TOOLS

Microsoft heeft de laatste paar jaar een flinke hoeveelheid nieuwe technologieën over de wereld van de programmeurs uitgestort. Termen als .NET, WPF, XAML vliegen ons dagelijks om de oren. Tijd om op de rijdende trein van het nieuwe tijdperk te springen. En aangezien C# erg veel op C++ lijkt, moet dat niet al te moeilijk zijn. Maar de schijn bedriegt. Edwig Huisman becijfert in dit artikel hoeveel coderegels, manuren en problemen je tegenkomt alvorens de klus zou kunnen zijn geklaard.

Porting naar .NET, klapper of teleurstelling?

Van C++ naar C# is nog een hele klus

De afgelopen vier jaar heeft Microsoft een groot aantal technologische vernieuwingen laten zien. Eerst was daar in 2003 de komst van het .NET framework en de eerste versie van de C# taal. Maar daar bleef het niet bij. Spoedig volgden niet alleen de versies van dit nieuwe framework elkaar snel op. Er kwamen ook nieuwe technologieën bij. Windows Presentation Foundation (WPF), Communications foundation, Workflow foundation, AJAX.NET en recentelijk Silverlight zijn slechts een paar voorbeelden. Maar om van al die nieuwe heerlijkheden gebruik te kunnen maken, zal men wel minimaal van de basis, C# op het .NET platform gebruik moeten maken.
Als voorbeeld dient een applicatieserver voor ERP-applicaties onder de naam Pronto. Deze applicatieserver is geschreven in C++. Midden jaren negentig was dit nog de crème-de-la-crème, maar met alle nieuwe mogelijkheden is wellicht het moment gekomen om te bezien of zo'n strategische applicatie naar .NET, en dus naar C#, moet worden omgezet.
In de praktijk blijkt echter dat dit makkelijker gezegd en gedacht is, dan gedaan. De verwachting was dat het overzetten zoiets als een simpel boottochtje bij kalme zee zou zijn. Maar bij het uitwerken van het traject doemen ijsbergen op aan de horizon. Met onder het oppervlak gevaarlijke punten die tot schipbreuk kunnen leiden.

Applicatieserver Pronto

Pronto is een applicatieserver die gebruikt wordt om WOCAS4*all* (een ERP applicatie voor

Pronto in getallen

Een paar kengetallen om een indruk te geven van de omvang van het product.

Totale omvang pakket in C++ regels

C++ source	: 322.611
Header source	: 269.253
Totaal	: 591.864 regels

MFC (Microsoft Foundation Classes) Interface

Referenties naar CWnd	: 239
Referenties naar CWnd*	: 192

Geheugen allocatie/ de-allocatie

Aantal keer 'new'	: 1766
Aantal keer 'delete'	: 476
Aantal destructors	: 1387
Expliciete destructors	: 620

Collections: Omzetten MFC 'CMap' naar STL

151 keer CMap<iets>
709 keer std::map<iets,anders> of std::list<iets>

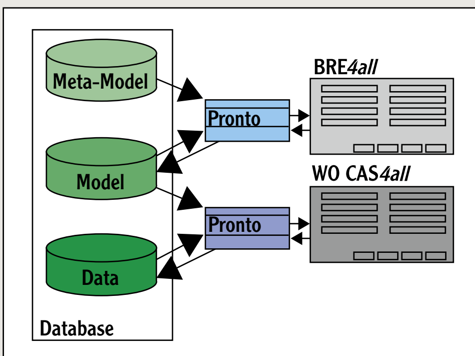
Kader 1

Edwig Huisman
is technisch product manager
bij Centric IT Solutions
Hij is te bereiken via
edwig.huisman@centric.nl

De werking van Pronto

Pronto laat een applicatie draaien door een model van de applicatie uit de database te lezen tezamen met gebruikers data, en dit op het beeldscherm te tonen in vensters. (onderste Pronto).

Uit diezelfde database kan een andere instantie van de applicatie server een meta-model lezen als applicatiemodel en de modellaag aanpassen. Dit gebeurt in de Business-Rules-Editor: BRE4*all*. (bovenste Pronto). Het wijzigen van het model in BRE4*all* is feitelijk equivalent aan het programmeren van een applicatie. De datalaag wordt tevens beheerd en gegenereerd met DDL (Data Definition Language).



Het opslaan van modellen in de database maakt het overbodig om compilatieslagen door te gaan voor het ontwikkelen van een ERP applicatie. Hierdoor kan snel en efficiënt worden geprogrammeerd en getest zonder op een compiler te wachten. Het uitleveren van programmatuur is veranderd van het uitleveren van gecompileerde executables in het leveren van een model in databasevorm.

Pronto is daarmee feitelijk de 'virtual machine' van de applicatie WOCAS4*all*.

Kader 2

Nederlandse woningcorporaties) en ITIL4*all* (een ITIL service applicatie voor automatiseringsbedrijven) te laten werken. Alle aspecten van zo'n applicatie worden daarbij ondersteund. Niet alleen de opvraag- en mutatievensters, maar ook de batchverwerking, rapportages en correspondentie output in de vorm van brieven horen daarbij. Koppelingen met andere applicaties via technieken als ODBC, (D)COM, Active/X, DDE, MS-Queue, MS-Outlook, MS-Word en MS-Excel zijn daarbij rijkelijk aanwezig. Pronto is feitelijk de 'virtual machine' van WOCAS4*all* en ITIL4*all*. Zie voor een nadere uitleg van de werking kader 2.

Omdat alle aspecten van het laten werken van een moderne applicatie worden ondersteund, is Pronto behoorlijk complex. Niet alleen is het programma groot qua absolute omvang (zie kader 1), maar ook zijn alle aspecten van het Windows operating systeem erin verweven. Een paar van de belangrijkste kenmerken zijn:

Alle user-interfaces worden op runtime gegenereerd aan de hand van de MFC bibliotheek. Statische resources komen nauwelijks voor, omdat alle interfaces gegeneerd worden aan de hand van een model in de database.

Gebruikersdata wordt middels een object-relational mapper in een database cache opgeslagen. Daarbij wordt overal rekening gehouden met audit-trails op de data en administratieve correcties.

Voor de inrichting van de functionaliteit wordt gebruik gemaakt van een Nederlandstalige scripttaal (ook Pronto genaamd). Hiermee zijn alle basisfuncties en klantspecifieke maatwerkaanpassingen geprogrammeerd.

Database acties en serverfuncties (stored procedures) worden modelmatig uitgeschreven in zogenaamde 'rekenregels'. Hiervoor is in principe geen kennis van SQL nodig.

Pronto porten

Terug naar de aanleiding van dit artikel: Zou het niet mogelijk moeten zijn om een C++ programma met enige inspanning om te zetten naar C# ? De programmeertalen lijken immers op het eerste gezicht sterk op elkaar! En zou het niet mooi zijn als ook Pronto kan profiteren van al die mooie nieuwe Microsoft producten en mogelijkheden die de laatste jaren over ons zijn uitgestort? En wat moeten we daar dan voor doen?

Deze laatste vraag hebben we uitgewerkt. Veel van de code is namelijk simpel over te zetten. Links en rechts moet er uiteraard nog wat pointer logica worden gladgestreken en wil je wat constructies wegwerken. Zo dachten we. Tijdens het vooronderzoek liepen we echter al tegen een aantal factoren aan die ons ernstig belemmerden in het porten van een complex programma als Pronto. In het kort gaat het om:

- Het geheugenbeheer en de destructie van objecten.
- Het omzetten van MFC algoritmen naar C# collecties.
- Het omzetten van STL algoritmen naar C# collecties.
- Het omzetten van de gegenereerde user interfaces.

Deze factoren worden hieronder per punt verder uitgewerkt, met een kwantificering van de inspanning.

Het porten van de geheugendestructie

Om maar met de deur in huis te vallen: de geheugen de-allocatie is één van de grootste problemen

bij het porten van Pronto. In C++ is het gebruikelijk om in de destructor van een klasse al het opruimwerk te doen. De andere objecten die van het object afhankelijk zijn worden opgeruimd, hetzij impliciet, hetzij expliciet. Een C++ programmeur is daarom gefocust op het eigendom van het object en waar het object het geheugen weer verlaat. Zou dat niet zo zijn, dan zijn geheugenlekken het gevolg.
Dit is een belangrijk verschil tussen C++ en C#. In C# wordt voor de destructie gebruikgemaakt van de 'Finalize' of de 'Dispose' methode om een override te doen van de bestaande destructie. In C# .NET is natuurlijk sprake van de garbage collector die onafhankelijk van het programma en de logica daarop kan ingrijpen en het geheugen opruimt. Met als gevolg dat het opruimen van de objecten op een niet door de programmeur te bepalen moment gebeurt en de 'omgeving' van de objecten dan veranderd kan zijn.
Natuurlijk kan de programmeur de garbage collector ook zelf aanroepen (met 'system.Collect()') maar daar zit niet de echte crux. Nee, het belangrijkste knelpunt is dat in C++ een compleet stuk logica zit in de klassen op een plek waar in C# geen tegenhanger is.
Alle afhandeling van logica bij het opruimen van een object moet dus geheel opnieuw worden doorlopen. Daar waar C++ automatisch de destructor aanroept bij het out-of-scope gaan van variabelen, zal dat nu bovendien expliciet aangeroepen moeten worden. Het minste of geringste foutje zal leiden tot problemen en een minder stabiele omgeving. Alle allocaties en de-allocaties van een object met een functionele destructor zullen dus nagekeken en uitgetest moeten worden.

Wat is de omvang van deze uitdaging? In totaal komen in Pronto bijna 600 klassen voor waarbij de destructor functionaliteit bevat. Per klasse moet dat omgeschreven en getest worden, waarvoor op een aantal uur per klasse wordt gerekend.
Grootste probleem bij deze calculatie: de destructors zijn wel te tellen. Het aantal variabelen dat zonder verdere code 'out-of-scope' gaat is echter niet simpelweg te tellen in de code. Hier is dus sprake van een zeer grote onzekerheidsfactor. Als variabelen van een bepaalde klasse erg veel gebruikt worden, dan zorgt het aanroepen van de nieuwe destructiemethode potentieel voor een veelvoud van de oorspronkelijke inspanning.

Het porten van de MFC algoritmen

De Microsoft Foundation Classes zijn jarenlang hét framework voor C++ programmeurs geweest. Niet alleen worden hierin user interfaces gebouwd, maar ook collecties zoals 'CMap' zijn uit MFC afkomstig. Oorspronkelijk waren er zo'n 800 klassen in Pronto aanwezig die hiervan gebruikmaakten. In de loop der jaren is een groot aantal hiervan al omgebouwd naar het modernere STL (zie verderop in dit artikel), maar er zijn er nog steeds zo'n 150 van over. Al deze klassen zullen naar een overeenkomstige .NET collectie moeten worden omgezet. De code waarin bijvoorbeeld door zo'n container heen gelopen wordt, verschilt behoorlijk qua opzet. Zie ook kader 3 met een voorbeeld van zo'n verschil.
Om hoeveel gaat het? In totaal komen er zo'n 150 klassen voor die gemiddeld drie keer gebruikt

Het porten van collecties

Een van de problemen bij het porten van C++ naar C# is het omzetten van containers met objecten. Deze containers worden ook wel collecties genoemd. Onderstaande voorbeelden laten de ontwikkeling in het denken over het programmeren van deze containers zien. Wat opvalt is de wijze waarop in de loop der jaren de complexiteit en het aantal code regels is teruggebracht.

Voorbeeld in MFC

```
typedef CMap<CString,MijnObject*> mijnMap;
mijnMap deMap;
POSITION pos;

pos = deMap.GetStartPosition();
while(pos)
{
  CString deString;
  MijnObject* obj;
  deMap.GetNextAssoc(pos,string,obj);
...
  (doe iets nuttigs met 'deString' en 'obj')
}
```

Voorbeeld in STL

```
typedef std::map<CString,MijnObject*> mijnMap;
mijnMap deMap;

for(deMap::iterator it = deMap.begin(); it !=
deMap.end(); ++it)
{
   CString deString = it->first;
   MijnObject* obj  = it->second;
   ...
   (doe iets nuttigs met 'deString' en 'obj')
}
```

Voorbeeld in C# .NET

```
MyCollection mijnCollectie = new MyCollection;

foreach (mijnObject obj in mijnCollectie)
{
    ...
    (Doe iets nuttigs met 'obj')
}
```

Kader 3

worden in de code. Er moeten dus zo'n 450 code omzettingen gedaan worden, inclusief testwerk.

Het porten van de STL algoritmen

Nu waren we al bezig geweest om de MFC containers in de code om te zetten naar het nieuwere STL. Dit staat voor Standard Template Library. Deze methodiek is in de tweede helft van de jaren negentig ontwikkeld en door het ANSI standaardcomité in de C++ taal opgenomen op verzoek van het ANSI standaard comité voor de C++ taal.
STL is een aantrekkelijke manier van programmeren van containers. Belangrijkste reden is dat de code rondom de containers erg makkelijk van container is te wijzigen. Voldoet een 'list' niet, dan nemen we een 'vector'. Voldoet een 'map' niet, dan nemen we een multimap (mmap). Vaak hoeft hiervoor alleen maar de declaratie van de container omgezet te worden, zonder de logica aan te passen.

Ook deze omzetting vergt veel inspanning. Van de oorspronkelijke 800 klassen zijn er al zo'n 650 omgezet in de afgelopen vier jaar. In die periode zijn er ook nog eens circa 50 klassen bijgekomen. Hierdoor zullen bij het porten naar C# in totaal 700 klassen moeten worden omgezet van STL naar C# collecties. Voor iedere klasse geldt ook hier weer dat ze gemiddeld zo'n 3 keer in de code worden doorlopen. Het omzetwerk inclusief testen loopt hier dus zeker op tot boven een mensjaar aan werk.

Omzetten van de gegenereerde user interfaces

In Pronto is een component aanwezig die we de 'interactiebouwsteen' noemen. Deze module genereert aan de hand van het applicatiemodel de daadwerkelijke user interface. Frames, velden, knoppen, lijsten en plaatjes worden op het beeldscherm getoond. De door de gebruiker ingegeven toetsaanslagen en muisbewegingen worden terugvertaald naar de applicatie.
Deze module maakt intensief gebruik van het MFC model van programmeren met de Windows message loop. Commando's en berichten van en naar de van CWnd afgeleide controls worden op die manier doorgegeven. Dat het hele venster uit run-time gegenereerde objecten bestaat, maakt het erg complex. Enkele voorbeelden zijn klassen als 'CEdit', 'CButton', 'CListBox' en 'CComboBox' en de afgeleide klassen die wij daarvan geproduceerd hebben.
Een tegenhanger in .NET aan de hand van WinForms betekent feitelijk het vanaf de grond af aan herontwerpen, herbedenken en herprogrammeren van deze module. Dit moet gebeuren aan de hand van de C# namespace 'System.Windows. Forms'. Los van het feit of dit allemaal op dezelfde manier kan en zonder functioneel verlies kan.

De hoeveelheid werk van het omzetten van de user interfaces is zeer moeilijk te kwantificeren. Het is niet te berekenen aan de hand van een telling van objecten en een schatting van hoeveel regels programmeer- en testwerk dit zou zijn. Zeker is dat bij de laatste grote verbouwing in deze module ongeveer een mensjaar werk is verstouwd, zonder nog helemaal van de grond af aan opnieuw te beginnen. Dat ene jaar werk zal het dus zeker kosten.

Klapper of teleurstelling?

Bij elkaar opgeteld leveren alle voorgaande factoren een werklast op van een aantal mensjaren. Bovendien gooi je het hele programmeerplatform om, waardoor je dit niet zomaar uurtje voor uurtje en fase voor fase kunt plannen en uitvoeren. Pas als Pronto opnieuw onder .NET opstart zal het eindresultaat te bewonderen zijn. Dan pas kan het model weer uit een database gelezen en uitgevoerd worden zodat er überhaupt iets zichtbaar wordt. Hierdoor heeft de overstap van C++ naar C# een soort big-bang scenario. Alles of niets. Dé grote klapper of totale teleurstelling?
Wat je daar omheen moet organiseren is nogal wat programmeer- en denkwerk. Dit artikel is beperkt tot een paar van de grootste en belangrijkste factoren, maar daarnaast zullen er nog vele kleinere factoren invloed hebben op het proces, voordat de applicatie weer draait.

Een complete releasecyclus

Inclusief testen komt de totale planning neer op een complete releasecyclus van één productiejaar van het product. De complete planning van een totale productieafdeling dient hiervoor te worden ingeruimd. Nog even los van de 'bijkomende' factoren zoals de omscholing, bijscholing, of de inhuur van nieuwe frisse .NET krachten die het project kunnen dragen.
De vraag dringt zich op wat de meerwaarde is van een dergelijk omzettingstraject naar C#. De omzetting vergt een compleet productiejaar, zonder dat in dat jaar toegevoegde waarde voor de klant wordt gerealiseerd. Om de meerwaarde te bepalen zijn de bedreigingen én kansen van het porting-traject onder de loep genomen.

Bedreigingen

Er zijn twee belangrijke bedreigingen voor een pakket als Pronto als het in C++ blijft voortbestaan.
Ten eerste is het met de komst van het .NET platform voor programmeurs heel aanlokkelijk om veel .NET kennis op het CV te hebben staan. De laatste jaren lijkt het de trend dat de

De overstap van C++ naar C# heeft een soort big-bang scenario

hoeveelheid .NET op je CV je kansen op de arbeidsmarkt en een hoger salaris vergroot. Dat dit niet ongegrond is, blijkt het laatste jaar uit de personeelsadvertenties op het gebied van de vraag naar programmeerkunde in 'de bladen'.
De kunde van C++ programmeren wordt hierdoor bedreigd of sterft zelfs langzaam uit. Op de langere termijn zal het waarschijnlijk net zo gaan als met de beschikbaarheid van 4GL, Fortran of Cobol programmeurs: er zullen er steeds minder zijn.

Ten tweede is op dit moment de hele MSDN documentatie van Microsoft en alle referenties alleen nog maar gericht op C# en hooguit VB programma's. Meer en meer worden alle referenties en documentatie van C++ programmatuur uit de omgeving verwijderd. Volg je een Microsoft cursus of masterclass dan lijkt het soms of er al helemaal niets anders meer bestaat dan C#.
Mocht het hele programmeerparadigma ooit in zijn totaliteit op C# gericht worden, dan gaat C++ een soort 'underground' bestaan leiden. Microsoft besluit wellicht nooit om de Windows SDK op .NET te baseren, maar aansluiting op nieuwe technologieën wordt dan steeds moeilijker.

Kansen

Er zijn belangrijke kansen voor een pakket als Pronto als het naar C# wordt omgezet.
Ten eerste is daar de toekomstwaarde. .NET is een blijvertje, zoveel is wel duidelijk. Door het product om te zetten naar .NET zou het in één klap weer toekomstvast worden;
Ten tweede zal het de integratie met andere applicaties vereenvoudigen. In de nabije toekomst zullen steeds meer producten in .NET geschreven worden. Het zou een investering in toekomstige integraties zijn;
Ten derde zal het gebruik van de nieuwe Presentation Foundation veel van de huidige zelfgebouwde user interface controls overbodig maken;
Ten vierde zal het inzetten van de Workflow Foundation ervoor kunnen zorgen dat onze eigen workflow module in één keer van een grafische interface voorzien kan worden. Die hoeven we dan niet zelf te implementeren;
Ten vijfde is daar de productiviteit. Uit proefnemingen blijkt dat programmeurs C# over het algemeen als intuïtief ervaren, waardoor de productiviteit kan stijgen.

C# in de ijskast

Na het eerste onderzoek is er vooralsnog niet voor gekozen Pronto om te zetten naar C#. Alle productiecapaciteit wordt eerst ingezet, daar waar de behoefte het grootst is: de implementatie van nieuwe functionaliteit voor onze klanten.
Porten naar C# betekent dat de complete productiecapaciteit van een gehele afdeling voor minimaal één jaar wordt opgesoupeerd, zonder dat er feitelijk nieuwe functionaliteit wordt toegevoegd aan het product.
Pas op het moment dat het toevoegen van nieuwe .NET gerelateerde functionaliteit in het gedrang komt, zal het overzetten naar C# .NET weer in het totale investeringsplaatje voorkomen. «