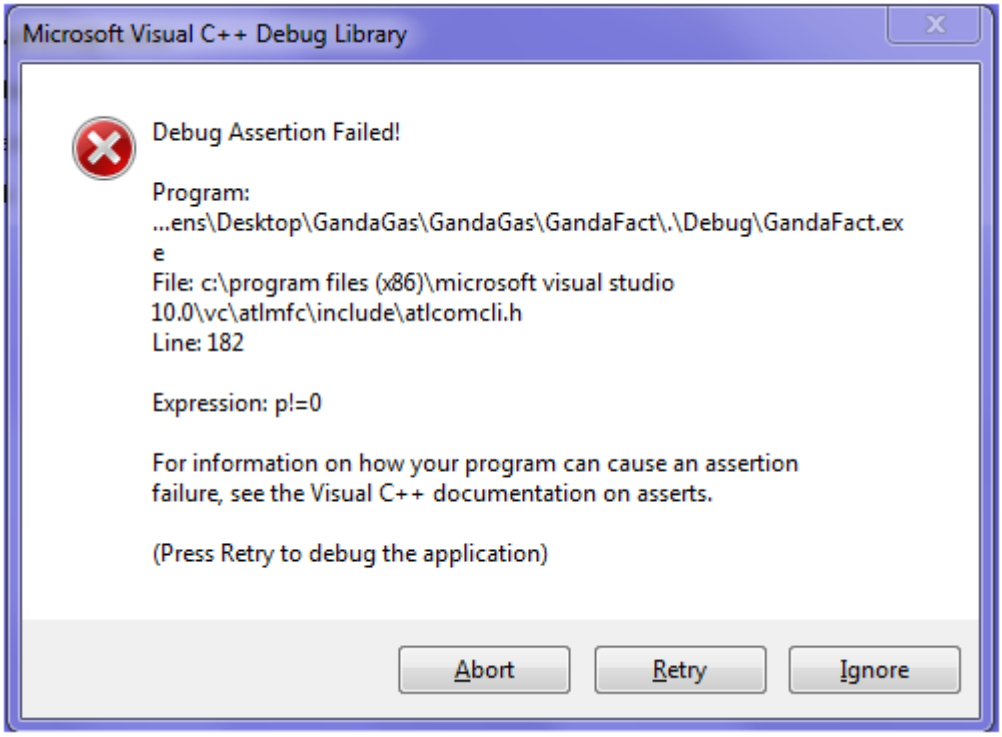
7.4 Proof of concept

In tegenstelling tot de andere proof of concepts in de scriptie, is de proof of concept in dit geval een daadwerkelijke oplossing van een opwaarderingsprobleem. Deze oplossing is in het kader van deze masterproef opgesteld omdat er bij het opwaarderen van de applicatie GandaFact op een onoverkomelijk probleem werd gestoten.

7.4.1 Probleemstelling

Na het opwaarderen van GandaFact zorgde het openen van een dialoogvenster er steeds voor dat er een reeks debug assertions werden getoond. Het doel van het dialoogvenster was om de gebruiker een keuze te laten maken uit een lijst van records opgehaald uit een databank. Dit dialoogvenster steunde daarvoor op de klasse `CGXAdoGrid` uit de externe Stingray libraries. In de methode `OpenRecordset` van de klasse wordt er gebruik gemaakt van de macro `GX_ADO_CHECK`. Tijdens het debuggen werd al snel duidelijk dat deze macro zorgde voor de vele debug assertions.

Door het invoeren van de nieuwe versie waren er een aantal zaken gewijzigd. Een van die wijzigingen was het gebruik van smartpointers, zoals `CComPtr`, in plaats van gewone pointers. Smartpointers hebben dezelfde functionaliteit als gewone pointers, maar bieden ook nog extra functionaliteit. `CComPtr`-objecten worden volgens Microsoft (2012b) gebruikt om COM interface pointers te beheren. Ze bevatten extra controles en tellen automatisch het aantal referenties. Hierdoor wordt het geheugenbeheer van het object, naar waar wordt verwezen, geautomatiseerd. Als er geen referenties meer zijn kan het object worden verwijderd. Het probleem met de overschakeling van gewone pointers naar `CComPtr`-objecten is dat ze ook



Figuur 7.3: De debug assertions opgeworpen bij het proberen openen van het dialoogvenster.

controleren op nullpointers aan de hand van ingebouwde assertions. Het zijn deze assertions die de debug assertions genereerden.

```
#define GX_ADO_CHECK(exp, recordset, succeded) \
    {                                          \
            //...
            ADOConnection *piConnection = 0;   \
            VARIANT var;\
            VariantInit(&var);\
            recordset->get_ActiveConnection(&var);\
            piConnection = (ADOConnection *) V_DISPATCH(&var);\
            //...
    }                                          \
```

Listing 7.5: Het gebruik van gewone pointers in de macro `GX_ADO_CHECK` in de oude versie van de Stingray libraries.

```
#define GX_ADO_CHECK(exp, recordset, succeded)  \
    {                              \
            //...            \
            CComPtr<ADOConnection> piConnection;    \
            CComVariant var;                       \
            recordset->get_ActiveConnection(&var);\
            piConnection = (ADOConnection *) V_DISPATCH(&var);\
            //...
    } \
```

Listing 7.6: Het gebruik van smartpointers in de macro `GX_ADO_CHECK` in de nieuwe versie van de Stingray libraries.

De functionaliteit van GandaFact steunde op het meegegeven van nullpointers aan de macro. Bij de nieuwe versie van de macro was dat niet mogelijk omdat de smartpointers

geen nullpointers accepteerden. Het herwerken van GandaFact, zodat het niet meer berust op nullpointers, zou meer goed dan kwaad hebben gedaan. De hoeveelheid werk was niet in te schatten en daarbij zouden veranderingen nieuwe bugs kunnen creëren. Omdat de macro enkel gebruikt werd in een enkel project ModuleVirtualGrid, bleek dit wel relatief makkelijk op te lossen door het gebruik van C++/CLI.

7.4.2 Strategie

Omdat het project ModuleVirtualGrid enkel gebruikt werd voor het weergeven van dit dialoogvenster, kon dit project vervangen worden door een Windows Forms Visual C# project dat een analoog venster weergaf. Er werd eerst geopteerd voor een WPF-dialoogvenster. Om alle functionaliteit uit het origineel venster te kunnen voorzien, was Windows Forms niet voldoende. Bij grote aantallen records was namelijk de performantie niet voldoende. Ook het implementeren van functies, zoals bijvoorbeeld zoeken op naam, bleken moeilijk te implementeren zonder merkbaar performantieverlies. Omdat in geen enkel ander programma van ICORDA WPF werd gebruikt viel de keuze uiteindelijk toch op Windows Forms, aangevuld met componenten uit een externe library: DevExpress. Deze componenten boden wel de nodige functionaliteit zonder performantieverlies.

Voor de gebruikers maakte het vervangen van het dialoogvenster geen verschil, want het venster zag er daarna nog steeds hetzelfde uit. Dit was een belangrijke voorwaarde vanuit ICORDA, omdat verandering vaak ongewenst is bij de klant. Verandering betekent immers dat gebruikers opnieuw moeten leren omgaan met het programma en eraan wennen.

Het vervangen door een Visual C# project zorgde ervoor dat projecten die het dialoogvenster aanriepen als managed moesten worden gecompileerd. In die projecten moesten dan alle aanroepen naar het origineel dialoogvenster gewijzigd worden naar aanroepen naar het nieuwe dialoogvenster. Gelukkig bevatte slechts enkel het startproject GandaFact aanroepen naar het dialoogvenster. Alle andere projecten moesten dus niet aangepast worden en bleven unmanaged.

7.4.3 Implementatie

Het project ModuleVirtualGrid werd eerst verwijderd en er werd een nieuw Windows Forms Visual C# project toegevoegd, GandaFactSelectionDlg genaamd. Het startproject GandaFact, waaruit het dialoogvenster werd opgeroepen, werd als managed project ingesteld met de optie "/clr". Er werd aan GandaFact ook een reference naar GandaFactSelectionDlg toegevoegd. Met behulp van het Entity Framework, een .NET objectrelationele mapper van Microsoft, werd er in GandaFactSelectionDlg een Visual C# model aangemaakt van de databank van GandaFact. De connectionstring voor de databank werd opgeslagen in de *App.config* van het project (listing 7.7 op p.66).

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3    <connectionStrings>
4      <add name="Multigas2010Entities"
5           connectionString="metadata=res://*/DataModel.csdl
6           |res://*/DataModel.ssdl
7           |res://*/DataModel.msl;
8           provider=System.Data.SqlClient;
9           provider connection string=&quot;
10          data source=ICORDA-DS3\SQL2005;
11          initial catalog=Multigas2010;
```

```
12          integrated security=True;
13          multipleactiveresultsets=True;
14          App=EntityFramework&amp;quot;"
15          providerName="System.Data.EntityClient" />
16    </connectionStrings>
17  </configuration>
```

Listing 7.7: De *App.Config* van het GandaFactSelectionDlg project. De connectionstring bevat whitespace om voor een beter weergave. In het bestand is er geen whitespace aanwezig in de connectionstring.

Aan de form zelf werd telkens een enumeration variabele meegegeven. Om dit te simuleren werden er analoge enums in GandaFactSelectionDlg aangemaakt (listing 7.9 op p.67). In het managed aanroepproject konden deze `enum`s meegegeven worden. De enumeration variabele gaf weer welke stored procedure opgeroepen moest worden om de records op te halen.

```
1  class CWrapperDlgSelection
2  {
3  //...
4  public:
5      enum SelectionType {
6          ARTICLE, CLIENT, TRANSPORTER, ORDER, INVOICE,
7          LEVADRES, CLIENTWITHSUPPR, STOCKCHANGE, CLIENTKV};
8  //...
9  }
```

Listing 7.8: De enumerations in het origineel ModuleVirtualGrid project.

```
1  namespace GandaFactSelectionDlg
2  {
3      public enum SelectionType {
4          ARTICLE, CLIENT, TRANSPORTER, ORDER, INVOICE,
5          LEVADRES, CLIENTWITHSUPPR, STOCKCHANGE, CLIENTKV
6      };
7  }
```

Listing 7.9: De enumerations in het GandaFactSelectionDlg project.

Via een '`switch case`'-constructie werd dan in GandaFactSelectionDlg de juiste blok code aangeroepen voor de meegegeven `enum`-variabele. De stored procedures werden gesimuleerd via Language Integrated Query (LINQ) queries op het gegenereerde Visual C# model van de databank. Omdat de teruggegeven kolommen van de stored procedures anders waren dan de kolommen weergegeven in het oorspronkelijk dialoogvenster, moest er voor elke kolommenstructuur uit het oorspronkelijk dialoogvenster een aparte klasse worden opgesteld. De gridcomponent[7] in het nieuwe dialoogvenster dat de records weergaf, werkte analoog als een `DataGridView`. Het bevatte een eigenschap `DataSource` waaraan een object kon gebonden worden. Dit object was een lijst van objecten van hetzelfde type. De kolommen in de component werden bepaald door de velden van de klasse van de objecten waaruit de lijst was opgebouwd. Voor elke kolomstructuur werd daarom een klasse opgesteld met als velden de kolommen (bijvoorbeeld listing 7.10 op p.68). De LINQ-expressies haalden voor elke record de nodige waarden op, staken ze in de velden van een object van de klasse dat de kolomstructuur representeert en staken die objecten op hun beurt in een lijstobject. Dat lijstobject werd dan gekoppeld aan de gridcomponent via diens `DataSource`-veld (listing 7.11 op p.68). Om alles zo snel mogelijk te doen verlopen werd gebruik gemaakt van de klasse `AsyncHelper`, opgesteld door ICORDA. Dit zorgde ervoor dat de records asynchroon werden opgehaald.

[7] Het wordt hier een gridcomponent genoemd, omdat het een rooster weergaf. De component was te vergelijken met een `DataGridView`.

```
public class ArticleModel : IModel
{
    public string Id { get; set; }
    public string Description { get; set; }
    public string Remark { get; set; }

    //...
}
```

Listing 7.10: De klasse `ArticleModel` bevat de kolomstructuur voor de stored procedure die wordt opgeroepen met de `enum`-variabele `ARTICLE`.

```
private void OnInitialize(SelectionType st)
{
    object models = null;
    AsyncHelper.RunAsync(() =>
    {
        Multigas2010Entities ent = new Multigas2010Entities(CONNECTIONSTRING);
        switch (st)
        {
            case SelectionType.ARTICLE:
            models = (from art in ent.Article
                select new ArticleModel() {
                Id = art.ID,
                Description = art.Description,
                Remark = art.Remark.Trim()
                }).ToList<ArticleModel>();
            break;
            // andere case statements...
            default:
                throw new ArgumentException(
                    "The first parameter must be a SelectionType enum variable."
                );
        }
        AsyncHelper.SyncBeginInvoke(this, () => gcData.DataSource = models);
    });
}
```

Listing 7.11: Het asynchroon ophalen van records.

Om ervoor te zorgen dat er in het aanroepend (managed Visual C++) project kon te weten worden gekomen of er daadwerkelijk iets was geselecteerd, werd er bij het afsluiten van het dialoogvenster ofwel een `DialogResult.Ok`- ofwel een `DialogResult.Cancel`-enumeratievariabele weergegeven door de methode `ShowDialog` (listing 7.14 op p.69). Als `DialogResult.Ok` werd teruggegeven betekende dit dat er een record was gekozen. Dit record kon dan worden opgehaald via de property `Result`. De property `Result` was van het type `IModel`. Dit was een interface dat door alle klassen voor de kolomstructuur werd geïmplementeerd. Het zorgde ervoor dat er geen '`switch case`'-constructies moesten worden opgesteld om het juiste type terug te geven. Als er dus nog een stored procedure zou moeten nagebootst worden, volstaat het om enkel een enumeratie variabele toe te voegen in listing 7.9 op p.67 en een `case` statement met de LINQ-expressie in de '`switch case`'-constructie in listing 7.11 op p.68.

Soms werden er ook parameters meegegeven. Om zo generiek mogelijk te zijn bevatte de constructor een `params`-variabele (listing 7.12 op p.68). Enige controle of alle parameters aanwezig waren voor een bepaalde LINQ query gebeurde dan in de '`switch case`'-constructie.

```
public DevExpressSelectionForm(SelectionType st, params object[] parameters)
{
    InitializeComponent();
    this.st = st;
```

```
5       this.parameters = parameters;
6   }
```

Listing 7.12: De constructor van het nieuw dialoogvenster.

Nu restte er enkel nog het vervangen van de aanroepen in het managed Visual C++ startproject GandaFact. Een veel voorkomende vervanging wordt weergegeven in listing 7.14 op p.69. In plaats van te kijken of er géén selectie was gebeurd[8] werd er gekeken of er wél een selectie was gebeurd[9]. Dit hield in dat de statements die in de originele code na de conditionele lus kwamen, in de nieuwe code binnenin de conditionele lus zaten. Ook hoefde de kolomnaam niet meer worden meegegeven. De waarde kon opgehaald worden door simpelweg het veld op te vragen dat overeenkwam met de kolom in de kolomstructuurklasse[10].

```
1   CWrapperDlgSelection DlgObject;
2   DlgObject.MakeSelection(CWrapperDlgSelection::TRANSPORTER);
3   if (!DlgObject.m_SelectionAvailable)
4       return;
5   m_pDoc->m_LeveringGasbottling->strTransporter =
        DlgObject.m_SelectionMap[_T("Nickname")];
6   (GetDlgItem(IDC_EDIT_TRANSPORTEUR))->SetWindowText(
7       m_pDoc->m_LeveringGasbottling->strTransporter
8   );
```

Listing 7.13: Het aanroepen van het nieuwe dialoogvenster.

```
1   GandaFactSelectionDlg::DevExpressSelectionForm^ DlgObject = gcnew
        GandaFactSelectionDlg::DevExpressSelectionForm(
2       GandaFactSelectionDlg::SelectionType::TRANSPORTER
3   );
4   if(DlgObject->ShowDialog() == System::Windows::Forms::DialogResult::OK){
5       GandaFactSelectionDlg::TransporterModel^ am  =
            (GandaFactSelectionDlg::TransporterModel^)DlgObject->Result;
6       m_pDoc->m_LeveringGasbottling->strTransporter = am->Nickname;
7       (GetDlgItem(IDC_EDIT_TRANSPORTEUR))->SetWindowText(
8           m_pDoc->m_LeveringGasbottling->strTransporter
9       );
10  }
```

Listing 7.14: Het aanroepen van het nieuwe dialoogvenster overeenkomstig met listing 7.13 op p.69.

Na de aanroep in listing 7.16 op p.70 werd er gebruik gemaakt van marshalfuncties. De marshalfunctie zorgde ervoor dat het .NET type werd omgezet naar het gevraagde unmanaged Visual C++ type. Volgens Microsoft (2012n) zijn de marshalfuncties aanwezig sinds Visual Studio 2008. Het hier gebruikte marshal object `marshal_context` kan een managed .NET string-object (`System::String^`) omzetten naar een '`const char*`'-, '`const wchar_t*`'- of een `BSTR`-object. De meest voorkomende managed en unmanaged types worden vaak automatisch omgezet.

```
1   CWrapperDlgSelection DlgObject;
2   DlgObject.MakeSelection(CWrapperDlgSelection::TRANSPORTER);
3   if (!DlgObject.m_SelectionAvailable)
4       return;
5   receiver->SetWindowText(DlgObject.m_SelectionMap[_T("Nickname")]);
6   if (transporterfullname != _T(""))
```

[8] "`if (!DlgObject.m_SelectionAvailable)`" in listing 7.13 op p.69

[9] "`if(DlgObject->ShowDialog() == System::Windows::Forms::DialogResult::OK)`" in listing 7.14 op p.69

[10] "`am->Nickname`" in listing 7.14 op p.69

```
7       transporterfullname = DlgObject.m_SelectionMap[_T("Name")];
```

Listing 7.15: Het aanroepen van het oude dialoogvenster.

```
1  GandaFactSelectionDlg::DevExpressSelectionForm^ DlgObject = gcnew
       GandaFactSelectionDlg::DevExpressSelectionForm(
2      GandaFactSelectionDlg::SelectionType::TRANSPORTER
3  );
4  if(DlgObject->ShowDialog() == System::Windows::Forms::DialogResult::OK){
5      GandaFactSelectionDlg::TransporterModel^ am  =
           (GandaFactSelectionDlg::TransporterModel^)DlgObject->Result;
6      msclr::interop::marshal_context context;
7      receiver->SetWindowText(context.marshal_as<const TCHAR*>(am->Nickname));
8      if (transporterfullname != _T(""))
9          transporterfullname = am->Name;
10 }
```

Listing 7.16: Het gebruik van Interop na het aanroepen van het nieuwe dialoogvenster overeenkomstig met listing 7.15 op p.69.

Een ingewikkeldere aanpassing bevond zich in listing 7.18 op p.70. Bij deze aanroep werden er parameters meegegeven voor de LINQ-expressies. Er werd een `long` meegegeven en een `CString`.

De `long`-variabele werd in de originele aanroep niet meegegeven omdat het vanuit ModuleVirtGrid kon gelezen worden. Het werd immers in *GandaFact.h* gedeclareerd. Dit is het headerbestand van de applicatie en bevatte de klasse `CGandaFactApp`, afgeleid van de MFC-klasse `CWinApp`. `CWinApp` bood functies aan voor het initialiseren en runnen van de solution. Er kon volgens Microsoft (2012f) slechts één object in de gehele solution ervan worden afgeleid. Dat afgeleid object bevatte dan alle globale variabelen en functies. Via de functie `AfxGetApp` kon er pointer naar dat object worden verkregen[11]. Omdat het Visual C# project GandaFactSelectionDlg geen toegang had tot het object moest de `long`-variabele worden meegegeven.

Omdat `CString` niet automatisch werd omgezet naar een .NET type, werd er een .NET string object aangemaakt met de inhoud van de `CString` om als parameter mee te geven. De `long`- en `enum`-variabele werden wel automatisch omgezet.

```
1  CWrapperDlgSelection DlgObject;
2  CString strhulp;
3  (GetDlgItem(IDC_EDIT_KLANT))->GetWindowText(strhulp);
4  strhulp.TrimLeft();
5  strhulp.TrimRight();
6  if (strhulp.GetLength()==7)
7      DlgObject.m_ClientID=strhulp;
8  else
9      DlgObject.m_ClientID=_T("");
10 DlgObject.MakeSelection(CWrapperDlgSelection::ORDER);
11 if (!DlgObject.m_SelectionAvailable)
12     return;
13 strhulp=DlgObject.m_SelectionMap[_T("ID")];
14 m_pDoc->m_LeveringGasbottling->lID = atol(strhulp);
15 (GetDlgItem(IDC_EDIT_LEVERINGSNUMMER))->SetWindowText(strhulp);
16 OnLookup();
```

Listing 7.17: Het aanroepen van het oude dialoogvenster.

```
1  CString strhulp;
2  CString strhulp2;
```

[11] Het moest dan wel nog gecast worden naar de klasse van dat object, in dit geval `CGandaFactApp`. Vandaar de static cast in listing 7.18 op p.70.

```
3  (GetDlgItem(IDC_EDIT_KLANT))->GetWindowText(strhulp);
4  strhulp.TrimLeft();
5  strhulp.TrimRight();
6  if (strhulp.GetLength()==7)
7      strhulp2=strhulp;
8  else
9      strhulp2=_T("");
10 GandaFactSelectionDlg::DevExpressSelectionForm^ DlgObject = gcnew
       GandaFactSelectionDlg::DevExpressSelectionForm(
11     GandaFactSelectionDlg::SelectionType::ORDER,
           static_cast<CGandaFactApp*>(AfxGetApp())->m_FirmID, gcnew
           System::String(strhulp2)
12 );
13 if(DlgObject->ShowDialog() == System::Windows::Forms::DialogResult::OK){
14     GandaFactSelectionDlg::OrderModel^ am  =
           (GandaFactSelectionDlg::OrderModel^)DlgObject->Result;
15     strhulp = am->Id.Value.ToString();
16     m_pDoc->m_LeveringGasbottling->lID = atol(strhulp);
17     (GetDlgItem(IDC_EDIT_LEVERINGSNUMMER))->SetWindowText(strhulp);
18     OnLookup();
19 }
```

Listing 7.18: Het aanroepen van het nieuwe dialoogvenster met parameters (overeenkomstig met listing 7.17 op p.70).

Hoofdstuk 8

COM als interface

De programma's van ICORDA steunen sterk op COM en COM+. De projecten van de solutions communiceren vaak met elkaar door COM. Hierdoor zijn er tussen de verschillende projecten vaste interfaces vastgelegd, waarvan handig gebruik kan gemaakt worden voor incrementele conversie naar Visual C#. Bij het opwaarderen van projecten naar Visual C# moet er dan enkel voor gezorgd worden dat de COM interfaces worden gerespecteerd.

Deze methode bleek één van de meest efficiëntste methodes en wordt op dit moment in ICORDA verder behandeld.

8.1 Strategie

De visual C++ 6.0 projecten van ICORDA solutions zijn vaak COM-componenten[1]. De interactie tussen de COM-projecten in de solution verloopt via interfaces. Daardoor maakt de inhoud van de projecten niets uit voor de andere projecten die deze via COM aanspreken, zolang de interface wordt gerespecteerd.

Dit laat toe om elk COM-project te vervangen door een alternatief Visual C# project dat dezelfde functionaliteit aanbiedt via dezelfde COM interfaces van het oorspronkelijke project. Dit heeft geen effect op andere projecten: de andere projecten hoeven zelfs niet als managed worden ingesteld. Ook de verdere ontwikkeling van de applicatie ondervindt geen hinder: er kan zelfs managed en unmanaged door elkaar gedebugged worden.

Het gebruik van COM van Visual C# wordt sterk ondersteund door het .NET framework dankzij de `System.Runtime.InteropServices` namespace. Deze namespace bevat alle nodige methodes en types om samenwerking tussen managed, unmanaged en op COM steunende projecten mogelijk te maken.

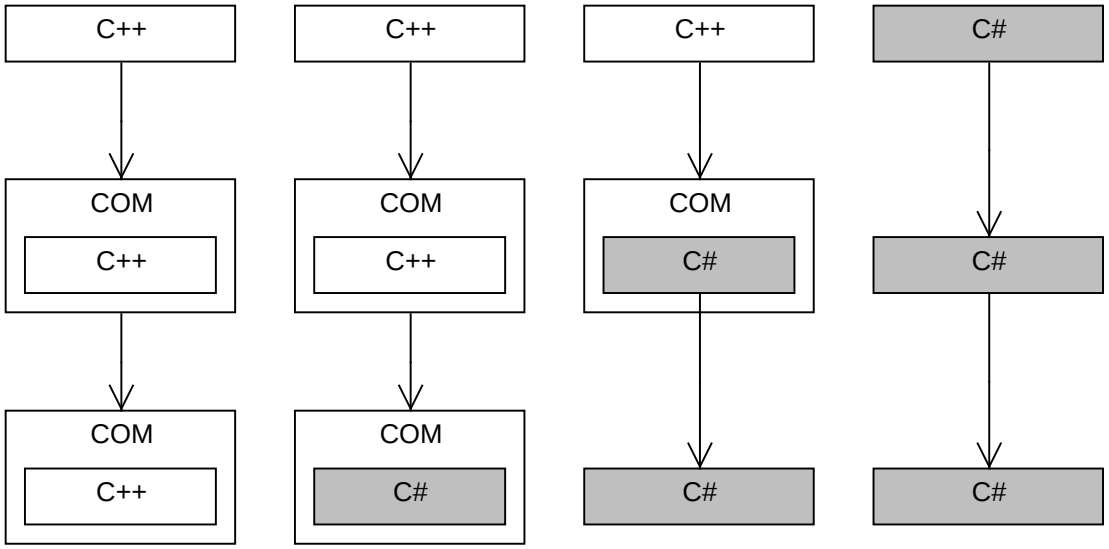
Als er dan een tweede project in de solution wordt omgezet naar Visual C#, hoeven de Visual C# applicaties onderling geen gebruik meer te maken van COM. De projecten zijn dan immers beide managed. Op deze manier wordt er voor gezorgd dat er uiteindelijk geen gebruik meer wordt gemaakt van COM (en COM+).

Omdat de namespace `System.Runtime.InteropServices` ook compabiliteit aanbiedt tussen native en managed code die geen COM-componenten zijn, zou de methode zou ook toepasbaar zijn met native Visual C++ projecten die geen gebruik maken van COM[2]. Op dit

[1] Niet alle projecten maken gebruik van COM. Zo is het startproject iRent is geen COM-project. Ook zijn er andere niet COM-projecten aanwezig, zoals SalesForce

[2] Zoals SalesForce in iRent.

moment is dit nog niet getest.



Figuur 8.1: Een overzicht van de strategie (van links naar rechts). De grijze blokken zijn managed projecten.

8.2 Proof of concept

Om deze methode te onderzoeken werd er een proof of concept opgesteld. Hiervoor werd het project IRServer van de applicatie iRent.Classic[3] gekozen. Dit project biedt via een COM interface methodes aan om stored procedures uit te voeren op een databank. Het kan vergeleken worden met een 3-tier structuur: het startproject iRent (presentatie) doet een beroep op het COM-project IRClient (bussiness), dat op zijn beurt voor toegang tot de databank beroep doet op het COM-project IRServer (data). Dit wordt duidelijk zichtbaar in figuur 8.1 op p.73. In deze figuur stelt het bovenste project iRent voor, het middelste IRClient en het onderste IRServer.

Dit project kan dus eigenlijk vervangen worden door een Visual C# project dat de databank aanspreekt via een Visual C# model van de databank en dat methodes aanbiedt met dezelfde functionaliteit als IRServer via een COM interface. Het Visual C# model van de databank wordt met behulp van het Entity Framework, een .NET objectrelationele mapper van Microsoft, net zoals bij GandaFact in hoofdstuk 7.4 op p.64.

Er werd een nieuw Visual C# project toegevoegd aan de iRent.Classic solution. Dit project, IRentServer genaamd, was dan het enige managed project van de solution. Om een Visual C# COM dll-bestand aan te maken moeten er eerst enkel instellingen worden ingesteld:

Register for COM interop

Deze instelling kan worden ingesteld bij de properties van het project (`Build` > `Output`

[3]iRent.Classic is de tijdens deze masterproef opgewaardeerde versie van de applicatie iRent.

> `Register for COM interop` aanvinken). Dit zorgt ervoor dat het project een COM-object aanmaakt dat kan aangesproken worden door andere COM-objecten, dit object is dus een soort COM-aanroepbaar *wrapperobject*[4]. Dit komt overeen met het manueel registreren van de dll via de Assembly Registration Tool (Regasm.exe) met de optie "/tlb" (en indien nodig ook met de optie "/codebase") op de command line. (Microsoft, 2010b)

Make assembly COM-Visible
Ook deze instelling wordt ingesteld bij de properties van het project (`Application` > `Assembly Information...` > `Make assembly COM-Visible` aanvinken). Deze optie zorgt ervoor dat alle publieke leden (klassen, methodes, variabelen, ...) zichtbaar worden via COM. Indien er enkele specifieke leden zichtbaar moeten zijn kan deze optie worden uitgevinkt en het `ComVisibleAttirbute`-attribuut worden gebruikt. Dit is enkel geldig op publieke leden. (Microsoft, 2012e)

IRentServer werd zodanig opgesteld dat niet de methodes, maar enkel de declaratie en initialisatie van pointers naar de COM interface zullen moeten worden aangepast.

```
1 IIRBewegingPtr pBeweging(__uuidof(IRData));
```

Listing 8.1: De declaratie en inialisatie van het pointerobject dat wijst naar het Visual C++ project IRServer.

```
1 IRentServer::Server_InterfacePtr pBeweging(__uuidof(IRentServer::Server));
```

Listing 8.2: De declaratie en inialisatie van het pointerobject dat wijst naar het Visual C# project IRentServer.

Op deze manier hoeven er (bijna[5]) geen aanpassingen aan de aanroepende COM-componenten.

8.2.1 De basisstructuur

Eerst en vooral moest er eerst een COM interface worden opgesteld zoals in listing 8.3 op p.74.

```
1  using System;
2  using System.Runtime.InteropServices;
3
4  namespace IRentServer
5  {
6      [Guid("009F5A85-E11F-4F95-825A-C5E568F4F6A3")]
7      public interface Server_Interface
8      {
9      }
10 }
```

Listing 8.3: De COM interface van IRentServer.

De Globally Unique Identifier (GUID) werd gegenereerd in Visual Studio zelf via de "Create GUID"-wizard figuur 8.2 op p.75.

[4]Een wrapperobject is een object dat een ander object incapsuleert. Daardoor komt er tussen het origineel object en de buitenwereld een tussenlaag. In dit geval zorgt het wrapperobject ervoor dat andere objecten via COM contact kunnen leggen met de tussenlaag, die op zijn beurt beroep doet op de functionaliteit in het oorspronkelijk object.

[5]In sommige gevallen zijn er enkele kleine wijzigen nodig. Dit wordt besproken in hoofdstuk 8.2.3 op p.79

Figuur 8.2: De "Create GUID"-wizard.

Voor de duidelijkheid werd de interface in een eigen broncodebestand *Server_Interface.cs* gestoken. Op basis van deze interface wordt er dan bij het builden een COM interface gegenereerd.

Er werd ook een bestand *Server_Events.cs* aangemaakt indien er gebruik zou gemaakt worden van COM events. Dit bestand bevat een interface `Server_Events` waarin de events kunnen worden gedefinieerd (listing 8.3 op p.74). Net als bij de interface `Server_Interface` werd er een GUID meegegeven, gegenereerd met de Visual Studio GUID wizard. Dit werd gevolgd ook een extra attribuut `InterfaceType(ComInterfaceType.InterfaceIsIDispatch)`. Deze optie duidt aan dat er, op basis van de interface `Server_Interface`, een `IDispatch` interface moet worden gegenereerd voor *late binding*[6].

```
1  using System;
2  using System.Runtime.InteropServices;
3
4  namespace IRentServer
5  {
6      [
7          Guid("BF6DB3F2-363F-488D-A2DB-C173A8185DD9"),
8          InterfaceType(ComInterfaceType.InterfaceIsIDispatch)
9      ]
10     public interface Server_Events
11     {
12     }
13 }
```

Listing 8.4: De COM interface voor COM events van IRentServer.

Een voorbeeld van het definiëren van COM events in Visual C# is gegeven in bijlage D op p.162.

Voor de implementaties van de functies in de interface wordt er een apart broncodebestand aangemaakt, namelijk *Server.cs*. Het bestand *Server.cs* bevat een klasse `Server` dat de interface `Server_Interface` uit het bestand *Server_Interface.cs* implementeert. Ook deze klasse krijgt een GUID mee, gegenereerd door de GUID wizard van Visual Studio. Daarnaast worden ook de opties `ClassInterface(ClassInterfaceType.None)` en `ComSourceInterfaces(typeof(Server_Events))` meegegeven:

[6]Late binding is het binden aan een object terwijl het programma loopt. Het alternatief is early binding, waarbij het binden met een object tijdens het compileren gebeurt.

`ClassInterface(ClassInterfaceType.None)`
: Deze optie geeft aan dat er geen COM interface moet worden gegenereerd voor de klasse zelf. Er worden dan enkel COM interfaces gegenereerd gebaseerd op de expliciet door de klasse geïmplementeerde interfaces. Indien de klasse geen interfaces geïmplementeerd hebben, wordt er een `IDispatch` interface gegenereerd voor late binding. (Microsoft, 2012c)

`ComSourceInterfaces(typeof(Server_Events))` Deze optie wijst erop dat de interface `Server_Events` de definities van de COM events bevat aanwezig in de klasse.

```
1  using RecordsetNet;
2  using System;
3  using System.Collections.Generic;
4  using System.Diagnostics;
5  using System.Linq;
6  using System.Runtime.InteropServices;
7
8  namespace IRentServer
9  {
10     [
11         Guid("33058759-9972-4B57-9900-040FD48C8131"),
12         ClassInterface(ClassInterfaceType.None),
13         ComSourceInterfaces(typeof(Server_Events))
14     ]
15     public class Server : Server_Interface
16     {
17     }
18 }
```

Listing 8.5: De COM interface van IRentServer.

8.2.2 Het Visual C# databankmodel

Alvorens er werd gekeken naar het toevoegen van methodes moest er eerst een Visual C# model van de databank worden opgesteld. Omdat IRServer een project is dat hevig steunt op de databank, werd aan IRentServer een databankmodel toegevoegd, gebaseerd op de iRent databank. Dit model is een Visual C# weergave van de databank dat klassen en methodes bevat voor overeenkomstige tabellen, stored procedures (en hun resultaattypes), ... Het model werd gegenereerd door het Entity Framework, een .NET objectgerelationeerde mapper van Microsoft. Voor IRentServer gebruikte men Entity Framework 5.0.0. Dit verschilde met de voorgaande versie gebruikt bij GandaFact (hoofdstuk 7.4 op p.64) zodat er geen connectionstring kon worden meegegeven met de constructor van het databankmodel[7]. Er werd dus enkel gekeken naar de connectionstring in de *App.config* van IRentServer listing 8.6 op p.76.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3    <connectionStrings>
4      <add name="irentEntities"
5           connectionString="metadata=res://*/Model.csdl
6           |res://*/Model.ssdl|res://*/Model.msl;
7           provider=System.Data.SqlClient;
8           provider connection string=&amp;quot;
9           data source=WS-DE-STG01\DEV;
```

[7]In 'GandaFact werd er een oudere versie gebruikt dan hier, omdat de proof of concept voor iRent enkele maanden later werd opgesteld dan die van GandaFact. In dat tijdsbestek was er dus een nieuwe versie van het Entity Framework, versie 5.0.0, uitgebracht.

```
10          initial catalog=irent;integrated security=True;
11          MultipleActiveResultSets=True;
12          App=EntityFramework&amp;quot;"
13          providerName="System.Data.EntityClient" />
14    </connectionStrings>
15  </configuration>
```

Listing 8.6: De connectionstring voor de iRent databank in *App.config* (de connectionstring zelf bevat hier whitespace zodat alles wordt duidelijk weergegeven, in *App.config* is er geen whitespace aanwezig in de connectionstring).

Op het eerste zicht leek dit geen probleem, maar bij het opstellen van de methodes werd duidelijk dat de connectionstring vaak werd meegegeven. Er werd gepoogd de methodes te implementeren enkel op basis van de *App.config* door de parameter met de connectionstring te negeren, maar dit gaf geen gewenste resultaten.

Eerst en vooral worden de connectionstrings al meegegeven in een configuratiebestand *irentconnectionc.xml* in het opstartproject iRent in de solution. Het gebruik van een *App.config* bestand in het IRentServer project zorgde voor redundantie. Deze redundantie zou zich bij elk omgezet naar Visual C# voordoen. Het ineens aanpassen van code zodat na omzetting van alle projecten naar Visual C# er slechts een enkel *App.config* bestand nodig is (voor het startupproject) in de gehele solution zou een enorme en foutgevoelige klus worden wegens de gebrekkige modulariteit en aanwezigheid van grote hoeveelheden redundante code in iRent.Classic.

Het tweede probleem is de onduidelijkheid over de scope van *App.config* in managed en unmanaged delen van de solution. Bij het aanroepen van de methode in het unmanaged iRent opstartproject[8] werd het gewenst resultaat niet teruggegeven. Het bleek dat er in de methode in IRenstServer bij het aanroepen van het datanbankmodel telkens de fout "No connection string named 'Entities' could be found in the application config file." werd opgeworpen. Indien dezelfde methode werd opgeroepen vanuit een ander Visual C# testproject bleek de methode wel correct uitgevoerd te worden. Het Visual C# testproject moest dan wel ook beschikken over een *App.config* bestand met dezelfde connectionstring als in IRentServer, anders werd dezelfde fout opgeworpen. Hierdoor werd het duidelijk dat het bij het oproepen van de methode vanuit unmanaged code, het unmanaged project dat de aanroep doet ook moest beschikken over de connectiestring. Als antwoord hierop werd er een *app.config*[9] toegevoegd aan het iRent startproject, maar dit had blijkbaar geen effect. Dit was niet geheel onverwacht aangezien het automatisch inlezen van *App.config* bestanden enkel gebeurt in managed projecten. Omdat de *App.config* bestanden worden hernoemd naar '¡projectnaam¿.exe.config', werd er geprobeerd het configuratiebestand *IRentServer.dll.config* te hernoemen naar *iRent.exe.config* en in dezelfde folder[10] als *iRent.exe* te plaatsen. Ook dit had geen effect.

Het was dus duidelijk dat er een manier moest worden gevonden om toch de connectiestring te kunnen meegegeven aan de constructor van het databankmodel. Het antwoord hier was partiële klassen. Een partiële klasse is een klasse waarvan de definitie is opgesplitst in meerdere delen. Alle klassedefinities in dezelfde scope met dezelfde klassenaam, voorgegaan door het keyword `partial`, vormen samen de definitie van de klasse. Er werd gebruik gemaakt van partiële klassen omdat het databanmodel enkel uit gegenereerde code bestaat. Telkens als er

[8] Meer informatie over het implementeren van methodes in de Visual C# projecten en hun aanroepen bevindt zich in hoofdstuk 8.2.3 op p.79.

[9] In Visual C++ projecten heeft de naam geen hoofdletters.

[10] De standaardlocatie voor ".exe.config"- of ".dll.config"-bestanden is hetzelfde als de locatie van het overeenkomstig exe- of dll-bestand.

iets aan veranderd wordt, wordt de code ook opnieuw gegenereerd. Door gebruik te maken van partiële klassen kan de zelfgeschreven code en de gegenereerde code van elkaar worden gescheiden. Zo wordt de manueel geschreven code niet overschreven als de code van het databankmodel opnieuw wordt gegenereerd. Dit wordt ook gebruikt bij Windows Forms en WPF.

> "*Having the declaration of a class distributed over separate segments of program text can be useful if these segments are produced or maintained in different contexts. For instance, one part of a class declaration may be machine generated, whereas the other is authored manually. Textual separation of the two prevents updates by one from conflicting with updates by the other.*" (HejlsBerg et al., 2011)

De belangrijkste klasse van het databankmodel[11] erft over van de klasse `DbContext` en heeft een default constructor listing 8.7 op p.78. Het is deze klasse die wordt aangeroepen in de methodes. Het wordt beschreven in het ".Designer.cs"-bestand van het databankmodel.

```
1  //------------------------------------------------------------------------------
2  // <auto-generated>
3  //     This code was generated from a template.
4  //
5  //     Manual changes to this file may cause unexpected behavior in your application.
6  //     Manual changes to this file will be overwritten if the code is regenerated.
7  // </auto-generated>
8  //------------------------------------------------------------------------------
9
10 namespace IRentServer
11 {
12     using System;
13     using System.Data.Entity;
14     using System.Data.Entity.Infrastructure;
15     using System.Data.Objects;
16     using System.Data.Objects.DataClasses;
17     using System.Linq;
18
19     public partial class irentEntities : DbContext
20     {
21         public irentEntities()
22             : base("name=irentEntities")
23         {
24         }
25
26         protected override void OnModelCreating(DbModelBuilder modelBuilder)
27         {
28             throw new UnintentionalCodeFirstException();
29         }
30
31         internal DbSet<C_VeldTypes> C_VeldTypes { get; set; }
32         internal DbSet<Algemeen> Algemeen { get; set; }
33         internal DbSet<Artikel> Artikel { get; set; }
34         internal DbSet<ArtikelNTC> ArtikelNTC { get; set; }
35         // ...
```

Listing 8.7: De belangrijkste klasse van het databankmodel `irentEntities`.

Zoals te zien in listing 8.7 op p.78 was de klasse `irentEntities` een partiële klasse en kon deze worden uitgebreid via andere partiële klassendefinities. Om zelf een constructor toe te voegen werd er in het project een broncodebestand *PartialEntities.cs* toegevoegd waarin de klasse werd uitgebreid met een extra constructor listing 8.8 op p.79. Deze constructor riep dan een constructor van de klasse `DbContext` op dat een connectionstring aanvaardde.

[11] In het project IRentServer werd het `irentEntities` genoemd.

```
using System.Data.Entity;

namespace IRentServer
{
    public partial class irentEntities : DbContext
    {
        public irentEntities(string connectionstring)
            : base(connectionstring)
        {
        }
    }
}
```

Listing 8.8: Het bestand *PartialEntities.cs*.

Dit liet dan toe om een connectionstring mee te geven aan de constructor van het databankmodel.

8.2.3 Implementatie van methodes

Het implementeren van methodes voor de COM interface in Visual C# COM-projecten is op zich relatief eenvoudig. De signatuur van de methode wordt opgenomen in de interface, voorafgegaan door een `DispId`-attribuut. De implementatie van de methode komt in de klasse die de interface definieert.

```
namespace IRentServer
{
    [Guid("009F5A85-E11F-4F95-825A-C5E568F4F6A3")]
    public interface Server_Interface
    {
        [DispId(1)]
        string HelloWorld();
    }
}
```

Listing 8.9: De signatuur van een COM-methode in de interface.

```
using System;
using System.Runtime.InteropServices;

namespace IRentServer
{
    [
        Guid("33058759-9972-4B57-9900-040FD48C8131"),
        ClassInterface(ClassInterfaceType.None),
        ComSourceInterfaces(typeof(Server_Events))
    ]
    public class Server : Server_Interface
    {
        public string HelloWorld()
        {
            return "Hello World!";
        }
    }
}
```

Listing 8.10: De implementatie van een COM-methode in de klasse.

Het `DispId`-attribuut staat eigenlijk voor “DispatchId”. Het is een nummer dat wordt toegekend aan de methode. Bij late binding gebruikt de COM-component die de methode wil aanroepen dit nummer om de functie aan te roepen. De interface die hier wordt gegenereerd erft over van `IDispatch`. `IDispatch` bevat twee methodes: `GetIDsOfNames` en `Invoke`. In

de aanroepende COM-component koppelt `GetIDsOfNames` de DispatchId nummers aan de functienamen zodat de functies dan via de functienaam kunnen worden aangeroepen. `Invoke` voert de methode uit overeenkomstig met het DispatchId. Deze twee methodes vormen de basis van late binding. Vaak worden deze methodes achter de schermen aangeroepen, zoals bij MFC. De moeilijkheid is het uitwisselen van objecten tussen de managed en unmanaged COM-componenten. Het is hier dat de namespace `System.Runtime.InteropServices` en alle daarinliggende klassen en methodes hun grootste nut tonen.

Als de Visual C# COM-component IRentServer wordt gecompileerd, wordt er een tlb-bestand gegenereerd (door de "Register for COM interop" optie (zie p.73). Het is dit tlb-bestand, de type library, dat in de aanroepende COM-componenten wordt geïmporteerd (listing 8.11 op p.80). Het `import` statement zorgt ervoor dat er op basis van het geïmporteerd tlb-bestand twee bestanden worden gegenereerd: een tlh- en een tli-bestand, ook wel respectievelijk de primary en secundary type library files genaamd. Deze bestanden bevatten de aangeboden functionaliteit, zoals de methodes in de COM interfaces, in Visual C++ code. Het tli-bestand wordt geïncludeerd in het tlh-bestand, dat op zijn beurt automatisch wordt geïncludeerd door de compiler. Het tlh-bestand bevat de *raw* methodes. Dit zijn de methodes met returntype `HRESULT` en prefix "`raw_`". Deze methodes worden in het tli-bestand ingekapseld in methodes[12] met de originele[13] naam en het originele returntype[14]. In de raw methodes is de laatste (out) parameter een pointer naar het origineel returntype. De raw methodes hebben als returntype `HRESULT` omdat in de ingekapselde methodes aan de hand van de returnwaarde wordt vastgesteld of er een fout is opgetreden. De ingekapselde methoden bevatten dus foutafhandeling en worden daarom aanbevolen om te gebruiken. Indien gewenst kunnen ook de raw methodes worden aangeroepen.

```
#import "C:\Dev2\CIACfleet\trunk\main\"
        "IRent.Server\bin\x86\Debug\IRentServer.tlb"
        named_guids
```

Listing 8.11: Het importeren van het tlb-bestand.

```
struct __declspec(uuid("f9b2da70-986e-11d5-b7f6-0050da07e52c"))
IIRGebruiker : IUnknown
{
    //
    // Wrapper methods for error-handling
    //

    //...

    IUnknownPtr GebruikerGetAll (
        _bstr_t ConnectionStringFromClient );

    //
    // Raw methods provided by interface
    //

    //...

      virtual HRESULT __stdcall raw_GebruikerGetAll (
```

[12]Eigenlijk zijn deze wrappermethodes al gedefinieerd in het tlh-bestand. Enkel de implementatie van de wrapper methodes zitten in het tli-bestand.

[13]Met origineel wordt hier telkens 'zoals gedefinieerd in de overeenkomstige COM interface' bedoeld.

[14]De returnvariabele van de wrapper methode is eigenlijk de variabele waarnaar deze pointer wijst. Indien de originele returntype `void` is is er geen dergelijke parameter voor de returnvariabele en wordt de returntype van de wrapper methode `HRESULT`.

```
20          /*[in]*/ BSTR ConnectionStringFromClient,
21          /*[out,retval]*/ IUnknown * * ppadoRS ) = 0;
22  };
23
24  //...
25
26  //
27  // Wrapper method implementations
28  //
29
30  #include "irserver.tli"
```

Listing 8.12: De ruwe methodes en declaratie van de wrapper methodes in het tlh-bestand.

```
1   //
2   // interface IIRGebruiker wrapper method implementations
3   //
4
5   //....
6
7   inline IUnknownPtr IIRGebruiker::GebruikerGetAll ( _bstr_t ConnectionStringFromClient
        ) {
8       IUnknown * _result = 0;
9       HRESULT _hr = raw_GebruikerGetAll(ConnectionStringFromClient, &_result);
10      if (FAILED(_hr)) _com_issue_errorex(_hr, this, __uuidof(this));
11      return IUnknownPtr(_result, false);
12  }
```

Listing 8.13: De implementatie van de wrapper methodes in het tli-bestand.

In het origineel Visual C++ project IRServer is er een (manueel geschreven) idl-bestand aanwezig dat de COM interface definieert. Dit bestand wordt tijdens het compilen van IRServer door de Microsoft Interface Definition Language (MIDL) compiler gecompileerd naar een tlb-bestand. Daarnaast worden er nog extra bestanden gegenereerd zoals een interface proxy bestand, een header, een interface Universal Unique Identifier (UUID)-bestand en een interface registratie bestand. Het gegenereerde tlb-bestand zorgt op zijn beurt voor de generatie van de primary en secundary type library files door het `import` statement in het Visual C++ startproject iRent[15]. Door te kijken naar de verschillen en gelijkenissen tussen tlh-bestanden van IRentServer (Visual C#) en IRServer (Visual C++) kan er worden afgeleid of de methodes in de COM interface van IRentServer overeenkomen met die van IRServer. Er moet enkel worden gekeken naar de raw methodes in de tlh-bestanden. Immers als die gelijk zijn, zullen de daarop gebaseerde methodes in het tli-bestand ook gelijk zijn.

Dezelfde signatuur genereren voor methodes in Visual C# als de originele Visual C++ COM-methodes is niet vanzelfsprekend. Zo steunen de methodes in Visual C++ hevig op pointers, welke in Visual C# niet gebruikelijk is. Om toch de gewenste parameter- en returntypes te kunnen aanbieden wordt er in Visual C# gebruik gemaakt van attributen gedefinieerd in de namespace `System.Runtime.InteropServices`.

Voor de proof of concept werden er drie methodes geïmplementeerd (listing 8.14 op p.81).

```
1   using System;
2   using System.Runtime.InteropServices;
3
4   namespace IRentServer
5   {
6       [Guid("009F5A85-E11F-4F95-825A-C5E568F4F6A3")]
7       public interface Server_Interface
```

[15]Door de buildorder van de solution en de dependencies van het project wordt het project iRent als laatste gebuild, na IRServer en IRentServer

```
8       {
9           [DispId(1)]
10          [return: MarshalAs(UnmanagedType.IUnknown)]
11          Object GebruikerGetAll(string connectionstring);
12
13          [DispId(2)]
14          void WagenGetAll(
15              string connectionstring,
16              int id,
17              [MarshalAs(UnmanagedType.IUnknown)] out Object par1,
18              [MarshalAs(UnmanagedType.IUnknown)] out Object par2,
19              [MarshalAs(UnmanagedType.IUnknown)] out Object par3
20          );
21
22          [DispId(3)]
23          [return: MarshalAs(UnmanagedType.IUnknown)]
24          Object BewegingGetForWagenHistoriek(
25              string connectionstring,
26              string NummerplaatWeergave,
27              [MarshalAs(UnmanagedType.Bool)] bool IsKosten,
28              int GroepID, DateTime DatumVan,
29              DateTime DatumTot,
30              [MarshalAs(UnmanagedType.Bool)] bool IsBoekhoudkundig
31          );
32      }
33  }
```

Listing 8.14: De volledig uitegewerkte interface van IRentServer.

Meeste standaard Visual C# datatypes worden automatisch vertaald naar hun Visual C++ tegenhanger (tabel 8.1 op p.82). Voor andere types is er wat meer hulp nodig en daar komen de attributen uit de namespace `System.Runtime.InteropServices` van pas.

Visual C# type	Visual C++ type
`string`	`BSTR`
`int`	`long`
`DateTime`	`DATE`

Tabel 8.1: De automatische vertaling voor de standaard types in listing 8.14 op p.81.

Om een `IUnknown**` dubbele pointer weer te geven wordt bijvoorbeeld gebruik gemaakt van het `return`-attribuut `MarshalAs(UnmanagedType.IUnknown)` toegepast op het type `Object`. Om andere types van returnvariabelen en parameters juist te vertalen kunnen dezelfde attributen gebruikt worden. Zo wordt bijvoorbeeld het attribuut `MarshalAs(UnmanagedType.Bool)` gebruikt om variabele van het type `bool` om te zetten naar `long`[16]. Sommige methodes hebben meerdere returnvariabelen. Deze worden in Visual C++ dan geïmplementeerd als output parameters via pointers. In Visual C# worden output parameters, aangeduid met het keyword `out`, automatisch omgezet naar (output) pointers. Er moet dan wel gespecificeerd worden naar welk type de pointers moeten wijzen via analoge attributen als voor de andere parameters en return variabelen.

Niet alle types in Visual C++ kunnen worden vertaald naar een Visual C++ tegenhanger. Zo zullen generieke en nullable types niet worden aanvaard door de Visual C# compiler.

Indien er wordt gebruik gemaakt van het Visual C# `Object`-type om een Visual C++ pointer (of dubbele pointer) te bekomen, moet er rekening worden gehouden met wat het

[16] In Visual C++ zijn booleaanse waarden eigenlijk integers. 0 staat voor `false` en alle andere waarden voor `true`.

Visual C# type	Interop attribuut	Visual C++ type
`bool`	`MarshalAs(UnmanagedType.Bool)`	`long`
`Object`	`MarshalAs(UnmanagedType.IUnknown)`	`**IUnknown`

Tabel 8.2: De manueel vertaling voor types in listing 8.14 op p.81.

`Object`-type inhoudt en hoe de pointer in de aanroepende Visual C++ COM-component wordt gebruikt. Bij de methodes in listing 8.14 op p.81 worden alle `Object`-types omgezet naar `IUnknown**` dubbele pointers. Indien de `Object`-variabele een returnvariabele is, is het returntype van de Visual C++ wrappermethode een `IUnknownPtr`, een smartpointer overeenkomstig met `IUnknown*`. Als de `Object`-variabele een `out`-parameter is blijft het een `IUnknown**` pointer. De pointers worden dan in de aanroepende Visual C++ COM-componenten gecast naar `_RecordsetPtr` pointers.

```
IUnknownPtr pUnk = pBeweging->BewegingGetForWagenHistoriek(GetConnectionString(),
    m_NummerplaatWeergave, m_IsKosten, m_GroepID, m_bDatumVan ? m_DatumVan : 0,
    m_bDatumTot ? m_DatumTot : 0, m_IsBoekhoudkundig);
_RecordsetPtr RS = pUnk;
//TraceRecordset(RS);

if(RS != nullptr
   && !RS->adoBOF
   && !RS->adoEOF)
{
    RS->MoveFirst();
}
```

Listing 8.15: Een voorbeeld van het inlezen van een `_RecordsetPtr` aan de hand van een wrapper methode gegenereerd op basis van de methodes in listing 8.14 op p.81.

```
HRESULT hr = pDB->raw_GebruikerGetAll(bstrConnectionString, &pIUnk);
// pIUNK is een *IUnknown pointer
if (SUCCEEDED(hr))
{
    _RecordsetPtr RS = pIUnk;
    // verdere bewerkingen met RS
}
```

Listing 8.16: Een voorbeeld van het inlezen van een `_RecordsetPtr` aan de hand van een raw methode gegenereerd op basis van de methodes in listing 8.14 op p.81 en een extra `IUnknown*` pointer variabele.

```
hr = pWagen->raw_WagenGetAll(GetConnectionString(), ID, ppRSWagen, ppRSContractKT,
    ppRSContractLT);

HRESULT CWagen::GetRSs(long ID, IUnknown **ppRSWagen, IUnknown **ppRSContractKT,
    IUnknown **ppRSContractLT)
{
    HRESULT hr = E_FAIL;
    IRentServer::Server_InterfacePtr pWagen(__uuidof(IRentServer::Server));

    hr = pWagen->raw_WagenGetAll(GetConnectionString(), ID, ppRSWagen,
        ppRSContractKT, ppRSContractLT);
    // ppRSWagen, ppRSContractKT en ppRSContractLT zijn **IUnknown pointers
    pWagen = NULL;

    return hr;
}

IUnknown* pUnkWagen = NULL;
```

```
16 IUnknown* pUnkContractKT = NULL;
17 IUnknown* pUnkContractLT = NULL;
18 hr = GetRSs(ID, &pUnkWagen, &pUnkContractKT, &pUnkContractLT);
```

Listing 8.17: Een voorbeeld van het inlezen van drie _RecordsetPtr-variabelen aan de hand van een raw methode gegenereerd op basis van de methodes in listing 8.14 op p.81 en drie extra *IUnknown pointer variabelen.

Om ervoor te zorgen dat de Object-variabelen worden opgevuld met 'ADODB.Recordset'-types werd er beroep gedaan op een externe library RecordsetNet. In IRentServer moest er dan enkel een referentie naar RecordsetNet en ADODB worden toegevoegd. RecordsetNet bood slechts één enkele extensiemethode ToRecordset aan listing 8.18 op p.84.

```
1 public static ADODB.Recordset ToRecordset<T>(this IEnumerable<T> input)
```

Listing 8.18: De extensiemethode ToRecordset.

Deze methode kan worden toegepast op hetvtype IEnumerable<T> en geeft een ADODB.Recordset met de inhoud van de 'IEnumerable<T>'-variabele terug. Op deze manier kan de methode overeenkomstig met de stored procedure[17] in het databankmodel aangeroepen worden en het resultaat teruggeven via COM in een formaat dat in de aanroepende Visual C++ COM-component wordt gebruikt (listing 8.19 op p.84). Soms geven stored procedures meerdere tabellen terug. Dit kan niet worden opgevangen door enkel een Object-returntype. Vaak wordt hier beroep gedaan op out-parameters. Omdat dergelijke stored procedures vaak steunen op andere stored procedures is de meest voor de hand liggende oplossing om de out-parameters met die stored procedures op te vullen (listing 8.20 op p.84).

```
1  public Object GebruikerGetAll(string connectionstring)
2  {
3      TextWriterTraceListener myTextListener = new
           TextWriterTraceListener(@"C:\log.txt");
4      Trace.Listeners.Add(myTextListener);
5      try
6      {
7          using (irentEntities model = new irentEntities("metadata=res://*/Model.csdl|"
               + "res://*/Model.ssdl|res://*/Model.msl;" +
               "provider=System.Data.SqlClient;" + "provider connection string='data
               source=WS-DE-STG01\\DEV;" + "initial catalog=irent;" + "integrated
               security=True;" + "MultipleActiveResultSets=True;" +
               "App=EntityFramework'"))
8          {
9              var res = model.usp_GebruikerGetAll().ToList();
10             var test = res.ToRecordset();
11             return test;
12         }
13     }
14     catch (Exception e)
15     {
16         Trace.WriteLine(e.Message);
17         return new List<Object>().ToRecordset();
18     }
19 }
```

Listing 8.19: Een voorbeeld van het opvullen en teruggeven van een ADODB.Recordset met behulp van het databankmodel en de extensiemethode ToRecordset in Visual C#.

```
1 public void WagenGetAll(string connectionstring, int id, out Object par1, out Object
      par2, out Object par3)
2 {
```

[17] De stored procedure dat ook werd aangeroepen in de originele methode in IRServer

```
3       TextWriterTraceListener myTextListener = new
            TextWriterTraceListener(@"C:\log.txt");
4       Trace.Listeners.Add(myTextListener);
5       try
6       {
7           using (irentEntities model = new irentEntities("metadata=res://*/Model.csdl|"
                + "res://*/Model.ssdl|res://*/Model.msl;" +
                "provider=System.Data.SqlClient;" + "provider connection string='data
                source=WS-DE-STG01\\DEV;" + "initial catalog=irent;" + "integrated
                security=True;" + "MultipleActiveResultSets=True;" +
                "App=EntityFramework'"))
8           {
9               par1 = model.usp_WagenGet(id).ToList().ToRecordset();
10              par2 = model.usp_ContractKTGetForWagen(id).ToList().ToRecordset();
11              par3 = model.usp_ContractLTGetForWagen(id).ToList().ToRecordset();
12          }
13      }
14      catch (Exception e)
15      {
16          Trace.WriteLine(e.Message);
17          par1 = new List<Object>().ToRecordset();
18          par2 = new List<Object>().ToRecordset();
19          par3 = new List<Object>().ToRecordset();
20      }
21  }
```

Listing 8.20: Een voorbeeld van het opvullen en teruggeven van een `ADODB.Recordset` met behulp van het databankmodel en de extensiemethode `ToRecordset` in Visual C#, indien de overeenkomstige stored procedure meerdere tabellen weergeeft op basis van andere stored procedures.

Er kunnen soms enkele problemen opduiken. Zo was er een stored procedure die meerdere kolommen dezelfde naam gaf. Het Entity Framework kon hier niet mee om en voegde een nummer toe achteraan de namen van kolommen met een duplicate naam. Dit leidde tot inconsistentie en daarmee gepaard gaande fouten tussen databank en IRentServer. De oplossing was dan om gewoonweg de stored procedure aan te passen zodat er geen duplicate kolomnamen meer waren. Met duplicate kolomnamen kunnen ze toch nergens worden aangesproken, dus was er zekerheid dat het aanpassen van de duplicaten geen problemen zou veroorzaken. Ook waren er soms fouten door het feit dat de pointer naar de record in de recordset niet bovenaan stond. In dat geval kan de pointer juist worden ingesteld met behulp van de `adoBOF`- en `adoEOF`-leden en de methode `MoveFirst` van de recordset zoals in listing 8.15 op p.83. Sommige methodes zoals `TraceRecordset` zetten de recordpointer automatisch goed.

8.2.4 Opmerkingen

Na de succesvolle implementatie van de proof of concept bleef er telkens een (first chance) exception opduiken (listing 8.21 op p.86). Na het testen met verschillende frameworkversies van het Visual C# COM-project bleek dat de fout enkel voorkwam bij versies 4.0 en 4.5. Volgens Dore (2010) is dit een ongedocumenteerde fout aangeduid als "`CLRDBG_NOTIFICATION_EXCEPTION_CODE`". Het wordt gebruikt als aanvulling voor het Inter-Process Communication (IPC)-protocol dat door de managed debugger in frameworkversies 4.0 en 4.5 wordt gebruikt. Het IPC-protocol is een protocol voor het uitwisselen van informatie tussen threads van één of meer processen. Deze fout is dus geen echte fout en dient enkel om informatie door te geven. Het mag dus genegeerd worden.

```
1 First-chance exception at 0x7697C41F (KernelBase.dll) in Program.exe: 0x04242420
      (parameters: 0x31415927, 0x6F310000, 0x00BBDAE8).
```

Listing 8.21: Een ongedocumenteerde fout bij het aanroepen van een Visual C# methode vanuit een unmanaged Visual C++ COM-component.

> "*Out of curiosity I did a little digging and found that this is actually an undocumented exception (CLRDBG_NOTIFICATION_EXCEPTION_CODE) that is apparently an addition to the IPC protocol used by the managed debugger in the 4.0 CLR. It should be entirely safe to ignore.*" (Dore, 2010)

Hoofdstuk 9

Rc2Form

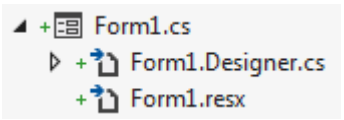
Visual C++ solutions bevatten grote hoeveelheden code die niet hoeven (en meestal ook niet kunnen) omgezet worden naar Visual C#. Een voorbeeld hiervan is code gebaseerd op de MFC en ATL libraries. In Visual C# kunnen dergelijke delen vervangen worden door code gesteund op de .NET libraries zoals WPF, Windows Forms, Interop, ...

In het stagebedrijf ICORDA werd het al vlug duidelijk dat klanten geen veranderingen willen aan de gebruikersinterface. Vaak worden dergelijke vernieuwingen door de gebruiker als hindernissen beschouwd, omdat de klanten opnieuw met de software moeten leren werken.

Als antwoord hierop werd in het kader van deze masterproef Rc2Form ontwikkeld. Rc2Form is een applicatie die de GUI uit Visual C++ 6.0 projecten extraheert en omzet naar Visual C# Windows Forms bestanden. Deze bestanden kunnen dan opgenomen worden in een Visual C# project.

Op dit moment worden enkel `DIALOG`- en `DIALOGEX`-structuren verwerkt. Voorbeelden van `DIALOG`(`EX`)-structuren bevinden zich respectievelijk in listings 9.1 en 9.2. Om andere, gedetailleerdere structuren te parsen moeten enkel de configuratiebestanden worden aangepast. In tegenstelling tot iSE is bij Rc2Form nooit een aanpassing in de code nodig[1].

Deze structuren worden omgezet naar Windows Forms bestanden. Een Windows Form bestaat uit 3 bestanden: één bestand met broncode, één met de *code behind*[2] en één met resources. (figuur 9.1 op p.87).



Figuur 9.1: De structuur van een Windows Form.

Type bestand	Extensie	Voorbeeld
Code	.cs	Form1.cs
Code behind	.Designer.cs	Form1.Designer.cs
Resources	.resx	Form1.resx

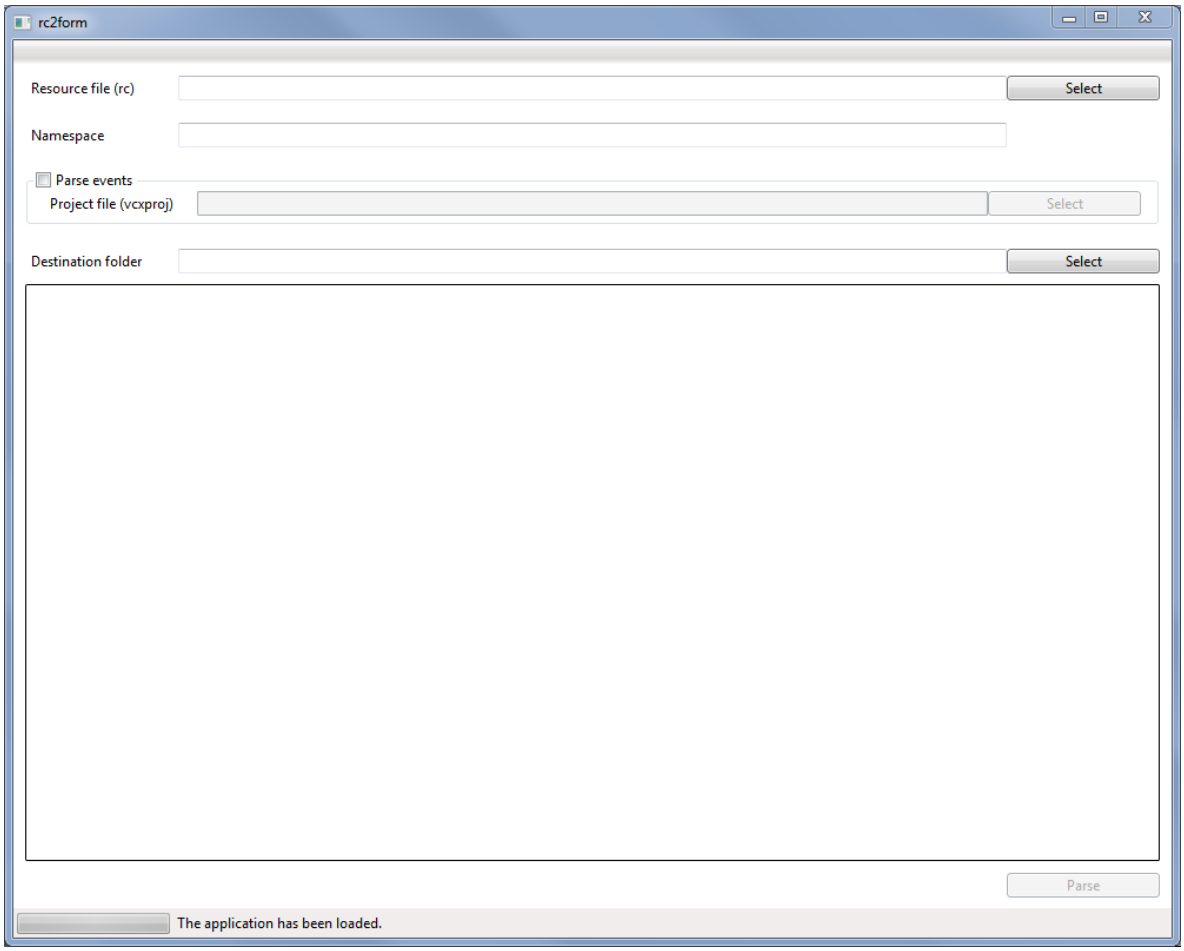
Tabel 9.1: De structuur van een Windows Form.

[1] Tenzij er een ingrijpende verandering of uitbreiding aan het programma gewenst is.
[2] De code behind is Windows Forms code dat automatisch wordt gegenereerd door Visual Studio.

Het cs-bestand bevat de code dat normaal gezien door de programmeur wordt ingevoerd. De logica achter de form en eventhandlers worden hier beschreven. Het ".Designer.cs"-bestand bevat code dat wordt gegenereerd door Visual Studio bij het opstellen en aanpassen van forms. De programmeur hoeft hier normaal gezien niet naar om te kijken, vandaar de benaming code behind. resx-bestanden bevatten bronnen voor de form. Die resx-bestanden bevatten in tegenstelling tot de andere bestanden geen Visual C#, maar XML en worden zelden manueel door de programmeur aangepast. Om het onderscheid te verduidelijken zal in het vervolg code aangeduid worden als implementatiecode en code behind als designercode.

9.1 Gebruikswijze

Het gebruik van Rc2Form is eenvoudiger dan iSE. Er is alleen een hoofdscherm (figuur 9.2 op p.88). Men moet hier enkel de nodige gegevens invoeren, op de "parse"-knop drukken en Rc2Form doet de rest. Er is daarna geen verdere tussenkomst van de gebruiker nodig. Na het indrukken van de "Parse"-knop, wordt de "Cancel"-knop beschikbaar. De "Cancel"-knop biedt de mogelijkheid het parsen te stoppen. Dit kan handig zijn indien er te laat wordt beseft dat enkele instellingen verkeerd zijn.



Figuur 9.2: Het hoofdscherm van Rc2Form.

De velden in het hoofdscherm zijn:

Resource file
Het rc-bestand dat geparset moet worden.

Namespace
De namespace waartoe de code in de gegenereerde Windows Forms bestanden moet behoren. Meestal is dit de namespace van het project waarin de bestanden zullen worden geïmporteerd.

Parse events
Indien dit wordt aangevinkt worden alle broncodebestanden uit het project doorzocht naar events verbonden aan de structuren uit het geselecteerde rc-bestand.

project file
Het vcxproj-bestand van het project waartoe het opgegeven rc-bestand hoort. Dit moet (en kan) enkel worden ingevuld indien de "Parse events"-checkbox is aangevinkt.

Destination folder
De folder waarin de gegenereerde bestanden worden opgeslagen.

Als alle nodige velden ingevuld zijn wordt de "Parse"-knop beschikbaar. Als hierop wordt gedrukt, wordt het rc-bestand geparset en worden de Windows Forms bestanden gegenereerd.

9.2 Locatie van informatie

De benodigde informatie voor het genereren van de Visual C# Windows Forms bestanden bevindt zich op verschillende plaatsen in het project. Statische informatie voor de forms bevindt zich in Visual C++ resource script bestanden (rc-bestanden), terwijl informatie over events zich in de broncodebestanden bevindt.

9.2.1 Resources

Resources worden in Visual C++ opgeslagen in rc-bestanden. Dit zijn script bestanden met een gelijkaardige syntax als C++[3]. rc-bestanden worden door de Resource Compiler gecompileerd tot res-bestanden. Resources worden in rc-bestanden vastgelegd met behulp van *resource-definition statements*. Resource-definition statements bepalen de syntax waaraan men moet voldoen om een bepaalde bron te definiëren. Volgens Microsoft (2012q) zijn er drie categorieën resource-definition statements: resources, controls en statements. Een overzicht van alle resource-definition statements bevindt zich in tabel 9.2 op p.90.

Rc2Form parset op dit moment enkel `DIALOG(EX)`-structuren als proof of concept. Door het instellen van de configuratiebestanden kunnen ook andere structuren geparset worden. Hiervoor hoeft geen code aangepast te worden (zie hoofdstuk 9.3 op p.93 voor meer informatie). Voorbeelden van `DIALOG(EX)`-structuren bevinden zich respectievelijk in listings 9.1 en 9.2.

[3]Er wordt slechts een subset aanvaard van de preprocessor directives, defines en pragmas.

Resources	Controls	Statements
ACCELERATORS	AUTO3STATE	CAPTION
BITMAP	AUTOCHECKBOX	CHARACTERISTICS
CURSOR	AUTORADIOBUTTON	CLASS
DIALOG	CHECKBOX	EXSTYLE
DIALOGEX	COMBOBOX	FONT
FONT	CONTROL	LANGUAGE
HTML	CTEXT	MENU
ICON	DEFPUSHBUTTON	MENUITEM
MENU	EDITTEXT	STYLE
MENUEX	GROUPBOX	VERSION
MESSAGETABLE	ICON	
POPUP	LISTBOX	
PLUGPLAY	LTEXT	
RCDATA	PUSHBOX	
STRINGTABLE	PUSHBUTTON	
TEXTINCLUDE	RADIOBUTTON	
TYPELIB	RTEXT	
User-Defined	SCROLLBAR	
VERSIONINFO	STATE3	
VXD		

Tabel 9.2: Een overzicht van all resource-definition statements per categorie. (Microsoft, 2012q)

```
IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 235, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About TxKassa"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT           "TxKassa Version 1.0",
                IDC_STATIC,40,10,119,8,SS_NOPREFIX
    LTEXT           "Copyright (C) 2001",IDC_STATIC,40,25,119,8
    DEFPUSHBUTTON   "OK",IDOK,178,7,50,14,WS_GROUP
END
```

Listing 9.1: Een voorbeeld van een `DIALOG`-structuur.

```
IDD_TXMAIL_MAIL_OUT_DIALOG DIALOGEX 0, 0, 271, 241
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Mail Out"
FONT 8, "MS Sans Serif", 0, 0, 0x1
BEGIN
    LTEXT           "Datum",IDC_STATIC,11,11,40,12,SS_CENTERIMAGE
    EDITTEXT        IDC_EDIT_DATUM,21,27,60,14,ES_AUTOHSCROLL |
                ES_READONLY |
                    NOT WS_TABSTOP,WS_EX_STATICEDGE
    GROUPBOX        "",IDC_STATIC,11,91,150,40,BS_FLAT
    LTEXT           "Mail from :",
                IDC_STATIC,16,107,35,12,SS_CENTERIMAGE
    COMBOBOX        IDC_COMBO_VAN_KASSA,56,107,90,84,CBS_DROPDOWN |
    LTEXT           "Mail type :",
                IDC_STATIC,16,203,35,12,SS_CENTERIMAGE
    COMBOBOX        IDC_COMBO_MAIL_TYPE,56,203,90,67,CBS_DROPDOWN |
                    WS_VSCROLL | WS_TABSTOP
```

```
18      DEFPUSHBUTTON   "F6 : Send",IDC_BUTTON_ENTER,181,27,70,14
19      PUSHBUTTON      "Esc : Cancel",IDC_BUTTON_CLOSE,181,43,70,14
20      PUSHBUTTON      "Button1",
21                  IDC_BUTTON_MAIL_OUT,191,121,50,50,BS_ICON |
22                  NOT WS_TABSTOP
23  END
```

Listing 9.2: Een voorbeeld van een `DIALOGEX`-structuur.

Voor het parsen van resources is dus enkel het rc-bestand nodig.

9.2.2 Eventhandlers

Eventhandlers gekoppeld aan resource-definition statements bevinden zich niet in het rc-bestand. De events worden daarentegen gedefinieerd in de broncodebestanden.

In het header-bestand wordt gedefinieerd over welk resource het gaat. Voor elke resource is er een constante in het project gedefinieerd met als naam de naam van de resource en als waarde een uniek nummer (listing 9.3 op p.91). Deze constantes liggen vast in *Resource.h*, een bestand dat wordt gegenereerd door Visual Studio.

```
1
2   //{{NO_DEPENDENCIES}}
3   // Microsoft Developer Studio generated include file.
4   // Used by TxKassa.rc
5   //
6   #define IDD_ABOUTBOX                    100
7   #define IDD_TXKASSA_FORM                101
8   #define IDB_SPLASH                      102
9   #define IDR_MAINFRAME                   128
10  #define IDR_TXKASSTYPE                  129
11  //...
12  #define IDD_AANWEZIGHEDEN_FORM          148
13  //...
14  #define IDS_KASTICKETPRT_PRIJS          32950
15  #define IDS_KASTICKETPRT_PRIJS_F          32951
16  #define IDS_TERUGNAMEPRT_NUMMER          32952
17  #define IDS_TERUGNAMEPRT_NUMMER_F          32953
18
19  // Next default values for new objects
20  //
21  #ifdef APSTUDIO_INVOKED
22  #ifndef APSTUDIO_READONLY_SYMBOLS
23  #define _APS_3D_CONTROLS                     1
24  #define _APS_NEXT_RESOURCE_VALUE        224
25  #define _APS_NEXT_COMMAND_VALUE         32797
26  #define _APS_NEXT_CONTROL_VALUE         1713
27  #define _APS_NEXT_SYMED_VALUE           103
28  #endif
29  #endif
```

Listing 9.3: Een voorbeeld van de definitie van een resource in het project.

De klasse waarin de logica wordt beschreven van een bepaalde resource wordt aan de resource gekoppeld door een publieke `enum` te definiëren dat de variabele `IDD` bevat. Deze variabele krijgt dan de unieke waarde van de resource (listing 9.4 op p.91).

```
1   class CDlgPostNummers : public CDialog
2   {
3   //...
4   public:
5       //{{AFX_DATA(CAanwezighedenFormView)
6       enum { IDD = IDD_AANWEZIGHEDEN_FORM };
7           // NOTE: the ClassWizard will add data members here
```

```
8       //}}AFX_DATA
9   //...
10      // Generated message map functions
11      //{{AFX_MSG(CAanwezighedenFormView)
12      afx_msg void OnButtonSave();
13      afx_msg LRESULT OnAfterWizardSwitch(WPARAM wParam, LPARAM lParam);
14      afx_msg LRESULT OnBeforeWizardSwitch(WPARAM wParam, LPARAM lParam);
15      afx_msg void OnZoek();
16      //}}AFX_MSG
17      DECLARE_MESSAGE_MAP()
18      //...
19  };
```

Listing 9.4: Een voorbeeld van een koppeling tussen een klasse en een resource in een header-bestand.

Terwijl in het header-bestand de koppeling tussen de resource en de controleklasse is vastgelegd, is in het overeenkomstige cpp-bestand de koppeling tussen de verschillende events van de control en methodes van de controleklasse vastgelegd. Deze methodes zijn dan de eigenlijke eventhandlers. Dergelijke methodes worden aangeduidt door het keyword `afx_msg` voor de signatuur van de methode in het header-bestand.

In het overeenkomstige cpp-bestand worden dan de eventhandlers beschreven. Om een event te koppelen aan een functie wordt een `MESSAGE_MAP`-structuur gebruikt (bv. listing 9.5 op p.92). `MESSAGE_MAP`-structuren zijn (en kunnen enkel aanwezig zijn) in klassen afgeleid van de klasse `CWnd`, een klasse dat een venster voorstelt. Deze structuur bevat voor elke geïmplementeerde eventhandler een lijn. Op deze lijn wordt gedefinieerd voor welk type event (message) de entry geldt, het uniek nummer van de resource dat deze event op zou werpen(vastgelegd in een constante), welke methode er moet worden opgeroepen indien dit event zich voordoet en eventueel de parameters van die methode. Zo wordt op lijn 3 in listing 9.5 op p.92 de event `BN_CLICKED`[4] vanuit de resource `IDC_BUTTON_SAVE` (een drukknop) gekoppeld aan de methode `OnButtonSave`.

```
1   BEGIN_MESSAGE_MAP(CAanwezighedenFormView, CTKFormView)
2       //{{AFX_MSG_MAP(CAanwezighedenFormView)
3       ON_BN_CLICKED(IDC_BUTTON_SAVE, OnButtonSave)
4       ON_MESSAGE (WM_WIZARD_AFTER, OnAfterWizardSwitch)
5       ON_MESSAGE (WM_WIZARD_BEFORE, OnBeforeWizardSwitch)
6       ON_COMMAND(ID_ZOEK, OnZoek)
7       //}}AFX_MSG_MAP
8   END_MESSAGE_MAP()
```

Listing 9.5: Een voorbeeld van een `MESSAGE_MAP` in een cpp-bestand.

De events worden voorgesteld door *messages*. Messages zijn constanten met een bepaalde numerieke waarden. Veel messages zijn al standaard aanwezig in MFC, maar ze kunnen ook zelf gedefinieerd worden. Omdat messages niet moeten gekoppeld worden met een resource, kunnen niet alle messages en hun gekoppelde methodes worden opgenomen in de Visual C# Windows Forms. Zonder de afzender kunnen immers de events in Windows Forms niet aan de juiste control worden gekoppeld. Voorbeeld van dergelijke `MESSAGE_MAP` entries zijn lijnen 4, 5 en 6 in listing 9.5 op p.92. Hier wordt enkel het type message gekoppeld aan een methode. Er is dus geen specifieke afzender, de message kan van overal afkomstig zijn.

[4] '`ON_BN_CLICKED(...)`' is een macro dat wordt vervangen door '`ON_CONTROL(BN_CLICKED, ...)`'

9.3 Configuratiebestanden

Rc2Form is zodanig opgesteld dat de code zo weinig mogelijk moet worden aangepast voor het implementeren van extra functionaliteit. Om dit te verwezenlijken wordt in Rc2Form intensief gebruik gemaakt van configuratie- en templatebestanden. Templatebestanden zijn bestanden die aangeven hoe de inhoud van de gegenereerde inhoud er moet uitzien. Het zijn simpele tekstbestanden waarin een aantal *markers* zijn opgenomen. Markers zijn bepaalde tekstfragmenten die bij het genereren van output worden vervangen door gegenereerde tekst. Configuratiebestanden zijn dan XML-bestanden die alle informatie voor het parsen van resources en eventhandlers bevatten, en het genereren van code dat die markers in de templatebestanden vervangen. Omdat de structuur van de configuratiebestanden redelijk complex is, wordt hier in de hoofdstukken 9.3.1 en 9.3.2 op respectievelijk p. 94 en 99 dieper op ingegaan.

FormCodeTemplate.txt
: De template die wordt gebruikt om de Windows Forms implementatiecode bestanden aan te maken. Dit zijn de bestanden met als extensie ".cs" (behalve de extensie ".Designer.cs"). Hier worden de markers `$namespace$`, `$name$` en `$events$` vervangen door respectievelijk de opgegeven namespace, de naam van de geparsete structuur en de code van de eventhandlers.

FormDesignerCodeTemplate.txt
: De template voor de Windows Forms designercode bestanden. Dit zijn de bestanden met als extensie ".Designer.cs". In deze template worden de volgende markers vervangen:

  $namespace$
  : De opgegeven namespace.

  $name$
  : De naam van de geparsete structuur.

  $declarations$
  : De declaraties overeenkomstig met de geparsete controls die horen tot de geparsete `DIALOG(EX)`-structuur.

  $initializations$
  : De initializaties overeenkomstig met de geparsete controls die horen tot de geparsete `DIALOG(EX)`-structuur.

  $controls$
  : De designercode overeenkomstig met de geparsete controls die horen tot de geparsete `DIALOG(EX)`-structuur.

  $width$
  : De breedte van de geparsete `DIALOG(EX)`-structuur.

  $height$
  : De hoogte van de geparsete `DIALOG(EX)`-structuur.

  $controlsonform$
  : De designercode die de controls overeenkomstig met de geparsete controls koppelt aan de form zelf.

$text$
De text (titel) van de geparsete `DIALOG(EX)`-structuur.

$events$
De designercode die de events van de geparsete controls of van de geparsete `DIALOG(EX)`-structuur zelf, gedefinieerd in de implementatiecode van de form, koppelt aan de events van controls aanwezig op de form.

FormResourceTemplate.txt
De template voor het resource-bestand. Dit bestand heeft als extensie ".resx" en is voor elke gegenereerde form hetzelfde[5].

ParserConfiguration.xml
Het configuratiebestand dat alle gegevens voor het parsen van het resourcebestand bevat in XML-formaat. In hoofdstuk 9.3.2 op p.99 komt de inhoud uitgebreid aan bod.

ParserEventsConfiguration.xml
Het configuratiebestand dat alle gegevens voor het parsen van eventhandlers in het project in in XML-formaat. In hoofdstuk 9.3.2 op p.99 komt de inhoud uitgebreid aan bod.

Deze bestanden bevinden zich in de folder "ConfigurationFiles". Men kan de relatieve locatie ten opzichte van *Rc2Form.exe* wijzigen door in de *App.config* van het Rc2Form-project de waarden in het attribuut `value` in de `add`-tags aan te passen (listing 9.6 op p.94).

```
<?xml version="1.0"?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0"
                          sku=".NETFramework,Version=v4.0"/>
    </startup>
    <appSettings>
        <add key="ParserConfigurationPath"
             value="Configurationfiles\ParserConfiguration.xml"/>
        <add key="FormCodeTemplatePath"
             value="Configurationfiles\FormCodeTemplate.txt"/>
        <add key="FormDesignerCodeTemplatePath"
             value="Configurationfiles\FormDesignerCodeTemplate.txt"/>
        <add key="FormResourceTemplatePath"
             value="Configurationfiles\FormResourceTemplate.txt"/>
        <add key="ParserEventsConfigurationPath"
             value="Configurationfiles\ParserEventsConfiguration.xml"/>
    </appSettings>
</configuration>
```

Listing 9.6: De inhoud van *App.config* in het Rc2Form-project.

9.3.1 ParserConfiguration.xml

De XML-structuur van *ParserConfiguration.xml* bestaat uit 2 grote delen: `Controls` en `Properties` (listing 9.7 op p.95). Deze opdeling biedt de grootste hergebruikbaarheid van de verschillende onderdelen en verhindert redundante code. Aanpassingen hoeven zo vaak slechts op een enkele plaats te gebeuren en bij het invoegen van nieuwe controls kan men vaak reeds de aanwezige informatie over properties gebruiken.

[5]Op dit moment worden er immers nog geen resources behalve `DIALOG(EX)`-structuren geparset

```
<ParserConfiguration>
  <Controls>
    <!-- ... -->
  </Controls>
  <Properties>
    <!-- ... -->
  </Properties>
</ParserConfiguration>
```

Listing 9.7: De structuur van een het configuratiebestand *ParserConfiguration.xml*.

Properties

De `Propeties`-tag is het tweede gedeelte van het configuratiebestand. Hier bevindt zich in elke `Property`-tag de informatie over een property. De structuur van een `Property`-tag bestaat uit verschillende delen (listing 9.8 op p.95):

```
<Property name="DropDownStyle">
    <Depends property="Name" />
    <Depends property="DropDownStyle" />
    <Code>
        <![CDATA[
            this.$name$.DropDownStyle = $dropdownstyle$;
        ]]>
    </Code>
    <Marker>
        <![CDATA[
            $dropdownstyle$
        ]]>
    </Marker>
    <PossibleValues>
        <Value content="CBS_DROPDOWNLIST">
            <![CDATA[
                System.Windows.Forms.ComboBoxStyle.DropDownList
            ]]>
        </Value>
        <Value content="CBS_DROPDOWN">
            <![CDATA[
                System.Windows.Forms.ComboBoxStyle.DropDown
            ]]>
        </Value>
    </PossibleValues>
</Property>
```

Listing 9.8: De structuur van een `Property`-tag.

`Depends`
: Hier wordt verwezen naar de nodige waarden van properties om de markers in de `Code`-tag te kunnen vervangen, zodat alle markers zijn vervangen.

`Code`
: De designercode die de overeenkomstige Visual C# property instelt.

`Marker`
: De marker die wordt geassocieerd met deze variable en wordt gebruikt in de `Code`-tags van eigen en andere `Property`-tags.

`PossibleValues`
: Soms is de waarde van de property geen getal of string, maar een keuze uit vaste

waarden. Indien dit het geval is wordt er voor elke mogelijke waarde een `Value`-tag gedefinieerd in de `PossibleValues`-tag. Het `content`-attribuut duidt aan welke ingelezen waarde overeenkomt met deze `Value`-tag. De inhoud van de `Value`-tag is het Visual C# designercode equivalent. De `PossibleValues`-tag is optioneel en er staat geen limiet op het aantal `Value`-tags dat het bevat.

Het `name`-attribuut in de `Property`-tag is het attribuut naar waar wordt verwezen vanuit het eerste gedeelte van het configuratiebestand.

Controls

De `Controls`-tag is het eerste gedeelte van het configuratiebestand. Een typische control ziet er uit als in listing 9.9 op p.96. Er zijn drie delen te onderscheiden: `Regex`-, `Options`-, `Code`-tags. De `Options`-tag is optioneel.

```
<Control type="ComboBox">
    <Regex>
        <Expression>
            <![CDATA[
                ^\s*(?<type>COMBOBOX)\s+(?<name>\w+)\s*, ...
            ]]>
        </Expression>
        <NameCaption name="name" property="Name" />
        <OptionsCaption name="options" />
        <Caption name="left" property="Left" />
        <Caption name="top" property="Top" />
        <Caption name="width" property="Width" />
        <Caption name="height" property="Height" />
    </Regex>
    <Options>
        <Option>
            <Regex>
                <Expression>
                    <![CDATA[
                        (?<option>CBS_DROPDOWNLIST)
                    ]]>
                </Expression>
                <Caption name="option" property="DropDownStyle" />
            </Regex>
        </Option>
    </Options>
    <Code>
        <Declaration>
            <![CDATA[
                private System.Windows.Forms.ComboBox $name$;
            ]]>
        </Declaration>
        <Initialization>
            <![CDATA[
                this.$name$ = new System.Windows.Forms.ComboBox();
            ]]>
        </Initialization>
        <CodeWithoutParameters>
            <![CDATA[
                //
                // $name$
                //
            ]]>
        </CodeWithoutParameters>
        <Properties>
            <Property name="Location"/>
            <Property name="Name"/>
```

```
48              <Property name="Size"/>
49              <Property name="DropDownStyle" />
50          </Properties>
51      </Code>
52  </Control>
```

Listing 9.9: De structuur van een `Control`-tag.

De `Regex`-tag bevat alle informatie om controls uit `DIALOG(EX)`-structuren te parsen. In de `Expression`-tag bevindt zich een reguliere expressie die beantwoord aan de opmaak van een control in `DIALOG(EX)`-structuren. Daarnaast zijn er ook nog `Caption`- en `NameCaption`-tags. Deze tags duiden aan welke named capture group uit de reguliere expressie (het `name`-attribuut) informatie bevat voor welke property (het `property`-attribuut). Het `property`-attribuut komt overeen met het `name`-attribuut van de overeenkomstige `Property`-tag in het `Properties`-gedeelte van het configuratiebestand. De `NameCaption`-tag is een speciale variant van de `Caption`-tag. De syntax en functionaliteit is hetzelfde, maar de `NameCaption`-tag is verplicht en dient uitsluitend om aan te duiden waar de naam zich bevindt in de reguliere expressie. De `NameCaption`-tag kan in plaats van het `name`-attribuut ook een `standard`-attribuut bevatten. Dit attribuut bevat dan een standaardnaam die wordt gebruikt indien de named capture group voor de naam leeg zou zijn. Een ander speciaal geval is de `OptionsCaption`-tag. Soms kunnen er aan de controls in het rc-bestand opties worden meegegeven. De `OptionsCaption`-tag geeft aan welke named capture group de opties voor de control bevat.

Om de verschillende opties in de opties named capture group te kunnen detecteren kan men voor elke optie in de `Options`-tag een `Option`-tag voorzien. Deze bevat een `Regex`-tag analoog aan de `Regex`-tag hierboven. Dit laat toe om extra informatie uit de control structuur uit het rc-bestand te extraheren. De `Options`-tag is optioneel en kan meerdere `Option`-tags bevatten.

Terwijl de gegevens in de `Regex`-tag aanduiden hoe men data kan extraheren uit het resourcebestand, duiden de gegevens in de `Code`-tag aan hoe men data kan genereren voor output. De `Code`-tag bestaat op zijn beurt uit verschillende tags:

`Declaration`
: Het codefragment dat de Visual C# variant van de control declareert.

`Initialization`
: Het codefragment dat de Visual C# variant van de control initialiseert.

`CodeWithoutParameters`
: Het codefragment dat in elk geval moet worden verwerkt in de gegenereerde output. Dit fragment mag enkel de marker `$name$` bevatten, omdat de naam van de control dankzij de verplichte `NameCaption`-tag sowieso wordt opgehaald.

`CodeBeforeFragment`
: Soms is het nodig om voor bepaalde stukken code in de resulterende output tekst in te voegen. Om hieraan tegemoet te komen is het mogelijk om een `CodeBeforeFragment` te gebruiken (listing 9.10 op p.97). Dit bestaat uit twee delen: `Fragment`, het stuk tekst in de resulterende output waarvoor er moet ingevoegd worden, en `Code`, de tekst die moet worden ingevoegd. `CodeBeforeFragment`-tags zijn optioneel en hebben geen limiet op het aantal..

```
1  <CodeBeforeFragment>
```

```
    <Fragment>
        <![CDATA[
            this.SuspendLayout();
        ]]>
    </Fragment>
    <Code>
        <![CDATA[
            ((System.ComponentModel.ISupportInitialize)
            (this.$name$)).BeginInit();
        ]]>
    </Code>
</CodeBeforeFragment>
```

Listing 9.10: De structuur van een `CodeBeforeFragment`-tag.

`CodeAfterFragment`
: `CodeAfterFragment`-tags hebben een analoge betekenis en structuur als de `CodeBeforeFragment`-tags. Alleen voegen `CodeAfterFragment`-tags in ná bepaalde tekstfragmenten, in plaats van ervóór zoals bij `CodeBeforeFragment`-tags. Ook `CodeAfterFragment`-tags zijn optioneel en hebben geen limiet op het aantal.

`Properties`
: De `Property`-tags in deze tag bepalen welke properties moeten worden opgenomen in de gegenereerde output. Het `name`-attribuut verwijst naar het `name`-attribuut van de overeenkomstige `Property`-tag in het `Properties`-gedeelte van het configuratiebestand. De waarden waarop de overeenkomstige properties steunen voor het genereren van output moeten allemaal ingelezen worden via de reguliere expressie, zoniet wordt er geen output voor die propertie gegenereerd.

Voor alle `Control`-tags komt er eerst een speciaal geval, namelijk de `Form`-tag (listing 9.11 op p.98). Deze tag omvat alle informatie om een `DIALOG(EX)`-structuur te parsen. De structuur van de `Form`-tag komt grotendeels overeen met die van een gewone `Control`-tag, maar toch zijn er een aantal verschillen. De `Regex`-tag heeft naast de `NameCaption`-tag en de `OptionsCaption`-tag ook nog een andere verplichte tag: de `ControlsCaption`-tag. Deze duiden de named capture groups in de reguliere expressies aan die respectievelijk de opties en controls van de `DIALOG(EX)`-structuur bevatten. De `Code`-tag bevat slechts een enkele tag, de `Child`-tag. Deze tag bevat de Visual C# code om de Visual C# varianten van geparsete controls te koppelen aan de Visual C# form die overeenstemt met de geparsete `DIALOG(EX)`-structuur.

```
<Form>
    <Regex>
        <Expression>
            <![CDATA[
                ^\s*(?<name>\w+)\s+(?<type>DIALOG(EX)?) ...
            ]]>
        </Expression>
        <NameCaption name="name" property="Name" />
        <OptionsCaption name="options" />
        <ControlsCaption name="controls" />
        <Caption name="left" property="Left" />
        <Caption name="top" property="Top" />
        <Caption name="width" property="Width" />
        <Caption name="height" property="Height" />
    </Regex>
    <Options>
        <Option>
```

```
            <Regex>
                <Expression>
                    <![CDATA[
                        CAPTION\s+"(?<caption>.*?)"
                    ]]>
                </Expression>
                <Caption name="caption" property="Text" />
            </Regex>
        </Option>
    </Options>
    <Code>
        <Child>
            <![CDATA[
                this.Controls.Add(this.$child$);
            ]]>
        </Child>
    </Code>
</Form>
```

Listing 9.11: De structuur van de `Form`-tag.

9.3.2 ParserEventsConfiguration.xml

Het configuratiebestand *ParserEventsConfiguration.xml* bevat alle informatie voor het parsen en verwerken van eventhandlers in het project. De structuur van dit bestand is eenvoudiger dan dat van *ParserConfiguration.xml*. Voorafgaand de informatie voor eventhandlers is er een `FormNameMarker`-tag (listing 9.12 op p.99). Deze tag bevat een marker die gebruikt wordt voor het genereren van de Visual C# varianten van eventhandlers, gekoppeld aan de `DIALOG(EX)`-structuur zelf. De marker wordt dan vervangen door de naam van de overeenkomstige `DIALOG(EX)`-structuur.

```
<FormNameMarker>
    <![CDATA[
        $formname$
    ]]>
</FormNameMarker>
```

Listing 9.12: De `FormNameMarker`-tag.

Daarna komt de `Events`-tag. Deze tag bevat `Event`-tags die op hun beurt informatie bevatten voor het parsen en verwerken van bepaalde eventhandlers. De structuur van een dergelijke `Event`-tag ziet er uit als in listing 9.13 op p.99.

```
<Event>
    <Regex>
        <Expression>
            <![CDATA[
                ON_EN_KILLFOCUS\(\s*(?<name>.*?)\s*,\s*(?<method>.*?)\s*\)
            ]]>
        </Expression>
        <NamedCaption name="name" marker="$name$" />
        <Caption name="method" marker="$method$" />
    </Regex>
    <Regex>
        <Expression>
            <![CDATA[
                ON_CBN_KILLFOCUS\(\s*(?<name>.*?)\s*, ...
            ]]>
        </Expression>
        <NamedCaption name="name" marker="$name$" />
        <Caption name="method" marker="$method$" />
    </Regex>
```

```
20      <!-- ... -->
21      <ImplementationCode>
22          <![CDATA[
23              private void $name$_Leave(object sender, EventArgs e)
24          ]]>
25      </ImplementationCode>
26      <DesignerCode>
27          <![CDATA[
28              this.$name$.Leave += new System.EventHandler(this.$name$_Leave);
29          ]]>
30      </DesignerCode>
31  </Event>
```

Listing 9.13: De structuur van een `Event`-tag.

Een `Event`-tag bestaat uit drie soorten tags:

`Regex`
: De structuur en het doel van deze tag zijn analoog aan de `Regex`-tags in *ParserConfiguration.xml*. De reguliere expressie dient om te detecteren of het om deze eventhandler gaat en om informatie te extraheren. Deze tag kan meerdere keren voorkomen. Er zijn wel enkel verschillen met de `Regex`-tags in *ParserConfiguration.xml*. Ten eerste is de `NameCaption`-tag niet verplicht. Eventhandlers die invloed hebben op de `DIALOG(EX)`-structuur zelf zullen deze niet implementeren, er is immers geen naam van een control aanwezig in de overeenkomstige entry in de `MESSAGE_MAP`-structuur. Ten tweede is er geen `property`-attribuut, maar wel een `marker`-attribuut. Het `marker`-attribuut geeft aan welke markers in de `ImplementationCode`- en `DesignerCode`-tags, in dezelfde `Event`-tag, moeten vervangen worden door welke named capture groups in de reguliere expressie[6]. Ten slotte is de `Event`-tag de enige in beide configuratiebestanden tag die meerdere `Regex`-tags kan bevatten.

`ImplementationCode`
: De Visual C# implementatiecode overeenkomstig met de eventhandler. Deze kan enkel markers bevatten die zijn opgegeven in elke `Regex`-tag van de `Event`-tag.

`DesignerCode`
: De Visual C# designercode overeenkomstig met de eventhandler. Ook deze kan enkel markers bevatten die zijn opgegeven in elke `Regex`-tag van de `Event`-tag.

Door de aard van het MFC messages systeem kunnen meerdere MFC eventhandlers worden vertaald naar hetzelfde type eventhandler in Visual C#. Omdat de aanwezige informatie in de `MESSAGE_MAP`-lijn voor bijna alle eventhandlers overeenkomt kan men al deze Visual C++ eventhandlers samennemen in een enkele `Event`-tag. De enige voorwaarde is dat de nodige named capture groups in de `ImplementationCode`- en `DesignerCode`-tags in elke reguliere expressie aanwezig zijn.

9.3.3 Opmerkingen

Codefragmenten en reguliere expressies in de configuratiebestanden met XML-structuur worden steeds tussen `CDATA`-tags geplaatst. Zo worden tekens als “<”, “>”, “`&`” en “`%`” niet

[6] De enige mogelijke waarden van markers zijn strings of numerieke waarden. Er moet dus niet met properties gewerkt worden om andere variabelentypes te ondersteunen, zoals in *ParserConfiguration.xml*.

opgevat als XML en kunnen er geen ongewenste fouten optreden. Het heeft als nadeel dat informatie uit `CDATA`-tags opgehaald via XPath-expressies nog moeten worden getrimd op de karakters "\t" (tab), "\n" (newline), "\r" (carriage return), " " (spatie) en al hun mogelijke combinaties. Om dit te vereenvoudigen wordt gebruik gemaakt van een extensie methode zoals in listing 9.14 op p.101.

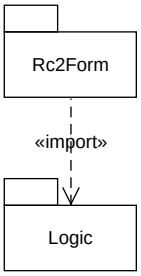
```
1 internal static class StringExt
2 {
3     public static string TrimEscape(this string text)
4     {
5         return text.Trim('\t', '\n', '\r', ' ');
6     }
7 }
```

Listing 9.14: Een extensie methode op het type `String` voor het trimmen van bepaalde karakters.

9.4 Achter de schermen

9.4.1 Overzicht

Rc2Form bestaat uit 2 projecten: Rc2Form en Logic (figuur 9.3 op p.101). In tegenstelling tot iSE is hier geen Data project aanwezig. Alle Input/Output (IO)-methodes komen immers uit het .NET framework. Er zijn dus geen zelfgeschreven methodes nodig.



Figuur 9.3: De verschillende projecten in Rc2Form.

Rc2Form
: Het project Rc2Form bevat de GUI van het project. Net als iSE maakt de GUI gebruik van WPF. De opbouw van de GUI is analoog aan iSE. Ook hier worden er methodes aan events van het project Logic gekoppeld om real time uitvoer te verkrijgen.

Logic
: Het project Logic bevat alle methodes om het rc-bestand te verwerken. Het bevat ook methodes om meer informatie uit het project zelf te halen (Dit is nodig om events gekoppeld aan structuren in het rc-bestand te detecteren en te verwerken, zie voor meer informatie hoofdstuk 9.2.2 op p.91).

De gehele applicatie steunt op een enkele datastructuur: `Dictionary<string, Logic.FormInformation> forms`. Deze structuur is de ruggengraat van Rc2Form. Het

zal aan alle methodes worden meegegeven als een by reference waarde en telkens worden aangevuld. Uiteindelijk wordt het meegegeven aan de methode verantwoordelijk voor het genereren van de bestanden dat er dan alle nodig informatie zal uithalen.

Dankzij deze structuur kunnen de methodes voor het vergaren van informatie over resources en het vergaren van informatie over events onafhankelijk van elkaar in een willekeurige volgorde worden uitgevoerd. In eerdere versies werd er nog geen gebruik gemaakt van een centrale figuur en berustten alle methodes op elkaar. De volgorde van functies lag dus vast en er was geen mooie splitsing tussen de verschillende onderdelen van de applicatie. Als er ergens aanpassingen nodig waren, liet dit zich voelen doorheen de gehele applicatie. Door het ontkoppelen van de methodes werd het makkelijker om functionaliteit toe te voegen of aan te passen.

9.4.2 De centrale structuur

De centrale datastructuur heeft als vorm `Dictionary<string, Logic.FormInformation>`. De sleutels zijn de namen van de overeenkomstige `DIALOG(EX)`-structuren (en de op basis daarvan gegenereerde Visual C# Windows Forms). Omdat `DIALOG(EX)`-structuren een unieke naam moeten hebben in het originele Visual C++ project, is het zeker dat er geen conflicten zullen optreden door meerdere entries in de `Dictionary` met eenzelfde sleutel. De waarden zijn instanties van de `FormInformation`-klasse in het project Logic. Elke instantie van deze klasse bevat alle gegenereerde informatie voor een bepaalde Windows Form overeenkomstig met een bepaalde `DIALOG(EX)`-structuur. De structuur van `FormInformation` wordt weergegeven in figuur 9.4 op p.104.

De leden van de klasse `FormInformation` bevatten de informatie van de Visual C# form zelf. `name` bevat de naam van de form, `formEventsCode` de gegenereerde implementatie- en designercode voor eventhandlers van de form zelf (niet van controls op de form) en `formCode` de gegenereerde code voor de form zelf (bijvoorbeeld voor properties zoals positie van de form). De methode `HasFormCode` geeft aan of er gegenereerde code is opgeslagen voor de form in `formCode`[7].

Om bij te houden welke (gegenereerde) Visual C# controls behoren tot de form, is er in de klasse ook een veld `controls` aanwezig. Dit veld is analoog aan de centrale structuur. Het is een `Dictionary<string, ControlInformation>` waarin de sleutel de naam van de geparsete control (en dus ook de naam van de daaruit gegenereerde Visual C# control) bevat. De waarde is een `ControlInformation`-object.

Het `ControlInformation`-object bevat alle gegenereerde code voor een Visual C# variant van een geparsete control in de `DIALOG(EX)`-structuur overeenkomstig met de `FormInformation`-instantie waartoe het `ControlInformation`-object behoort. Net als bij de klasse `FormInformation` bevat de klasse `ControlInformation` velden voor het opslaan van informatie voor de Visual C# control. `name` bevat de naam van de control, `declaration` alle gegenereerde code voor het declareren van de control en `initialization` bevat dan alle gegenereerde code voor het initialiseren van de control. De gegenereerde code voor het instellen van properties voor de control bevindt zich in het veld `code`. De implementatie- en designercode voor eventhandlers op de control worden opgeslagen in `controlEventsCode`. Gegenereerde codefragmenten die voor of achter bepaalde codefragmenten moeten worden ingevoegd in de gegenereerde bestanden, worden dan respectievelijk opgeslagen in de velden

[7] Er wordt dus niet gekeken naar eventuele gegenereerde code voor eventhandlers in `formEventsCode`.

`codeBeforeFragments` en `codeAfterFragments`. Er zijn ook nog twee methodes aanwezig: `HasControlCode` en `HasEventsCode`. `HasControlCode` kijkt na of er gegenereerde code aanwezig is in het veld `code`. `HasEventsCode` gaat na of er gegenereerde code voor eventhandlers op de control aanwezig is in het veld `ControlEventsCode`.

De volledige structuur van de `FormInformation`- en `ControlInformation`-klasse en hun relaties wordt weergegeven in figuur 9.4 op p.104.

9.4.3 Opstarten

In tegenstelling tot iSE worden de configuratiebestanden hier nog niet ingelezen. Dit gebeurt pas in de methodes zelf voor het parsen van rc-bestanden, de eventhandlers of het uitschrijven. Dit komt door de complexe structuur en het feit dat men zelden echt alle informatie uit de configuratiebestanden nodig heeft. Op grote schaal is dit duidelijk bij het parsen van eventhandlers. Indien men dit niet wenst hoeft het bestand *ParserEventsConfiguration.xml* niet te worden ingelezen. Ook in configuratiebestanden die wel worden gebruikt wordt vaak niet alle aanwezige informatie aangewend.

Bij het opstarten wordt slechts een enkele belangrijke taak vervuld en dat is het aanmaken van de centrale datastructuur: `Dictionary<string, Logic.FormInformation> forms`.

9.4.4 Het parsen van eventhandlers

Voor het parsen van eventhandlers wordt er gebruikt gemaakt van de klasse `HeaderParser`. Deze klasse bevat slechts één publieke methode, namelijk `ParseHeaders` (listing 9.15 op p.103).

```
1 ParseHeaders(ref Dictionary<string, FormInformation> forms)
```

Listing 9.15: De `ParseHeaders`-methode.

Het ophalen van informatie

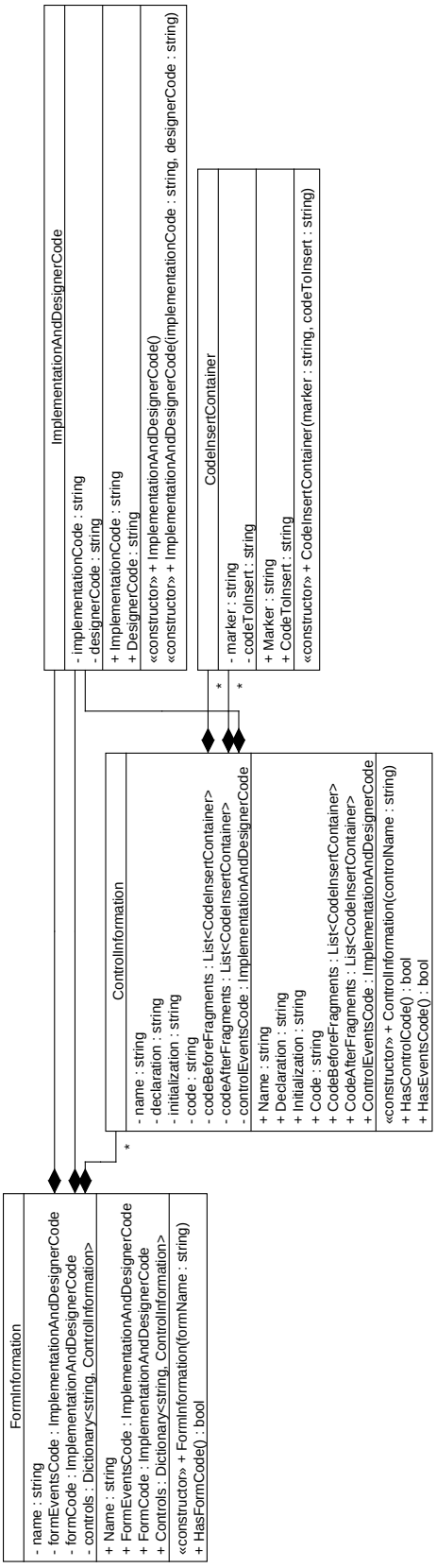
Bij de creatie van een `HeaderParser`-instantie wordt er een `XmlDocument`-object aangemaakt op basis van het meegegeven vcxproj-bestand. `XmlDocument` is een .NET klasse dat toelaat om informatie uit XML-structuren te halen via XPath-expressies. Omdat er in vcxproj-bestanden gebruik gemaakt wordt van de default XML namespace "`http://schemas.microsoft.com/developer/msbuild/2003`" moet dit ook worden doorgegeven aan het `XmlDocument`. Dit gebeurd via de `XmlNameSpaceManager` (listing 9.16 op p.103).

```
1 XmlDocument vcxprojXmlDoc = new XmlDocument();
2 vcxprojXmlDoc.Load(projectPath);
3 XmlNamespaceManager xmlNamespaceManager = new
      XmlNamespaceManager(vcxprojXmlDoc.NameTable);
4 xmlNamespaceManager.AddNamespace("rs",
      "http://schemas.microsoft.com/developer/msbuild/2003");
```

Listing 9.16: Het aanmaken van een `XmlDocument` dat een vcxproj-bestand vertegenwoordigt.

Bij elke XPath bewerking op het XmlDocument waarin een element van de namespace voorkomt, moet de `XmlNameSpaceManager` meegegeven worden als extra parameter[8] (listing 9.17 op p.105).

[8] Omdat het in het vcxproj-bestand om een default namespace gaat, behoren alle elementen van het `XmlDocument` tot de namespace.



Figuur 9.4: De structuur van de klasse `FormInformation`. De klasseweergaven bevatten in elk deel van boven naar onder de attributen, properties en functies. Om het schema overzichtelijk te houden worden de verbanden van properties niet weergegeven omdat deze analoog zijn aan die van de attributen.

```
XmlNodeList headersInformation =
    vcxprojXmlDoc.SelectNodes("/rs:Project/rs:ItemGroup/rs:ClInclude",
    xmlNamespaceManager);
```

Listing 9.17: Een voorbeeld van een XPath expressie met elementen die behoren tot de `rs`-namespace in listing 9.16 op p.103.

Naast het vcxproj-bestand wordt ook het configuratiebestand *ParserEventsConfiguration.xml* ingeladen als een `XmlDocument`-object. Uit het configuratiebestand wordt dan de inhoud van de `FormNameMarker`-tag opgehaald.

In het vcxproj-bestand worden alle header-bestanden opgehaald die deel uitmaken van het project dat het vcxproj-bestand vertegenwoordigt. Voor elke header bestaat er een `ClInclude`-tag (De XPath-expressie naar een dergelijke `ClInclude`-tag bevindt zich in listing 9.17 op p.105). Het `Include`-attribuut van de `ClInclude`-tag bevat dan de locatie van het header-bestand relatief ten opzichte van het vcxproj-bestand (listing 9.18 op p.105).

```
<ClInclude Include="AanwezighedenFormView.h" />
```

Listing 9.18: Een voorbeeld van een `ClInclude`-tag.

De inhoud van elk gedetecteerd header-bestand wordt *gematcht*[9] met de reguliere expressie in listing 9.19 op p.105. Als de match succesvol blijkt te zijn, wil dit zeggen dat de klasse gedefinieerd in de header overeenkomt met een `DIALOG(EX)`-structuur. De named capture group `dialogname` bevat dan de naam van de `DIALOG(EX)`-structuur.

```
/enum\s*{\s*IDD\s*=\s*(?<dialogname>.*?)\s*}/ms
```

Listing 9.19: De reguliere expressie voor het detecteren naar IDD `enum`-instanties in header-bestanden.

Als de match succesvol is wordt het overeenkomstige cpp-bestand ingelezen[10] en doorzocht op de aanwezigheid van een `MESSAGE_MAP`-structuur. Ook hier wordt er gebruikt gemaakt van een reguliere expressie (listing 9.20 op p.105).

```
/
BEGIN_MESSAGE_MAP\(\s*(?<class>\w+)\s*,\s*(?<baseclass>\w+)\s*\)
    (?<events>.*?)
END_MESSAGE_MAP\(\)
/xms
```

Listing 9.20: De reguliere expressie voor het detecteren van `MESSAGE_MAP`-structuren in cpp-bestanden.

Beide reguliere expressies krijgen de opties `RegexOptions.Multiline` en `RegexOptions.Singleline` mee:

`RegexOptions.Multiline`
: De operatoren `^` en `$` wijzen op het begin en eind van elke lijn in de tekst waarmee wordt gematcht, in plaats van enkel het begin en eind van de tekst.

[9] Matchen staat hier voor het vergelijken met een reguliere expressie. Indien de match succesvol is, betekent dit dat de waarde overeenkomt met de reguliere expressie en de reguliere expressie eventueel bepaalde delen van de waarde heeft opgehaald.

[10] Er wordt hierbij uitgegaan dat cpp- en header-bestanden in dezelfde folder staan. De locatie van de cpp-bestanden wordt dus nergens ingelezen of opgehaald, maar simpelweg bekomen door de ".h"-extensie van de locatie van de header-bestanden te veranderen naar de ".cpp"-extensie.

`RegexOptions.Singleline`
: De operator `.`(dot) matcht nu ook het newline karakter "`\n`".

Deze opties zorgen ervoor dat de reguliere expressies een zo groot mogelijke vorm van vrijheid hebben en niet als beperkend worden ervaren. Ze zorgen er ook voor dat er bij het opstellen van de reguliere expressies geen rekening moet gehouden worden met wat er precies op een enkele lijn staat. Aangezien de aard van Visual C++ kunnen de `enum`- en `MESSAGE_MAP`-structuren over zoveel lijnen worden uitgespreid als de programmeur dat wil.

Om overhead te vermijden, wordt eerst nagegaan of er nuttige informatie in de `MESSAGE_MAP`-structuur zit en dus niet enkel bestaat uit comments[11] of whitespace[12].

De naam van de overeenkomstige `DIALOG(EX)`-structuur werd afgeleid via listing 9.19 op p.105. Nu moet enkel de inhoud van de named capture group `events` uit listing 9.20 op p.105 verder worden onderzocht op aanwezige bruikbare eventhandlers.

Om informatie gegenereerd tijdens het parsen op te slaan is er een instantie van de `FormInformation`-klasse nodig. Er wordt nagegaan of er al een instantie in de centrale structuur aanwezig is voor de te verwerken `DIALOG(EX)`-structuur, waarmee het header- en cpp-bestand overeenkomen. Als dit het geval is wordt de reeds aanwezige `FormInformation`-instantie aangevuld, anders wordt er een nieuw `FormInformation`-object aangemaakt. In het laatste geval moet dit object op het eind van de methode worden ingevoegd in de centrale structuur.

Het verwerken van de opgehaalde informatie

Voor het verwerken van de inhoud van de `events` named capture group wordt intensief gebruik gemaakt van het *ParserEventsConfiguration.xml* configuratiebestand. Er wordt voor elke reguliere expressie van elke `Event`-tag in het configuratiebestand nagegaan of er positieve matches zijn in de `events` named capture group.

Indien er een positieve match optreedt worden de `ImplementationCode`- en `DesignerCode`-tags ingelezen. Dan wordt voor alle aanwezige `Caption`-tags de markers (aangeduid door het `marker`-attribuut) in de ingelezen waarden vervangen door de waarden van de overeenkomstige named capture groups in de reguliere expressie (aangeduid door het `name`-attribuut). Indien er een `NamedCaption`-tag aanwezig is wordt hetzelfde gedaan voor de `NamedCaption`-tag. De aanwezigheid van de `NamedCaption`-tag geeft weer dat het niet gaat om een eventhandler op de `DIALOG(EX)`-structuur zelf. Uiteindelijk wordt ook nog de marker, opgehaald uit de `FormNameMarker`-tag, vervangen door de naam van de `DIALOG(EX)`-structuur, dat reeds werd opgehaald via de reguliere expressie in listing 9.19 op p.105.

Als het om een eventhandler op een control gaat, moet de informatie worden opgeslagen in een `ControlInformation`. Ook hier wordt nagegaan of er al een `ControlInformation`-object voor de control aanwezig is in het `FormInformation`-object[13]. Zo ja, dan worden de ingelezen en verwerkte waarden opgeslagen in het `ControlInformation`-object. Zo nee, dan wordt de

[11] Op dit moment worden comments in de vorm '`/* ... */`' nog niet als comments beschouwd wegens hun complexe aard. Enkel comments in de vorm '`// ...`' worden als niet zinvolle informatie beschouwd.

[12] Deze controle wordt dus uitgevoerd op de inhoud van de `events` named capture group in listing 9.20 op p.105

[13] Dit is mogelijk omdat de `NamedCaption`-tag ervoor zorgt dat de naam van de control kan achterhaald worden.

informatie opgeslagen in een nieuwe instantie van de `ControlInformation`-klasse. Daarna wordt de nieuwe instantie toegevoegd aan het `FormInformation`-object.

De informatie wordt opgeslagen in het `ImplementationCode`- en `DesignerCode`-veld van het `ControlInformation`- of `FormInformation`-object.

Na het verwerken wordt ook het `FormInformation`-object ingevoegd in de centrale structuur, indien dit een nieuw aangemaakte instantie is dat nog niet in de centrale structuur zit.

9.4.5 Het parsen van het rc-bestand

Net zoals er voor het parsen van header- en cpp-bestanden naar eventhandlers een klasse bestaat, bestaat er voor het parsen van het rc-bestand zelf een klasse met alle nodige methodes: de `formParser`-klasse. De klasse bevat analoog aan de `HeaderParser`-klasse een enkele publieke methode, namelijk `ParseDialogs`.

Ook bij de creatie van een `formParser`-instantie wordt er gebruik gemaakt van de .NET klasse `XmlDocument`. Hier wordt het XML-bestand *ParserConfiguration.xml* ingelezen. Voor elke `Property`-tag wordt de overeenkomstige marker in de `Marker`-tag en de naam in het `name`-attribuut als koppel opgeslagen in een '`Dictionary<string, string>`'-structuur. Het koppel wordt enkel opgeslagen indien er een `Marker`-tag aanwezig is. Sommige properties hebben geen marker omdat hun waarde wordt samengesteld door andere waarden in Visual C# Windows Forms. Een voorbeeld is listing 9.21 op p.107.

```
1 this.$name$.Location = new System.Drawing.Point($left$, $top$);
```

Listing 9.21: Een voorbeeld van de Visual C# code van een samengestelde property.

Er is geen marker nodig voor de property `Location`, want de markers `$left$` en `$top$` zijn voldoende en al gedefinieerd.

In tegenstelling tot de `HeaderParser`-constructor worden hier ook twee templatebestanden ingeladen: *FormDesignerCodeTemplate.txt* en *FormCodeTemplate.txt*.

Het parsen van `DIALOG(EX)`-structuren

Het rc-bestand wordt volledig gematcht met de reguliere expressie in de `Regex`-tag in de `Form`-tag van het configuratiebestand listing 9.22 op p.107.

```
1  /
2  ^\s*(?<name>\w+)
3  \s+(?<type>DIALOG(EX)?)
4  \s+(?<left>\d+)\s*,
5  \s*(?<top>\d+)\s*,
6  \s*(?<width>\d+)\s*,
7  \s*(?<height>\d+)
8  \s*(?<options>.*?)
9  (^\s*BEGIN|{)
10 (?<controls>.*?)
11 (^\s*END|})
12 /xms
```

Listing 9.22: De reguliere expressie voor het detecteren van `DIALOG(EX)`-structuren.

Elke gevonden match wordt verder verwerkt. Allereerst wordt de naam van de `DIALOG(EX)`-structuur opgehaald uit de reguliere expressie[14].

[14] Dankzij de `NameCaption`-tag bij de regex is de bekend. Indien de naam leeg blijkt te zijn wordt de waarde van het `standard`-attribuut in de `NameCaption`-tag gebruikt.

Net als in de `ParseHeaders`-methode wordt er nagegaan of de centrale, structuur reeds een `FormInformation`-object heeft voor deze `DIALOG(EX)`-structuur aan de hand van de opgehaalde naam. Indien er al een dergelijk object aanwezig is wordt dat verder gebruikt en aangevuld. In het andere geval wordt er een nieuw object aangemaakt en op het eind toegevoegd aan de centrale structuur.

Aan de hand van de `OptionsCaption`- en `ControlsCaption`-tags worden de named capture groups voor de opties en controls elk in een `string`-object ingelezen. Voor alle andere `Caption`-tags worden de waarden van properties ingelezen en opgeslagen. Er wordt telkens een koppel gegenereerd met als sleutel de naam van de property en als waarde de ingelezen waarde van de property. Deze koppels worden opgeslagen in een '`Dictionary<string, string>`'-structuur[15].

Na het inlezen van de options en controls moeten deze op hun beurt nog verwerkt worden. De options worden eerst geparset. Voor elke `Option`-tag in de `Form`-tag van het configuratiebestand wordt de zojuist ingelezen opties-variabele gematcht met de reguliere expressie van in de `Regex`-tag van de `Option`-tag. Indien de match succesvol is duidt dit aan dat de optie overeenkomstig met de `Option`-tag voorkomt in de opties-variabele. Net als bij de reguliere expressie voor `DIALOG(EX)`-structuren worden de bijhorende `Caption`-tags overlopen en '(naam,waarde)'-koppels toegevoegd aan de 'naam-waarden'-structuur.

Voor elk koppel in de marker-structuur worden alle markers van dat koppel in de templates vervangen door de overeenkomstige waarden in de 'naam-waarden'-structuur.

Het parsen van de controls

Nu rest nog het parsen van de controls. Het parsen van de controls is het grootste onderdeel van het parsen van een `DIALOG(EX)`-structuur. De werkwijze om de ingelezen `string`-variabele met de controls informatie te parsen komt overeen met de werkwijze om `DIALOG(EX)`-structuren te parsen uit een rc-bestand.

De ingelezen controls variabele wordt gematcht met de reguliere expressie van elke `Control`-tag in het configuratiebestand. Elke match betekent dat er een instantie van dat type control aanwezig is in de overeenkomstige `DIALOG(EX)`-structuur en wordt verder verwerkt. Ook hier wordt dankzij de `NameCaption`-tag de naam opgehaald. In tegenstelling tot `DIALOG(EX)`-structuren zijn de namen van controls niet uniek. Een `DIALOG(EX)`-structuur kan meerdere controls met dezelfde naam hebben. Dit zijn controls die niet in de code worden aangesproken, zoals bijvoorbeeld labels (de controls `LTEXT`, `CTEXT` en `RTEXT`). Omdat controls met dezelfde naam in een Windows Form niet worden aanvaard zal Rc2Form een unieke naam genereren. Indien er al een control is geparset met dezelfde naam, zal Rc2Form een cijfer als suffix toevoegen. Elke keer er daarna een control met diezelfde naam wordt geparset, wordt het cijfer verhoogt. Omdat dergelijke controls nooit worden aangesproken in de Visual C++ code en dus ook geen messages kunnen ontvangen (en daarom geen eventhandlers hebben), zal dit geen problemen opleveren in de resulterende Visual C# Windows Forms bestanden.

Analoog aan de `FormInformation`-objecten wordt de gegenereerde informatie van controls opgeslagen in `ControlInformation`-objecten. Er wordt eerst nagegaan of er al een

[15] Dit is niet dezelfde structuur als de `Dictionary<string, string>` aangemaakt in de constructor. De in de constructor aangemaakte structuur bevat de koppels '(naam, marker)' en zal verder worden aangeduid als de marker-structuur. De hier aangemaakt structuur bevat voor een specifieke `DIALOG(EX)`-structuur de koppels '(naam, ingelezen waarde)' en wordt verder aangeduid als de 'naam-waarden'-structuur.

`ControlInformation`-object aanwezig is via de (gegenereerde) unieke naam van de control[16]. Indien er al een dergelijk object aanwezig is wordt dat aangevuld, in het ander geval wordt er een nieuw object aangemaakt en later toegevoegd aan het huidig gebruikte `FormInformation`-object onder de (gegenereerde) unieke naam.

Daarna worden de `Declaration`-, `Initialization`- en `CodeWithoutParameters`-tags ingelezen als `string` variabelen. Ook wordt er een 'naam-waarden'-structuur opgesteld voor '(naam, ingelezen waarde)'-koppels van properties op de control. De structuur is een `Dictionary<string, string>` net zoals bij de `DIALOG(EX)` structuur. Via de reguliere expressies en de `Caption`-tags wordt deze structuur opgevuld. Er zijn vier speciale gevallen: de properties `Width`, `Height`, `Top` en `Left`. De afmetingen van MFC-objecten en Windows Forms objecten hebben geen 1-op-1 relatie. Als de waarden van deze properties ongewijzigd worden gebruikt zullen de Windows Forms en hun controls kleiner geschaald zijn dan hun MFC-equivalenten. Om dit tegen te gaan wordt er gebruikt gemaakt van twee offsets zoals in listing 9.23 op p.109.

```
1  const double MULOFFSET = 1.618;
2  const double ADDOFFSET = 0;
3  string data = matchControl.Groups[captionNode.Attributes["name"].Value].Value;
4  if (captionNode.Attributes["property"].Value == "Width"
5      || captionNode.Attributes["property"].Value == "Height"
6      || captionNode.Attributes["property"].Value == "Top"
7      || captionNode.Attributes["property"].Value == "Left")
8  {
9  data = Convert.ToString(
10         Convert.ToInt32(
11         Convert.ToDouble(
12             matchControl.Groups[captionNode.Attributes["name"].Value].Value
13         ) * MULOFFSET + ADDOFFSET
14         )
15 );
```

Listing 9.23: De offsets en hun bewerkingen voor het schalen van de resulterende Windows Forms objecten.

Bij het uitproberen van talloze waarden bleken de waarde 1.618[17] voor `MULOFFSET` en 0 voor `ADDOFSET` de meest gewenste resultaten te geven.

Een control beschreven in het rc-bestand bevat ook vaak opties. Via de `Optionscaption`-tag en de reguliere expressie worden de opties ingelezen in één grote `string`-variabele. Voor elke `Option`-tag in de `Control`-tag wordt de `string`- variabele gematcht met diens reguliere expressie. Indien er sprake is van een match worden alle `Caption`-tags van de reguliere expressie in de `Option`-tag overlopen. Telkens wordt er nagegaan of de `property`-tag waaraan de `Caption`-tag wordt gelinkt via het `Property`-attribuut een `PossibleValues`-tag bevat, gevuld met `Value`-tags. De aanwezigheid van een `PossibleValues`-tag betekent dat de waarde van de property geen letterlijk getal of string is dat zomaar kan overgenomen worden uit het rc-bestand, maar vertaald moet worden naar een Visual C# Windows Forms variant. De inhoud van de `Value`-tag, waarvan de waarde van het `content`-attribuut overeenkomt met de inhoud van de named capture group voor die optie, wordt dan opgeslagen als 'ingelezen' waarde in de 'naam-waarde'-structuur van de control. Op deze manier bevat de 'naam-waarde'-structuur

[16] Doordat controls telkens in dezelfde volgorde worden verwerkt, is de gegenereerde unieke naam voor controls steeds dezelfde.

[17] 1.618 is tevens bekend als de Gulden Snede φ. Omdat de meest gewenste waarde dicht aanleunde bij 1.6, werd als aardigheid besloten dit getal te gebruiken. De keuze voor dit getal heeft echter geen enkele wetenschappelijke betekenis.

van de control zowel 'naam-waarde'-koppels voor de basis properties als eventuele optionele properties. Het vervolg van de methode moet dan geen onderscheid maken tussen basis en optionele properties.

De 'naam-waarde'-koppels voor alle aanwezige properties van de control in het rc-bestand zijn ingelezen, maar moeten nog verwerkt worden. Voor elke `Property`-tag van de `Control`-tag moet er nog output worden gegenereerd en opgeslagen. De rol van `Property`-tags is aanduiden welke properties van de control er moeten worden uitgeschreven. Omdat er niet altijd genoeg informatie beschikbaar is voor het uitschrijven van properties wordt er gekeken naar de `Depends`-tags van de overeenkomstige `property`-tag in het `Properties`-gedeelte van het configuratiebestand. Als de 'naam-waarde' structuur voor elke propertie beschreven in de `Depends`-tag een 'naam-waarde'-koppel heeft, is er voor de propertie genoeg informatie beschikbaar en kan er output voor worden gegenereerd. Aan het `Code`-attribuut van het `ControlInformation` object wordt dan een nieuwe lijn toegevoegd met de code beschreven in de `Code`-tag van de `Property`-tag in het `Properties`-gedeelte van het configuratiebestand.

Nu rest enkel nog het opslaan van de `CodeBeforeFragment`- en `CodeafterFragment`-tags. De inhoud van de `Fragment`- en `Code`-tag in de `CodeBeforeFragment`- en `CodeafterFragment`-tags worden opgeslagen in `CodeInsertContainers` die op hun beurt worden opgeslagen in de `CodeBeforeFragment`- en `CodeafterFragment`-leden van het `ControlInformation`-object.

Als laatste worden alle markers in de ingelezen en gegenereerde waarden vervangen door hun waarde opgeslagen in de 'naam-waarde'-structuur.

Als het gebruikte `ControlInformation`-object voor het parsen van de control nieuw werd aangemaakt, wordt het opgeslagen onder de (gegenereerde) unieke naam van die control in het `FormInformation`-object van de overeenkomstige `DIALOG(EX)`-structuur. Indien dat object zelf nieuw is aangemaakt, wordt het toegevoegd in de centrale structuur onder de naam van de overeenkomstige `DIALOG(EX)`-structuur.

9.4.6 Het uitschrijven naar Windows Forms bestanden

Zoals bij het parsen van van eventhandlers en het parsen van het rc-bestand, bestaat er ook voor het genereren van de Windows Forms bestanden een aparte klasse met een enkele methode. Die klasse noemt `OutputGenerator` en de methode `GenerateOutput` (listing 9.24 op p.110).

```
1 public void GenerateOutput(ref Dictionary<string, FormInformation> forms, string
      givenNamespace)
```

Listing 9.24: De `GenerateOutput`-methode.

Bij het aanmaken van een instantie van de klasse `OutputGenerator` wordt het configuratiebestand *ParserConfiguration.xml* ingeladen in een `XmlDocument`. Dit is het enige configuratiebestand dat in de klasse `OutputGenerator` wordt gebruikt.

In de methode wordt de meegegeven centrale structuur, in tegenstelling tot de andere methodes, niet opgevuld. Er wordt ervan uitgegaan dat alle geparsete informatie aanwezig is in de centrale structuur.

Per entry in de centrale structuur wordt de informatie in het `FormInformation`-object (en de daarmee verbonden objecten) verwerkt tot de Windows Forms bestanden. Allereerst wordt er in de implementatie- en designercode van de form de marker `$namespace$` vervangen door de opgegeven namespace. Indien er geen namespace is opgegeven wordt deze stap overgeslaan. Als de gebruiker de bestanden dan toevoegt en compileert zullen de nog niet vervangen markers fouten genereren en de gebruiker erop wijzen dat deze nog moeten aangepast worden.

De volgende stap is het toevoegen van de code, gegenereerd voor de controls van de form, aan de code van de form. Dit gebeurt door het `control`-veld van het `FormInformation`-object te overlopen. De gegenereerde code van de controls voor declaratie, initialisatie en de control zelf worden elk geconcateneerd in een variabele. De geconcateneerde waarden zijn gescheiden door een newline karakter via het statement `Environment.Newline`. Er is dan per form een variabele met de geconcateneerde waarden voor de declaratie, een variabele voor initialisatie en een variabele voor de code van de controls zelf. Als de control ook code bevat voor events, worden deze ook toegevoegd aan een variabele met de geconcateneerde waarden voor eventhandlers in de implementiecode en een variabele voor de geconcateneerde waarden van eventhandlers in de designercode. Naast al deze waarden is er nog een speciale variabele. Voor elke verwerkte control wordt de waarde van de `Child`-tag in de `Form`-tag uit het configuratiebestand opgehaald (listing 9.11 op p.98). Deze tag bevat een codetemplate om de control te koppelen aan de form. In deze template is er slechts één marker aanwezig: `$child$`. Deze marker wordt vervangen door de naam van de control. Al deze codefragmenten worden net als de andere codefragmenten geconcateneerd met een newline karakter in een variabele.

Een entry in de `controls`-structuur van het `FormInformation`-object wordt enkel verwerkt als de methode `HasControlCode` van het `ControlsInformation`-object in de entry "`true`" teruggeeft. Het kan soms gebeuren dat er eventhandlers worden gedeclareerd in de `MESSAGE_MAP` voor controls die niet in het rc-bestand zijn gedeclareerd. Bij dergelijke 'spookcontrols' zit er in het overeenkomstige `ControlInformation`-object enkel gegenereerde code voor de eventhandlers, maar niet in de velden `code`, `initialization`, `declaration`, `codeBeforeFragments` en `codeAfterFragments`[18].

In de designercode worden dan de markers `$declarations$`, `$initializations$`, `$controls$`, `$controlsonform$` en `$events$` vervangen door de variabelen met de geconcateneerde waarden van declaraties van de controls, initialisaties van de controls, code van de controls zelf, code voor het koppelen van de controls aan de form en de designercode van eventhandlers van controls en de form zelf. In de implementatiecode van de form wordt de marker `$events$` vervangen door de variabele met de geconcateneerde implementatiecode van eventhandlers van de controls en de form zelf.

Hierna moeten de codefragmenten verwerkt worden die voor of achter bepaalde fragmenten moeten worden ingevoegd. Hiervoor worden opnieuw alle entries in het `controls`-veld van het `FormInformation` overlopen. Voor elk `ControlInformation`-object worden eerst de `codeBeforeFragments` behandeld en daarna de `codeAfterFragments`. Omdat invoegingen worden uitgevoerd volgens de volgorde van de controls in het `controls`-veld van het `FormInformation`, kunnen invoegingen genest worden. Als bijvoorbeeld twee controls in een form iets moeten invoegen voor `XXX` en na `YYY` dan zal dit eruit zien zoals in listing 9.25 op p.111. Als ze beiden voor `XXX` en voor `YYY` moeten invoegen zal dit er uitzien zoals in listing 9.26 op p.112. Het nadeel is dat het codefragment waarvoor of waarachter moet ingevoegd worden, niet zelf een in te voegen fragment mag zijn. De resultaten daarvan zijn niet altijd te voorspellen. Zeker als daarbij ook de volgorde in de `codeBeforeFragments`- en `codeAfterFragments`-velden van het `ControlInformation`-object bij komt kijken.

```
1 BBB
2 AAA
3 XXX
```

[18] `code`, `initialization`, `declaration` hebben dan als waarde `null`. `codeBeforeFragments` en `codeAfterFragments` hebben dan een leeg `List<CodeInsertContainer>`-object.

```
4 ...
5 YYY
6 AAA
7 BBB
```

Listing 9.25: Een voorbeeld van geneste invoegingen.

```
1 AAA
2 BBB
3 XXX
4 ...
5 AAA
6 BBB
7 YYY
```

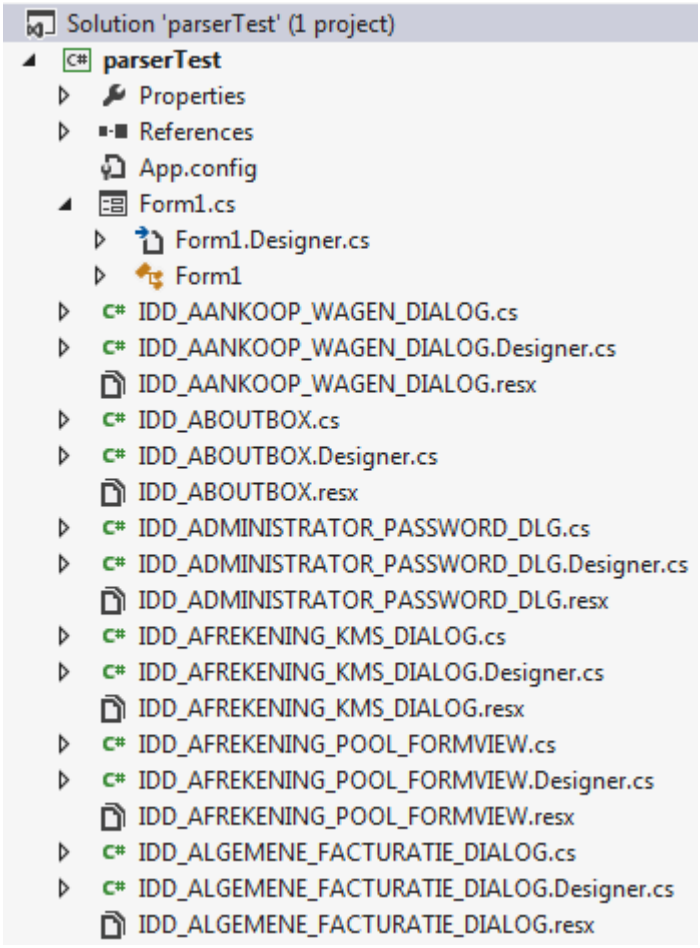
Listing 9.26: Een voorbeeld van invoegingen gebaseerd op de volgorde.

Tot slot worden de Windows Forms bestanden uitgeschreven. Ze krijgen als naam de naam van de geparsete `DIALOG(EX)`-structuur (tevens ook de naam van de Visual C# Windows Form). Het cs-bestand bevat dan de resulterende implementatiecode en het "Designer.cs"-bestand de designercode. Het resx-bestand wordt gekopïeerd van het *FormResourceTemplate.txt* bestand.

9.5 Opmerkingen

Indien er een grote hoeveelheid Visual C# Windows Forms in één keer in een project wordt ingeladen, dan worden deze niet juist ingeladen. De implementatiecode-, designercode- en resourcebestanden worden als aparte bestanden beschouwd en niet als een geheel (figuur 9.5 op p.113).

Om hieraan tegenmoet te komen werd er door de externe begeleider, Tom Eeraerts, een programma geschreven dat de bestanden inlaadt door het csproj-bestand rechtstreeks aan te passen.



Figuur 9.5: De forms (“IDD_AANKOOP_WAGEN_DIALOG, . . .) worden hier verkeerd ingeladen. Enkel Form1, dat al aanwezig was, wordt als een Windows Form weergegeven.

Hoofdstuk 10

Het vertalen van code

Omdat het via Rc2Form mogelijk is om GUI van een Visual C++ om te zetten naar een Visual C# variant werd er ook gekeken of het mogelijk was om de logica om te zetten naar Visual C#. Door het gebruik van Rc2Form en een methode voor de logica zou het mogelijk zijn om een programma bijna helemaal om te zetten met zo weinig mogelijk menselijke tussenkomst. Helaas bleek deze piste niet haalbaar.

10.1 Syntax-directed translation

Een veelbelovende methode bleek Syntax-Directed Translation. Deze methode voegt regels toe aan de grammatica van een programmeertaal.

Een context-vrije grammatica van een programmeertaal bestaat uit volgens Aho et al. (2007) uit vier onderdelen:

Terminalen
Elementaire symbolen van de programmeertalen (bv. de "**;**" in C++).

Niet-terminalen
Een nonterminal beschrijft een waaier aan mogelijke codefragmenten.

Producties
Producties bestaan uit de vorm $hoofd \rightarrow lichaam$. Het hoofd is altijd een niet-terminaal. Het lichaam daarentegen kan bestaan uit zowel terminalen als niet-terminalen (of het nulsymbool ϵ). Het lichaam geeft aan hoe de niet-terminaal in het hoofd wordt uitgeschreven.

Het startsymbool
Een niet-terminaal dat wordt aangeduid als het startsymbool van de grammatica.

Deze onderdelen definiëren samen de programmeertaal. Door vanaf het startsymbool telkens niet-terminalen te vervangen door hun lichamen tot er enkel terminalen overschieten kunnen alle mogelijke codefragmenten van de taal worden bekomen. Dit wordt het afleiden van het startsymbool genoemd. Al deze mogelijke codefragmenten samen vormen dan de taal gedefinieerd door de grammatica. Het probleem om te bepalen of een codefragment behoort tot een taal is het parseprobleem. Er moet worden nagegaan of het mogelijk is om vanaf het startsymbool het codefragment af te leiden. Indien het codefragment niet afleidbaar is van

het startsymbool behoort het niet tot de programmeertaal en is er sprake van een syntaxfout. Het afleiden naar een codefragment kan worden voorgesteld door een boomstructuur, namelijk een *parse tree*.

Bij syntax-directed translations worden de producties aangevuld met regels. Er zijn twee vormen syntax-directed translation:

Syntax-directed definitions
Elk grammaticasymbool kan een aantal variabelen bevatten. De waarde van de variabelen wordt dan bepaald door regels overeenkomstig met producties van dat grammaticasymbool.

$$E \to E_1 + T \qquad E.code = E_1.code \| T.code \| '+'$$

Figuur 10.1: Een voorbeeld van een syntax-directed definition.

Syntax-directed translation schemes
Binnenin producties kunnen er codefragmenten voorkomen. De plaats van de codefragmenten bepaalt wanneer deze worden uitgevoerd bij een van links naar rechts diepte-eerst (preorder) doorgang van de parse tree.

$$E \to E_1 + T \ \{print\ '+'\}$$

Figuur 10.2: Een voorbeeld van een syntax-directed translation scheme.

Het gebruik van syntax-directed translation schemes leek het meest nuttige hulpmiddel voor de vertaling van Visual C++ naar Visual C#.

10.2 ANTLR

Omdat het schrijven van lexers en parser enorm tijdsverslindend kan zijn, werd er besloten te werken met de parsergenerator ANTLR[1]. ANTLR had Visual C# als mogelijke doeltaal (naast C++, Java, ...) als taal van de gegenereerde lexer en parser. Het leek het best mochten deze in Visual C# worden geïmplementeerd, omdat het doel van de lexer en parser het achterwege laten van Visual C++ is.

ANTLR biedt ook ondersteuning voor het genereren en overlopen van Abstract Syntax Tree (AST)-structuren in parsers.

Dankzij de StringTemplate engine kunnen door ANTLR gegenereerde vertalers vertalen naar verschillende talen. Op deze manier zou het theoretisch mogelijk zijn om met een enkele gegenereerde parser niet enkel te vertalen naar Visual C# maar ook bijvoorbeeld naar Java. De vertaler zou dan zelfs niet opnieuw moeten worden gecompileerd, verwisselen van templatebestand is voldoende. Voor syntax-directed translation is dit erg handig.

Het opstellen van grammatica's in ANTLR is relatief eenvoudig en beidt ondersteuning voor reguliere expressie constructies zoals '`(...)*`' en '`(...)+`'.

[1] `http://www.antlr.org/`

```
grammar SimpleCalc;

tokens {
    PLUS     = '+' ;
    MINUS    = '-' ;
    MULT     = '*' ;
    DIV = '/' ;
}

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

expr            : term ( ( PLUS | MINUS )  term )* ;

term            : factor ( ( MULT | DIV ) factor )* ;

factor          : NUMBER ;

/*------------------------------------------------------------------
 * LEXER RULES
 *------------------------------------------------------------------*/

NUMBER          : (DIGIT)+ ;

WHITESPACE      : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+    { $channel = HIDDEN; } ;

fragment DIGIT  : '0'..'9' ;
```

Listing 10.1: Een voorbeeld van een simpele ANTLR-grammatica.

ANTLR is een LL(k) recursive descent parser. De categorie waartoe de grammatica van C++ (en Visual C++) behoren is niet echt duidelijk. C++ behoort niet tot LL of LR aangezien het *ambiguities* bevat. Ambiguities zijn onduidelijkheden in de grammatica, waardoor voor bepaalde constructies meerdere parse trees kunnen worden opgesteld. Omdat ANTLR een speciale versie van LL-grammatica's gebruikt, is deze toch in staat om C++ parsers op te stellen. Recursive descent parsers zijn veel duidelijker en leesbaarder opgesteld dan LR-parsers.

> "*ANTLR uses a new parsing strategy that makes it possible to develop natural, easy-to-read grammars for difficult languages like C++. ANTLR uses pred-LL(k)grammars, which is a LL(k)for ¿1grammar augmented with predicates. Predicates allow arbitrary semantic and syntactic information to direct the parse.*" (Parr & Quong, 1994)

Ook Stroustrup (1995) vermeld dat hij beter een recursive descent parser had geschreven in plaats van een LALR(1) parser te laten genereren met Yet Another Compiler-Compiler (YACC).

> "*In 1985 when I first planned Cfront, I wanted to use a recursive descent parser because I had experience writing and maintaining such a beast, because I liked such parsers' ability to produce good error messages, and because I liked the idea of having the full power of a general-purpose programming language available when decisions had to be made in the parser. Al Aho and Steve Johnson (...) convinced me that writing a parser by hand was most old-fashioned, would be inefficient use of my time, would almost certainly result in a hard-to-understand and hard-to-maintain parser, and would be prone to unsystematic and therefore*

> *unreliable error recovery. the right way to use an LALR(1) parser generator, so I used Al and Steve's YACC.*
>
> *For most projects, it would have been the right choice. For almost every project writing an experimental language from scratch, it would have been the right choice. In retrospect, for me and C++ it was a bad mistake. (...) Even Steve Johnson's PCC, which was the preeminent C compiler at the time, cheated at details that were to prove troublesome to C++ parser writers. (...) My bias towards top-down parsing has shown itself many times over the years in the form of constructs that are hard to fit into a YACC grammar. To this day, Cfront has a YACC parser supplemented by much lexical trickery relying on recursive descent techniques. On the other hand, it is possible to write an efficient and reasonably nice recursive descent parser for c++. Several modern C++ compilers use recursive descent.*" (Stroustrup, 1995)

Samen met de aanwezigheid van een reeds bestaande grammatica voor C++ was er voldoende bewijs dat de ANTLR parser generator in staat is om een C++ parser te genereren.

Er bestaan ook GLR-parsers. Deze parsers zijn een uitbreiding van LR-parsers en kunnen ook grammatica's met ambiguities verwerken. Dit komt omdat bij conflicten alle mogelijkheden verder worden onderzocht. Hierdoor zijn GLR-parsers minder performant, maar wel bruikbaar voor bijvoorbeeld C++. Het gebruik van GLR-parser werd in deze masterproef niet onderzocht, omdat ANTLR reeds alle nodige functionaliteit bevatte.

10.3 Problemen

Helaas ging het opstellen van een methode niet van een leien dakje. Al snel kwamen er verschillende problemen opduiken waardoor het onmogelijk werd om zelfs maar een parser te genereren.

10.3.1 Het vinden van een juiste grammatica voor Visual C++

Alhoewel er voor ANTLR al een grammatica bestond bleek deze niet van toepassing. Het ging over een grammatica voor standaard C++ en was het laatst aangepast in 2011. Ook had de grammatica als doeltaal C++. Het aanpassen van deze grammatica zou een enorm tijdrovende en foutgevoelige klus zijn. Als alternatief werd er gekeken of er een grammatica kon worden gegenereerd aan de hand van documentatie.

De MSDN voldoet niet

Het vinden van een grammatica gericht op Visual C++ (en niet op standaard C++) bleek geen sinecure. Op MSDN werd er enkel voor Visual C++ 2003 een volledige grammatica gevonden. De MSDN voor Visual C++ 2005 en hoger geeft niet meer de volledige grammatica weer, maar enkel de Microsoft Extensions. Het was dan ook onduidelijk of de grammatica van Visual C++ 2005 en hoger niet meer was veranderd sinds Visual C++ 2003 buiten de Microsoft Extensions of dat buiten e Microsoft Extensions de Visual C++ graammatica van versie 2005 en hoger compleet conform is met de standaard c++ grammatica.

Om meer duidelijkheid te krijgen werd er een nieuwe thread aangemaakt op het MSDN-forum[2] Daar werd er door Xie (2012) aangeraden om feedback in te dienen bij Microsoft. Na het indienen van de feedback werd als antwoord ontvangen dat de MSDN op dat vlak inderdaad tekortschiet en de MSDN aangepast zal worden. Microsoft (2012p)

> "*Thank you for the improvement suggestion, we take these very seriously. I have evaluated the current content for this area and it does indeed need work, so I created a content task to address this shortcoming. It will take some time to generate the content and get it up on MSDN, in the meantime here is some potentially useful advice from our experts:*
> *'If I were intent on doing this, I would start with the official C++ grammar:* `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3376.pdf`. *I think most of our extensions are fairly self-contained. For example, by and large I think you could just ignore all __declspecs and it wouldn't affect the syntax of the program at all. So once you've got the standard C++ grammar implemented you can probably just browse through the list of Microsoft Extensions and exclude/skip over all of them. (Or fail the translation. What else could you do with an asm block anyway?)'* " (Microsoft, 2012p)

De standaard C++ grammatica als backup

Op aanraden van Microsoft (2012p) in het antwoord op de feedback werd er dan overgeschakeld op de standaard C++ grammatica. Helaas kwamen er toen nieuwe problemen aan het licht. Omdat de grammatica van C++ is opgesteld met als doel zo duidelijk mogelijk te zijn is deze grammatica eigenlijk niet echt geschikt als basis voor een parser. De grammatica bevat een aantal niet nader gespecificeerde producties. Bijlage C op p. 160 bevat een overzicht van dergelijke functies. Voor het gemak wordt de eerste productie nogmaals weergegeven in 10.1.

$$
\begin{aligned}
\mathit{preprocessing-token} \rightarrow\ & \mathit{header-name} \\
\mid\ & \mathit{identifier} \\
\mid\ & \mathit{pp-number} \\
\mid\ & \mathit{character-literal} \\
\mid\ & \mathit{user-defined-character-literal} \quad \text{C++0x} \\
\mid\ & \mathit{string-literal} \\
\mid\ & \mathit{user-defined-string-literal} \quad \text{C++0x} \\
\mid\ & \mathit{preprocessing-op-or-punc} \\
\mid\ & \text{each non-white-space character} \\
& \quad \text{that cannot be one of the above}
\end{aligned}
\tag{10.1}
$$

De eigenlijke ongespecificeerde productie wordt niet cursief weergegeven en is hier vervat in zijn eigen productie in 10.2.

$$
\begin{aligned}
\mathit{preprocessing-token} \rightarrow\ & \text{each non-white-space character} \\
& \quad \text{that cannot be one of the above}
\end{aligned}
\tag{10.2}
$$

[2]Meer specifiek in de tak *Microsoft Developer Network > Forums Home > Visual Studio Languages Forums > C++ Standards, Extensions, and Interop.*

Het is heel moeilijk om dergelijke producties te implementeren. Producties zoals 10.2 zijn ontstaan doordat de grammatica is opgesteld met als doel om te informeren. Dit maakt deze grammatica's eigenlijk niet ideaal om te gebruiken als basis voor een parser (of parsergenerator). Helaas is het enig alternatief het zelf schrijven van een grammatica, maar dat is niet haalbaar voor een enkel persoon in de tijdsperiode van de masterproef.

Ongespecificeerde producties berusten ook op specifieke implementaties. De invulling van de ongespecificeerde lichamen hangt vaak af van het taalplatform zoals Visual C++, standaard C++, ... Ongespecifiëerde producties die steunen op de *source character set*[3], de verzameling bruikbare karakters voor bronbestanden, zijn al wat makkelijker te implementeren dankzij informatie van Microsoft (2012a):

> "*Microsoft Specific*
> *For the Microsoft C/C++ compiler, the source and execution character sets are both ASCII.*
> *The basic source character set consists of 96 characters: the space character; the control characters that represent horizontal tab, vertical tab, formfeed, and newline; and the following 91 characters:*
> *abcdefghijklmnopqrstuvwxyz*
> *ABCDEFGHIJKLMNOPQRSTUVWXYZ*
> *0123456789*
> *_ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \"*
> *The basic execution character set consists of the characters in the basic source characters set, and also the control characters that represent alert, backspace, carriage return, and null.*
> *END Microsoft Specific*" (Microsoft, 2012a)

Dankzij deze informatie en de informatie in het lichaam van de ongespecifiëerde productie kunnen dan nuttige overeenkomstige producties worden opgesteld.

Zo is het newline karakter in productie C.10 op p.161 "`\r\n`" voor niet-UNIX systemen en "`\n`" voor UNIX systemen.

Voor de producties C.1, C.4 en C.9 in bijlage C op p.160 zijn er geen aanvaardbare alternatieven gevonden. De kleinste aanpassing van een grammatica kan grootste gevolgen hebben, terwijl het testen van grammatica's zeer moeilijk is. Dit geldt zeker voor complexe talen: Er moet niet alleen worden getest op juiste code dat als fout wordt aanzien, maar ook foute code dat als juist wordt aanzien. Bij aanpassingen kunnen dergelijke fouten exponentieel oplopen. Om een grammatica volledig te testen zouden alle mogelijk afleidbare structuren van het startsymbool moeten worden nagegaan. Het implementeren van alternatieven zouden hoogst waarschijnlijk de grammatica schade aanbrengen. Doordat er geen zicht is op de gevolgen van wijzigingen werd uiteindelijk besloten dat deze piste geen toekomst had.

Een ander gevolg van het educatief karakter van de standaard C++ grammatica is dat er niet-terminalen voorkomen die zelf geen productie hebben. Een voorbeeld hiervan is de niet-terminaal `alignment-expression`. De niet-terminaal komt zelf in de lichamen van een aantal producties voor, maar er bestaat geen productie met deze niet-terminaal in het hoofd. Er resten dan twee opties: zelf producties voor deze niet-terminaal schrijven of de niet-terminaal overal in de grammatica te vervangen door een andere terminaal. Net zoals bij ongespecifiëerde producties zijn de gevolgen van beide opties niet te onderschatten en ook

[3]Producties C.2, C.3, C.5, C.6, C.8 in bijlage C op p.160.

niet te overzien. Op dit moment is `alignment-expression` de enigste niet-terminaal zonder producties.

10.3.2 Links-recursieviteit

Zowel de grammatica's van Microsoft als standaard C++ bevatten links-recursieve structuren. Links-recursie houdt in dat er producties zijn waarbij het eerste atomair deel van de lichaam dezelfde niet-terminaal is als het hoofd van de productie. Dit is een recursie op hetzelfde niveau (productie 10.3 op p.120).

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n \tag{10.3}$$

Figuur 10.3: Een voorbeeld van een links-recursie op hetzelfde niveau. Hoofdletters staan voor niet-terminalen, Griekse letters staan voor een reeks van nul of meer terminalen en niet-terminalen. (Aho et al., 2007)

Sommige types parsers (en dus ook parsergenerators, want deze creëren parsers aan de hand van grammatica's) kunnen niet tegen links-recursie. Links-recursieve grammatica's kunnen ervoor zorgen dat dergelijke parsers in een oneindige lus terechtkomen. ANTLR is een LL($*$) recursive-descent parsergenerator en kan daarom niet tegen links-recursie.

Door het invoeren van een extra niet-terminaal en diens productie kan men de links-recursiviteit verwijderen. De oplossing voor productie 10.3 op p.120 is productie 10.4 op p.120.

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' &\rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{aligned} \tag{10.4}$$

Figuur 10.4: De niet-links-recursieve oplossing voor productie 10.3 op p.120. (Aho et al., 2007)

Hiermee is het probleem niet van de baan. Links-recursie kan ook op meerdere niveaus voorkomen in afleidingen bestaande uit twee of meer stappen.

$$\begin{aligned} S &\rightarrow A\ a \mid b \\ A &\rightarrow A\ c \mid S\ d \mid \epsilon \end{aligned} \tag{10.5}$$

Figuur 10.5: Een voorbeeld van links-recursie op meerdere niveaus ($S \Rightarrow Aa \Rightarrow Sda$). (Aho et al., 2007)

Volgens Aho et al. (2007) kan men alle linksrecursie verwijderen uit een grammatica met het algoritme 1 op p. 121, op voorwaarde dat de grammatica geen lussen of ϵ-producties bevat.

Een negatief effect van het resultaat van het algoritme is dat de resulterende grammatica rechts-recursief is. Dit vervormt de associativiteit van aritmetische operatoren, waardoor de grammatica uiteindelijk niet bruikbaar is.

Rangschik de niet-terminalen in een volgorde A_1, A_2, ..., A_n.
for *each i from 1 to n* **do**
  for *each j from 1 to* $i-1$ **do**
    vervang elke productie van de vorm $A_i \to A_j\gamma$ door de producties
    $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$, waar $A_j \to \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ alle huidige A_j-producties zijn.
  end
  verwijdere links-recursie op hetzelfde niveau voor alle A_i-producties
end
Algorithm 1: Het verwijderen van links-recursie op alle niveaus. (Aho et al., 2007)

10.4 Het opstellen van een vertaler

De vooraf besproken nadelen werden pas gaandeweg duidelijk tijdens het opstellen van een proof of concept. Het was de bedoeling om na te gaan of er een vertaler kon worden opgesteld om Visual C++ te vertalen naar Visual C#. In eerste instantie zou worden nagegaan of het mogelijk was voor standaard C++ en daarna voor Visual C++.

Theoretische schets

De verwachte moeilijkheden achter een mogelijke vertaler bestonden uit drie niveaus:

1. Het 1-op-1 vertalen van statements.
2. Het vertalen van methodes en klassen van Visual C++ (bv. uit de STL, MFC, ATL, ...) naar Visual C# .NET varianten.
3. Het omzetten van het importeren van headerstanden in Visual C++ naar het gebruik van referenties in Visual C#.

Het eerste niveau is het vertalen van constructies in Visual C++ naar Visual C# . Veel structuren kunnen gelijk blijven, zoals `i++;`, maar andere structuren, zoals pointers, moeten worden vertaald. Hier zouden de syntax-directed translations voor dienen.

Het tweede niveau en het derde niveau hangen sterk samen. Visual C++ en Visual C# steunen elk op een eigen set van klassen. Eerst moet er een onderscheid worden gemaakt welke klassen zelf zijn aangemaakt in het project, afkomstig zijn uit externe bibliotheken of horen tot standaardklassen van de taal. Enkel eigen klassen moeten worden vertaald. Voor externe bibliotheken en standaardklassen moeten er alternatieven worden gezocht met gelijkaardige functionaliteit. Dit geldt ook voor hun methodes. Om hieraan tegemoet te komen werd er gedacht aan een database die voor elke Visual C++ standaard klasse of methode informatie bevat en informatie voor een waardig Visual C# alternatief. De vertaler zou dan bij elke klasse- of methode-aanroep de databank raadplegen en kijken of er een Visual C# is gekend. Indien er geen record in de databank zit voor de aanroep, weet de vertaler dat het gaat om ofwel een zelfgeschreven klasse ofwel een klasse uit een externe-library of standaard-klasse zonder waardig alternatief.

Het opstellen

Er werd begonnen met het eerste niveau. Het doel was om via ANTLR een vertaler te genereren op basis van een gramatica aangevuld met syntax-directed translation schemes. Door de problemen vermeld in hoofdstuk 10.3.1 op p.117 en hoofdstuk 10.3.2 op p.120 werden er geen geschikte C++ grammatica's gevonden. Voor het ophalen van de standaard grammatica werd er een perlscript geschreven om de informatie op te halen van een website (`http://www.nongnu.org/hcb/`) op het internet. De site bevat een hyperlinked standaard C++ grammatica. Het script parset de brondcode van de webpagina en genereert op basis daarvan de overeenkomstige ANTLR-compatibele grammatica. Om dan de links-recursiviteit op meerdere niveaus te verwijderen werd er een ander perlscript geschreven dat ANTLR-grammatica's inleest, daarop het algoritme 1 op p.121 uitvoert en het resultaat terug als ANTLR-grammatica uitschrijft. Beide scripts zijn terug te vinden op de, bij deze masterproef, meegeleverde dvd.

Het beëindigen van het opstellen

Uiteindelijk vormden de nadelen van de grammatica besproken in hoofdstuk 10.3.1 op p.117 en hoofdstuk 10.3.2 op p.120 dergelijk grote obstakels dat ze de voordelen overtreffen. De kosten wogen immers niet op tegen de baten. Op de website van ANTLR worden er al grammatica's aangeboden om parsers te genereren[4]. Bij het bekijken van de documentatie van de recentste reeds bestaande ANTLR-grammatica voor C++ werd duidelijk hoeveel werk werd gestoken in het maken van een ANTLR-grammatica voor C++. Bovendien zou het resultaat van dit alles een vertaler zijn dat Visual C++ code omzet naar Visual C# code, wat niet zoveel voordeel biedt als men zou denken. Alle gegenereerde code moet sowieso worden nagekeken en er wordt niet geoptimaliseerd. De vertaling is te letterlijk waardoor nieuwe structuren en technologieën niet worden aangewend. Het werd uiteindelijk duidelijk dat deze piste niet de gewenste resultaten zou opleveren en de poging tot het opstellen van een vertaler werd beëindigd bij het proberen behalen van een bruikbare grammatica voor ANTLR.

[4]Helaas waren deze niet echt bruikbaar omdat ze al syntax-directed translations bevatten voor het genereren van C++ klassen.

Besluit

In deel I werd het duidelijk dat het converteren van Visual C++ 6.0 naar Visual C++ 2010 geen eenvoudige taak is. Het opzoeken van informatie over het opwaarderen gaf aan dat er bijna geen vaste methodes of hulpmiddelen beschikbaar zijn. In de meeste gevallen werd het opwaarderen manueel uitgevoerd.

De moeilijkheden voor het opwaarderen werden veroorzaakt door breaking changes tussen de specificaties van verschillende Visual C++ versies, samen met een gewijzigde projectstructuur. Naast de talloze breaking changes in de Visual C++ specificaties waren er vaak ook problemen met externe libraries en projectinstellingen in Visual Studio.

Om de projectstructuur op te waarderen kan er gebruik gemaakt worden van tools ingebouwd in Visual Studio zoals de Visual Studio Conversion Wizard. Voor het oplossen van breaking changes bestonden er nog geen hulpmiddelen. Als antwoord hierop werd in het kader van deze masterproef de applicatie iSE ontwikkeld.

iSE is een applicatie dat aan de hand van sln-bestanden breaking changes in de solution detecteert en dan aanbiedt om ze op te lossen. Het spreekt hiervoor de commandline compiler aan van de gewenste Visual Studio doelversie en spoort breaking changes op door de output van de commandline compiler te filteren met reguliere expressies. Er kunnen oplossingen en tools aan iSE worden toegevoegd of worden uitgebreid om het aantal breaking changes die opgelost kunnen worden uit te breiden. iSE heeft zichzelf talloze malen bewezen tijdens het manueel opwaarderen van projecten tijdens het onderzoek. Dit toonde aan dat het de gebruiker tijd bespaart en effectief helpt. Wel moet de gebruiker de door iSE aangeboden oplossingen steeds nakijken.

Tijdens het onderzoek werd wel duidelijk dat niet alle handelingen geautomatiseerd kunnen worden. Het aanpassen of opwaarderen van externe libraries en het aanpassen van projectinstellingen moeten nog steeds manueel door de gebruiker worden ingesteld. Ook moeten soms de oplossingen door iSE worden aangepast. Om het manueel werk te verlichten werden alle fouten verzameld en gedocumenteerd.

Dankzij de tools in Visual Studio, iSE en de opgedane kennis tijdens de manuele opwaarderingen in het onderzoek is men er in geslaagd om alle Visual C++ 6.0 projecten op te waarderen naar Visual C++ 2010.

Voor het converteren van applicaties in Visual C++ 2010 naar Visual C# 2010, werden er in deel II twee verschillende strategieën opgesteld en getest aan de hand van proof of concepts. Ook werd er gewerkt aan het opstellen van een tweetal hulpmiddelen om het converteren te vergemakkelijken.

De proof of concepts van de twee startegieën slaagden beiden in hun opzet:

- Het gebruik van managed code zorgde ervoor dat de ICORDA applicatie GandaFact kon worden opgewaardeerd. Dankzij deze strategie kunnen solutions project per project

geconverteerd worden. Telkens als er een project wordt geconverteerd naar Visual C# moeten enkel de projecten die het geconverteerde project aanroepen als managed worden ingesteld en hun aanroepen worden aangepast.

De proof of concept maakte duidelijk dat door het gebruik van managed code, de solution de kracht van het .NET framework kan aanwenden. Het .NET framework en al zijn technologieën zoals de class library en garbage collection kunnen worden aangesproken. Zo wordt er in het vervangende Visual C# project in GandaFact gebruik gemaakt van recente technologieën zoals een Visual C# databankmodel, gegenereerd door de .NET objectrelationele mapper Entity Framework van Microsoft.

Helaas brengt dit ook een kleine kost met zich mee. In Visual Studio 2010 was er geen Intellisense voor managed Visual C++ aanwezig. Ook moesten er enkele opties zoals "Edit and Continue" en "Minimal rebuild" worden uitgeschakeld waardoor debuggen moeilijker wordt en het compileren langer duurt. Daarnaast moet ook het .NET framework aanwezig zijn op de systemen waar het programma moet geïnstalleerd worden.

- Het aanwenden van COM interfaces voor conversie bleek uiteindelijk de meest belovende strategie te zijn. Er moet geen unmanaged code als managed code worden gecompileerd, waardoor de nadelen van de eerste strategie wegvallen. Door COM-projecten in de solution te vervangen door Visual C# alternatieven die de COM interfaces respecteren en functies met dezelfde functionaliteit aanbieden, kan de solution stapsgewijs worden geconverteerd. Bij het aanroepen van COM-methodes van een Visual C# COM-project vanuit unmanaged code, moet enkel de pointer naar het COM-project gewijzigd worden zodat deze wijst naar het Visual C# COM-project in plaats van het oud unmanaged COM-project.

  Het gebruik van COM kan dan uiteindelijk worden weggewerkt. De omzetting tussen Visual C# en Visual C++ types is mogelijk dankzij het gebruik van de functionaliteit in de .NET namespace `System.Runtime.InteropServices`.

  In de proof of concept werd dankzij deze strategie het project IRServer van de solution iRent succesvol geconverteerd. De strategie wordt nu verder onderzocht in ICORDA.

Bij ICORDA leerde de ervaring dat gebruikers geen verandering in de interface van de applicaties wensen. Daarom werd er in het kader van deze masterproef de applicatie Rc2Form ontwikkeld die de GUI in de Visual C++ applicaties omzet naar Visual C# Windows Forms. Omdat er veel verschillende controls en daarbij horende opties bestaan, wordt er gebruik gemaakt van uitgebreidde xml-configuratiebestanden. Deze configuratiebestanden bevatten alle nodige informatie om control- en vensterstructuren uit rc-bestanden en eventhandlers uit de brondcodebestanden te parsen. Bij testen op applicaties van ICORDA bewezen de uitgebreidde XML-bestanden talloze keren hun nut. Veel fouten in het programma konden simpelweg opgelost worden door de informatie in de configuratiebestanden te wijzigen, zonder aanpassingen in de code van Rc2Form.

Omdat Rc2Form eigenlijk zorgt voor conversie van de GUI code, werd er gezocht naar een oplossing om de code voor de logica te converteren. Er werd een poging gedaan om een Visual C++ naar Visual C# vertaler op basis van syntax-directed translation op te stellen met behulp van de parser generator ANTLR. Helaas heeft deze poging niet veel opgebracht. Er was nergens een grammatica te vinden die door ANTLR kon gebruikt worden. Er werd geprobeerd de Visual C++ grammatica van de MSDN en de standaard C++ grammatica te

herwerken, zodat ze konden ingelezen worden door ANTLR. Helaas vormden hinderpalen zoals links-recursie, niet-terminalen zonder duidelijke producties en zelfs niet-terminalen zonder producties een onoverkomelijke hindernis. Doordat het zelf schrijven van een grammatica teveel tijd in beslag zou nemen, werd besloten deze piste te verlaten.

Deel III

Bijlagen

Bijlage A

Voorbeelden van project- en solutionbestanden

```
Microsoft Developer Studio Workspace File, Format Version 6.00
# WARNING: DO NOT EDIT OR DELETE THIS WORKSPACE FILE!

###############################################################################

Project: "CZAM"=..\..\Libraries\CZAM\CZAM.dsp - Package Owner=<4>

Package=<5>
{{{
    begin source code control
    "$/VC60WNET/Libraries/CZAM", EXHAAAAA
    ..\..\libraries\czam
    end source code control
}}}

Package=<4>
{{{
}}}
...
```

Listing A.1: Een voorbeeld van een dsw-bestand.

```
# Microsoft Developer Studio Project File - Name="TxKassa" - Package Owner=<4>
# Microsoft Developer Studio Generated Build File, Format Version 6.00
# ** DO NOT EDIT **

# TARGTYPE "Win32 (x86) Application" 0x0101

CFG=TxKassa - Win32 Debug
!MESSAGE This is not a valid makefile. To build this project using NMAKE,
!MESSAGE use the Export Makefile command and run
!MESSAGE
...
```

Listing A.2: Een voorbeeld van een dsp-bestand.

```

Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 2012
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "IRClient",
    "IRClient\IRClient.vcxproj", "{768CDFE4-7A0C-41EB-8BEF-131AF6F2AAD9}"
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "IRClientWeb",
    "IRClientWeb\IRClientWeb.vcxproj", "{F342EABE-8B32-4354-80A8-515426331B82}"
```

```
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "IRExcel",
    "IRExcel\IRExcel.vcxproj", "{5999C827-2FBC-4B27-AEC5-C91DE8382168}"
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "IRFTP", "IRFTP\IRFTP.vcxproj",
    "{1380DE08-13A6-481C-9174-5DCB230A595A}"
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "IRLookUp",
    "IRLookUp\IRLookUp.vcxproj", "{74A5F287-78A7-422E-BB06-985DDB4B103F}"
EndProject
Project("{8BC9CEB8-8B4A-11D0-8D11-00A0C91BC942}") = "IRServer",
    "IRServer\IRServer.vcxproj", "{88BA470E-B264-4A75-851C-46D7221C157A}"
EndProject
...
```

Listing A.3: Een voorbeeld van een sln-bestand (versie 2012).

```
<?xml version="1.0" encoding="Windows-1252"?>
<VisualStudioProject
    ProjectType="Visual C++"
    Version="8,00"
    Name="iRent"
    ProjectGUID="{297D72C7-A482-42FC-9E6F-631DC0D4B2E6}"
    RootNamespace="iRent"
    SccProjectName="&quot;$/VC2005/CIACRent/iRent&quot;, PKMDAAAA"
    SccLocalPath="."
    SccProvider="MSSCCI:Microsoft Visual SourceSafe"
    Keyword="MFCProj"
    AssemblyReferenceSearchPaths="&quot;..\..\Libraries\Poco&quot;"
    >
    <Platforms>
        <Platform
            Name="Win32"
        />
    </Platforms>
    <ToolFiles>
    </ToolFiles>
    <Configurations>
        <Configuration
            Name="Release|Win32"
            OutputDirectory=".\Release"
            IntermediateDirectory=".\Release"
            ConfigurationType="1"
            InheritedPropertySheets=
            "$(VCInstallDir)VCProjectDefaults\UpgradeFromVC60.vsprops"
            UseOfMFC="2"
            ATLMinimizesCRunTimeLibraryUsage="false"
            CharacterSet="2"
            >
...
```

Listing A.4: Een voorbeeld van een vcproj-bestand.

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="Build" ToolsVersion="4.0"
    xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
    <ItemGroup Label="ProjectConfigurations">
        <ProjectConfiguration Include="Debug|Win32">
            <Configuration>Debug</Configuration>
            <Platform>Win32</Platform>
        </ProjectConfiguration>
        <ProjectConfiguration Include="Release MinDependency|Win32">
            <Configuration>Release MinDependency</Configuration>
            <Platform>Win32</Platform>
        </ProjectConfiguration>
    </ItemGroup>
```

```
13      <PropertyGroup Label="Globals">
14          <ProjectGuid>{768CDFE4-7A0C-41EB-8BEF-131AF6F2AAD9}</ProjectGuid>
15          <RootNamespace>IRClient</RootNamespace>
16          <SccProjectName>
17          </SccProjectName>
18          <SccLocalPath>
19          </SccLocalPath>
20          <SccProvider>
21          </SccProvider>
22          <Keyword>AtlProj</Keyword>
23      </PropertyGroup>
24      <Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />
25  ...
```

Listing A.5: Een voorbeeld van een vcxproj-bestand.

Bijlage B

Voorgekomen fouten bij opwaardering

B.1 Build errors

C4430

error C4430: missing type specifier - int assumed. Note: C++ does not support default-int

Tot Visual C++ 2005 was het niet nodig om een type identifier op te geven als het om een integer ging. Als er ergens een type identifier ontbrak, dan werd er een integer type identifier (int) impliciet ingevoegd door de compiler. Omdat dit niet het geval was in standaard C++ werd deze feature vanaf Visual C++ 2005 verwijderd en is men verplicht om steeds type identifiers op te geven, ook al betreft het een integer.

> "*Compiler will not inject int as the default type in declarations.*
>
> *Code that is missing the type in a declaration will no longer default to type int the compiler will generate Compiler Warning C4430 or Compiler Warning (level 4) C4431. Standard C++ does not support a default int and this change will ensure that you get the type you really want.*" (Microsoft, 2005)

```
1 const CHAR_MASK = 0xFF;
```

Listing B.1: In Visual C++ 6.0 was het niet nodig om een type definition op te geven bij integers.

```
1 const int CHAR_MASK = 0xFF;
```

Listing B.2: In hogere versies dan Visual C++ 6.0 is het wel nodig om type definitions op te geven bij integers.

C2065

error C2065: 'i' : undeclared identifier

Dit heeft te maken met fout C4430 (zie B.1). Dit wordt vaak veroorzaakt door het feit dat er geen type definition werd opgegeven voor een lusvariabele omdat het om een integer gaat. Soms werd dit actief gebruikt om een `while`-lus te simuleren zoals in listing B.5 op p.131.

```
for (i = 0; i < lengte; i++)
{
    CString Hulp = AlleNummers.GetAt(i);
    TRACE(Hulp);
}
```

Listing B.3: Het ontbreken van een type definition.

```
for (int i = 0; i < lengte; i++)
{
    CString Hulp = AlleNummers.GetAt(i);
    TRACE(Hulp);
}
```

Listing B.4: De aangepaste code.

```
for (int i = 0; (i < aantal) && (WriteText(leeg)); i++);
// verdere bewerking met i;
```

Listing B.5: Het foutief gebruik van de lusvariabele.

```
int i = 0;
while((i < aantal) && (WriteText(leeg))){
    i++;
}
```

Listing B.6: De aangepaste code.

Dit werd ook een enkele keer tegengekomen bij een pointer (listing B.7 op p.131).

```
for (lpwData = (LPWORD)lpData; (LPBYTE)lpwData < ((LPBYTE)lpData)+unBlockSize;
    lpwData+=2)
{
    if (((*lpwData)&0x00FF) == (wLangId&0x00FF))
    {
        dwId = *((DWORD*)lpwData);
        return TRUE;
    }
}
```

Listing B.7: De oorpsronkelijke code.

```
for (LPWORD lpwData = (LPWORD)lpData; (LPBYTE)lpwData < ((LPBYTE)lpData)+unBlockSize;
    lpwData+=2)
// LPWORD type identifier toegevoegd
{
    if (((*lpwData)&0x00FF) == (wLangId&0x00FF))
    {
        dwId = *((DWORD*)lpwData);
        return TRUE;
    }
}
```

Listing B.8: De aangepaste code.

C2259

error C2259: 'CException' : cannot instantiate abstract class

Deze fout komt boven bij een verkeerde implementatie van foutafhandeling in MFC. Er wordt geprobeerd (vaak in het `catch`-blok) een `CException`-object te initialiseren, maar dit is een abstracte klasse. Blijkbaar werd deze fout door de Visual C++ 6.0 compiler nooit gedetecteerd. Nochtans worden er in MFC enkel `CException`-pointers opgeworpen bij fouten.

Deze fout kan worden verholpen door de initialisatie van het `Cexception`-object om te zetten naar een pointerdeclaratie. Alhoewel dit in MFC-projecten zelden gebeurt, moeten deze pointers ook weer worden verwijderd (tenzij deze doorgegeven worden). Hiervoor mag echter niet het kernwoord `delete` worden gebruikt. Om een `CException`-pointer te verwijderen, moet de `Delete`-methode worden aangeroepen op het pointerobject.

```
catch(CException CEx)
{
    TCHAR   szCause[255];
    CString strFormatted;
    CEx.GetErrorMessage(szCause, 255);

    strFormatted = _T("CException : ");
    strFormatted += szCause;

    TRACE_AND_LOG(strFormatted);
    TRACE_AND_LOG(_T("\n"));

    *pLastError = CZAM_VIC107_ERROR_MSG_USER_ERROR_RETREIVING_MESSAGE_01;
        TRACE_AND_LOG(CZAM_VIC107_ERROR_MSG_USER_ERROR_RETREIVING_MESSAGE_01)
    return FALSE;
}
```

Listing B.9: Het verkeerd gebruik van een `Cexception`-object.

```
catch(CException * CEx)
{
    TCHAR   szCause[255];
    CString strFormatted;
    CEx->GetErrorMessage(szCause, 255);

    strFormatted = _T("CException : ");
    strFormatted += szCause;

    TRACE_AND_LOG(strFormatted);
    TRACE_AND_LOG(_T("\n"));

    *pLastError = CZAM_VIC107_ERROR_MSG_USER_ERROR_RETREIVING_MESSAGE_01;
        TRACE_AND_LOG(CZAM_VIC107_ERROR_MSG_USER_ERROR_RETREIVING_MESSAGE_01)

    CEx->Delete();

    return FALSE;
}
```

Listing B.10: Het verkeerd juist van een `Cexception`-object.

C2240

from 'unsigned char' to 'CString'

error C2440: 'type cast' : cannot convert from 'unsigned char' to 'CString'

Er wordt geprobeerd een variabele van het type '`unsigned char`' te casten naar een variabele van het type `CString` om een `CString`-variabele in te vullen. Dit wordt niet ondersteund. Wel kan dit met het type `char` in plaats van het type '`unsigned char`'.

```
unsigned char byte = (TCHAR)((*pRD)[iOffset + i]);
CString string = (CString)byte;
```

Listing B.11: Een foute cast van '`unsigned char`' naar `CString`.

```
unsigned char byte = (TCHAR)((*pRD)[iOffset + i]);
CString string = (char)byte;
```

Listing B.12: De aangepaste code.

from 'void (__thiscall CDlgSelectionNew::*)(UINT,LONG)' to 'LRESULT (__thiscall CWnd::*)(WPARAM,LPARAM)'

error C2440: 'static_cast' : cannot convert from 'void (__thiscall CDlgSelectionNew::)(UINT,LONG)' to 'LRESULT (__thiscall CWnd::*)(WPARAM,LPARAM)'*

Dit is het gevolg van een breaking change in ATL en MFC 7.0 in verband met de `ON_MESSAGE`-macro in `MESSAGE_MAP`-structuren.

> "*The function parameter in the `ON_MESSAGE` macro must match the type `afx_msg LRESULT (CWnd::*)(WPARAM, LPARAM)`.*" (Microsoft, 2008a)

```
BEGIN_MESSAGE_MAP(CDlgSelectionNew, CDialog)
    //{{AFX_MSG_MAP(CDlgSelectionNew)
    ON_CBN_SELCHANGE(IDC_COMBO_SEARCHKEY, OnSelchangeComboSearchkey)
    ON_EN_CHANGE(IDC_EDIT_SEARCH, OnChangeEditSearch)
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_CAPCO_GRID_LCLICK, OnLClickGrid) // error
    ON_MESSAGE(WM_CAPCO_GRID_LDBCLICK, OnLDblClickGrid) // error
END_MESSAGE_MAP()
```

Listing B.13: De `MESSAGE_MAP`-structuur.

```
void CDlgSelectionNew::OnLClickGrid(UINT wParam, LONG lParam)
{
    GetDlgItem(IDC_EDIT_SEARCH)->SetFocus();
    return 1;
}

void CDlgSelectionNew::OnLDblClickGrid(UINT wParam, LONG lParam)
{
    OnOK();
    return 1;
}
```

Listing B.14: De bijhorende eventhandlers.

Binnenin de `ON_MESSAGE`-macro is er een cast vervangen door een static cast waardoor de cast verstrengd is.

```
// from afxmsg_.h VC++ 6
#define ON_MESSAGE(message, mbrFn) \
    {message, 0, 0, 0, AfxSig_lwl, \
        (AFX_PMSG)(AFX_PMSGW)(LRESULT \
        (AFX_MSG_CALL CWnd::*)(WPARAM, LPARAM))&mbrFn },
```

Listing B.15: De Visual C++6.0 `ON_MESSAGE`-macro.

```
// for Windows messages, from afxmsg_.h
#define ON_MESSAGE(message, memberFxn) \
    { message, 0, 0, 0, AfxSig_lwl, \
        (AFX_PMSG)(AFX_PMSGW) \
        (static_cast< LRESULT (AFX_MSG_CALL CWnd::*)(WPARAM, LPARAM) > \
        (memberFxn)) },
```

Listing B.16: De vernieuwde `ON_MESSAGE`-macro.

Dit komt vaak voor om fouten beter te vermijden en duidelijker aan de programmeur te kunnen melden wat en waar er iets is fout gegaan. Foute casts kunnen zeer moeilijk op te sporen zijn. Door de static cast wordt het meteen duidelijk bij compileren dat er een pointer is meegegeven dat wijst naar een fout type.

> "*Explicit type conversion, often called casting, is occasionally essential. However, traditionally it is seriously overused and a major source of errrors. The* `static_cast` *operator converts between related types such as one pointer type to another in the same class hierarchy, an integral type to an enumeration, or a floating-point type to an integral type. (...) This distinction allows the compiler to apply some minimal type checking for* `static_cast` *(...).*" (Stroustrup, 2000)

Er zijn twee manieren om dit probleem op te lossen: in de `MESSAGE_MAP`-structuur of in de implementaties.

In de `MESSAGE_MAP`-structuur kan er ofwel gecast worden naar "`LRESULT(AFX_MSG_CALL CWnd::*)(WPARAM, LPARAM))`", ofwel de macro `ON_MESSAGE_VOID` worden gebruikt.

```
BEGIN_MESSAGE_MAP(CDlgSelectionNew, CDialog)
    //{{AFX_MSG_MAP(CDlgSelectionNew)
    ON_CBN_SELCHANGE(IDC_COMBO_SEARCHKEY, OnSelchangeComboSearchkey)
    ON_EN_CHANGE(IDC_EDIT_SEARCH, OnChangeEditSearch)
    //}}AFX_MSG_MAP
    ON_MESSAGE(WM_CAPCO_GRID_LCLICK, (LRESULT(AFX_MSG_CALL CWnd::*)(WPARAM,
        LPARAM))OnLClickGrid)
    ON_MESSAGE(WM_CAPCO_GRID_LDBCLICK, (LRESULT(AFX_MSG_CALL CWnd::*)(WPARAM,
        LPARAM))OnLDblClickGrid)
END_MESSAGE_MAP()
```

Listing B.17: De eerste oplossing met behulp van casts.

```
BEGIN_MESSAGE_MAP(CDlgSelectionNew, CDialog)
    //{{AFX_MSG_MAP(CDlgSelectionNew)
    ON_CBN_SELCHANGE(IDC_COMBO_SEARCHKEY, OnSelchangeComboSearchkey)
    ON_EN_CHANGE(IDC_EDIT_SEARCH, OnChangeEditSearch)
    //}}AFX_MSG_MAP
    ON_MESSAGE_VOID(WM_CAPCO_GRID_LCLICK, OnLClickGrid)
    ON_MESSAGE_VOID(WM_CAPCO_GRID_LDBCLICK, OnLDblClickGrid)
END_MESSAGE_MAP()
```

Listing B.18: De tweede oplossing met behulp van de macro `ON_MESSAGE_VOID`.

Voor het aanpassen van implementaties moet enkel het returntype worden aangepast naar een `LRESULT`. De waarde dat de returnvariabele moet bevatten hangt af van de methode en de context waarin deze wordt gebruikt.

```
1  LRESULT CDlgSelectionNew::OnLClickGrid(UINT wParam, LONG lParam)
2  {
3      GetDlgItem(IDC_EDIT_SEARCH)->SetFocus();
4      return 0L;
5  }
6
7  LRESULT CDlgSelectionNew::OnLDblClickGrid(UINT wParam, LONG lParam)
8  {
9      OnOK();
10     return 0L;
11 }
```

Listing B.19: De aangepaste eventhandlers.

from 'long' to 'ATL::CStringT<BaseType,StringTraits>'

error C2440: 'initializing' : cannot convert from 'long' to 'ATL::CStringT<BaseType,StringTraits>'

Een `long`-variabele wordt verkeerd gecast naar een `CString`-variabele. Er werd beroep gedaan op een implicite casts, maar deze geldt niet meer. Via een truc met de `Format`-methode van `CString` kan de `long`-variabele toch worden ingelezen.

```
1  CString tempPersoneelsNummer = (*It).m_PersoneelID;
```

Listing B.20: Een verkeerde impliciete cast.

```
1  CString tempPersoneelsNummer;
2  tempPersoneelsNummer.Format("%ld",(*It).m_PersoneelID);
```

Listing B.21: Een truc om de `long`-variabele in de `CString`-variabele te steken

from 'System::Nullable<T>' to 'long'

error C2440: '=' : cannot convert from 'System::Nullable<T>' to 'long'

Deze fout kwam voor bij het aanroepen van het Visual C# project IRentServer in het managed Visual C++ startproject iRent in iRent.Classic. De variabele '`am->AddressId`' is een *nullable* `int`. Dit type is een `int`-type dat naast gehele getallen ook de waarde `null` kan bevatten. In Visual C++ bestaat dit type niet, dus treedt er een fout op in lijn 10 in listing B.22 op p.135. Dit kan opgelost worden door de waarde van de variabele '`am->AddressId`' op te halen met de `Value`-methode op de variabele '`am->AddressId`' (listing B.23 op p.136).

```
1  CString strhulp2 = m_pDoc->m_OrderGandagas->strClient;
2  GandaFactSelectionDlg::DevExpressSelectionForm^ DlgObject
3      =   gcnew GandaFactSelectionDlg::DevExpressSelectionForm(
4              GandaFactSelectionDlg::SelectionType::CLIENTKV, gcnew
                   System::String(strhulp2)
5          );
6          if(DlgObject->ShowDialog() == System::Windows::Forms::DialogResult::OK){
7              GandaFactSelectionDlg::TankModel^ am
8                  = (GandaFactSelectionDlg::TankModel^)DlgObject->Result;
9              m_pDoc->m_OrderGandagas->strTankID = am->TankId;
10             m_pDoc->m_OrderGandagas->lLeveringsadres = am->AddressId; // FOUT
11             GetKlantLeveringsGegevens(m_pDoc->m_OrderGandagas->lLeveringsadres);
12             UpdateData(FALSE);
13         }
```

Listing B.22: Het gebruik van de variabele '`am->AddressId`' zonder de methode `Value`.

```
CString strhulp2 = m_pDoc->m_OrderGandagas->strClient;
GandaFactSelectionDlg::DevExpressSelectionForm^ DlgObject
    =   gcnew GandaFactSelectionDlg::DevExpressSelectionForm(
            GandaFactSelectionDlg::SelectionType::CLIENTKV, gcnew
                System::String(strhulp2)
        );
        if(DlgObject->ShowDialog() == System::Windows::Forms::DialogResult::OK){
            GandaFactSelectionDlg::TankModel^ am
                = (GandaFactSelectionDlg::TankModel^)DlgObject->Result;
            m_pDoc->m_OrderGandagas->strTankID = am->TankId;
            m_pDoc->m_OrderGandagas->lLeveringsadres = (am->AddressId).Value;
            GetKlantLeveringsGegevens(m_pDoc->m_OrderGandagas->lLeveringsadres);
            UpdateData(FALSE);
        }
```

Listing B.23: Het gebruik van de variabele '`am->AddressId`' met de methode `Value`.

C2593

error C2593: 'operator ==' is ambiguous

In Visual C++ bestaan er verschillende types voor string-variabelen: `char*`, `wchar_t*`, `_bstr_t`, `CComBSTR`, `CString`, `basic_string`, `String`, `BSTR`, ... In listing B.24 op p.136 worden de types `CString` en `_bstr_t` met elkaar vergeleken met de "`==`"-operator. Meerdere definities van de "`==`"-operator zijn in dit geval geldig en de compiler weet niet welke hij moet kiezen. Om ervoor te zorgen dat de compiler weet welke definitie te gebruiken wordt de `CString`-variabele gecast naar het type `LPCTSTR`.

De vergelijking is dan mogelijk omdat beide types in dit project steunen op het type `wchar_t*`. Dit is enkel het geval als de instelling "Character Set" van het project ingesteld staat op "Use Unicode Character Set". In dat geval steunt het type `BSTR` (het type `_bstr_t` incapsuleert het type `BSTR`) op het type `wchar_t*`. Als het daarentegen op "Use Multi-Byte Character Set" staat ingesteld, steunt het type `BSTR` op het type `char*`. In het laatste geval zijn de types `BSTR` (en dus ook `_bstr_t`) en `LPCTSTR` incompatibel.

```
_variant_t v = Index;
_bstr_t type = v; //type
VariantInit(pVal);
pVal->vt = VT_DISPATCH;
long c = m_coll.size();
BSTR hulp;
CString txt; //txt
for (long i = 0; i < c; i++)
{
    (m_coll[i])->get_Type(&hulp);
    txt = hulp;
    if (txt == type)
    //...
```

Listing B.24: Een foute vergelijking tussen de verschillende string-types `CString` en `_bstr_t`.

```
_variant_t v = Index;
_bstr_t type = v; //type
VariantInit(pVal);
pVal->vt = VT_DISPATCH;
long c = m_coll.size();
BSTR hulp;
CString txt; //txt
for (long i = 0; i < c; i++)
{
```

```
10      (m_coll[i])->get_Type(&hulp);
11      txt = hulp;
12      if ((LPCTSTR)txt == type) // FIXED
13      //...
```

C2664

from 'std::_Vector_iterator<_Ty,_Alloc>' to 'CollComPtr<Intf> *'

*error C2664: 'CollImplEx<Intf,MainIntf>::ExtraErase' : cannot convert parameter 1 from 'std::_Vector_iterator<_Ty,_Alloc>' to 'CollComPtr<Intf> *'*

Deze fout werd veroorzaakt door een breaking change in de Standard C++ Library van Visual C++ 2003. Vanaf versie 2003 werden iterators niet meer geïmplementeerd als pointers. In Visual C++ 6.0 was dit wel nog het geval en werd daar ook vaak gebruik van gemaakt zoals in listing B.25 op p.137. Dit kan opgelost worden door de waarde van de iterator op te halen, een referentie naar die waarde te genereren en die referentie expliciet te casten.

```
1  // it is een iterator
2  if (! ExtraErase(it) )
3  {
4      return E_FAIL;
5  }
```

Listing B.25: De oorspronkelijke code.

```
1  // it is een iterator
2  if (! ExtraErase( (CollComPtr<Intf>*) (&(*it)) ) )
3  {
4      return E_FAIL;
5  }
```

Listing B.26: De aangepaste code.

from 'const CollComPtr<Intf> *' to 'ITransito **'

*error C2664: '_CopyVectorItem<Intf>::copy' : cannot convert parameter 2 from 'const CollComPtr<Intf> *' to 'ITransito **'*

Deze fout wordt opgeworpen in het header-bestand *atlcom.h* van Microsoft (listing B.27 op p.137).

```
1      STDMETHOD(get_Item)(long Index, ItemType* pvar)
2      {
3          //Index is 1-based
4          if (pvar == NULL)
5              return E_POINTER;
6          if (Index < 1)
7              return E_INVALIDARG;
8          HRESULT hr = E_FAIL;
9          Index--;
10         CollType::const_iterator iter = m_coll.begin();
11         while (iter != m_coll.end() && Index > 0)
12         {
13             iter++;
14             Index--;
```

```
15          }
16          if (iter != m_coll.end())
17              hr = CopyItem::copy(pvar, &*iter);
18              /* exception */
19          return hr;
20      }
```

Listing B.27: De methode in het header-bestand *atlcom.h* waarin de fout optreed.

Maar de echte fout zit in de eigen code (listing B.28 op p.138).

```
1  template <class ElmtIntf> class CICORDACopyVectorItem {
2  public:
3      static HRESULT copy(VARIANT* p1, ElmtIntf** p2)
4      {
5          CComVariant var = *p2;
6          return VariantCopy(p1, &var);
7      }
8      static void init(VARIANT* p) {p->vt = VT_EMPTY;}
9      static void destroy(VARIANT* p) {VariantClear(p);}
10 };
11
12 template <class ElmtIntf> class CICORDACollComPtr : public CComPtr<ElmtIntf>
13 {
14 public:
15     CICORDACollComPtr() : CComPtr<ElmtIntf>( ){};
16     CICORDACollComPtr(ElmtIntf* lp) : CComPtr<ElmtIntf>(lp){};
17     CICORDACollComPtr(const CComPtr< ElmtIntf >& lp) : CComPtr<ElmtIntf>(lp){};
18     CICORDACollComPtr(const CICORDACollComPtr< ElmtIntf >& lp) :
           CComPtr<ElmtIntf>(lp){};
19     ElmtIntf** operator&()
20     {
21         //maak dat er geen assert komt...
22         return &p;
23     };
24 };
25
26 #define CollImpl(ElmtIntf, I) ICollectionOnSTLImpl<I,
       std::vector<CICORDACollComPtr<ElmtIntf> >, VARIANT, \
27     CICORDACopyVectorItem<ElmtIntf>, CComEnumOnSTL<IEnumVARIANT, &IID_IEnumVARIANT,
           VARIANT, CICORDACopyVectorItem<ElmtIntf>,
           std::vector<CICORDACollComPtr<ElmtIntf> > > >
```

Listing B.28: De eigen code die de fout veroorzaakt.

Er wordt een `ICollectionOnSTLImpl`-object aangemaakt. Aan dergelijk object mag er een `CopyItem` worden meegegeven (listing B.29 op p.138). Door deze implementatie kunnen de `ICollectionOnSTLImpl`-objecten gebruikt worden voor diverse klassetypes. `CopyItem` is een klasse met methodes waarin wordt gedefinieerd wat er moet gebeuren bij het aanmaken, kopiëren en vernieten van een `Item` in het `ICollectionOnSTLImpl`-object.

```
1  template <class T, class CollType, class ItemType, class CopyItem, class EnumType>
2  class ICollectionOnSTLImpl : public T
```

Listing B.29: Het `ICollectionOnSTLImpl`-object.

In listing B.27 op p.137 lijn 17 verwacht men eigenlijk als eerste parameter een '`ItemType*`'-pointer en als tweede parameter een constante pointer naar het inhoudstype van de meegegeven STL-container (`CollType` in listing B.29 op p.138). Dit is dus een parameter van het type '`const CICORDACollComPtr<ElmtIntf>*`'.

Om ervoor te zorgen dat de compiler het parametertype '`const CICORDACollComPtr<ElmtIntf>*`' begrijpt, moet er een forward declaration worden

aangebracht of de volgorde van de klassen `CICORDACollComPtr` en `CICORDACopyVectorItem` worden omgewisseld zoals in listing B.30 op p.139.

```
template <class ElmtIntf> class CICORDACollComPtr : public CComPtr<ElmtIntf>
{
public:
    CICORDACollComPtr() : CComPtr<ElmtIntf>( ){};
    CICORDACollComPtr(ElmtIntf* lp) : CComPtr<ElmtIntf>(lp){};
    CICORDACollComPtr(const CComPtr< ElmtIntf >& lp) : CComPtr<ElmtIntf>(lp){};
    CICORDACollComPtr(const CICORDACollComPtr< ElmtIntf >& lp) :
        CComPtr<ElmtIntf>(lp){};
    ElmtIntf** operator&()
    {
        //maak dat er geen assert komt...
        return &p;
    };
};

template <class ElmtIntf> class CICORDACopyVectorItem {
public:
    static HRESULT copy(VARIANT* p1, const CICORDACollComPtr<ElmtIntf>* p2)
    {
        CComVariant var = *p2;
        return VariantCopy(p1, &var);
    }
    static void init(VARIANT* p) {p->vt = VT_EMPTY;}
    static void destroy(VARIANT* p) {VariantClear(p);}
};

#define CollImpl(ElmtIntf, I) ICollectionOnSTLImpl<I,
    std::vector<CICORDACollComPtr<ElmtIntf> >, VARIANT, \
    CICORDACopyVectorItem<ElmtIntf>, CComEnumOnSTL<IEnumVARIANT, &IID_IEnumVARIANT,
        VARIANT, CICORDACopyVectorItem<ElmtIntf>,
        std::vector<CICORDACollComPtr<ElmtIntf> > > >
```

Listing B.30: De aangepaste code.

from 'const CObject *' to 'CObject *'

*error C2664: 'POSITION CObList::AddTail(CObject *)' : cannot convert parameter 1 from 'const CObject *' to 'CObject *'*

Hier gaat het om een simpele cast fout. De methode `AddTail` aanvaardt enkel '`CObject*`'-objecten, maar de methode `GetNext` geeft een '`const CObject*`'-object terug. Dit wordt opgelost door te casten naar het type `CObject*`.

```
m_UndoList.AddTail(rhs.m_UndoList.GetNext(pos));
```

Listing B.31: De oorspronkelijke code.

```
m_UndoList.AddTail((CObject*)rhs.m_UndoList.GetNext(pos));
```

Listing B.32: De aangepaste code.

from 'System::String ^ ' to 'LPCTSTR'

error C2664: 'CWnd::SetWindowTextA' : cannot convert parameter 1 from 'System::String ^ ' to 'LPCTSTR'

Deze fout kwam voor bij het aanroepen van het Visual C# project IRentServer in het managed Visual C++ startproject iRent in iRent.Classic. '`am->Nickname`' is een '`System::String^`'-object, maar de methode `SetWindowText` vraagt een `LPCTSTR`-object. Er moet een beroep worden gedaan op een '`msclr::interop::marshal_context`'-object om de variabele '`am->Nickname`' van het type '`System::String^`' om te zetten naar het type `LPCTSTR` (listing B.34 op p.140).

```
GandaFactSelectionDlg::DevExpressSelectionForm^ DlgObject
    =   gcnew GandaFactSelectionDlg::DevExpressSelectionForm(
            GandaFactSelectionDlg::SelectionType::TRANSPORTER
        );
    if(DlgObject->ShowDialog() == System::Windows::Forms::DialogResult::OK){
        GandaFactSelectionDlg::TransporterModel^ am
            = (GandaFactSelectionDlg::TransporterModel^)DlgObject->Result;
        receiver->SetWindowText(am->Nickname);
        //...
```

Listing B.33: Het gebruik van de variabele '`am->Nickname`' zonder het '`msclr::interop::marshal_context`'-object.

```
#include <msclr\marshal.h>

GandaFactSelectionDlg::DevExpressSelectionForm^ DlgObject
    =   gcnew GandaFactSelectionDlg::DevExpressSelectionForm(
            GandaFactSelectionDlg::SelectionType::TRANSPORTER
        );
    if(DlgObject->ShowDialog() == System::Windows::Forms::DialogResult::OK){
        GandaFactSelectionDlg::TransporterModel^ am
            = (GandaFactSelectionDlg::TransporterModel^)DlgObject->Result;
        msclr::interop::marshal_context context;
        receiver->SetWindowText(context.marshal_as<const TCHAR*>(am->Nickname));
        //...
```

Listing B.34: Het (foutieve) gebruik van de variabele '`am->Nickname`' met het '`msclr::interop::marshal_context`'-object.

Meer informatie hierover bevindt zich in hoofdstuk 7.4.3 op p.66.

from 'unsigned short **' to 'LPOLESTR *'

*error C2664: 'ATL::IDispatchImpl<T,piid,plibid>::GetIDsOfNames' : cannot convert parameter 2 from 'unsigned short **' to 'LPOLESTR *'*

Deze fout wordt gegenereerd door het type van parameter `pps` in listing B.35 op p.140. De fout kan opgelost worden door het wijzigen van het type van de parameter `pps` van '`unsigned short**`' naar `LPOLESTR*`. Dit kan omdat het type `LPOLESTR` gelijk is aan het type `WCHAR*`, dat op zijn beurt gelijk is aan het type `wchar_t*`. `wchar_t` is volgens Microsoft (2012t) door MIDL gedefinieerd als een 16 bit `unsigned short` data object.

```
virtual long __stdcall GetIDsOfNames(const struct _GUID &rg,unsigned short **
    pps,unsigned int i,unsigned long l,long * pl){
    return CIDOI::GetIDsOfNames(rg, (LPOLESTR*)pps, i, l, pl);
};
```

Listing B.35: De originele code, waarbij de parameter `pps` van het type '`unsigned short**`' is.

```
virtual long __stdcall GetIDsOfNames(const struct _GUID &rg,LPOLESTR* pps,unsigned
    int i,unsigned long l,long * pl){
```

```
2     return CIDOI::GetIDsOfNames(rg, pps, i, l, pl);
3 };
```

Listing B.36: De code zonder foutmelding, waarbij de parameter `pps` van het type `LPOLESTR*` is.

from 'unsigned short **' to 'BSTR *'

*error C2664: 'CIICORDADataObjectImpl<T,pclsid,I,piid,plibid>::get_LastError' : cannot convert parameter 1 from 'unsigned short **' to 'BSTR *'*

Deze fout is analoog aan "*error C2664: 'ATL::IDispatchImpl<T,piid,plibid>::GetIDsOfNames' : cannot convert parameter 2 from 'unsigned short **' to 'LPOLESTR *'*" en kan ook op dezelfde manier worden opgelost, alleen wordt het type omgezet naar `BSTR*` in plaats van `LPOLESTR*` (listing B.38 op p.141). Dit kan omdat, net als het type `LPOLESTR`, ook het type `BSTR` neerkomt op het type `wchar_t*` en volgens Microsoft (2012t) het type `wchar_t` door MIDL is gedefinieerd als een 16 bit `unsigned short` data object.

```
1 virtual long __stdcall get_LastError(unsigned short ** ppus){
2     return CIDOI::get_LastError((BSTR*)ppus);
3 }; // Martijn
```

Listing B.37: De originele code, waarbij de parameter `pps` van het type '`unsigned short**`' is.

```
1     //virtual long __stdcall get_LastError(unsigned short ** ppus){return
          CIDOI::get_LastError((BSTR*)ppus);}; // Martijn
2     virtual long __stdcall get_LastError(BSTR* ppus){
3     return CIDOI::get_LastError(ppus);
4 };
```

Listing B.38: De code zonder foutmelding, waarbij de parameter `pps` van het type `BSTR*` is.

LNK1104

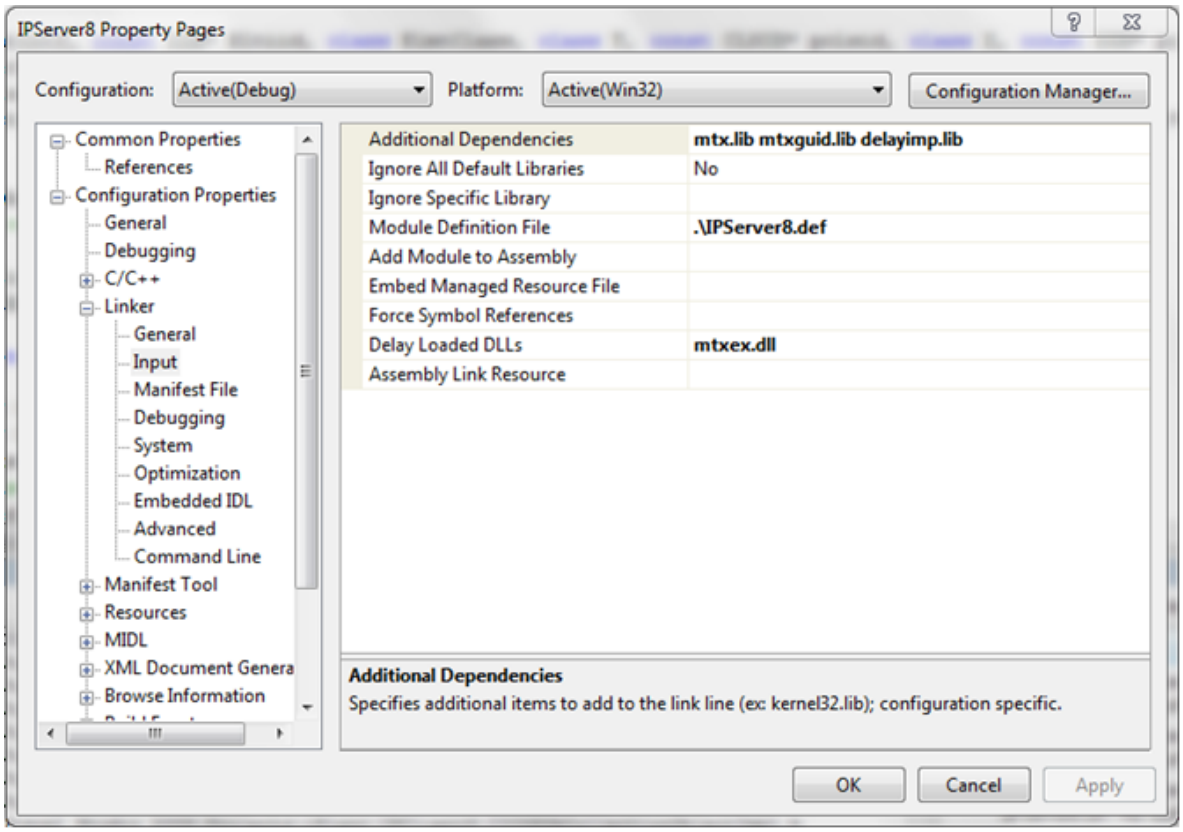
file 'mtxguid.lib'

fatal error LNK1104: cannot open file 'mtxguid.lib'

Volgens Microsoft (2001) is *mtxguid.lib* een library dat de GUID bevat van objecten en interfaces in de Microsoft Transaction Server (MTS) Application Programming Interface (API). Indien bij creatie van een ATL project ondersteuning voor MTS wordt ingeschakeld worden automatisch de bestanden *mtx.lib* en *mtxguid.lib* toegevoegd aan de optie "Additional Dependencies" in de linkeropties (figuur B.1 op p.142). In recentere versies van Visual Studio is *mtxguid.lib* niet meer aanwezig, dit mag dan ook uit de optie "Additional Dependencies" worden verwijderd.

Bij sommige solutions van ICORDA moest de solution daarna ook nog worden gecleaned. Ander werd er de error "error PRJ0019: A tool returned an error code from "Performing registration"" opgeworpen tijdens het compileren. Deze slaat op de buildstep in listing B.39 op p.141.

```
1 regsvr32 /s /c "$(TargetPath)"
2 echo regsvr32 exec. time > "$(OutDir)regsvr32.trg"
3 echo Execute mtxrereg.exe before using MTS components in MTS
```



Figuur B.1: *mtxguid.lib* in de optie "Additional Dependencies".

```
4 cscript //job:install iCorda.iRenT.wsf "$(SolutionDir)" "Debug"
```

Listing B.39: De buildstep met als beschrijving "Performing registration".

file 'opends60.lib'

error LNK1104: cannot open file 'opends60.lib'

Dit bestand is niet meer aanwezig in SQL server 2008. Het werd gebruikt om te werken met extended stored procedures. Deze feature is nu deprecated. Volgens Kent (2009) kan men dit bestand van een oudere versie van SQL server halen. Dit bestand moet dan opgenomen worden in de solution en ook alle verwijzingen naar dit bestand, zowel in de code als in de instellingen, moeten aangepast worden.

C1083

secall.h

fatal error C1083: Cannot open include file: 'secall.h': No such file or directory

Dit wordt veroorzaakt door het ontbreken van de Stingray Objective Toolkit libraries in Stingray versie 11 (waarvan *secall.h* deel uitmaakt) bij de overgang van de oude Stingray

2006 libraries naar de nieuwe Stingray 11 libraries (zie hoofdstuk 2 op p.15). ICORDA beschikt over een eigen library die dezelfde functionaliteit als de ontbrekende Stingray Objective Toolkit libraries aanbiedt. Deze heeft een bepaalde structuur en naamgeving zodat ze de Stingray Objective Toolkit klassen kunnen vervangen in enkele stappen:

1. Eerst moeten de bestanden van de ICORDA libraries worden ingeladen in het startproject van de solution.

2. De lijnen, in listing B.40 op p.143, in het bestand *StdAfx.h* van het startproject van de solution moeten vervangen worden door de lijnen in listing B.41 op p.143. Zo wordt er verwezen naar de ICORDA libraries en de geïnstalleerde Stingray Objective Grid versie 11 libraries. Die laatste hebben een andere mappenstructuur dan versie 2006, vandaar de toevoeging van "`grid/`" in listing B.41 op p.143.

```
#include "secall.h"
#include "gxall.h"
```

Listing B.40: De te vervangen lijnen in *StdAfx.h*.

```
#include "icdall.h"
#include "grid/gxall.h"
```

Listing B.41: De aangepaste lijnen in *StdAfx.h*.

3. In de namen van alle gebruikte Stingray Objective Toolkit klassen en variabelen in de solution moet de substring "SEC" vervangen worden door "ICD".

4. Sommige rc-bestanden steunen ook op de Stingray Objective Toolkit libraries. In die rc-bestanden moet de lijn "`#include "secres.h"`" worden verwijderd en de lijn "`#include "gxresrc.h`" vervangen door "`#include "grid\gxresrc.h"`".

5. Voor die rc-bestanden moeten dan ook de resource includes aangepast worden (`View > Other Windows > Resource View > rechtermuisknop rc-bestand > Resource Includes...`). Daar moet in het veld "Read-only symbol directives" de lijn "`#include "secres.h"`" verwijderd worden en de lijn "`#include "gxresrc.h"`" vervangen worden door "`#include "grid\gxresrc.h"`". In het veld "Compile-time directives" moet dan de lijn "`#include "gxres.rc"`" vervangen worden door "`#include "grid/gxres.rc"`" en binnen de '`if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_ENU)`'-structuur geplaatst worden (Zie listing B.42 op p.143 en listing B.43 op p.144.

```
// Read-only symbol directives
#include "afxres.h"
#include "secres.h"          // Objective Toolkit Resource Symbols
#include "gxresrc.h"         // Objective  Grid

// ------------------------------------------------
// Compile-time directives
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
```

```
#pragma code_page(1252)
#endif //_WIN32
#include "res\TRR2.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources
#endif

#include "secres.rc"    // Objective Toolkit Resources
#include "gxres.rc"    // Objective Toolkit Resources
```

Listing B.42: De originele lijnen in de resource includes.

```
// Read-only symbol directives
#include "afxres.h"
#include "grid/gxresrc.h"

// ------------------------------------------------
// Compile-time directives
#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif //_WIN32
#include "res\TxKassa.rc2"  // non-Microsoft Visual C++ edited resources
#include "QuickBox.rc2"
#include "afxres.rc"         // Standard components
#include "afxprint.rc"       // printing/print preview resources
#include "grid/gxres.rc"
#endif
```

Listing B.43: De aangepaste lijnen in de resource includes.

PostNummers.dll

fatal error C1083: Cannot open type library file: .\..\..\Libraries\PostNummers \ReleaseMinDependency\PostNummers.dll: No such file or directory

Het bestand *PostNummers.dll* wordt niet gevonden. Dit kan doordat het project PostNummers niet (foutloos) is gebuild of omdat het project wordt gebuild in een andere configuratie. Solutions en projecten kunnen gebuild worden in verschillende configuraties zoals Debug, Release, ... Voor elke configuratie worden de gecompileerde bestanden in een eigen folder opgeslagen. Indien de configuraties van de projecten in een solution niet consistent zijn, kan het gebeuren dat er wordt gekeken naar bestanden in de verkeerde folder en daardoor niet worden gevonden.

gxstndrd.h

error C1083: Cannot open include file: 'Grid\config\gxstndrd.h': No such file or directory

Men is in Visual Studio 2010 vergeten de property sheet "SS-Win32-PropSheet"[1] van de Stingray libraries vergeten toe te voegen aan het project. Men kan dit toevoegen via de property manager (`View > Other Windows > Property Manager > Het project openklappen > rechtsklikken op '"Debug | Win32" (of een andere modus-folder indien gewenst) > Add Existing Project Property Sheet...`) . De property sheet bevat de nodige informatie voor Visual Studio om de Stingray libraries te vinden.

[1]Dit is de 32 bit versie die verwijst naar de 32 bit libraries.

grid/gxall.h

fatal error C1083: Cannot open include file: 'grid/gxall.h': No such file or directory

De oorzaak en oplossing zijn hetzelfde als bij "*error C1083: Cannot open include file: 'Grid\config\gxstndrd.h': No such file or directory*".

C2146 & C4430

error C2146: syntax error : missing ';' before identifier 'iterator'
error C4430: missing type specifier - int assumed. Note: C++ does not support default-int

`ElmtIntf` is een typename. Dan is het keyword `typename` nodig om aan te duiden dat er typenames worden gebruikt. Deze fout werd niet opgevangen in de Visual C++ 6.0 compiler.

> "*(...) unless otherwise stated, an identifier is assumed to refer to something that is not a type or template. If we want to state that something should be treated as a type, we can do so using the* `typename` *keyword (...) The* `typename` *keyword can be placed in front of a qualified name to state that the entity named is a type. in this it resembles* `struct` *and* `class`." (Stroustrup, 2000)

```
1 typedef std::vector<CICORDACollComPtr<ElmtIntf> >::iterator iterator;
```

Listing B.44: De oorspronkelijke code.

```
1 typedef typename std::vector<CICORDACollComPtr<ElmtIntf> >::iterator iterator;
```

Listing B.45: De aangepaste code.

C2555

error C2555: 'CSECWizardTabWnd::ActivateTab': overriding virtual function return type differs and is not covariant from 'SECTabWndBase::ActivateTab'

Een virtuele functie wordt overschreven, maar de return type verschilt van de oorspronkelijke returntype en voldoet niet aan de *covariant return rule*:

> "*The type of an overriding function must be the same as the type of the virtual function it overrides, except that the return type may be relaxed. That is, if the original return type was* `B*`*, then the return typ of the overriding function may be* `D*`*, provided* `B` *is a public base of* `D`*. Similarly, a return type of* `B&` *may be relaxed to* `D&`*. This is sometimes called 'the covariant return rule.'*" (Stroustrup, 2000)

In het volgende fragment worden de virtuele methodes `ActivateTab` overschreven. Deze hebben als returntype `void`. De functie die hierbij wordt overschreven heeft echter als returntype `BOOL`. `void` is geen afgeleidde van `BOOL` en is ook niet gelijk. Er is dus niet voldaan aan de signatuur van de virtuele basisfunctie of de covariant return rule. De oplossing hierbij is om simpelweg de signatuur en body aan te passen naar het type `BOOL`.

```
1  public:
2      CObject* GetTab(const long &index);
3      CWnd* GetPreviousSelectedTabAsCWnd(int& OldIndex);
4      virtual void ActivateTab(int nIndex);
5      virtual void ActivateTab(CWnd* pWnd);
6      virtual void ActivateTab(CWnd* pWnd, int nIndex);
7      virtual ~CSECWizardTabWnd();
8
9      // Generated message map functions
10 protected:
11     virtual BOOL CreateTabCtrl(DWORD dwStyle, UINT nID);
12     //{{AFX_MSG(CSECWizardTabWnd)
13     //}}AFX_MSG
14     DECLARE_MESSAGE_MAP()
15 };
16
17 void CSECWizardTabWnd::ActivateTab(CWnd *pWnd, int nIndex)
18 {
19     // ...
20     return;
21 }
```

Listing B.46: De oorspronkelijke code.

```
1  public:
2      CObject* GetTab(const long &index);
3      CWnd* GetPreviousSelectedTabAsCWnd(int& OldIndex);
4      virtual BOOL ActivateTab(int nIndex);
5      virtual BOOL ActivateTab(CWnd* pWnd);
6      virtual BOOL ActivateTab(CWnd* pWnd, int nIndex);
7      virtual ~CSECWizardTabWnd();
8
9      // Generated message map functions
10 protected:
11     virtual BOOL CreateTabCtrl(DWORD dwStyle, UINT nID);
12     //{{AFX_MSG(CSECWizardTabWnd)
13     //}}AFX_MSG
14     DECLARE_MESSAGE_MAP()
15 };
16
17 BOOL CSECWizardTabWnd::ActivateTab(CWnd *pWnd, int nIndex)
18 {
19     //...
20     return false;
21 }
```

Listing B.47: De aangepaste code.

C2248

private member declared in class 'ATL::_NoAddRefReleaseOnCComPtr<T>'

error C2248: 'ATL::_NoAddRefReleaseOnCComPtr<T>::Release' : cannot access private member declared in class 'ATL::_NoAddRefReleaseOnCComPtr<T>'

Deze lijn wordt veroorzaakt door het proberen aanroepen van de `Release`-methode van de interface bevat in een `CComPtr`, in plaats van het aanroepen van de `Release`-methode op de `CComPtr` zelf. In listing B.48 op p.147 moet in lijn 3 de "->"-operator vervangen worden door de "."-operator, zoals in listing B.49 op p.147.

```
if(m_piFields)
{
    m_piFields->Release();
    m_piFields = NULL;
}
```

Listing B.48: Het foutief aanroepen van de `Release`-methode van de interface terwijl men die van de `CComPtr` wil aanroepen.

```
if(m_piFields)
{
    m_piFields.Release();
    m_piFields = NULL;
}
```

Listing B.49: Het juist aanroepen van de `Release`-methode.

private typedef declared in class 'CIICORDADataObjectImpl<T,pclsid,I,piid,plibid>'

error C2248: 'CIICORDADataObjectImpl<T,pclsid,I,piid,plibid>::CIDOI' : cannot access private typedef declared in class 'CIICORDADataObjectImpl<T,pclsid,I,piid,plibid>'

De nodige `typedef` is enkel gedefinieerd in het `private`-gedeelte van de klasse. De oplossing is om de definitie te verplaatsen naar het `public`-gedeelte.

```
template <class T , const CLSID* pclsid,  class I ,  const IID* piid , const GUID*
    plibid /* = &CComModule::m_libid*/  > class ATL_NO_VTABLE CIICORDADataObjectImpl
    : public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<T, pclsid>,
    public IDispatchImpl<I, piid, plibid>
{
    typedef CIICORDADataObjectImpl<T, pclsid, I, piid, plibid> CIDOI;
public:

    //...
}
```

Listing B.50: De oorspronkelijke locatie van de `typedef`.

```
template <class T , const CLSID* pclsid,  class I ,  const IID* piid , const GUID*
    plibid /* = &CComModule::m_libid*/  > class ATL_NO_VTABLE CIICORDADataObjectImpl
    : public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<T, pclsid>,
    public IDispatchImpl<I, piid, plibid>
{
    //typedef CIICORDADataObjectImpl<T, pclsid, I, piid, plibid> CIDOI;
public:
    typedef CIICORDADataObjectImpl<T, pclsid, I, piid, plibid> CIDOI;
    //...
}
```

Listing B.51: De aangepaste code.

C2600

error C2600: 'CSECWizardTabWnd::~CSECWizardTabWnd' : cannot define a compiler-generated special member function (must be declared in the class first)

De deconstructor was niet gedefinieerd in de klasse zelf, maar wel geïmplementeerd.

Als de compiler geen declaratie voor een destructor vindt zal die zelf impliciet een deconstructor declareren en implementeren. Er werd dus geprobeerd de door compiler gegenereerde destructor te implementeren, terwijl de compiler dit al doet. De oplossing is om zelf de deconstructor te declareren in de klasse. Zo ziet de compiler dat de deconstructor niet meer impliciet moet worden aangemaakt en zijn er geen conflicten.

C3876

error C3867: 'CIPData3::GetLabCategoriesIntern': function call missing argument list; use '&CIPData3::GetLabCategoriesIntern' to create a pointer to member

Door een breaking change in Visual C++, om tegemoet te komen aan standaard c++, moeten pointers naar functieleden vergezeld zijn met de adresoperator (`&`) en de volledige gequalificeerde naam.

> "*Pointer-to-members now require qualified name and &*
> *Code written for previous versions of the compiler that just used the method name will now give Compiler Error C3867 or Compiler Warning C4867. This diagnostic is required by Standard C++. Now, to create a pointer to a member function, the address of operator (&) must be used with the fully qualified name of the method. Not having to use the & operator and the fully qualified name of the method can lead to logic bugs in code due missing parentheses in function calls. Using the function's name without an argument list results in a function pointer which is convertible to several types. This code would have compiled, leading to unexpected behavior at runtime.*" (Microsoft, 2005)

```
1 return SaveRSExternWithoutRefresh(ConnectionStringFromClient, padoRS,
      GetLabSubstratesIntern, L"usp_LabSubstratePut", L"usp_LabSubstrateDelete", Parms,
      AantParms(Parms), L"@LabSubstrateID");
```

Listing B.52: De oorspronkelijke code.

```
1 return SaveRSExternWithoutRefresh(ConnectionStringFromClient, padoRS,
      &CIPData3::GetLabSubstratesIntern, L"usp_LabSubstratePut",
      L"usp_LabSubstrateDelete", Parms, AantParms(Parms), L"@LabSubstrateID");
```

Listing B.53: De aangepaste code.

Indien in listing B.53 op p.148 de prefix "`CIPData3::`" niet aanwezig is, maar enkel de adresoperator (`&`), wordt de fout "*error C2276: '&' : illegal operation on bound member function expression*" opgeworpen.

C1189

This file requires _WIN32_WINNT to be #defined at least to 0x0403

error C1189: #error : This file requires _WIN32_WINNT to be #defined at least to 0x0403. Value 0x0501 or higher is recommended.

De macro `_WIN32_WINNT`, in het bestand *StdAfx.h* van het project, wordt gebruikt om aan te geven wat de laagste Windows versie is waarop dit programma wordt ondersteund.

Omdat de applicatie wordt opgewaardeerd, worden versies lager dan 0x0403 niet meer ondersteund. De waarde van de macro moet dus aangepast worden naar 0x0501 (Windows XP) zoals in listing B.55 op p.149.

```
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0400
#endif
```

Listing B.54: Het aangeven van de laagste Windows versie waarvoor dit programma is gemaakt.

```
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0501
#endif
```

Listing B.55: Het aangeven van de laagste Windows versie waarvoor dit programma is gemaakt (Windows XP).

_WIN32_WINNT settings conflicts with _WIN32_IE setting

fatal error C1189: #error : _WIN32_WINNT settings conflicts with _WIN32_IE setting

Deze fout heeft dezelfde oorsprong en oorzaak als "*error C1189: #error : This file requires _WIN32_WINNT to be #defined at least to 0x0403. Value 0x0501 or higher is recommended.*"

C1853

fatal error C1853: '.\Debug/CZAM.pch' precompiled header file is from a previous version of the compiler, or the precompiled header is C++ and you are using it from C (or vice versa)

Het pch-bestand is nog een oud bestand. Door de oplossing te cleanen (of rebuilden) wordt dit bestand verwijderd.

c1010001

manifest authoring error c1010001: Values of attribute "level"not equal in different manifest snippets.

Het betekent dat de waarden van de optie "UAC Execution Level" in de manifest-bestanden van projecten in een solution niet overeenkomen. Dit kan eenvoudig opgelost worden door deze optie overal op dezelfde waarde in te stellen (`Rechtermuisknop op project > Properties > Configuration Properties > Linker > Manifest File > UAC Execution Level`).

C2668

error C2668: 'copy_if' : ambiguous call to overloaded function

In het project is er een functie `copy_if` gedefinieerd. Microsoft heeft ondertussen zelf een functie `copy_if` toegevoegd in de header `algorithm`. Deze functies gaan met elkaar

in conflict. Omdat niet kan aangetoond worden dat de functies dezelfde functionaliteit aanbieden en hetzelfde resultaat weergeven wordt de zelfgeïmplementeerde methode hernoemd naar `icdcopy_if` doorheen het hele project.

LNK2005 & LNK1169

error LNK2005: _atof already defined in ICORDADoubleHelpers.obj
error LNK1169: one or more multiply defined symbols found

Net als de functie `copy_if`, is er in het project een functie `_ttof` gedefinieerd. Microsoft heeft ondertussen zelf een functie `_atof` gedefinieerd. Omdat hier aangetoond werd dat de functies gelijkaardig zijn en hetzelfde resultaat geven werd de declaratie en implementatie van de zelfgeschreven functie `_ttof` uitgecommentariëerd. De foutmelding geeft de functie `atof` weer. Eigenlijk is `_ttof` een macro, die afhankelijk van de waarde in de optie "Character Set" van het project, omgezet wordt naar `atof` or `_wtof`.

C2679

*error C2679: binary '=' : no operator found which takes a right-hand operand of type 'COrdersPerCustomerGridColumn *' (or there is no acceptable conversion)*

De fout werd veroorzaakt door het door elkaar gebruiken van pointers en iterators. Bij versies van Visual C++ hoger dan 6.0 zijn sommige iteratoren niet meer geïmplementeerd als pointers.

In listing B.56 op p.150 wordt er geprobeerd een datastructuur te doorlopen vanaf een bepaalde positie tot het eind van die structuur, met behulp van een `for`-lus. Om ervoor te zorgen dat de `for`-lus start vanaf een bepaalde waarde, wordt de iterator `theIteratorColumn` geïnitialiseerd met de pointer `theIteratorColumnPrice`. In Visual C++ 2003 en hoger wordt de toewijzing van de pointer aan de iterator (lijn 7 in listing B.56 op p.150) niet meer ondersteund. Om toch in de methode dezelfde functionaliteit te behouden, wordt in listing B.57 op p.151 de iterator eerst ingesteld op de beginpositie van de te overlopen datastructuur (lijn 7 in listing B.57 op p.151). Daarna wordt met behulp van een `while`-lus de iterator op de juiste beginpositie geplaatst (lijn 9 in listing B.57 op p.151). Op deze manier moet er in de `for`-lus geen pointer worden toegewezen aan de iterator, maar wordt wel de functionaliteit behouden.

```
1  void COrdersPerCustomerGrid::UpdateYearPrice(COrdersPerCustomerGridRow*
       pOrdersPerCustomerGridRow, COrdersPerCustomerGridColumn *theIteratorColumnPrice,
       double NieuwePrice)
2  {
3      ORDERSPERCUSTOMERGRIDCOLUMNVECTOR::iterator theIteratorColumn;
4
5      BOOL bFirst = TRUE;
6
7      for (theIteratorColumn = theIteratorColumnPrice; theIteratorColumn !=
           m_OrdersPerCustomerGridColumn.end(); theIteratorColumn++)
8      {
9          if (bFirst)
10         {
11             bFirst = FALSE;
12         }
13         else
14         {
```

```
15              if (theIteratorColumn->m_WeekNummer == 0)
16              {
17                  break;
18              }
19              UpdatePrice(pOrdersPerCustomerGridRow, &*theIteratorColumn,  NieuwePrice);
20          }
21      }
22  }
```

Listing B.56: Het (foutief) door elkaar gebruik van pointers en iteratoren.

```
1   void COrdersPerCustomerGrid::UpdateYearPrice(COrdersPerCustomerGridRow*
        pOrdersPerCustomerGridRow, COrdersPerCustomerGridColumn *theIteratorColumnPrice,
        double NieuwePrice)
2   {
3       //ORDERSPERCUSTOMERGRIDCOLUMNVECTOR::iterator theIteratorColumn;
4
5       BOOL bFirst = TRUE;
6
7       ORDERSPERCUSTOMERGRIDCOLUMNVECTOR::iterator theIteratorColumn =
            m_OrdersPerCustomerGridColumn.begin();
8
9       while(theIteratorColumn != m_OrdersPerCustomerGridColumn.end() &&
            (&(*theIteratorColumn)) != theIteratorColumnPrice){
10          theIteratorColumn++;
11      }
12
13      //for (theIteratorColumn = theIteratorColumnPrice; theIteratorColumn !=
            m_OrdersPerCustomerGridColumn.end(); theIteratorColumn++)
14      for (theIteratorColumn; theIteratorColumn != m_OrdersPerCustomerGridColumn.end();
            theIteratorColumn++)
15      {
16      {
17          if (bFirst)
18          {
19              bFirst = FALSE;
20          }
21          else
22          {
23              if (theIteratorColumn->m_WeekNummer == 0)
24              {
25                  break;
26              }
27              UpdatePrice(pOrdersPerCustomerGridRow, &*theIteratorColumn,  NieuwePrice);
28          }
29      }
30  }
```

Listing B.57: Het gescheiden houden van iteratoren en pointers.

'`(&(*theIteratorColumn))`' geeft de referentie weer van de waarde waar de iterator `theIteratorColumn` naar wijst. Dit zorgt ervoor dat de `while`-lus in listing B.57 op p.151 kan kijken wanneer de iterator op de gewenste startwaarde staat voor de `for`-lus.

RC1015

fatal error RC1015: cannot open include file 'toolkit\secres.h'.

Dit probleem heeft te maken met de overgang van de Stingray Objective Toolkit library naar de eigen ICORDA library. *secres.h* moet vervangen worden door *icdres.h*. Zie hoofdstuk 2 op p.15 en de uitleg van '*fatal error C1083: Cannot open include file: 'secall.h': No such file or directory*' voor meer informatie.

CVT1100

CVTRES : fatal error CVT1100: duplicate resource. type:MENU, name:20620, language:0x0409

Dit werd veroorzaakt door de lijn "`#include ïcdres.h"`" in het rc-bestand van de applicatie bij de overgang van de Stingray libraries naar de ICORDA libraries. Dit zorgde ervoor dat het bestand *icdres.h* tweemaal werd opgenomen in de applicatie. De oplossing is om in het rc-bestand de lijn te verwijderen.

LNK1123

fatal error LNK1123: failure during conversion to COFF: file invalid or corrupt

Deze fout werd veroorzaakt door dezelfde oorzaak als "*CVTRES : fatal error CVT1100: duplicate resource. type:MENU, name:20620, language:0x0409*". De oplossing is dan ook hetzelfde.

RC1004

fatal error RC1004: unexpected end of file found

Na het toevoegen van headerguards in een header-bestand moet er een lege lijn komen, anders wordt deze foutmelding weergegeven. Headerguards (lijn 1,2 en 9 in listing B.58 op p.152) zorgen ervoor dat zaken in een header-bestand slechts één keer in een project kunnen gedeclareerd worden.

```
1 #ifndef __ICDRES_H__
2 #define __ICDRES_H__
3
4 #define ICD_IDR_SHORCUTLIST_MENU        20620
5 #define ICD_ID_SHORTCUTLIST_LARGEICON   43054
6 #define ICD_ID_SHORTCUTLIST_SMALLICON   43055
7 #define IDC_TOOLBAR_HORZDRAG            20692
8
9 #endif __ICDRES_H__
```

Listing B.58: Een voorbeeld van headerguards in een header-bestand.

C2668

error C2668: 'ATL::CStringT<BaseType,StringTraits>::CStringT' : ambiguous call to overloaded function

De methode `GetValue` op de pointer `pControl` wordt in listing B.59 op p.152 verkeerd gebruikt. De methode vult een meegegeven `CString`-variabele (`s`) op met de inhoud van de gevraagde cel van de `GridView`. Het is dan ook deze `CString` die moet worden teruggegeven worden, niet de return-waarde van de `GetValue`-methode, zoals in listing B.60 op p.153. Op deze manier wordt de huidige staat van de cel bewaard.

```
1 CString CGridViewTarief::GetCellValue(ROWCOL nRow, ROWCOL nCol)
2 {
3     if (IsCurrentCell(nRow,nCol) && IsActiveCurrentCell())
```

```
4       {
5           CString s;
6           CGXControl* pControl = GetControl(nRow,nCol);
7           return pControl->GetValue(s);
8       }
9       else
10          return GetValueRowCol(nRow,nCol);
11  }
```

Listing B.59: De originele methode `GetValue`.

```
1   CString CMyGrid::GetCellValue(ROWCOL nRow, ROWCOL nCol)
2   {
3       if (IsCurrentCell(nRow, nCol) && IsActiveCurrentCell())
4       {
5           CString s;
6           CGXControl* pControl = GetControl(nRow, nCol);
7           pControl->GetValue(s);
8           return s;
9       }
10      else
11          return GetValueRowCol(nRow, nCol);
12  }
```

Listing B.60: De aangepaste methode `GetValue`.

Zonder foutcode

error : 0x2 trying to open file <IREventMessages>. Mc

De macro `$(InputName)` in Visual Studio 2005 is gewijzigd in Visual Studio 2008 naar `$(InputName)`. De build step verbonden aan het mc-bestand moet dus gewijzigd worden (`Rechtermuisknop op het mc-bestand in Visual Studio > Properties > Configuration Properties > Custom Build Tool > Command Line`). Deze fout komt ook voor bij de macro `$(Filename)` in Visual Studio 2008, dat in Visual Studio 2010 is hernoemd naar `$(Identity)`.

B.2 Warnings

C4715

warning C4715: 'CSECWizardTabWnd::ActivateTab' : not all control paths return a value

Deze foutmelding wijst erop dat niet alle mogelijke uitvoeringen van de code eindigen met een `return` statement dat een variabele (of constante) van hetzelfde type als het returntype teruggeeft. Deze waarschuwing kan dus niet worden opgegooid voor functies met als returntype `void`. De oplossing is om de uitvoeringen die geen `return` statement bevatten er van één te voorzien.

Een ander geval waarbij deze waarschuwing kan optreden zijn recursieve functies die nooit terugkeren (en dus oneindig lang verder recurseren tot alle aanwezige bronnen zoals Random-Access Memory (RAM)-geheugen zijn opgebruikt. Indien dit toch gewenst is kan de waarschuwing worden vermeden door het keyword `__declspec(noreturn)` te gebruiken in de signatuur van de recursieve functie.

C4018

warning C4018: '>' : signed/unsigned mismatch

Er wordt een signed variabele vergeleken met een unsigned variabele. Bij signed variabele wordt het eerste bit van de variabele gebruikt om aan te duiden of het om een positief of negatief getal gaat. Signed variabelen doen dit niet. Dit zorgt ervoor dat unsigned variabelen zowel negatieve als positieve getallen kunnen bevatten en unsigned enkel positieve. Zo is het bereik van een '`unsigned int`' van vier bytes [0, 4294967295] en van een '`signed int`' van vier bytes [2147483648, 2147483647]. De `signed` en `unsigned` keywords kunnen gebruikt worden bij elke numeriek type behalve `bool`.

Signed type	Unsigned type	Andere naam
`int`	`unsigned int`	
`__int8`	`unsigned __int8`	`char`
`__int16`	`unsigned __int16`	`short`
`__int32`	`unsigned __int32`	`int`
`__int64`	`unsigned __int64`	`long long`
`short`	`unsigned short`	
`long`	`unsigned long`	
`long long`	`unsigned long long`	

Tabel B.1: Een overzicht van alle signed en unsigned variabelen in Visual C++. De '`__int<nummer>`'-types staan voor types waarvan de grootte exact '`<nummer>`' bits bedraagt. (Microsoft, 2012g)

De waarschuwing wijst erop dat er problemen kunnen onstaan door het gebruik van signed en unsigned types door elkaar. Immers worden de bits door een signed type anders gelezen dan door een signed type. Zo zal in listing B.61 op p.154[2] de waarde van `b` uiteindelijk -1 bedragen. 129 wordt immers in een '`unsigned char`' opgeslagen als $(10000001)_b$. Als de '`signed char`' `b` deze bits overneemt, interpreteert deze de eerste bit als het signbit. Dus betekent $(10000001)_b$ voor b -1.

```
1 unsigned int a = 129;
2 signed int   b;
3 b = a;
```

Listing B.61: Een fout gebruik van signed en unsigned variabelen.

Dit kan vaak worden opgelost door het gebruik van de `signed` en `unsigned` keywords. In listing B.63 op p.155 is de variabele `i` een '`signed long`'-variabele en geeft de methode `size` een '`unsigned long`'-variabele terug. Om een juiste overeenstemming te krijgen moet `i` ook een '`unsigned long`' zijn zoals in listing B.62 op p.154.

```
1 for (long i = 0; i<pPD->size(); i++)
2 {
3     char nummer = (*pPD)[i];
4     hulp.Format(_T("%d /"), nummer);
5     AlleNummers += hulp;
6 }
```

Listing B.62: De oorspronkelijke code.

[2]Omdat het om een voorbeeld gaat is hier de grootte van `int` vastgelegd op acht bits.

```
for (unsigned long i = 0; i<pPD->size(); i++)
{
    char nummer = (*pPD)[i];
    hulp.Format(_T("%d /"), nummer);
    AlleNummers += hulp;
}
```

Listing B.63: De aangepaste code.

C4244

warning C4244: 'initializing' : conversion from 'double' to 'int', possible loss of data

De compiler waarschuwt voor mogelijk verlies van data. Een `double` variabele heeft een groter bereik dan een variabele van het type `int` en kan ook cijfers na de komma bevatten. Het type `int` kan geen cijfers na de komma bevatten. Dus bij conversie van een `double` naar een `int` gaat de informatie na de komma verloren. Om de compiler duidelijk te maken dat de conversie zo bedoeld is, moet deze expliciet worden aangeduid met een cast (listing B.65 op p.155). Dit zorgt ervoor dat de waarschuwing niet meer wordt weergegeven.

```
double amount = 123;
int TestAmount = amount * 100;
```

Listing B.64: De originele code zonder expliciete cast.

```
double amount = 123;
int TestAmount = (int)(amount * 100);
```

Listing B.65: De aangepaste code.

C4996

This function or variable may be unsafe.

warning C4996: 'strcpy': This function or variable may be unsafe. Consider using strcpy_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.

Voor sommige functies is er in de C Run-Time Libraries (CRT) library een veiliger alternatief. Deze alternatieve functies hebben dezelfde naam als de originele functie, gevold door een suffix `_s`. Als er voor een functie een dergelijke alternatieve functie bestaat zal de compiler daarop wijzen met een C4996 waarschuwing.

Deze veiligere functies bieden een foutdetectie aan. Ze controleren de parameters van de functie of ze voldoen aan de voorwaarden van de functie.

> "*The secure functions do not prevent or correct security errors; rather, they catch errors when they occur. They perform additional checks for error conditions, and in the case of an error, they invoke an error handler (see Parameter Validation).*" (Microsoft, 2012s)

Nemen we als voorbeeld `strcpy` (listing B.66 op p.156). Deze functie kopieert de C-stijl string `strSource` naar `strDestination`. Als `strDestination` niet groot genoeg is om

er een kopie van `strSource` op te slaan wordt buiten de grenzen van `strDestination` geschreven. Wat daar overschreven wordt is onbekend en dit kan voor serieuze problemen zorgen. Om ervoor te zorgen dat dergelijke fouten aan het licht komen wordt best de veiligere `strcpy_s` functie gebruikt (listing B.67 op p.156). Deze functie vraagt een extra parameter `numberOfElements` waarin de grootte van `strDestination` wordt weergegeven[3]. Zo kan de functie controleren of er tijdens het kopiëren buiten de grenzen van `strDestination` wordt geschreven. Indien dit gebeurt zal de invalid parameter handler worden opgeroepen. Standaard zal deze de applicatie doen stoppen en een foutmelding geven. Bij het debuggen worden er assertions opgeworpen.

```
char *strcpy(
    char *strDestination,
    const char *strSource
);
```

Listing B.66: De functie `strcpy`.

```
errno_t strcpy_s(
    char *strDestination,
    size_t numberOfElements,
    const char *strSource
);
```

Listing B.67: De functie `strcpy_s`.

De beste oplossing is om de functie te vervangen door zijn veiliger alternatief. In sommige gevallen is het niet mogelijk om de grootte van de doelbuffer te achterhalen. In dat geval kan de originele functie best blijven staan. De waarschuwing wijst er immmers enkel op dat er een veiligere functie bestaat:

> "*Many old CRT functions have newer, more secure versions. If a secure function exists, the older, less secure version is marked as deprecated and the new version has the* `_s` *("secure") suffix.*
> *In this context, 'deprecated' just means that a function's use is not recommended; it does not indicate that the function is scheduled to be removed from the CRT.*"
> (Microsoft, 2012s)

Om de overgang gemakkelijker te maken kan er gebruik gemaakt worden van macro's zoals bijvoorbeeld `_CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES`. Als deze wordt ingesteld op 1, worden de functies automatisch overschreven door hun veiligere varianten (listing B.68 op p.156).

```
#define _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES 1

//...

char szBuf[10];
strcpy(szBuf, "test"); // ==> strcpy_s(szBuf, 10, "test")
```

Listing B.68: Het gebruik van de macro `_CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES`. (Microsoft, 2012r)

[3]Er moet ook rekening gehouden worden met het afsluitende null karakter in `strSource`.

No longer needed

warning C4996: 'CWinApp::Enable3dControls': CWinApp::Enable3dControls is no longer needed. You should remove this call.

De functie-aanroep naar '`CWinApp::Enable3dControls`' is niet meer nodig en mag worden weggelaten.

Clashes with future language keyword

warning C4996: 'CFileException::generic': CFileException::generic clashes with future language keyword generic and should not be used. Use CFileException::genericException instead.

In Visual C++ 2005 is de enumeratievariabele '`CFileException::generic`' hernoemt naar '`CFileException::genericException`'. '`CFileException::generic`' moet dus vervangen worden door '`CFileException::genericException`'.

LNK4222

warning LNK4222: exported symbol 'DllCanUnloadNow' should not be assigned an ordinal

De "ordinals", de '@-nummers', moeten uit het def-bestand van het project verwijderd worden, zoals in listing B.70 op p.157.

```
1 ; TKData.def : Declares the module parameters.
2
3 LIBRARY      "TKData.DLL"
4
5 EXPORTS
6     DllCanUnloadNow     @1 PRIVATE
7     DllGetClassObject   @2 PRIVATE
8     DllRegisterServer   @3 PRIVATE
9     DllUnregisterServer @4 PRIVATE
```

Listing B.69: Het def-bestand met ordinals

```
1 ; TKData.def : Declares the module parameters.
2
3 LIBRARY      "TKData.DLL"
4
5 EXPORTS
6     DllCanUnloadNow     PRIVATE
7     DllGetClassObject   PRIVATE
8     DllRegisterServer   PRIVATE
9     DllUnregisterServer PRIVATE
```

Listing B.70: Het def-bestand zonder ordinals

D9035

option 'Fr'

Command line warning D9035 : option 'Fr' has been deprecated and will be removed in a future release cl

De compileroptie "Fr" zal in de volgende versies van Visual C++ niet meer worden ondersteund. In de huidige release wordt deze optie wel nog aanvaard.

Deze foutmelding wordt gevolgd door de fout "*Command line warning D9036 : use 'FR' instead of 'Fr' cl*". Dit geeft aan dat de optie "Fr" in volgende releases van Visual C++ kan vervangen worden door de optie "FR".

De optie kan ingesteld worden in de eigenschappen van het project (`Configuration Options > C/C++ > Browse Information > Enable Browse Information`).

option 'Wp64'

Command line warning D9035 : option 'Wp64' has been deprecated and will be removed in a future release cl

De optie "Wp64" zorgt voor detectie van 64 bit portabiliteitsproblemen en is deprecated in Visual Studio 2010.

C4273

warning C4273: 'atof' : inconsistent dll linkage

Dit heeft dezelfde oorzaak en oplossing als de fouten "*error LNK2005: _atof already defined in ICORDADoubleHelpers.obj*" en "*error LNK1169: one or more multiply defined symbols found*".

LNK4199

warning LNK4199: /DELAYLOAD:mtxex.dll ignored; no imports found from mtxex.dll

mtxex.dll mag verwijderd worden uit de project-opties voor alle modussen, zoals Debug, Release, ... (`Rechtermuisknop op project -> Properties -> Configuration Properties -> Linker -> Input -> Delay Loaded Dlls`).

C4603

warning C4603: '_ICDDLL_IMPL' : macro is not defined or definition is different after precompiled header use

Dit kan opgelost worden door de lijn "`#undef _ICDDLL_IMPL`" voor de macro-definitie te plaatsen.

C4627

cstring skipped

warning C4627: '#include <cstring>': skipped when looking for precompiled
warning C4627: '#include <exception>': skipped when looking for precompiled

De `include`-statements moeten ná de lijn "`#include "stdafx.h"`" komen zoals in listing B.71 op p.159.

```
#include "stdafx.h"
#include <cstring>
#include <exception>
```

Listing B.71: De `include` van *stdafx.h* moet voor die van `cstring` en `exception` staan.

acces denied

warning : Access to the path 'C:\Dev\Van Den Broeck\trunk\main\iBomat.Classic \iBoMatData\Debug\iBoMatData.dll' is denied.

Het dll-bestand wordt vastgehouden door het proces *dllhost*. Dit kan opgelost worden door het proces te beëindigen in *taakbeheer*.

C4278

warning C4278: 'GetUserName': identifier in type library '..\ModuleGFData\DebugU\ModuleGFData.dll' is already a macro; use the 'rename' qualifier

De Stingray property sheets moeten nog toegevoegd worden (zie de fout "*error C1083: Cannot open include file: 'Grid\config\gxstndrd.h': No such file or directory*").

C4068

warning C4068: unknown pragma

In listing B.72 op p.159 wordt de pragma `chMSG` gebruikt, maar deze wordt nergens in de applicatie gedefinieerd. Deze waarschuwing mag gewoon genegeerd worden of de lijn met de pragma kan in commentaar worden gezet.

```
m_BreedteFactorFont((long)1.863)
#pragma chMSG("Hier een kommagetal meegeven heeft geen zin: 't is een long !!")
{

}
```

Listing B.72: De onbekende pragma `chMSG`.

Bijlage C

Niet-gespecifiëerde producties

$$
\begin{aligned}
\mathit{preprocessing-token} \rightarrow\ & \mathit{header-name} \\
| \ & \mathit{identifier} \\
| \ & \mathit{pp-number} \\
| \ & \mathit{character-literal} \\
| \ & \mathit{user-defined-character-literal} \quad \text{C++0x} \\
| \ & \mathit{string-literal} \\
| \ & \mathit{user-defined-string-literal} \quad \text{C++0x} \\
| \ & \mathit{preprocessing-op-or-punc} \\
| \ & \text{each non-white-space character} \\
& \quad \text{that cannot be one of the above}
\end{aligned} \tag{C.1}
$$

$$
\begin{aligned}
h-\mathit{char} \rightarrow\ & \text{any member of the source character set} \\
& \quad \text{except new-line and } >
\end{aligned} \tag{C.2}
$$

$$
\begin{aligned}
q-\mathit{char} \rightarrow\ & \text{any member of the source character set} \\
& \quad \text{new-line and "}
\end{aligned} \tag{C.3}
$$

$$
\begin{aligned}
\mathit{identifier-nondigit} \rightarrow\ & \mathit{nondigit} \quad \text{C++0x} \\
| \ & \mathit{universal-character-name} \quad \text{C++0x} \\
| \ & \text{other implementation-defined characters} \quad \text{C++0x}
\end{aligned} \tag{C.4}
$$

$$
\begin{aligned}
c-\mathit{char} \rightarrow\ & \text{any member of the source character set} \\
& \quad \text{except the single quote ', backslash \textbackslash, or new-line character} \\
| \ & \mathit{escape-sequence} \\
| \ & \mathit{universal-character-name}
\end{aligned} \tag{C.5}
$$

$$
\begin{aligned}
s-char \;\rightarrow\; & \text{any member of the source character set} \\
& \quad \text{except the double-quote ", backslash \textbackslash, or new-line character} \\
& \mid escape-sequence \\
& \mid universal-character-name
\end{aligned}
\tag{C.6}
$$

$$
\begin{aligned}
r-char \;\rightarrow\; & \text{any member of the source character set,} \\
& \quad \text{except a right parenthesis) followed by} \\
& \quad \text{the initial d-char-sequence (which may be empty)} \\
& \quad \text{followed by a double quote ". C++0x}
\end{aligned}
\tag{C.7}
$$

$$
\begin{aligned}
d-char \;\rightarrow\; & \text{any member of the basic source character set,} \\
& \quad \text{except: space, the left parenthesis (, the right parenthesis),} \\
& \quad \text{the backslash and the control characters} \\
& \quad \text{representing horizontal tab, vertical tab, form feed, and newline.} \\
& \quad \text{C++0x}
\end{aligned}
\tag{C.8}
$$

$$
lparen \;\rightarrow\; \text{a (character not immediately preceded by white-space} \tag{C.9}
$$

$$
new-line \;\rightarrow\; \text{the new-line character}
$$

Bijlage D

Voorbeeld van een COM event in Visual C#

```
using System;
using System.Runtime.InteropServices;
namespace EventSource
{
    public delegate void ClickDelegate(int x, int y);
    public delegate void ResizeDelegate();
    public delegate void PulseDelegate();

    // Step 1: Defines an event sink interface (ButtonEvents) to be
    // implemented by the COM sink.
    [GuidAttribute("1A585C4D-3371-48dc-AF8A-AFFECC1B0967") ]
    [InterfaceTypeAttribute(ComInterfaceType.InterfaceIsIDispatch)]
    public interface ButtonEvents
    {
        void Click(int x, int y);
        void Resize();
        void Pulse();
    }
    // Step 2: Connects the event sink interface to a class
    // by passing the namespace and event sink interface
    // ("EventSource.ButtonEvents, EventSrc").
    [ComSourceInterfaces(typeof(ButtonEvents))]
    public class Button
    {
        public event ClickDelegate Click;
        public event ResizeDelegate Resize;
        public event PulseDelegate Pulse;

        public Button()
        {
        }
        public void CauseClickEvent(int x, int y)
        {
            Click(x, y);
        }

        public void CauseResizeEvent()
        {
            Resize();
        }
        public void CausePulse()
        {
            Pulse();
        }
```

```
45      }
46  }
```

Listing D.1: Voorbeeld van de implementatie van een COM event in Visual C#. (Microsoft, 2012h)

Lijst van figuren

Lijst van tabellen

Listings

Literatuurlijst

Adharapurapu, P. (2009). Framework multi-targeting for vc++ projects. http://blogs.msdn.com/b/visualstudio/archive/2009/11/22/framework-multi-targeting-for-vc-projects.aspx.

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers Principles, Techniques, & Tools Second Edition*. Boston: Pearson Education - Addison-Wesley.

Dore, E. (2010). Re: .net 4.0 : First-chance exception 0x04242420 [online forum content]. http://social.msdn.microsoft.com/Forums/eu/clr/thread/bca092d4-d2b5-49ef-8bbc-cbce2c67aa89.

Fourie, M. (2006). Msbuild / visual studio aware error messages and message formats [web log post]. http://blogs.msdn.com/b/msbuild/archive/2006/11/03/msbuild-visual-studio-aware-error-messages-and-message-formats.aspx.

Gregory, K. (2004). *Microsoft Visual C++ .NET 2003 Kick Start*. Indianapolis: Sams Publishing.

HejlsBerg, A., Torgerson, M., Wiltamuth, S., & Golde, P. (2011). *The C# Programming Language*. Boston: Pearson Education - Addison-Wesley.

Jabes, B. (2009). Re: Rebuilding intellisense [web log comment]. http://blogs.msdn.com/b/vcblog/archive/2009/05/27/rebuilding-intellisense.aspx.

Kent, G. (2009). Sql server 2008 sdk directory does not contain the opends60.lib file required for the creation of extended stored procedures [web log post]. http://blogs.msdn.com/b/grahamk/archive/2009/09/29/sql-server-2008-sdk-directory-does-not-contain-the-opends60-lib-file-required-for-the-c
aspx.

Microsoft (2005). *Breaking Changes in the Visual C++ 2005 Compiler*. http://msdn.microsoft.com/en-us/library/ms177253%28v=vs.80%29.aspx.

Microsoft (2008a). *ATL and MFC Changes: ATL 7.0 and MFC 7.0*. http://msdn.microsoft.com/en-us/library/w1sc4t4k(VS.90).aspx.

Microsoft (2008b). *Managed Extensions for C++ Syntax Upgrade Checklist*. http://msdn.microsoft.com/en-us/library/b23b94s7%28v=vs.90%29.aspx.

Microsoft (2010a). */clr Restrictions*. http://msdn.microsoft.com/en-us/library/ffkc918h%28v=vs.100%29.aspx.

Microsoft (2010b). *How to: Register a Component for COM Interop*. http://msdn.microsoft.com/en-us/library/w29wacsy%28v=vs.100%29.aspx.

Microsoft (2010c). *Language Features for Targeting the CLR*. http://msdn.microsoft.com/en-us/library/xey702bw%28v=vs.100%29.aspx.

Microsoft (2010d). *Visual C++*. http://msdn.microsoft.com/en-us/library/vstudio/60k1461a%28v=vs.100%29.aspx.

Microsoft (2010e). *Walkthrough: Creating a Standard C++ Program (C++)*. http://msdn.microsoft.com/en-us/library/ms235629%28v=vs.100%29.aspx.

Microsoft (2010f). */Za, /Ze (Disable Language Extensions)*. http://msdn.microsoft.com/en-us/library/0k0w269d%28v=vs.100%29.aspx.

Microsoft (2011). *.NET Framework System Requirements*.

Microsoft (2012a). *C++ Character Constants*. http://msdn.microsoft.com/en-us/library/6aw8xdf2.aspx.

Microsoft (2012b). *CComPtr Class*. http://msdn.microsoft.com/en-us/library/ezzw7k98%28v=vs.110%29.aspx.

Microsoft (2012c). *ClassInterfaceType Enumeration*. http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.classinterfacetype.aspx.

Microsoft (2012d). *Common Type System*. http://msdn.microsoft.com/en-us/library/zcx1eb1e.aspx.

Microsoft (2012e). *ComVisibleAttribute Class*. http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.comvisibleattribute%28v=vs.110%29.aspx.

Microsoft (2012f). *CWinApp Class*. http://msdn.microsoft.com/en-us/library/362kaah4%28v=vs.110%29.aspx.

Microsoft (2012g). *Data Type Ranges*. http://msdn.microsoft.com/en-us/library/s3f49ktz%28v=vs.110%29.aspx.

Microsoft (2012h). *Expand How to: Raise Events Handled by a COM Sink*. http://msdn.microsoft.com/en-us/library/dd8bf0x3%28v=vs.110%29.aspx.

Microsoft (2012i). *Getting Started with the .NET Framework*. http://msdn.microsoft.com/en-us/library/hh425099.aspx.

Microsoft (2012j). *How to: Instantiate Classes and Structs*. http://msdn.microsoft.com/en-us/library/ke3a209d%28v=vs.110%29.aspx.

Microsoft (2012k). *managed, unmanaged*. http://msdn.microsoft.com/en-us/library/0adb9zxe%28v=vs.110%29.aspx.

Microsoft (2012l). *.NET Framework System Requirements*. http://msdn.microsoft.com/en-us/library/8z6watww%28v=vs.110%29.aspx.

Microsoft (2012m). *.NET Framework Versions and Dependencies*. `http://msdn.microsoft.com/library/bb822049.aspx`.

Microsoft (2012n). *Overview of Marshaling in C++*. `http://msdn.microsoft.com/en-us/library/bb384865.aspx`.

Microsoft (2012o). *Overview of the .NET Framework*. `http://msdn.microsoft.com/en-us/library/zw4w595w.aspx`.

Microsoft (2012p). Re: Visual c++ 2012 grammar [comment]. `http://connect.microsoft.com/VisualStudio/feedback/details/771358/visual-c-2012-grammar`.

Microsoft (2012q). *Resource-Definition Statements (Windows)*. `http://msdn.microsoft.com/en-us/library/windows/desktop/aa381043%28v=vs.85%29.aspx`.

Microsoft (2012r). *Secure Template Overloads*. `http://msdn.microsoft.com/en-us/library/ms175759.aspx`.

Microsoft (2012s). *Security Features in the CRT*. `http://msdn.microsoft.com/en-us/library/8ef0s5kh.aspx`.

Microsoft (2012t). *wchar_t attribute (Windows)*. `http://msdn.microsoft.com/en-us/library/windows/desktop/aa367308%28v=vs.85%29.aspx`.

Microsoft (ca. 2001). *MTS Support in ATL Projects*. `http://msdn.microsoft.com/en-us/library/aa234086%28v=vs.60%29.aspx`.

Nagel, C. (2005). C++/cli value types and memory location [web log post]. `http://weblogs.asp.net/cnagel/archive/2005/02/28/381542.aspx`.

Parr, T. J. & Quong, R. W. (1994). Antlr: A predicated-ll(k) parser generator. *Software Practice and Experience*, 25, 789–810.

Rich, A. (2006). Conformance testing and breaking changes [web log post]. `http://blogs.msdn.com/b/vcblog/archive/2006/10/24/conformance-testing-and-breaking-changes.aspx`.

Shao, L. (2008). Msbuild task [web log post]. `http://blogs.msdn.com/b/vcblog/archive/2008/12/16/msbuild-task.aspx`.

Shao, L. (2009). C++ native multi-targeting [web log post]. `http://blogs.msdn.com/b/vcblog/archive/2009/12/08/c-native-multi-targeting.aspx?PageIndex=2`.

Spolksy, J. (2000). Things you should never do, part i [web log post]. `http://www.joelonsoftware.com/articles/fog0000000069.html`.

Stroustrup, B. (1995). *The design and evolution of C++*. Addison-Wesley.

Stroustrup, B. (2000). *The C++ Programming Language Special Edition*. Indianapolis: Pearson Education - Addison-Wesley.

Xie, E. (2012). Re: Visual c++ 2012 grammar [online forum content]. `http://social.msdn.microsoft.com/Forums/en-US/vclanguage/thread/3f02e061-5848-4b8f-ad52-e3d2d115167f/`.

Alle internetartikels waren actueel op 28 januari 2013.