MAGAZINE
Software Developers Group Netherlands
nummer 72
juni 2002

Multithreading, doing more than one thing at a time.

IN DIT NUMMER

Interview with Jason Vokes at CttM 2002

Magazine voor Softwareontwikkelaars • Twaalfde jaargang • No. 72 • juni 2002 • Een uitgave van Software Developers Group Netherlands

advertentie

Colofon

SDGN Magazine Nr. 72

Uitgave:

Software Developers Group Netherlands

Ontmoetingsplaats en platform voor professionele ontwikkelaars:

Postbus 506,
7100 AM Winterswijk.

Telefoon : (0543) 51 80 58
Telefax : (0543) 51 53 99

Internet : www.sdgn.nl

Email:

Bestuur : bestuur@sdgn.nl
Sectiehoofden : secthfd@sdgn.nl
Redactie : redactie@sdgn.nl
WebSite Team : webteam@sdgn.nl
Secretariaat : info@sdgn.nl

Bestuur van SDGN:

Ad van de Lisdonk, voorzitter
Joop Pecht, secretaris
Rob Suurland, penningmeester
Pepijn Smits, vice-voorzitter
Ed Richard, bestuurslid

Sectiehoofden:

VO : Ed Richard
Delphi : Harry Maes
Microsoft-Development : Remi Caron
Internet-Development : Guus Hofstede

Redactie SDGN-Magazine:

Jan van der Graaf (eindredacteur)
Mark Blomsma
Johan Parent
Erik Visser
Rob Willemsen

SDGN Web Team:

Cor Fransen
Erik Visser
Joop Muis
Ed Sonneveld

Met medewerking van:

Janno Hordijk, Bob Swart, Rod da Silva, Paul Maskend and Andy Kramek, Edwin Holkers, Aram Janigian, Ginny Caughey, Remi Caron, Peter van Ooijen, Chad Z. Hower, Meinhard Schnoor-Matriciani, Bert Dingemans, Remi Caron, Ad van de Lisdonk en Joop Pecht, Frits Bosschert , Ed van Akkeren

Vormgeving en opmaak:
Reclame-adviesbureau JE/ES, Winterswijk (www.je-es.nl)

Druk:

Drukkerij Loor b.v., Varsseveld (www.loor.nl)

Inhoudsopgave

Adverteerders

Listings

Zie de WebSite voor eventuele source files uit deze uitgave.

Adverteren?

Informatie over adverteren en de advertentietarieven kunt u vinden op www.sdgn.nl onder de rubriek Magazine. Deze informatie is tevens telefonisch of per email verkrijgbaar via het SDGN secretariaat.

Software Developers Group Netherlands

Wanneer ik met buitenlandse sprekers sta te praten, bijvoorbeeld op een CttM, dan hebben we het nog wel eens over de verschillen tussen landen. In grote lijnen zou je kunnen zeggen dat ze bijvoorbeeld in Amerika niet hard rijden, geen verstand hebben van koffie, bang zijn voor intimiteit, en geweld verheerlijken (of toch in ieder geval de gewoonste zaak van de wereld vinden). In Nederland is dat precies andersom verzuchten ze dan een beetje jaloers, terwijl ze aan hun bakje koffie lurken. Het zal voor mij dit jaar een stuk moeilijker worden om trots te glimlachen, denk ik.

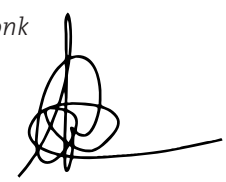
Precies een week voor de aanvang van CttM 2002 is Pim Fortuyn neergeschoten. Wellicht dat men er tegen de tijd dat u dit leest achter is gekomen wat de dwaas die hem vermoordde nou eigenlijk dacht. Op dit moment weten we nog niet veel meer dan dat de vermoedelijke dader gek is op beestjes en dat Fortuyn zou hebben gezegd dat wat hem betreft het verbod op het fokken van pelsdieren wel kon worden opgeheven. Al met al blijkbaar reden genoeg om iemand weloverwogen vijf kogels in zijn lijf te jagen.

Diezelfde avond staan er politiebussen bij ons in de straat en heeft de BVD onopvallend in een onopvallende auto op een onopvallende plek post gevat. Het enige waar je aan kunt zien dat het om onopvallende BVD-ers gaat, is het feit dat ze de hele dag in hun auto zitten, terwijl de meeste mensen bij ons in de straat liever gewoon naar binnen gaan. ...En aan dat verplaatsbare toilet (zo'n Dixie) natuurlijk waar ze zo af en toe hun koffie en donuts naartoe brengen. Zo'n groot blauw toilet is namelijk best opvallend. Temeer daar de meeste mensen bij ons in de straat al een toilet in huis hebben. Nou moet u weten dat mijn overbuurman ene mijnheer Paul Rosenmöller is.

Desgevraagd blijkt het hier inderdaad de beveiliging van de buurman te betreffen. Hij is nu ook (meer dan normaal !?!?!?) bedreigd. Zijn vrouw en kinderen zijn ondergebracht in het buitenland en naast onze vrienden met de Dixie, logeert er ook een BVD-er in zijn huis. De volgende dag is het al raak. Terwijl mijn vrouw op straat loopt, wordt zij klemgezet door wat intimiderende lieden met kaalgeschoren koppies en flessen bier in de ene hand en zo'n fijne vechthond aan de riem in de andere hand die haar vragen waar 'die Rosenmöller' woont. Op dat moment gaat natuurlijk de Dixie open en krijgt mijn vrouw hulp...

Nederland is veranderd roept iedereen op de TV. Misschien. Maar gelukkig zijn er ook altijd nog dingen waar u op kunt rekenen en vertrouwen. Bij SDGN mag u programmeren in de taal die u het liefste spreekt. En u wordt er alleen maar meer om gewaardeerd indien u iets te vertellen hebt.

Ad van de Lisdonk



Panta rei, oeden menei

CttM is .Net achter de rug en we hebben een hoop informatie kunnen absorberen. Informatie die we zo hard nodig hebben om onze visie scherp en alert te houden. We moeten continu bewust zijn van onze omgeving en open staan voor veranderingen. Alles stroomt, niets blijft. Alleen bedrijven die verstandig met veranderingen omgaan overleven in deze snel veranderende wereld. Gelukkig heb ik kunnen zien dat veel SDGNers hier serieus mee bezig zijn.

Tijdens de CttM heb ik veel niet Microsoft developers een bezoek zien brengen aan .NET sessies. Ook in de wandelgangen heb ik veel mensen horen discussiëren over "wat nu met .NET". Ik ben dan ook blij dat het voor iedereen duidelijk is dat deze techniek niet meer weg te denken is ongeacht welk platform of ontwikkeltaal je op dit moment gebruikt. U zult vroeg of laat op één of andere manier moeten veranderen.

Er zijn niet alleen nieuwe syntaxen maar ook de manier van ontwikkelen wordt anders. Niets blijft en dat geldt ook voor onze klanten. Onze klanten veranderen namelijk ook. Daarom moeten we pragmatisch ontwikkelen in korte cycli zodat veranderingen snel doorgevoerd kunnen worden. Reserveer binnenkort eens een aantal uur om u te verdiepen in eXtreme programming en probeer eens met een aantal collega's te filosoferen over deze methode. Iedereen heeft zijn eigen methode maar wellicht zijn er een aantal of misschien wel alle aspecten die zullen aanspreken van eXtreme programming.

Ook het landschap van mobile devices is aan het veranderen. De infrastructuur is er klaar voor. Te denken valt aan fatsoenlijke dataoverdracht door middel van GPRS en later UMTS en voor onze positie GPS. Met deze infrastructuur kunnen deze 'lite' devices nu al serieuze oplossingen bieden. Webservices worden belangrijk voor de 'mobile' omgeving. U moet bij mobile devices niet alleen denken aan een luxe GSM toestel of een palmtop. Het zal steeds vaker geïntegreerd worden in bestaande apparaten. Zo is Microsoft zeer serieus bezig in de auto-industrie. Het is toch prettig als u via uw navigatiesysteem actuele informatie kunt opvragen bij bijvoorbeeld een webservice als MapPoint. Webservice in combinatie met eMbedded software zal steeds belangrijker worden.

Scherp en alert zijn is de 'key', want de cyclus wordt steeds korter. Probeer af en toe even wat hoger op te stijgen als u in een helikopterview naar uw organisatie, ontwikkelomgeving of platform kijkt. Probeer die horizon te verbreden en pas u aan het landschap aan. Het landschap zal zich niet aan u aanpassen wat een "shake-out" tot gevolg kan hebben.

Guus Hofstede

Software Developers Group Netherlands

tools

- ✔ Internet
- ✔ HTML
- ✔ Perl
- ✔ CGI

auteur

JANNO HORDIJK

HTML, CGI-BIN, PERL en lokale Unix scripts

Op mijn werk bij het Dijkzigt ziekenhuis te Rotterdam werd me gevraagd om een paar simpele HTML-pagina's te maken waarmee je eenvoudig kleine unix-scriptjes kan aanroepen. Dit lijkt gemakkelijker gezegd dan gedaan. Er is een duidelijke scheiding tussen wat publieke gebruikers (internetters bijvoorbeeld) en lokale gebruikers (zoals bijvoorbeeld een supervisor) mag doen. In principe mag een internetter maar heel weinig. Dus aan internetters toegang geven tot bepaalde programma's kan gelukkig niet zomaar. Daarvoor is CGI-bin beschikbaar. CGI betekent Common Gateway Interface en zorgt voor de brug tussen het internet en het lokale netwerk. Vele internetservers kunnen geconfigureerd worden om CGI te ondersteunen. Om een CGI-pagina aan te roepen kan je html-code gebruiken en dit plaatsen in een bestand met de extensie .cgi.

Apache server

Om aan bijvoorbeeld Apache te vertellen dat CGI moet worden ondersteund staat in de directory */opt/apache/etc* (of waar apache bij jou is geïnstalleerd) een configuratiefile met de naam *httpd.conf*. Hierin is de ondersteuning van CGI geactiveerd door middel van *ScriptAlias /cgi-bin/ "/home/www/lib/cgi-bin/"*. Dit vertelt Apache dat direct onder de DocumentRoot de directory *cgi-bin* bestaat die door browsende internet gebruikers aangeroepen kan worden. Fysiek staan de bestanden in dit geval op */home/www/lib/cgi-bin*.

Er kan meestal niet direct een cgi-script worden aangeroepen

De DocumentRoot wordt dmv de regel *DocumentRoot "/home/www/lib/htdocs"* ingesteld. Dit even terzijde.

De ingang van de gebouwde pagina's is de index.html die in de *"/home/www/lib/htdocs"* staat. Doordat dit de DocumentRoot is en de index.html bij de standaard documenten hoort kan eenvoudig in de browser http://[domeinnaam] worden ingegeven.

Redirect

Er kan meestal niet direct een cgi-script worden aangeroepen, daarom bevat index.html een automatische redirection naar index.cgi die in de cgi-bin directory staat:

```
<html>
<head>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="-1">
<meta HTTP-EQUIV='REFRESH' CONTENT="0;
        URL=http://noc2.azr.nl//index.cgi">
</head>
<body>redirecting....</body>
</html>
```

Flow van verwerkers

Alle cgi-scripten worden, als ze worden aangeroepen, geïnterpreteerd door Apache (de webserver). Op de eerste regel van een script hoort de regel *#!/opt/perl5/bin/perl* te staan. Dit betekent dat de rest van de tekst moet worden gestuurd naar Perl welke in de */opt/perl5/bin* directory staat. Hier kunnen ook andere commando's worden geplaatst dan Perl. Maar vanwege dit artikel ga ik uit van Perl. Bovenaan de uitvoer van Perl hoort de tekst 'Content type: text/html' te staan met een lege regel daaronder. Perl verwerkt dus dit script en de uitvoer wordt teruggestuurd naar Apache. Uiteindelijk verwerkt Apache de uitvoer van Perl en geeft deze terug aan de client-browser.

Netscape en Perl

De Netscape browser verwacht minimaal dat in het CGI-script een *<html>* tag wordt teruggeven. Wordt deze tag verplaatst naar een Unix-script dat wordt aangeroepen vanuit deze CGI-pagina dan geeft Netscape alleen maar de pure HTML-code weer en doet dit niet, zoals verwacht, evalueren (renderen). Ineternet Explorer heeft hier geen last van.

Met deze constructie van een CGI-script dat door Perl wordt gehaald kan je dus dynamisch HTML code laten genereren. Voor de duidelijkheid, de uitvoer van een script is HML-code wat weer wordt doorgestuurd naar de webserver. Deze verwerkt de HTML-code en vervangt eventuele specifieke webserver code weer door andere HMTL en geeft dit uiteindelijk door aan de lokale browser.

Algemene opbouw van een CGI-script in geval van een aanroep via Perl

```
1       #!/opt/perl5/bin/perl
2       do "../../scripts/init_perl.pl";
3       $Titel='TCP/IP adres pingen';
4       print "Content type: text/html\n";
5       print "\n";
6       print "<script SRC='$DOC_ROOT/javascript/default.js'>
                    </script>\n";
7       do "../../scripts/include.pl";
8       use CGI;
9       <html>
10      <body>
rest van de html code….
100     END
101     do '../../scripts/mainmenu.pl';
```

Regel 1 bevat, zoals opgemerkt, de instructie voor Apache dat dit script voor Perl bestemd is. Dit wordt ook wel de 'Shebang'-regel genoemd. **Regel 2** bevat een aanroep van een extern perl-script welke variabelen definieert **Regel 3** bevat een variabele declaratie die de Titel van deze pagina aangeeft. Op **regel 4 en 5** wordt de genoemde 'Content type: text/html" geprint. Daarna wordt op **regel 6** een standaard javascript uitgevoerd. Dus **regel 4** geeft de standaard code op welke de Apache webserver begrijpt. Zonder deze regel en de daaropvolgende lege regel (5) wordt direct een Internal Server error gegeven zodra het bewuste script wordt opgeroepen in de webbrowser.

Het script geeft een standaard header terug, activeert CSS, en geeft als titel van het document de inhoud van de variabele Title, in dit geval 'TCP/IP adres pingen'.

Regel 7 bevat de <*html*> tag, dit is speciaal voor Netscape. Deze tag is voor alle document hetzelfde en was daartoe verhuisd naar de header.sh. Onder Internet Explorer gaat dit gewoon goed, maar blijkbaar wil Netscape dit al in het originele document hebben staan.

Regel 8 bevat een commando bestemd voor Perl om CGI ondersteuning voor dit script aan te zetten.

Regel 9 bevat het eigenlijke begin van de html code, vandaar de <html> tag.

Regel 10 bevat de <body> tag, alles na deze tag bevat gewone html code.

Regel 100 bevat END, kijkend naar regel 4 betekent dit het einde van de redirect van alle uitvoer voor het printcommando.

Regel 101 bevat een aanroep van een extern perl-script om een standaard link naar het hoofdmenu op te nemen.

Publieke variabelen gebruiken in Perl-programma's

Ik heb niet zo gauw een manier kunnen ontdekken om variabelen globaal te kunnen definiëren die kunnen worden gebruikt in de diverse scripts. Als een tijdelijke oplossing (wie weet wel de permanente) heb ik elk perl-programma uitgebreid met een aanroep van do "../../scripts/init_perl.pl". Dit scriptje is vrij kort en het zet alleen maar wat variabelen:

```
1       #!/opt/perl5/bin/perl
2       # init_perl.pl
3       #
4       $TestOmgeving="1";
5       $SERVERNAME="http://noc2.azr.nl";
6       $INTRANET="http://intranet.azr.nl";
7       $DOC_ROOT="$SERVERNAME/test";
8       $CGI_ROOT="$SERVERNAME/testcgi";
```

Zoals je ziet worden een aantal variabelen gezet waaronder de DOC_ROOT en CGI_ROOT variabelen. Nu kan ik op een centrale plek onderscheid maken in de locatie van mijn 'echte' omgeving en mijn testomgeving. Gewoon even de juiste paden intikken en daarna gaan alle verwijzingen correct naar bijvoorbeeld een testomgeving. Let op dat de variabelen net zo lang worden geëvalueerd totdat ze geen verwijzingen meer bevatten naar andere variabelen. Dus CGI_ROOT bevat eerst SERVERNAME met daarachter /test. Dit wordt de tweede evaluatie dus http://noc2.azr.nl/test.

In het Perl-script kan ik eenvoudig een link naar een andere pagina coderen als volgt:

```
print "<td><a href='$CGI_ROOT/email/enter_email.cgi'>E-
mail adres controleren<a><br></td>\n";
```

De $CGI_ROOT wordt netjes vertaald naar **http://noc2.azr.nl**/test dus deze regel wordt uiteindelijk in html het volgende:
<td><a href='http://noc2.azr.nl/test/email/enter_email.cgi'>
E-mail adres controleren<a>
</td>

Standaard header door middel van een Perl script

```
1       #!/opt/perl5/bin/perl
2       print "<HEAD>\n";
3       print "<TITLE>$Titel</TITLE>\n";
4       print "</HEAD>\n";
5       print "<LINK rel='stylesheet'
                                href='$DOC_ROOT/css/azr.css'>\n";
6       print "<STRONG>Supportpagina   -  $Titel</STRONG>\n";
7       print "<HR>\n";
```

Dus door gebruik te maken van de DOC_ROOT variabele wordt op de juiste locatie een Cascading Stylesheet geladen. De variabele Titel wordt ook hier weer gebruikt voor het geven van een mooie titel boven al mijn pagina's. Deze oplossing levert ook veel minder code op en wordt dus efficiënter uitgevoerd door de webserver.

Lokaal script of lokale functie aanroepen vanuit Perl

Dit is nog redelijk te begrijpen, toch? Nu gaan het wat ingewikkelder maken door vanuit een Perl script een Unix-commando aan te roepen via een Pipe. Het Perl script krijgt een aantal variabelen binnen die in de URL aanroep zijn opgenomen:

http://blabla/hw_ping.cgi?pingadres=172.16.37.239.

Het vraagteken achter de cgi-pagina (hw_ping.cgi) geeft aan dat er hierachter een variabele is gedefinieerd. In dit geval is dat pingadres met de waarde 172.16.37.239. Perl moet weer zo slim zijn om dit te ontrafelen uit de URL en doet dat als volgt:

```
foreach (split('&', $ENV{'QUERY_STRING'})) {
  s/\+/ /g ;
  ($name, $value)= split('=', $_, 2) ;
  $name=~ s/%(..)/chr(hex($1))/ge ;
  $value=~ s/%(..)/chr(hex($1))/ge ;
  $in{ $name} .= "\0" if defined($in{ $name} ) ;  # concatenate multiple vars
  $in{ $name} .= $value ;
}
```

Kort uitgelegd betekent dit dat de QUERY_STRING (de complete URL) wordt opgevraagd uit ENV (de environment van Perl met daarin variabelen). Het split commando neemt deze tekst en kijkt of ampersands (&) in voorkomen. Als ik namelijk meerdere variabelen wil doorgeven dan scheidt ik deze met behulp van de ampersand. Bijvoorbeeld:

http://blabla/hw_ping.cgi?pingadres=172.16.37.239&mode=1.

Dus de splitfunctie zou nu twee resultaten opgeven, te weten pingadres=172.16.37.239 en mode=1. In de array 'In' worden uiteindelijk de afzonderlijke variabele naam en de bijbehorende waarde (kijk naar de split op het isteken) opgeslagen zodat we die later weer kunnen gebruiken.

```
$pingadres=$in{ 'pingadres'} ;
$pingparam="$pingadres -n 1";
$pingpath="/usr/sbin/ping";

open (PINGPONG, "$pingpath $pingparam  |");
@Output2=<PINGPONG>;
close(PINGPONG);
$pingcount=`echo @Output2[ 1]  | grep -c "bytes"`;

print "<br>Pingen naar: <strong>$pingadres</strong><br>";
print "<br><br>";
if ( $pingcount != "0" )
{
  print "De netwerkinterface card (NIC) reageert goed";
} else{
  print "Er wordt niet geantwoord!!!";
}
```

Het bovenstaande wordt gelezen als volgt: de variabele pingadres wordt gehaald uit de array in met als zoekparameter pingadres. De variabele pingparam krijgt dit adres met als opties –n 1. De variabele pingpath bevat de volledige verwijzing naar het ping-commando. Uiteindelijk wordt dit gevoerd aan het open commando met een pipe daarachter (kijk naar het filterteken). De handle die ik terugkrijg van dit opencommando heet PINGPONG. Ja, ik hou wel van een grapje.

Het Perl script krijgt een aantal variabelen binnen die in de URL-aanroep zijn opgenomen

In de array (@) Output2 wordt alle uitvoer van het pingcommando via de handle PINGPONG toegekend. Daarna wordt de handle netjes weer gesloten. We gebruiken nu een leuke truck om door middel van quotes een extern programma's even aan te roepen. Dus de uitvoer wordt getoond, de uitvoer daarvan gaat door een pipe naar het unix grep commando. De optie –c van grep betekent Count (oftewel, tel het aantal voorkomens). En dan het criteria waarop gekeken moet worden in dit geval het woord 'bytes'. Ik doe dit omdat bij een succesvolle ping het volgende bijvoorbeeld wordt uitgevoerd:

```
PING 172.16.37.239: 64 byte packets
64 bytes from 172.16.37.239: icmp_seq=0. time=0. ms
```

en bij een onsuccesvolle bijvoorbeeld:

```
/usr/sbin/ping: unknown host 172.16.372.39
```

Om te weten of een ping is geslaagd kunnen we uit de uitvoer dat afleiden door te kijken of er het word 'bytes' in voorkomt. Daarvoor is dus die grep –c. Het resultaat van dit commando wordt weer gevoerd aan de variabele pingcount. Daarna wordt gekeken of pingcount ongelijk is aan nul en wordt een tekst getoond anders wordt er een andere tekst getoond.

Janno Hordijk is I & A manager bij F.S.M. Europe B.V. in Sittard.

In zijn vrije tijd is hij directeur van zijn eigen bedrijf I.S.D. (Innovative System Development) wat sinds 01/08/1998 is opgericht. Binnen I.S.D. ontwikkelt hij programmatuur met behulp van VO en ColdFusion.

Daarnaast is Janno veel terug te vinden op sdgn.visualobjects en op comp.lang.clipper.visual-objects. Zijn homepage is te vinden www.isdevelop.nl

auteur
Bob Swart

COM met Delphi
(Introduktie)

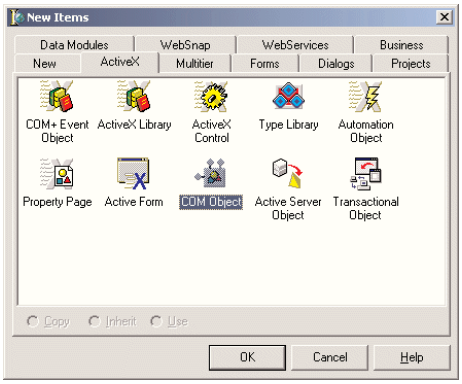
Elke ontwikkelaar heeft van COM gehoord, alhoewel niet iedere ontwikkelaar er evenveel ervaring mee heeft. Voor sommige programmeurs is COM nog iets engs, alhoewel het dat absoluut niet is. Het is net als zwemmen: je moet even doorkomen, maar als je er eenmaal inzit dan is het hartstikke leuk.
In dit artikel introduceer ik COM - Microsoft's Component Object Model - met behulp van Delphi. Ik zal COM Objecten en OLE Automation en behandelen, en de volgende keer verder gaan met MTS en COM+. Daarna zal ik de opgedane kennis en ervaring toepassen in een .NET platform (alhoewel Delphi daar tegen die tijd al wat meer mogelijkheden voor heeft)

COM Objects

Zoals ik in de introductie al aangaf: COM staat voor Component Object Model, en met behulp van COM kunnen we objecten bouwen (en gebruiken) die draaien in een Windows omgeving, en volledige ontwikkelomgeving of programmeertaal onafhankelijk zijn. We kunnen dus in Delphi een COM object bouwen en dat in Visual Basic of C++ gebruiken (en andersom). In dit artikel zal ik mij beperken tot bouwen en gebruiken vanuit Delphi zelf - raadpleeg de documentatie behorende bij je eigen favoriete Windows ontwikkelomgeving voor de mogelijkheden tot het importeren en gebruiken van deze COM objecten.

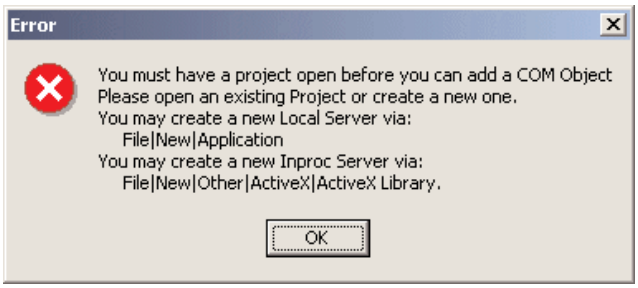
Nieuw COM Object

Om met een eenvoudig voorbeeld te beginnen, gaan we een nieuw generiek COM Object maken met Delphi 6 (straks zal blijken dat er nog meer "speciale" COM mogelijkheden zijn, zoals een Automation Object, een Transactional Object of een Activer Server Object). Start Delphi 6, en sluit het default project (dat willen we niet gebruiken). Doe nu *File | New - Other* en ga naar de ActiveX tab van de Object Repository.



figuur 1.

Hier staan de verschillende COM en ActiveX Wizards waarvan we er deze keer al een paar zullen gebruiken. We beginnen met een "normaal" COM Object, dus selecteer het icoontje en klik op OK. Dit geeft helaas een foutmelding, omdat COM Objecten alleen maar aan een project (executable of ActiveX DLL) toegevoegd kunnen worden. Dit worden dan resp. out-of-process (in een executable) en in-process (in een DLL) COM objecten overigens.

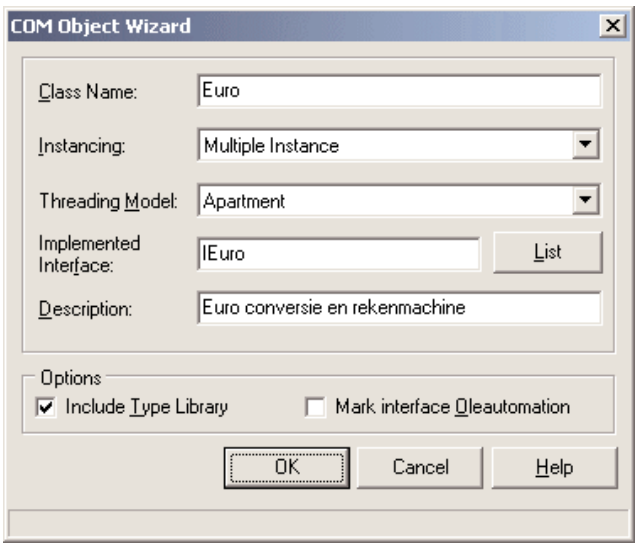


Figuur 2.

Laten we beginnen met een in-process server, dus doe nu weer File | New - Other, ga naar de ActiveX tab van de Object Repository en kies voor de ActiveX Library. Bewaar dit project in Euro42.dpr. Het bevat de volgende paar regels code:

```
library Euro42;
uses
  ComServ;

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.RES}

begin
end.
```

Nu we de container voor onze COM Objecten hebben, kunnen we (voor de tweede keer) | *New - Other* doen, en deze keer wel een COM Object toevoegen, wat resulteert in de volgende dialoog:

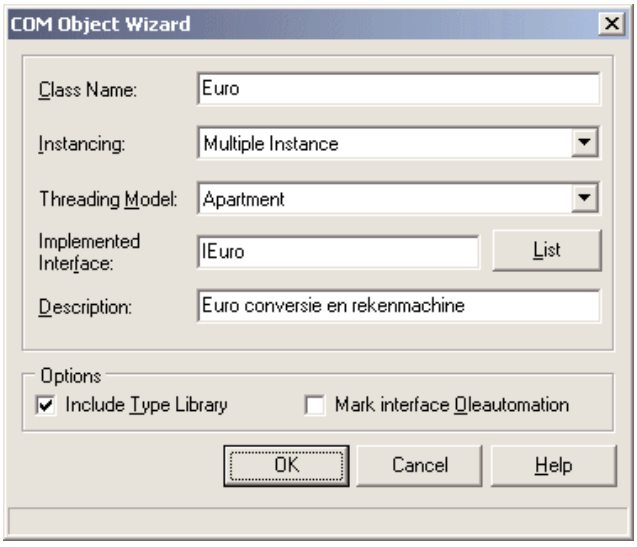


Figuur 3.

De Class Name zal gebruikt worden als interne en externe naam van het COM Object. Hierdoor is het niet aan te raden zomaar wat in te vullen, maar een beetje zinvolle naam, zoals "Euro" in dit voorbeeld. Het resultaat is een klasse TEuro, afgeleid van TTypedComObject, die het IEuro interface implementeert.

We kunnen dus in Delphi een COM object bouwen en dat in Visual Basic of C++ gebruiken (en andersom)

De Instancing staat default al op Multiple Instance (zodat er meerdere instanties van onzer COM Server tegelijkertijd aktief kunnen zijn), en het Threading Model staat op Apartment (zodat de verschillende instanties niet bij elkaars instance data kunnen - alleen globale variabelen zijn nog onveilig uiteraard). Zodra we de CoClass Name hebben ingevuld, zal automatisch de naam van het interface ook ingevuld zijn (in ons voorbeeld dus IEuro - afgeleid van IUnknown). Laat de checkbox "Include Type Library" aan staan, zodat we straks de Type Library Editor kunnen gebruiken om properties en methoden aan dit nieuwe interface toe te kennen.
De laatste optie "mark interface Oleautomation" ga ik nu nog niet gebruiken, dus het maakt niet uit of die aan of uit staat op dit moment (ik zelf zet hem nu uit).



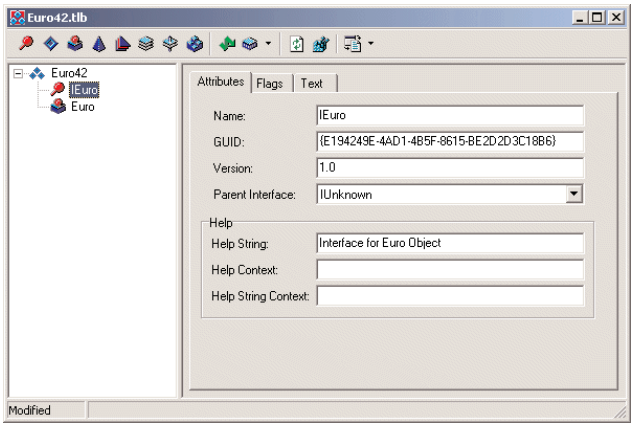
Figuur 4.

Na een druk op de OK knop zal de volgende unit gegenereerd worden (bewaar deze in unit Euro.pas):

```
unit Euro;
{$WARN SYMBOL_PLATFORM OFF}
interface
uses
  Windows, ActiveX, Classes, ComObj, Euro42_TLB, StdVcl;

type
  TEuro = class(TTypedComObject, IEuro)
  protected
  end;

implementation
uses
  ComServ;

initialization
  TTypedComObjectFactory.Create(ComServer,
    TEuro, Class_Euro, ciMultiInstance, tmApartment);
end.
```

Type Library Editor

Behalve de nieuwe unit, krijgen we ook meteen de Type Library Editor te zien. Dit is de plek om nieuwe properties en methoden toe te voegen aan het COM Object - of specifieker gezegd, aan het IEuro interface. De Type Library kan gezien worden als de definitie van onze COM Objecten - letterlijk het interface met o.a. de properties en methoden, die met behulp van de Type Library Editor op eenvoudige wijze aan te passen en te onderhouden zijn. De Type Library zelf heeft de naam van het project en de extentie .tlb, en de import unit is Euro42_TLB.pas (dit bestand wordt iedere keer opnieuw gegenereerd nadat we een wijziging in de type library hebben gemaakt, dus het is over het algemeen niet zinvol om zelf wijzigingen in de type library import unit te maken).

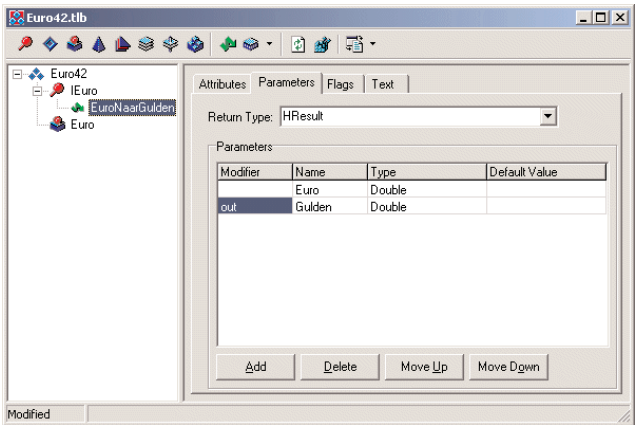


Figuur 5.

We kunnen nu zelf nieuwe properties of methoden toevoegen aan het IEuro interface. Voor een nieuwe methode moeten we op de knop met de groene pijl drukken (tip: je kan de toolbar uittrekken waardoor de labels onder de knoppen komen - handig voor wie nog niet zovaak met de Type Library Editor heeft gewerkt). Laten we de methode EuroNaarGulden toevoegen, voor wie nog wat moeite heeft met de euro te rekenen en in zijn/haar hoofd(stiekem) alles nog in guldens doet.

Ga op IEuro staan, druk op de New Method button, en noem de nieuwe methode dus EuroNaarGulden. Er moeten nog wel wat argumenten bijkomen, dus klik op de Parameters tab. Wat we daar aantreffen kan er op twee verschillende manieren uitzien: de "Pascal" manier of de "IDL" manier. De eerste laat alle types zien zoals Delphi ontwikkelaars die een beetje kennen (zoals WideString), terwijl de tweede juist de taal-onafhankelijke COM IDL (interface definition language) gebruikt - handig voor wie vaker met COM werkt. Ik heb zelf niet echt een voor-

keur, en switch regelmatig heen en weer, wat je kan doen via Tools | Environment Options en dan de Type Library tab (de optie "language" kan op Pascal of IDL gezet worden). Laten we uitgaan van de setting op Pascal, dan moeten we twee parameters toevoegen, dus druk tweemaal op de Add knop. Noem de eerste parameter Euro en de tweede Gulden. Ze zijn allebei van type Double. Tot slot moeten we via de modifier aangeven of het om input, output of input/output parameters gaat. Default staat deze uit, wat input betekent. Echter, de Guldens parameter is een output parameter, en we moeten daarom de Modifier op "out" zetten:



Figuur 6.

Als je langere tijd met de Type Library Editor werkt kan het geen kwaad om aan het eind even op de witte knop met de twee kleine groene pijlen te klikken (de "refresh implementation" knop), die zorgt er namelijk voor dat de unit Euro.pas met daarin de TEuro klasse weer synchroon loopt met de wijzigingen die we in de type library hebben aangebracht. In dit geval hebben we een nieuwe functie die als volgt gedefinieerd is:

```
type
  TEuro = class(TTypedComObject, IEuro)
  protected
    function EuroNaarGulden(Euro: Double;
        out Gulden: Double): HResult;
      stdcall;
  end;
```

Merk op dat ook het implementatie skelet er al staat (nog leeg, maar in tegenstelling tot event handlers blijven lege COM methoden gewoon staan gelukkig). De implementatie is uiteraard niet moeilijk:

```
function TEuro.EuroNaarGulden(Euro: Double;
                out Gulden: Double): HResult;
begin
  Gulden := Euro * 2.20371; // implementatie
end;
```

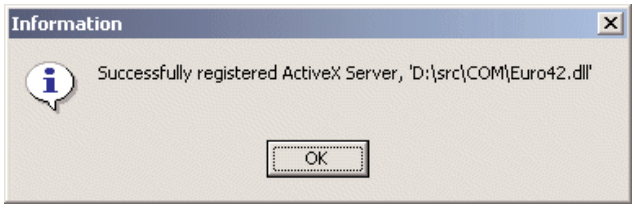
Compileer nu het Euro42 ActiveX project, wat tot een warning leidt: de return-waarde van de funktie EuroNaarGulden is niet gedefinieerd. Dat klopt, daar hebben we nog niks aan gedaan. Wie nog eens kijkt op Fig. 6 ziet dat daar als return type nog steeds de default type HResult staat. Waarom hebben we een funktieresultaat van type HResult eigenlijk nodig? Dat komt omdat het in de COM wereld een beetje gebruikelijk is om COM methoden een HResult waarde als mogelijke foutmelding terug te geven. Als HResult gelijk is aan S_OK dan is alles OK (dat lag voor de hand), en als HResult iets anders is, dan is er iets misgegaan. Omdat COM Objecten door vele verschillende omgevingen gebruikt kunnen worden, lijkt het me altijd beter om me maar aan de regels te houden, dus voeg ik één regel toe aan de implementatie van EuroNaarGulden, namelijk Result := S_OK.

```
function TEuro.EuroNaarGulden(Euro: Double;
                out Gulden: Double): HResult;
begin
  Gulden := Euro * 2.20371; // implementatie
  Result := S_OK;
end;
```

Nu kunnen we ons Euro42 project compileren, maar nog niet draaien, omdat het een DLL is en geen executable. COM Objecten worden gebruikt door zogenaamde COM Clients, en die gaan we nu bouwen.

Register COM Objects

Maar voordat we een COM Client gaan bouwen, moeten we eerst de COM Server gaan registreren. Dat kan met behulp van de *Run | Register ActiveX Server* menu optie. Hierdoor zijn meteen alle COM objecten in de Euro42.dll geregistreerd voor gebruik.



Figuur 7.

Als alternatief kun je regsvr32 aanroepen met de Euro42.dll als argument. Het extra argument /uninstall zorgt ervoor dat de COM Server weer verwijderd wordt.

COM Objecten gebruiken

Een COM Client kan zo'n beetje elk denkbaar Delphi project zijn, dus start maar een gewone nieuwe applicatie. Bewaar het main form in MainForm.pas en het project in EuroClient.dpr. We moeten nu zowel de ComObj unit als de Euro42_TLB.pas (type library import unit) aan de uses clause toevoegen, zodat het IEuro interface bekend is en gebruikt kan worden. Om het gebruik te demonstreren hebben we verder twee TEdits en een TButton nodig. Geef de TEdits de namen edtEuro en edtGulden, en de TButton de naam btnConvert. De code voor de OnClick event handler van de btnConvert is als volgt:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Euro: IEuro;
  Gulden: Double;
begin
  Euro := CreateCOMObject(Class_Euro) as IEuro;
  Euro.EuroNaarGulden(StrToFloatDef(edtEuro.Text,0),
                      Gulden);
  edtGulden.Text := Format('hfl. %.2f',[Gulden]);
end;
```

Als je dit compileert en uitvoert, dan zal de CreateCOMObject een instantie van onze COM Server genaamd "Class_ Euro" voor ons maken. De Type Library ►

► import unit bevat de definitie van de Class_Euro net als het IEuro interface zelf. Zodra de toepassing draait kunnen we 100 euro invullen en op de btnConvert button klikken voor het volgende resultaat:

Figuur 8.

Hoe zat dat nou met S_OK?

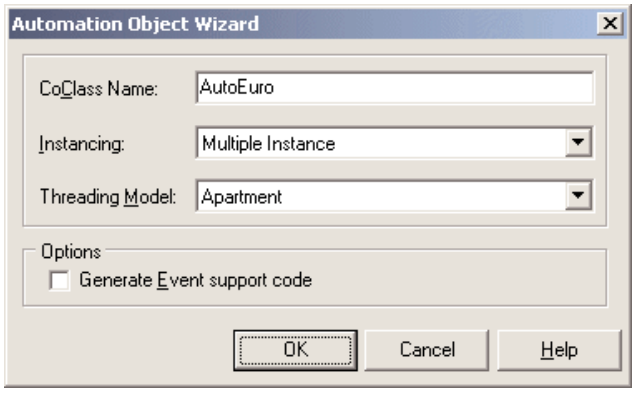
Eerder vertelde ik dat de EuroNaarGulden methode eigenlijk een funktie was die een HResult waarde teruggeeft - in ons geval altijd S_OK. Maar wat moeten we eigenlijk doen met deze HResult waarde? Welnu, het is "good practice" om iedere COM methode (die een HResult waarde teruggeeft) aan te roepen binnen een speciale wrapper genaamd OleCheck. Die controleert het resultaat, en als dat niet S_OK is krijgen we een foutmelding. Om toch maar netjes COM te gebruiken, zouden we dus de implementatie van de OnClick event handler als volgt kunnen herschrijven:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Euro: IEuro;
  Gulden: Double;
begin
  Euro := CreateCOMObject(Class_Euro) as IEuro;
  OleCheck(Euro.EuroNaarGulden(
           StrToFloatDef(edtEuro.Text,0),Gulden));
  edtGulden.Text := Format('hfl. %.2f',[Gulden])
end;
```

En als je echt netjes COM wil gebruiken moet je uiteraard ook een try-except blok om de CreateCOMObject heen zetten, voor het geval de COM Server niet gevonden kan worden op de client machine. Dat laat ik verder aan de lezer over, want zelf gaan we nu door met OLE Automation objecten.

Automation

Een Automation object is een speciaal COM object - het speciale zal straks duidelijk worden. We kunnen een nieuw automation object gewoon toevoegen aan onze bestaande Euro42 ActiveX Library (er kunnen meerdere COM objecten in dezelfde DLL opgenomen worden), met File | New - Other, en dan van de ActiveX tab het Automation Object icon. Dit geeft de volgende dialoog, die lijkt op die van een "normaal" COM Object:



Figuur 2.

We hoeven nu eigenlijk alleen maar weer de naam van de CoClass in te vullen, die ik op AutoEuro zet (we hebben Euro al gebruikt voor het normale COM Object). Instancing en Threading laat ik weer op hun default waarden staan, en events gebruik ik niet deze keer.

Als we op OK klikken (bewaar de gegenereerde unit in AutoEuro.pas) komen we weer in de Type Library Editor terecht, die nu het interface bevat van zowel IEuro als IAutoEuro. En als je die twee met elkaar vergelijkt dan zie je dat IEuro van IUnknown is afgeleid, terwijl IAutoEuro van IDispatch is afgeleid. Het verschil zit hem in de "dispatch", wat zoveel betekent dat de methoden van het interface dynamisch opgezocht, toegewezen en uitgevoerd kunnen worden. IDispatch is op zijn beurt namelijk weer afgeleid van IUnknown, maar bevat een aantal extra methoden, namelijk GetTypeInfoCount, GetTypeInfo, GetIDsOfNames en Invoke. Hier kom ik later op terug.
Ook de implementatie klassen verschillen van elkaar: TEuro is afgeleid van TTypedComObject, terwijl TAutoEuro afgeleid is van TAutoObject (die het IDispatch interface implementeert, dus we hoeven ons zelf niet druk te maken hierover).

Er is nog iets interessants te zien in de Euro42_TLB.pas unit, en dat is het feit dat er behalve de IAutoEuro interface definitie (afgeleid van IDispatch) ook een IAutoEuroDisp zogenaamd "dispinterface" aanwezig is, die dezelfde GUID heeft. Het effect hiervan zullen we straks zien, als we de AutoEuro op drie verschillende manieren kunnen gaan gebruiken.

Nieuwe methoden

Laten we eerst nog even twee methoden toevoegen. Dat gaat weer met behulp van de Type Library Editor net als met de vorige IEuro interface. Deze keer noem ik de methoden EuroToGuilder en GuilderToEuro, en geef ze wederom twee argumenten mee (het tweede argument weer van type "out"), die deze keer echter van type Currency zijn.
De implementatie van EuroToGuilder en GuilderToEuro zal niet zo'n verrassing zijn denk ik:

```
const
  GuilderPerEuro = 2.20371;

procedure TAutoEuro.EuroToGuilder(Euro: Currency;
                                  out Guilder: Currency);
begin
  Guilder := Euro * GuilderPerEuro;
end;

procedure TAutoEuro.GuilderToEuro(Guilder: Currency;
                                  out Euro: Currency);
begin
  Euro := Guilder / GuilderPerEuro;
end;
```

En hierna kunnen we het Euro42 project weer compileren en opnieuw registreren (zodat ook het AutoEuro automation object gevonden kan worden).

Automation Objecten gebruiken

En nu komen we bij een interessant deel van het artikel: waar we "normale" COM Objecten slechts op één manier kunnen gebruiken (met CreateCOMObject), daar kunnen we Automation Objecten op drie verschillende manieren gebruken, via variants (het meest flexibel, maar ook "gevaarlijkste"), via dispatch interfaces (ook wel de dispinterfaces) en via interfaces (de manier van de normale COM Objecten).

Variants

Waarom zouden we eigenlijk variants willen gebruiken? De belangrijkste reden is gemak. Variants maken gebruik van "late binding" om de methoden van het Automation Object aan te roepen. Ze hoeven dus niet tijdens compilatie al te weten welke methoden en argumenten beschikbaar (en geldig) zijn, maar zoeken dit tijdens het draaien wel uit. Helaas betekent dit ook dat er geen Code Insight ondersteuning meer is. Met alle gevolgen van dien, als iemand een tikfoutje gemaakt heeft, want de methode EuroToGulden bestaat echt niet.
Het voordeel is dat we geen Type Library import unit nodig hebben, en dat we ieder Automation Object op onze machine op deze manier kunnen benaderen (waaronder bijvoorbeeld de Office onderdelen), wat op z'n tijd best handig is. Voor het gebruik van de AutoEuro via Variants, is de code als volgt:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Euro: Variant;
  Guilder: Currency;
begin
  try
    Euro := GetActiveOleObject(EuroClassName);
  except
    Euro := CreateOleObject(EuroClassName);
  end;
  Euro.EuroToGuilder(StrToFloat(edtEuro.Text),Guilder);
  edtGuilder.Text := FloatToStr(Guilder);
end;
```

Hoe werkt die Variant dan? Welnu, die gebruikt het IDispatch interface, die via de GetIDsOfNames een methode naam gebruikt om een dispatch ID op te halen, en vervolgens via het dispatch ID de daadwerkelijke methode aanroept. Dit zoeken kost tijd, en de Variants oplossing is dan ook de langzaamste van alle drie de oplossingen die we zullen zien.

Dispatch Interfaces

Een stapje "hoger" (en sneller) is het gebruik van dispatch interfaces. Ook hierbij wordt late binding gebruikt. Echter, er is wel Code Insight ondersteuning aanwezig, dus we kunnen toch de aanwezige methoden en hun argumenten zien. Dat komt omdat het dispinterface alle methoden (en argumenten) definieert, zodat we hieruit kunnen kiezen. Op het moment dat de methoden worden aangeroepen, wordt via late binding het dispatch iD gebruikt om de juiste methode te vinden en aan te roepen. Er is geen GetIDsOfNames meer nodig (want het ID kennen we al), dus dit werkt niet alleen handiger (met Code Insight) dan de Variants oplossing, maar ook sneller.
De code voor het gebruik van AutoEuro via dispatch interfaces is als volgt:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  Euro: IEuro42Disp;
  Guilder: Currency;
begin
  Euro := CoEuro42.Create AS IEuro42Disp;
  Euro.EuroToGuilder(StrToFloat(edtEuro.Text),Guilder);
  edtGuilder.Text := FloatToStr(Guilder);
end;
```

Hier is overigens wel weer de Type Library import unit Euro42_TLB.pas voor nodig.

Interfaces

Het laatste voorbeeld maakt gebruik van de "normale" interfaces en early-binding zoals we die eerder zagen voor het IEuro object. De code voor het gebruik is hierbij als volgt:

```
procedure TForm1.Button3Click(Sender: TObject);
var
  Euro: IEuro42;
  Guilder: Currency;
begin
  Euro := CoEuro42.Create;
  Euro.EuroToGuilder(StrToFloat(edtEuro.Text),Guilder);
  edtGuilder.Text := FloatToStr(Guilder);
end;
```

Het verschil tussen de drie methoden van gebruik zal duidelijk zijn. Soms is het noodzakelijk om variants te gebruiken, maar als het even kan pak ik de dispatch interfaces. Los van het gebruik echter, is het hopelijk ook duidelijk dat Automation Objecten (vanwege het feit dat ze op meerdere manieren gebruikt kunnen worden) te prefereren zijn boven normale COM Objecten. Ik bouw zelden meer een "normaal" COM Object, maar kies bijna altijd voor een Automation Object, zodat er potentieel meer clients mee kunnen praten. En hoe meer clients, hoe meer vreugde, nietwaar?

Over meer clients gesproken: de volgende keer ga ik het hebben over MTS (Microsoft Transaction Server) en COM+. Tot dan!

Bob Swart is een freelance schrijver, trainer en webmaster van Dr.Bob's Delphi Clinic te http://www.drbob42.nl

auteur
Rod da Silva

A VO Framework For Bulletproofing Your Code

Introduction

In general, software developers spend too little time and effort dealing with the possibility of a software exception occurring in their applications. Even though VO has rich semantics for handling errors, few developers employ a strategy to make use of them fully and from the outset in their application design. This paper won't talk about VO's error handling system. Rather, it will outline a reusable framework that makes software robust in the face of exceptions. In today's modern software environment where distributed components with clients written in different languages are common, it is not enough to show an error box, or log the error to disk when something goes wrong. Catching the exception is the easy part. Exception propagation in a consistent and standard way throughout your application is what bulletproofing your code is really all about.

All software will fail in some way, eventually

This paper will attempt to prove to you that it is possible to write stable code from the beginning without having to distract your application logic with obtrusive error handling code. It is assumes you are already familiar with VO's error handling capabilities, and focuses instead on how to employ them to create a framework for bulletproof application code.

The Pessimistic Programming Model

Bugs happen. It's a fact of life. What most people don't appreciate is that most "bugs" are not logic errors or problems with the software per se. It's virtually impossible to write a non-trivial piece of code that does not depend in some way on the environment in which it is running. Whether it's some system resource such as memory or hard disk space, or data from a database, or even something as basic as time, programs have implicit and/or explicit dependencies on their runtime environment. Moreover developers make assumptions about these dependencies. Most problems with software, even well tested software, occur when they are faced with abnormal environments—environments where the basic assumptions made by the developer are being challenged.

It's taken me a decade of study at the school of hard knocks to learn my lessons about writing robust software, but I have learned them well. I have come to appreciate what I refer to as the Pessimistic Programming Model (PPM), which simply states that all software will fail in some way, eventually. By "fail" I mean behave in an unexpected way. Armed with this hard learned knowledge, I now always program very defensively, expecting and prepared for the worst a computer can throw at me at any time. I want to stress that it doesn't matter how good a developer you are, or how bug free your software appears to be, you cannot claim your software is robust if you haven't programmed using the PPM. This paper is dedicated to showing you how to write robust software without slowing you down or otherwise asking you to learn new programming methodologies.

Writing Robust Clients and Servers

Most non-trivial software can be loosely described using a client-server model. That is, in general, software can be divided into that which consume the services of other software such as external components, operating system APIs, or internal functions or subroutines—or else it is a provider of such services. The former can be thought of as the software clients and the latter as the software servers.

Exception handling has to be dealt with properly in both of these contexts. However, perhaps the most important thing that must be decided upon is how the two types of software will communicate exception context information between each other. This communication mechanism profoundly influences the implementation of both types of software and therefore must be decided upon as early in the software project as possible.

Exception Codes

There are a number of different ways to return exception context information from a service exposed by server software, to client software that is calling it:

- Return it as the service's return value
- Return it as one or more of the service's out parameters
- Write the information to a place that can be shared by both client and server (e.g.; disk, shared memory, registry, database, etc.,)

Regardless of how it is done, it is imperative that it is done for one important reason; knowing whether a ser-

vice call succeeded or not, and why if it did not, is imperative to writing robust, deterministic software. This means that having a service return simply TRUE or FALSE as to its success is not enough because while a result of TRUE is self explanatory, a result of FALSE generally is not. For example, consider a function such as IsThereEnoughDiskSpace("D:", 1024) which takes a drive letter and an amount of space in bytes and returns TRUE if the drive has free space equal to or greater then the amount requested. While a return result of TRUE is self explanatory, a return result of FALSE remains ambiguous, because we do not know for sure if the drive does not have the space, or whether the reality is something more fundamental such as there is no D: drive defined on the computer the software is running on.

The problem is not just with functions that return Boolean values either. Consider a related function such as DriveSpaceAvailable("D:") which is intended to return the number of bytes of free disk space on the supplied drive. Again the problem occurs when an ambiguous value of 0 is returned. Does it mean the drive is full, or that drive D: doesn't exist? Perhaps D: does exist but it is not formatted?

The issue here, in general, is that in order to properly interpret the result of a service call, the success of the call must be unambiguously communicated independently of any semantic values the service may produce.

Therefore a mechanism for communicating the service's success must be agreed upon by both parties to the contract—the client and server. Anyone of the methods listed above can be used, but for the purposes of this paper, and to retain compatibility with the Win32 API's recommend approach, I will use the return value of all server services to return a status code indicating success or failure of the call. This notification will be in the form of a signed 32-bit integer, referred to as an HRESULT by the Win32 API documentation (see the MSDN documentation for more information on the internal structure of a 32-bit HRESULT code). By convention, an HRESULT of value zero (S_OK) means the service succeeded, whereas a negative value indicates an unexpected result occurred (i.e.; an error). Finally, the value S_FALSE (defined as 1) indicates that while no error occurred, the call did not succeed semantically.

Applying this approach to the IsThereEnoughDiskSpace ("D:", 1024) function above, we could unambiguously describe the result of the call as S_OK meaning TRUE, S_FALSE meaning FALSE, and E_NO_SUCH_DRIVE, E_DRIVE_NOT_FORMATTED, etc., as independent negative error codes. To handle the DriveSpaceAvailable() function described above, you would have to redesign the function to return the number of bytes available as an out parameter of the function as in DriveSpaceAvailable ("D:", @dwNumBytes). Under this scheme, a return HRESULT of S_OK indicates that dwNumBytes contains the correct answer. Whereas any other negative error code such as E_NO_SUCH_DRIVE or E_DRIVE_NOT_FORMATTED indicates the corresponding error occurred and, importantly, that dwNumBytes is undefined.

Sacrificing Implementation Elegance for Robustness

I hope the above discussion sounds reasonable to you and you are considering using HRESULT status codes in the future to indicate the success or failure of all your server services. However, committing fully to this approach has some definite consequences that you need to be aware of and accept. These consequences necessarily give you less freedom in how you design your server software interfaces. However, it is my opinion that these constraints are the price of robust software and that you should resolve yourself to them sooner rather then later so you can get on with the business of writing bulletproof code.

Functions and Methods Only

The first constraint is a result of the requirement that all of your server services must return an HRESULT. This necessarily means you can't return anything else. The immediate consequence of this is that if your server software is implemented as a class, then you can only use methods to expose functionality. Specifically, you cannot use any ACCESSes or ASSIGNs to expose public properties of your class. To do so would forfeit your ability to unambiguously communicate whether the property access/assignment was successful. This is particularly important for the implementation of "virtual" properties—properties whose values are computed rather then stored—since often times the logic to compute these values is non-trivial and needs proper error handling.

[Note that an exception to this rule is the use of EXPORT instance variables as part of an exposed server class' public interface. EXPORTs can be safely used since presumably there is no status to check when accessing or assigning them. That is, an access or assignment to a concrete EXPORT instance variable is always assumed to succeed (the only erroneous case is where you attempt to assign a value of the wrong type to the instance variable, in which case the compile-time type checking process should catch the problem). However, exposing public instance variables on your class often has other undesirable consequences from a class encapsulation standpoint, and as such is generally frowned upon by the object-oriented programming community and should be avoided.]

The second constraint, which is really a corollary consequence to our first constraint, is that you forfeit your ability to expose entire class hierarchies (such as those

exposed by popular ActiveX servers like MS Word and MS Excel) as your server's interface. The reason once again is that your services must always return an HRESULT, which means they cannot return instances of other classes (objects) of the exposed class hierarchy, in the familiar and natural way that popular ActiveX server components do. The result is that you are forced to flatten your class hierarchies to look more like COM interfaces—collections of semantically related functions.
Finally, as has been mentioned already, all actual semantic data values produced by your services must be returned to the caller using out parameters (i.e.; via parameters passed by reference). For "property" services that manipulate single values, this necessarily means the adoption of a Get<PropertyName>(@Value) and Set<PropertyName>(Value) (or similar) programming style—reminiscent of C++ programming—which may take some getting use.

*[**WARNING:** You should be advised that Microsoft's JavaScript implementation known as JScript does not support the passing of values by reference and therefore cannot be used to call services written in the above style.]*

While these may seem like harsh restrictions, the payoff for having a consistent way to test for service failure is well worth it. What you have to realize (and what is not apparent until you get very far into the development cycle) is that debugging an n-tier architecture project for example is much more difficult then debugging traditional systems. This is because the code execution is split across multiple component layers often written in different computer languages. Trust me when I tell you that the information returned from these consistent status codes is critical when trying to find a problem.

Writing Robust Servers

There is no point in having your server services dutifully return status codes if you do not go to the effort of properly trapping the errors within those services in the first place. Simply stated, you must not allow your services to crash! I am dead serious. Absolutely without exception, each and every one of your services must be properly wrapped by a CA-Visual Objects local error handling context implemented with a BEGIN SEQUENCE...RECOVER USING...END construct. If you ignore all my other advice in this paper, listen to me now. I estimate that fully 40% of all the business logic in my last server component project was in support of proper error handling. And the results were well worth it. While I had lots of logic bugs to find (i.e.; the services returned lots of negative status codes) the component itself virtually never crashed. I can't tell you how this eases the debugging process of a distributed system!

You have to decide on a template to use for each and every service you write and then stick to it. The one I use looks something like:

```
METHOD SomeService( … ) CLASS ServerComponent
   LOCAL cbOrgErrorHandler      AS CODEBLOCK
   LOCAL uxError                AS USUAL
   LOCAL hResult                AS LONG

   // Immediately set up a local error handler
   cbOrgErrorHandler := ErrorBlock( {|e| _Break( e ) } )

   // Default to no error
   hResult := S_OK

   // Create local error handler context
   BEGIN SEQUENCE
         //
      // …Actual service code goes here…
      //
      // Call other support services
      // which also use THIS template
      hResult := SELF:SomeOtherSupportService(…)

      // You MUST check the result!
      IF FAILED( hResult )
          // Break with error code if something is wrong
         BREAK hResult
      ENDIF

      // If we make it here all is well in the world!
   RECOVER USING uxError
      // If we get a number we know we received a self-
      // induced assertion error, OTHERWISE it was an error
      // raised by the CA-Visual Objects' runtime
      IF IsNumeric( uxError )
         hResult := uxError
      ELSE
         hResult := E_UNEXPECTED // Win32 "catch all" error code
      ENDIF
      // Optional code to log the error can go here
   END
   // Restore previous error handler
   ErrorBlock( cbOrgErrorHandler )
   RETURN hResult
```

In each and every service, the first thing I do is set up a local error handler to trap any error that might occur. Then as various assertions (e.g.; validation checks) fail I explicitly call break with an appropriate error code to force execution to transfer to the RECOVER USING section of the local error handler. *(NOTE: the FAILED() function simply tests the passed parameter to see if it is negative)* From here I perform any required error logging and set the appropriate error code return value before returning from the service gracefully.
Under this scheme, the service can certainly fail but it is very hard to crash it.

you have to clean up from unwanted side effects when things go wrong

Writing Robust Clients

The second most prominent mistake I see developers make after not bulletproofing their server service code using techniques similar to above, is to exacerbate the situation by not checking the status of those service calls on the client side. If you are going to write a robust client you simply must check the success status of every server service call your client makes. Moreover, you have to clean up from unwanted side effects when things go wrong. More often then not, this is where the difficulty lies.

Many well-intentioned developers start off adding error handling code to their programs abandon the practice after a while simply because there code tends to become difficult to read and therefore maintain. We have all seen client code that looks something like:

```
IF oServer:Service1(…) == S_OK
   IF oServer:Service2(…) == S_OK
      IF oServer:Service3(…) == S_OK
         IF oServer:Service4(…) == S_OK
            // The successful case…
         ELSE
            // Clean up from Service3/2/1 calls
            lFailed := TRUE
         ENDIF
      ELSE
         // Clean up from Service2/1 calls
         lFailed := TRUE
      ENDIF
   ELSE
      // Clean up from Service1 call
      lFailed := TRUE
   ENDIF
ELSE
   lFailed := TRUE
ENDIF

IF lFailed
   // General error handling goes here
ENDIF
```

The problem here isn't with checking the status code, but rather in making sure that after every possible failure the state of the software is returned to that prior to running the code. The use of nested IF..ELSE..ENDIF constructs and their accompanying indentation levels makes the code more difficult to read and understand, especially if the clean up code is non-trivial which is often the case.

By using a local error handler on the client side as well, you can avoid this unsightly code, making the code much easier to read, write and debug. For example:

```
LOCAL cbOrgErrorHandler                         AS CODEBLOCK
LOCAL uxError                                   AS USUAL
LOCAL hResult                                   AS LONG

// Immediately set up a local error handler
cbOrgErrorHandler := ErrorBlock( {|e| _Break( e ) } )

// Default to no error
hResult := S_OK

// Create local error handler context
BEGIN SEQUENCE

   IF FAILED( hResult := oServer:Service1(…) )
      BREAK hResult
   ENDIF
   IF FAILED( hResult := oServer:Service2(…) )
      // Clean up from Service1 call
      BREAK hResult
   ENDIF
   IF FAILED( hResult := oServer:Service3(…) )
      // Clean up from Service2/1 calls
      BREAK hResult
   ENDIF
   IF FAILED( hResult := oServer:Service4(…) )
      // Clean up from Service3/2/1 calls
      BREAK hResult
   ENDIF
   // Only way to make it here is
   //if all calls where successful!!
```

```
RECOVER USING uxError
   // If we get a number we know we received a self-induced
   // assertion error, otherwise it was an error raised by
   // the CA-Visual Objects' runtime
   IF IsNumeric( uxError )
      hResult := uxError
   ELSE
      hResult := E_UNEXPECTED //Win32 "catch all" error
   ENDIF
   // Optional code to log the error can go here
END
// Restore previous error handler
ErrorBlock( cbOrgErrorHandler )
```

The secret to this code is that it takes a pessimistic approach and checks for the failure (as opposed to the success) of each service code, performing necessary clean up and breaking to a local error handler if found. This approach ensures you do not introduce superfluous nested IF constructs by allowing you to short circuit the program flow at any point something goes wrong using a BREAK.
Notice that I used the exact same error-handling template for the client that I did for the server. By creating a robust, general purpose, local error handler suitable for use with both client and server code, you will find it easier to adopt, and the results more consistent.

professional programmers can no longer afford to cut corners when it comes to error handling

Hiding Your Exception Handling Code

Looking back on my example client and server code you will notice that the bulk of it is devoted to the purpose of error handling. For example stripped away all error handling code, my server method would look like:

```
METHOD SomeService( … ) CLASS ServerComponent
   LOCAL hResult
   hResult := self:SomeOtherSupportService(…)
RETURN hResult
```

And my client code would look like:

```
LOCAL hResult     AS LONG
   hResult := Service1(…)
   hResult := Service2(…)
   hResult := Service3(…)
   hResult := Service4(…)
```

Clearly, it can be very tempting to be lazy since the error handling code can become very tedious, as it is always the same. However, it is precisely this kind of tedious code that is often a candidate for automation.

The technique I use is based on the little used *User Defined Command* (UDC) feature of CA-Visual Objects. By creating as a set of UDCs to encapsulate the error handling code, we can effectively remove it from our source code as a distraction without compromising the robustness of our software. Consider the following UDC file:

```
TRY  => LOCAL hResult:= S_OK    AS LONGINT ;; ;
        LOCAL cb                AS CODEBLOCK ;; ;
        LOCAL uxError           AS USUAL;; ;
        cb := ErrorBlock( {|e| _Break(e) } );; ;
        BEGIN SEQUENCE

ENDTRY =>  RECOVER USING uxError;; ;
           IF IsNumeric( uxError );; ;
               HResult := uxError ;; ;
           ELSE ;; ;
               HResult := E_UNEXPECTED ;; ;
           ENDIF ;; ;
           END ;; ;
           ErrorBlock( cbOrgErrorHandler ) ;; ;
           RETURN hResult

AFFIRM <exp>      =>       hResult := <exp> ;; ;
          IF hResult != S_OK;; ;
              BREAK hResult ;; ;
          ENDIF
```

By including this UDC file into our server software the previous longwinded service code can be rewritten as follows with a significant decrease in distractions due to error handling code yet, with no loss of robustness:

```
METHOD SomeService( … ) CLASS ServerComponent
  TRY
AFFIRM (self:SomeOtherSupportService(…))
  ENDTRY

Likewise, our client code would look like:

TRY
AFFIRM( Service1(…) )
AFFIRM( Service2(…) )
AFFIRM( Service3(…) )
AFFIRM( Service4(…) )
ENDTRY
```

You have no excuse to be lazy with respect to proper error handling when it is this easy to add to your code. There are several excellent reasons to move your error handling code into a UDC as I have done here:

1. Improved code readability (minimal distracting error handling code)
2. Improved programmer productivity (less code to type in)
3. Consistency of programming model (the same framework is always used)
4. Centralized maintenance of error handling framework

I have already demonstrated 1-3 above. However, the 4th point deserves special mention. By moving the error handling code to a centralized UDC file, your error handling framework can be maintained in a single place. This makes it easy to change the framework easily, and have those changes exert a global impact on your programs. For example, since you are using the framework

VFP6/7 Tip

Verschil in openmode van files

Wist je dat de VFP 6 editor een file standaard DenyWrite opent en de VFP 7 editor DenyRead? Probeer maar eens om een HTML-file dat (om beurten natuurlijk!) in één van beide editors geopend is, met IE te openen!

in virtually all your code (both client and server side) the framework becomes a prime candidate to introduce debugging aids such as trace or profiling code. Consider, the TRY and ENDTRY UDCs from above modified to including some simple debugging tracing logic:

```
TRY  =>   LOCAL hResult:= S_OK AS LONGINT ;; ;
        LOCAL cb             AS CODEBLOCK ;; ;
        LOCAL uxError        AS USUAL;; ;
        cb := ErrorBlock( {|e| _Break(e) } );; ;
        BEGIN SEQUENCE ;; ;
        #IFDEF __DEBUG__ ;; ;
           DebOut32( String2Psz( "Entering: " +__ENTITY__ ));; ;
        #ENDIF



ENDTRY => RECOVER USING uxError;; ;
       IF IsNumeric( uxError );; ;
          HResult := uxError ;; ;
       ELSE ;; ;
          HResult := E_UNEXPECTED ;; ;
       ENDIF ;; ;
       END ;; ;
       ErrorBlock( cb ) ;; ;
       #IFDEF __DEBUG__ ;; ;
           DebOut32( String2Psz( "Leaving: " +__ENTITY__  ) );; ;
       #ENDIF ;; ;
       RETURN hResult
```

By making these changes in the single UDC file, and reapplying the UDC to your program (NOTE: you must delete and then readd the UDC for changes to the file to take affect as VO caches the UDC file in the repository), your software will now dutifully show you a trace in a debug window of every server service called by every client.

Conclusion

With the complexity level of today's software continuing to grow, professional programmers can no longer afford to cut corners when it comes to error handling. This paper has attempted to show you just how easy and non-disruptive it can be to make your software bullet-proof through the use of robust error handling practices. By using and expanding upon the basic error handling framework presented in this paper, you will be building more robust, error free software in less time then ever before. I hope you are able to apply some of the ideas presented herein in your next software project.

Rod da Silva is the principal of Software Perspectives, Toronto, Canada. He has been a featured speaker at Developer's Conferences all over the world for several years. Rod has a Bachelor of Mathematics degree with Honors in Computer Science from the University of Waterloo, and his areas of interest include compilers, client/server DBMS system software and Windows component technologies. He is the architect and lead developer of CULE.NET (Component Unification Language Everywhere), a CA-Visual Objects compatible language targeting the .NET platform. He can be reached at RodDaSilva@SoftwarePerspectives.com .

auteur
Paul Maskens and Andy Kramek

Making Sense of Intellisense

One of the most eagerly awaited features of Visual FoxPro Version 7.0 has been the introduction of intellisense into the development environment. Although other Microsoft tools have long had this technology, it has been noticeably absent from Visual FoxPro. However, what has been introduced is probably far beyond what most of us expected and opens up a host of possibilities. The VFP Intellisense can do much more than just auto-complete text as you type, or prompting you with commands and their options...

Paul: Visual FoxPro 7 is what I'd call a developer's release, and I think the single most noticeable (and possibly most useful) new feature is IntelliSense. While dockable windows are nice, the modeless document view window is really useful and the new database events invaluable, the one thing that struck me immediately in the command window was IntelliSense. Just take a simple example, type in MODI COMM and it expands to MODIFY COMMAND.

The single most noticeable new feature is IntelliSense

Andy: But that looks like just command completion, we could do that with tools like the Cobb Editor Extensions (CEE) back in VFP 3!

Paul: Yes true, but I did choose a simple example, and I left out the really useful part. At each stage in the expansion IntelliSense also documents both the command, and where you are in the command, thus:

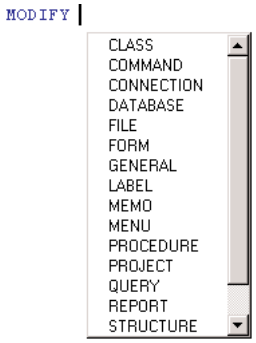


Fig. 1: MODIFY and the drop-down of available completing commands

Fig. 2: Add COMMAND and you see the most recently used list

Andy: That most recently used files list is useful. It applies to other commands as well, of course. Like USE and CD, all very productive while working on a typical large project with many files.

Paul: I found typing SYS() very useful. I can never remember all the SYS commands and, usually before I use one in VFP 6 I type SYS() then highlight it and press F1 to read the help!

Andy: And you know I prefer to use the real MESSAGEBOX() constants?

Paul: Yes, we discussed it before, I don't like magic numbers.

Andy: Well go on, try typing in `MESSAGEBOX("test"`, and see what you get.

Paul: Neat, but I can break it. I want to use combinations and don't want to add them up in my head. Being completely lazy I pressed down arrow to choose 4 and then pressed + and the IntelliSense stopped helping me!

Andy: You didn't read the documentation far enough <g> Press **Ctrl+I** at this point, and the list is re-displayed so you can choose another value.

Paul: I for *I*nfo I suppose, since this feature is called Quick Info (see, I did read the Help!).

Andy: Seems a reasonable guess. The next feature to look at is List Members (Ctrl+J) which displays the members of an object. Just type in **_SCREEN.** in the command window.

Paul: Very nice. A scrolling list of the properties, events and methods with incremental searching as I type.

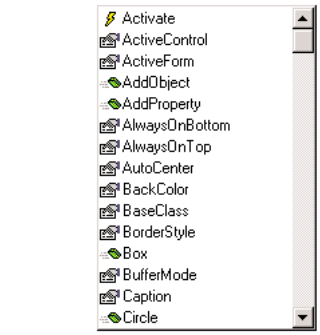


Fig. 3: Members list for _Screen

Andy: There's much, much more, of course. Create an Excel object reference and then start using that object reference, like this:

```
loxl=NEWOBJECT("excel.application")
loxl.
```

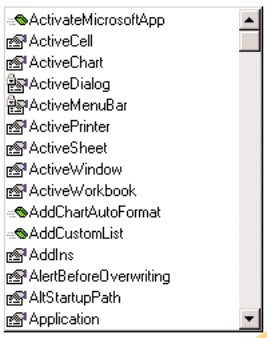


Fig. 4: Members list for Excel

Paul: Very nice. But I tried it in a .PRG and it doesn't work.

```
x.prg *
loxl=NEWOBJECT("excel.application")
loxl.
```

Fig. 5: No members in a .prg?

Andy: Ahah! I bet you forgot to declare the local variable.

Paul: So? I didn't declare it in the command window.

Andy: In code, you need to declare its type so that IntelliSense can look up the members. It's similar in intent to the VB syntax. Add `LOCAL loXL AS Excel.Application` as line 1 of your code.

```
x.prg *
LOCAL loXL AS Excel.Application
loxl=NEWOBJECT("excel.application")
loxl.
```

ActivateMicrosoftApp
ActiveCell
ActiveChart
ActiveDialog
ActiveMenuBar
ActivePrinter
ActiveSheet
ActiveWindow
ActiveWorkbook
AddChartAutoFormat
AddCustomList
AddIns
AlertBeforeOverwriting
AltStartupPath
Application

Fig. 6: Strong typing in action

Paul: I tried that myself, and there's an added bonus. VFP IntellliSense picks up the Excel. reference in the first line and offers what appears to be the top level of the Excel object model. But it looks as if VFP now has strong typing. Is that checked or enforced at runtime?

Andy: No, it's not – there is no change in the behaviour of variables at run time. Even at design time you can write code as ridiculous as this:

```
x.prg *
LOCAL loXL AS Excel.Application
loxl=NEWOBJECT("word.application")
loxl.
```

ActivateMicrosoftApp
ActiveCell
ActiveChart
ActiveDialog

Fig. 7: Programmer Stupidity beats VFP IntelliSense

and IntelliSense still uses the "LOCAL...AS" type declaration and ignores your line of code that assigns a Word object to the variable.

Paul: I suppose that's reasonable. The assignment won't happen until runtime so VFP can't check that I've been stupid while I'm writing the code.

Andy: Now, the ***really*** powerful feature of IntelliSense is that it's driven from a VFP table (named *FoxCode.dbf*), so we can alter its behaviour by modifying the entries in that table and even add completely new behaviours.

The really powerful feature of IntelliSense is that it's driven from a VFP table

Paul: That's much like CEE isn't it?

Andy: I suppose it is, but as well as simply expanding an abbreviation you can also write a script

Paul: Whoa! How do we define an abbreviation to expand, first?

Andy: Well, you could just open the FoxCode table and hack away, but I prefer to use the new "IntelliSense Manager" which is on the Tools pad of the main menu.

Paul: That's a neat little tool, and it can do quite a lot of interesting and cool things besides just adding abbreviations. It will more than repay the effort of studying it, but we don't have the space here to go into it in any great detail, so let's concentrate on adding our own abbreviations and scripts.

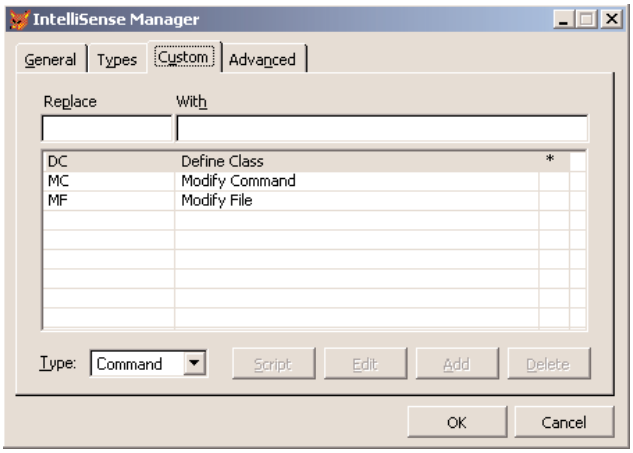


Fig. 8: The Custom tab of the IntelliSense manager

Andy: OK. Abbreviations are pretty straightforward. All you need to do is make sure that the correct type is selected in the combo box at the bottom left, to type your abbreviation in the 'Replace' box, the expansion in the 'With' box and click the 'Add' button. What do you want to add?

Paul: Hmmm. I have a little user-defined function called 'CalcFont' which takes a text string and a number of font-related parameters and pops up a little window giving me the information about how to size a textbox to display that string. I can never remember all of the parameters, or the order in which they come. It would be really nice to have it behave just like a proper VFP function and give me a list of parameters and then move along as I add them. Here is the calling prototype for the function:

```
LPARAMETERS tcString, tcFontName, tnFontSize, tcFontStyle
```

Andy: OK, and here's the step by step guide. We want this to behave as a function, so set the type combobox to 'Function'. Then enter the abbreviation to use (let's use 'CF') and the expanded form (which is, of course, 'CalcFont'). Now click on the 'Add' button.

Paul: Well that's easy enough. But hang on, when I type 'cf' + <space> in the command window nothing happens. It doesn't expand at all!

Andy: Of course not, we've told VFP that it is a Function. Try 'cf' + <(>.

Paul: Ah! That's got it, but I don't see any parameters. How do we get them in?

Andy: We have to add them. Go back to the Intellisense manager. Select your 'cf' item on the Custom tab and click the 'Edit' button. This will open up a browse window on the FoxCode.dbf table. We need to enter our information in the 'TIP' field, exactly as we want it to appear in the tooltip. Let's add:

```
cTestString [ , cFontName [ , nFontSize [ , cFontStyle] ] ]
```

and save the record. Now try typing 'cf(' at the command line....

Paul: Cool! It behaves just like a native function. The current parameter even gets highlighted as I type:

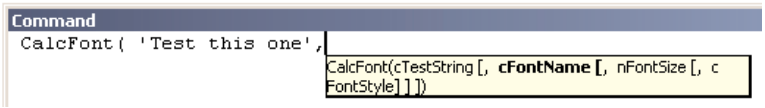


Fig. 9: A user defined function can look just like a native function

Andy: Nice, isn't it. Now what about these scripts?

Paul: Well, there are at least two ways of using scripts. First, you can use them to store FoxPro code in the FoxCode.Data field which is evaluated when the associated abbreviation is entered. Second, you can simply define an identifier in the FoxCode.Cmd field which points to another record which contains the actual code.

Andy: I understand the first one all right, but I am not so sure about the value of the second.

Paul: Think about it. If you write a generic routine you can call it from several different records. For example the FoxCode table has a script item named 'Color' which (simplified) displays the GetColor() dialog and returns the resulting selection. A set of records identifies the various color properties (Type = 'P') and all of these call the same 'color' script by including in their 'cmd' field the ID "{color}". So when you type any of these color related properties you get the GetColor() dialog.

Andy: Ahah! The curly braces indicate that the engine is to evaluate the named script. Very clever! I see that VFP actually defines quite a few already:

```
{ buildmenu}
{ cmdhandler}
{ color}
{ dbgetmenu}
{ funcmenu2}
{ funcmenu}
{ onkeymenu}
{ onoffmenu}
{ picture}
{ setmenu}
{ setsysmenu}
```

Paul: Yes. We'll leave it to the reader to investigate what they all do. So what sort of script do you want to produce?

Andy: What I would like is something that will allow me to put a standard function header into a PRG file when I am defining classes programmatically. This is what I like to use, but it is a real pain to do manually all of the time:

```
*****************************************************
*** FUNCTIONNAME(): Brief Comment here
*****************************************************
FUNCTION <FunctionName>( <Parameters> )
  LOCAL luRetVal

  RETURN luRetVal
ENDFUNC
```

Paul: OK, shouldn't be too hard to manage. We'll start by defining, in the IntelliSense Manager (Custom Tab) the abbreviation we want to use as type 'Other'. Let's use 'FHdr' for now.

Andy: Hang on, I thought this was going to be a script. Shouldn't we be using a type of 'Script'.

Paul: No. We have to use one of the other identifiers if we want to use an abbreviation to run the embedded code. A type of 'Script' indicates that the record contains code but does not permit an abbreviation to trigger the evaluation of the code.

Andy: OK – I think I get it, it's a little confusing at first though. So we create our record in the FoxCode table and then we edit it I suppose. Oh! I see it has been assigned a Type of "U" even though we chose 'Other' in the combo box. Oh well I suppose "U" stands for "User defined", or maybe "Undefined", I wonder why they didn't just use "O"?

Paul: I have no idea. Now pay attention please because there are some things that we must include in the new record. First the "cmd" field must be populated with empty curly braces like this: "{}". This indicates to the Intellisense engine that it must evaluate the text in the 'data' field of the current record.

Andy: I see, if a script name is included inside the curly braces then it will try and find a record of Type 'S' where the `'abbrev'` field contains the specified name. If it finds one, it will run the code from that record and if it doesn't find one, it throws an error. But when we omit the name it just uses the current record as the source.

Paul: Now for the actual code which goes into the "data" field. First we need a parameters statement to receive the reference to the FoxCode object. This object has a number of properties that we need to pay attention to (the full list is in the Help topic) specifically we need to check the Location and set the ValueType.

Andy: The "location" tells us which kind of editing window we are using. This allows us to stop stupid things happening (like expanding a code block in the command window). Here, for reference, are the valid locations:

Location ID	Editor Type
0	Command Window
1	PRG File Editor (called by Modify Command)
8	Menu Procedure Editor
10	Form or Class Designer Methode Code Editor
12	Database Stored Procedure Editor

So for this particular script we are going to use code like this:

```
IF !INLIST( oFoxCode.Location, 1, 12 )
          RETURN oFoxCode.UserTyped
  ENDIF
```

This will ensure that unless we are in the PRG file or Stored Procedure editors (which are the only places we would want to define a function header) the script will simply return whatever was typed in to trigger it.

Paul: And we also need to set the ValueType which is used to determine what happens when the script completes. There are currently only three options for this property as follows:

Value Type	Consequential Action
L	Displays a drop-down list populated from the values stored in the object's Items Array
T	Displays a "Quick Info" tip from the value stored in the object's ValueTip property
V	Displays whatever value is returned by the script

In this case we simply want whatever the script displays to be returned. So we will also need the following:

```
oFoxCode.ValueType = 'V'
```

The rest is standard FoxPro code with one little addition.

Andy: Would this little addition have anything to do with getting the values for the Name, Comment and Parameter list in there?

Paul: How astute of you. Yes, we have a new function in VFP 7.0 called InputBox() which displays a modal dialog and returns a single character string. We can use this in the script to get user input while the script is running like this:

```
LOCAL lcName, lcComment, lcParams
  STORE "" TO lcName, lcComment, lcParams

  lcName = INPUTBOX('Name this Function')
  lcComment = INPUTBOX('Describe this Function')
  lcParams = INPUTBOX(;
    'List the Parameters (if Any) for this Function')
```

Andy: Ah I see, so the whole thing looks like this when we are done:

```
  LPARAMETER oFoxCode

  *** Check we are in the appropriate editors
  IF !INLIST(oFoxCode.Location, 1, 12)
    RETURN oFoxCode.UserTyped
  ENDIF

  *** Set the value type
  oFoxcode.valuetype = "V"

  *** Need some locals here
  LOCAL lcTxt, lcName, lcComment, lcParams
  STORE "" TO lcName, lcComment, lcParams
  #DEFINE CRLF CHR(13)+CHR(10)

  *** Use the new InputBox to get User Entered Values
  lcName = INPUTBOX('Name this Function')
  lcComment = INPUTBOX('Describe this Function')
  lcParams = INPUTBOX( ;
    'List the Parameters (if Any) for this Function')

  *** This is standard FoxPro from here on in:
  lcTxt = lcTxt + "*****************************" + CRLF
  lcTxt = lcTxt + "*** [E] " + UPPER(ALLTRIM(lcName)) +;
    "(): " + lcComment + CRLF
  lcTxt = lcTxt + "*****************************" + CRLF
  lcTxt = lcTxt + "FUNCTION " + ALLTRIM(lcName) +;
    "(" + lcParams + ")" + CRLF
  lcTxt = lcTxt + "LOCAL luRetVal" + CRLF + CRLF
  lcTxt = lcTxt + "       RETURN luRetVal" + CRLF
  lcTxt = lcTxt + "ENDFUNC" + CRLF

  RETURN lcTxt
```

Now when I type FHDR at the beginning of a line, and hit the space bar, I get prompted for the Name, Comment and Parameters for the function and the whole header is constructed beautifully. That's really nice! But we're rapidly running out of space, what else is there to cover?

Paul: Wow! There's loads of things! But, there's one really neat new feature that I simply must mention. We now have the full range of C – like operators: ++, +=, --, -=, *=, /=.

Andy: You mean that you can type in `lnCount++` and it expands to `lnCount = lnCount + 1` ?

Paul: Exactly. Like you, I get lazy with counter variables because I hate to type very long variable names, particu-

larly twice in the same line. But even though that would be good enough in itself, it goes further than that. You can also type `lnCount+=2` and it expands to `lnCount = lnCount + 2`. Or leave it incomplete and lnCount += expands to `lnCount = lnCount +` so you can fill in whatever you like after the + sign.

Andy: Very cool! How is that done?

Paul: Well that's a bit of a mystery, I can't find it in FoxCode.dbf at all. There are no entries in the table for ++, +=, --, -= and so on. I think it has to be coded in the FoxCode.app itself.

Andy: Well, however its being done I guess that we should be glad that we finally have them! You know, there's so much power hidden away in this implementation of IntelliSense that it's going to take us all a long time to discover all of the features and possibilities. The on-line documentation is a good place to start and, we are delighted to say, that in VFP 7.0 it is probably the best Help System we have ever had.

Andy Kramek is an old FoxPro developer, FoxPro MVP, independent contractor and occasional author originally English but now resident in Akron, Ohio. Currently unemployed ...
Email: andykr@compuserve.com

Paul Maskens is a VFP specialist and FoxPro MVP, working as Web Development Manager for Euphony Communications Ltd, he's based in Oxford, England.
Email: pmaskens@compuserve.com

VB/Access

VB/Access: Gebruik van eigen error codes

Applicaties/componenten gebruiken vaak eigen error codes om fouten te identificeren die specifiek zijn voor de applicatie/component. Vaak wordt daarbij een eigen range gedefinieerd, b.v. 1001-1099. Op deze manier wordt echter het risico geïntroduceerd dat een error code gebruikt wordt die overeenkomt met een al bestaande error code van b.v (een component van) Access. Door intern bij je eigen errors altijd de constante vbObjectError te 'verrekenen' kun je naar de gebruiker toe de gewenste reeks gebruiken en toch uniciteit van de error code afdwingen.

Het 'raisen' van een error doe je dan als volgt:

```
Err.Raise 1001+vbObjectError, "SaveOrder", "Ongeldig Artikel"
```

Het afhandelen van de error in een error-Handler doe je als volgt:

```
MsgBox   "Fout " & Err.number-vbObjectError & ": " &_
Err.Desciption
```

auteur
Edwin Holkers

Ontwerpen van webapplicaties: Microsoft.NET en UML

Wat een (beginnend) webapplicatie ontwerper moet weten? *Inmiddels is er veel ervaring en informatie omtrent het ontwikkelen van webapplicaties met behulp van UML beschikbaar in de vorm van boeken en webpages. Wanneer een webapplicatie ontwikkeld wordt met Microsoft.NET zijn op het gebied van het ontwerp nog een aantal punten waarmee rekening gehouden dient te worden. Een aantal hebben met name betrekking op de specifieke mogelijkheden van Microsoft.NET, anderen hebben betrekking op het verschil in taalgebruik binnen UML en het .NET ontwikkelplatform, weer andere bepalen in sterke mate de architectuur met betrekking tot communicatie en het gebruik van classes.*

De in dit artikel beschreven items en onderwerpen zijn gebaseerd op basis van ervaring die voornamelijk berust op de Microsoft.NET bèta 2-release. Achtereenvolgens zullen de volgende punten aan de orde komen: Besluiten voor UML in combinatie met Microsoft.NET, (communicatie)architectuur bepalen, class-gebruik, spraakverwarring communicatie en wensen.

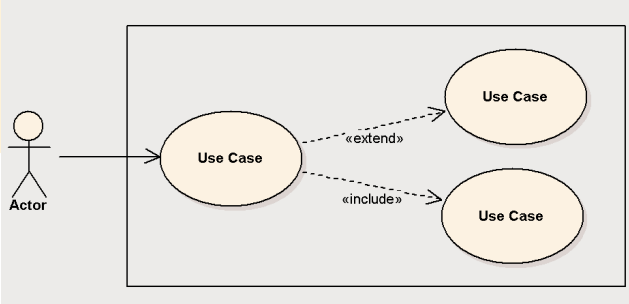
UML en Microsoft.NET

Is het wel handig om UML te gebruiken voor een .NET applicatie? Waarom UML geschikt is wanneer er gedacht wordt of besloten is te gaan ontwikkelen voor het Microsoft.NET platform, ligt meestal voor de hand, maar zal hier niet beschreven worden. In een aantal omgevingen is UML reeds ingeburgerd en is een vertrouwde ontwerptaal een handige basis voor het ontwikkelen met behulp van .NET. In de basis is UML geschikt voor elke mogelijke doeltaal, maar een ideale aansluiting vindt plaats wanneer modeltaal en doeltaal onderling mogelijkheden bieden voor zowel forward als backward-integratie. Ideaal zijn talen welke de eigenschap hebben verschillende softwarearchitecturen te ondersteunen, zoals n-tier, cbd, object oriëntatie, service based development, etc. Vooral op het gebied van deze aspecten speelt de samenhang tussen UML en Microsoft.NET een positieve rol in de softwareontwikkeling. Op zich natuurlijk geen nieuws, maar de samenwerking tussen UML en het .NET platform wordt nog eens versterkt doordat de verschillende UML-tool leveranciers zich inmiddels richten op het Microsoft.NET platform en hier ondersteuning voor leveren Zowel generatie als roundtrip-engineering zijn hierbij belangrijk. Een korte inventarisatie wijst uit dat met name op het gebied van C# de meeste tools al zijn uitgerust met tenminste enige vorm van generatiemogelijkheden.

Bruikbare diagrammen

Welke diagrammen zou je kunnen gebruiken? Bruikbare diagrammen voor het modelleren van web-applicaties zijn:
- Use-case diagram(men),
- Sequence diagrammen (eventueel aangepast),
- Klassediagram(men),
- Componentdiagram(men),
- Deployment diagram(men)

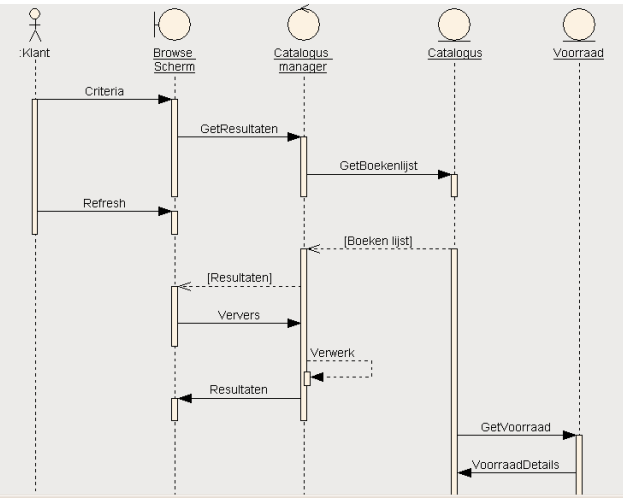
Use case diagrammen



Figuur1: Use-case diagram met use-case en use-case relaties

Use case diagrammen worden gebruikt om te kunnen communiceren met gebruikers over de gevraagde en/of geboden functionaliteit. De bedrijfsprocessen worden onderverdeeld in (systeem)functionaliteit beschreven in use-cases. Tevens kunnen use-case diagrammen als handvat voor latere componentisatie gebruikt worden. Door de eenvoudige diagrammen en de herkenbare elementen kan met behulp van use-case modellen goed gecommuniceerd worden met gebruikers. De onderlinge verdeling dient zodanig gekozen te worden, dat voorkomen wordt dat diverse use-cases elkanders functionaliteit overlappen. Het is altijd van groot belang dat een use-case een afgerond geheel aan functionaliteit beschrijft. Het belang is nog groter wanneer het gaat om een webapplicatie, immers de sessie met de toepassing dient kort en afgerond te zijn. In use-case diagrammen kan door use- en extend relaties al een goede basis voor componentisatie worden gelegd. De use-case modellen voorzien van een complete set van use-case diagram-

men, use-case reports etc. kunnen met name in de complexere systemen toch onvoldoende blijken om –door een gebruiker of opdrachtgever - te kunnen bepalen of de geboden en gemodelleerde functionaliteit inderdaad is wat de gebruiker en/of opdrachtgever voor ogen heeft. Het is daarom niet onverstandig om minimaal een overzicht van schermen te verstrekken naast het use-case model. Eventueel kan het use-case model worden verklaard met behulp van een prototype van de gebruikersinterface. Hierin kan nog een aantal hulpmiddelen gewenst worden. Vaak kunnen impressies van schermen weer een negatief effect hebben in de communicatie met de gebruiker en/of opdrachtgever. Wellicht is het wenselijk om een prototype van de user-interface te maken. Er zijn (nog) te weinig hulpmiddelen beschikbaar voor het prototypen van webforms.

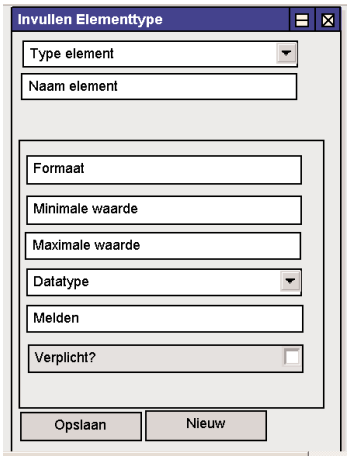


Figuur 2: Voorbeeld sequence diagram met 'refresh'-acties

Het is altijd van groot belang dat een use-case een afgerond geheel aan functionaliteit beschrijft

Sequence diagrammen

Op basis van de use-case diagrammen kunnen verschillende object-sequence-diagrammen gemaakt worden. De sequence diagrammen kunnen worden gebruikt voor het bepalen van de benodigde methoden voor diverse klassen. Tevens kan hieruit worden afgeleid welke klassen van elkaar afhankelijk zijn. Voor sequence diagrammen geldt een bewuste beslissing wanneer een webapplicatie ontworpen gaat worden. De meest gekozen architectuur (thin web cliënt) biedt de mogelijkheid om een zo groot mogelijk publiek te bereiken, maar er dienen wel een aantal regels in ogenschouw genomen te worden. Elke gebruikersactie is een vooruitgaande beweging. Met andere woorden het scherm wordt getekend, interactie met de gebruiker vindt plaats en zodra een ander scherm wordt geopend, kan niet meer worden teruggekeerd naar het vorige scherm, hooguit kan dit gedrag gesimuleerd worden door het vasthouden van de toestand van het vorige scherm. Anders geformuleerd: een gebruiker krijgt een scherm, vult hier gegevens op in en verstuurt de gegevens. De ontvangen gegevens genereren na enkele processtappen uiteindelijk een nieuw scherm. Dit gedrag kan vereenvoudigd gemodelleerd worden door het modelleren van 'refresh'-acties. Dit klinkt uitermate vanzelfsprekend, maar elke ontwerper bekend en ervaren met cliënt-server applicaties zal bevestigen dat dit een andere wijze van denken vereist. Een ander belangrijk punt waar een ontwerper onmiddellijk rekening mee dient te houden is het vasthouden van toestanden ook wel states.



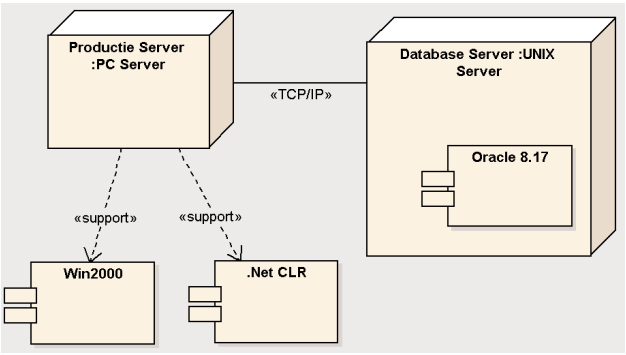
Figuur 3: Een globale schets van schermverloop kan een use-case goed verduidelijken.

In een cliënt-server omgeving geen enkel probleem om de toestand van de cliënt te bepalen, in een web omgeving moet gewerkt worden met sessiestate en de gevolgen hiervan voor de server. Uiteraard moet tijdens het ontwerp al nagedacht worden over de te transporten data. Immers lijncapaciteit is zeer beperkt. Tevens moet bedacht worden dat communicatie voornamelijk gebeurt op basis van vraag en antwoord. Vereenvoudigd betekent dit dat de cliënt vraagt en de server verwerkt het verzoek en reageert met een antwoord, in de vorm van data.

Klassendiagrammen

Op het gebied van klassendiagrammen dient er voldoende inzicht te zijn in de geboden architectuur op het gebied van klassen uit het .NET-framework. Elke aangeboden klasse dient beoordeeld te worden op bruikbaarheid voor het voorogen zijnde doel van de nieuw te ontwikkelen klasse. Met name in de communicatie met het ontwikkelteam is het verstandig om een use-case specifiek klassendiagram te maken. Een zogenaamde View-Of-Participating-Classes geeft tezamen met de overigens beperkte blik van sequence-diagrammen, een beperkt datamodel, etc., een beter beeld van de in de use-case gewenste functionaliteit. Uiteraard kan een volledig klassendiagram ook een bijdrage leveren, maar de overvloed aan klassen kunnen meer verwarring dan begrip opleveren. Een begrip van de context van de te ontwikkelen use-case kunnen onvolkomenheden in de modellen en detaillering opvangen.

Deployment diagram



Figuur 4: Deployment diagram met verdeling van componenten

In een deployment diagram kan de volledige applicatie architectuur worden weergegeven in samenhang met de (toekomstige) infrastructuur. In het deployment diagram dient duidelijk te worden weergegeven welke componenten op welke fysieke of logische node zal worden aangeboden. In het deployment diagram wordt nogmaals de communicatie tussen de verschillende componenten weergegeven. Indien de deployment plaats vindt over verschillende fysieke nodes zal tevens in combinatie met de sequence-diagrammen en klassen-diagrammen een beeld gevormd kunnen worden van de belasting ten aanzien van communicatie voor de infrastructuur.

(Communicatie)architectuur

Hoe gaan de verschillende objecten informatie uitwisselen? Onderscheid wordt gemaakt tussen de inter-layer communicatie en de inter-object communicatie. Wanneer applicatielagen (fysiek) worden gescheiden, dan dient een communicatie mechanisme te worden afgesproken. Op dit moment spelen SOAP en XML hier een belangrijke rol. De complexiteit van SOAP en XML en de invloed op de communicatie en performance zijn niet wenselijk wanneer geen fysieke grenzen worden overschreden. Het 'native'-communiceren tussen objecten is dan ook zeker aan te bevelen. Het is ook aan te bevelen om wel XML en SOAP communicatie over lagen mogelijk te maken, door (data)klassen serializeable te maken. Tevens dienen keuzes gemaakt omtrent de architectuur van de applicatie zelf; zo dient gekozen te worden voor wel of geen cliëntscripting, parameteroverdracht met cookies of andere technieken, etc. Deze beslissingen hebben alles te maken met herbruikbaarheid, performance en toekomstplannen.

Veiligheidsrisico's

Welke veiligheidsrisico's moet ik in het ontwerp mee rekening houden? In het kader van communicatie dient tevens gekeken te worden naar beveiliging. Hierbij moet nagedacht worden over het risico en benodigde investering van beveiliging. Daarnaast moet bij het bepalen van het risico meegenomen welke informatie nuttig is voor onbevoegden en daarmee hoe gevoelig deze informatie is voor de verantwoordelijke organisatie. Beveiligingsrisico's bestaan er voor elke gebruikte technologie:

Cookies	Cookies vormen een risico wanneer een site de informatie kan lezen uit de cookie van een andere website. Verder kan een cookie bekeken worden door een ander dan de (geautoriseerd) gebruiker. Indien de cookie persoonlijke informatie bevat kan wellicht de informatie gebruikt worden om zich voor te doen als een andere persoon. Belangrijke aspecten, die hierin meegenomen moeten worden zijn bijvoorbeeld de expiration date van cookies en welke informatie in een cookie mag worden opgenomen. Zo is het niet verstandig hierin een username en wachtwoord combinatie vast te houden.
JavaScript	JavaScript kan de integriteit van de persoonlijke informatie bedreigen. Tevens kan informatie van of over de gebruiker worden verzameld. Vanwege dit mogelijke gedrag kan een potentiële gebruikersomgeving de optie tot gebruik van JavaScript hebben uitgeschakeld, wat gevolgen kan hebben voor de ontwerp beslissingen.
Java applets	Files kunnen worden gelezen en worden geschreven op de cliënt. Er kunnen netwerk verbindingen worden aangelegd naar andere machines. Er kunnen andere programma's worden gestart of systeemaanroepen worden gedaan. Dit alles kan alleen als een applet 'signed' is. Wel zijn er nog verschillende bugs die mogelijke schade aan de cliënt kan aanrichten. Ook vanwege deze mogelijkheden kan het zijn dat de functionaliteit in een gebruikersomgeving is uitgeschakeld.
ActiveX controls	Gelijk aan die van java applets. Na signing tijdens laden, volledige toegang tot de cliënt.
Plug-ins en Mime-type viewers	Plug-ins en viewers zijn geïmplementeerd als native applicaties, met gelijke rechten. Nadat een plugin of viewer is gestart, is er weinig dat nog tegengehouden kan worden.
Firewalls	Strikt ingerichte firewalls in gebruikersomgevingen zullen nauw moeten worden bestudeerd en de gevolgen voor het ontwerp dienen in kaart gebracht te worden. Zo is het niet overal mogelijk om iedere beveiligingsoptie te gebruiken. Verrassingen in de zin van niet werkende applicaties als gevolg van inrichting in een gebruikersomgeving kunnen niet altijd voorspeld worden.

Autorisatie

Hoe scherm ik bepaalde functionaliteit af van ongeautoriseerde gebruikers? Tevens dient in de architectuur rekening gehouden te worden met het gedrag van een internet applicatie met betrekking tot navigatie. Het volstaat niet om de toegangsbeveiliging van een bepaalde functie af te schermen tijdens het navigeren. Een pagina is ten slotte ook middels de URL te bereiken, met andere woorden elke functie dient zelf verantwoordelijk te zijn voor haar autorisatie en beveiliging.

Cliëntarchitectuur

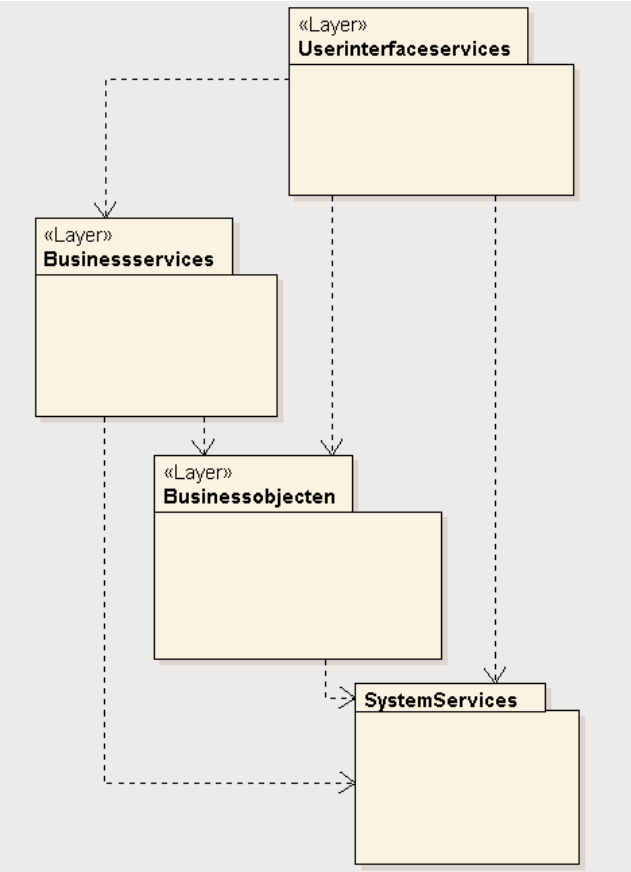
Welke basisarchitectuur is bruikbaar voor het systeem? Een beslissing dient genomen te worden omtrent de architectuur met betrekking tot de cliënt:

- Thin-web cliënt, veelal gebruikt voor internet-based applicaties, waar in de cliënt een zo klein mogelijke controle behoeft te hebben over de configuratie. Alle business logica wordt uitgevoerd op de server. Een cliënt behoeft alleen te beschikken over een web-browser. De browser is de cliënt.
- Thick web client, een significante hoeveelheid aan business logica wordt uitgevoerd op de cliënt. Er wordt gebruikt gemaakt van dynamic HTML, Java applets of ActiveX.
- Web Delivery. Het belangrijkste technologische middel voor web-delivery applicaties zijn webservices. Veel van de intelligentie wordt als 'normale' applicatie uitgevoerd op de cliënt. Kort door de bocht is dit een n-tier architectuur waarbij de webserver als middleware wordt gebruikt om via HTTP te kunnen connecten aan business-services.

Verticale opsplitsing

Tevens dient een beslissing op het gebied van architectuur genomen te worden met betrekking tot de verticale opdeling van de toepassing. Met verticale opdeling wordt bedoeld: bouwen we één applicatie of delen we de applicatie op naar gebruik of functioneel gebied. Bij opsplitsen op basis van gebruik kan bijvoorbeeld leiden tot een applicatie voor beheer, één voor raadplegen en één voor sturing. Bij opsplitsen op basis van functioneel gebied kan gedacht worden aan splitsing naar onderwerp, zoals bijvoorbeeld een applicatie voor het tonen van een catalogus en een voor het plaatsen van een bestelling en een voor een vraag en antwoord (news-achtige) faciliteit. Een verticale opsplitsing kan nuttig zijn voor het nauw scheiden van gebruik. Zo is het bijvoorbeeld mogelijk om een beheerapplicatie apart en strakker te beveiligen - bijvoorbeeld middels certificaattechnologie - dan de raadpleegapplicatie. Zelfs kan er dan voor worden gekozen om de beheerapplicatie als winform-applicatie te ontwikkelen en de raadpleegapplicatie als webform-applicatie. Het opsplitsen op basis van functioneel gebied kan een gefaseerde oplevering mogelijk maken, maak maakt communicatie tussen de functionele gebieden wel complexer.

Horizontale opsplitsing



Figuur 5: Een implementatie voorbeeld van het layered patroon

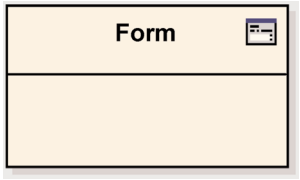
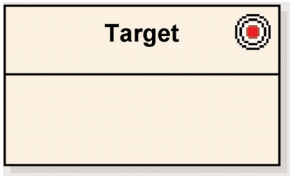
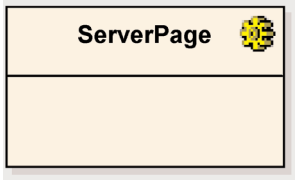
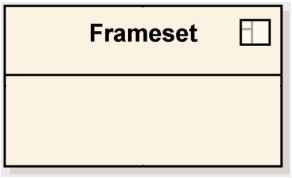
opsplitsen op basis van functioneel gebied kan een gefaseerde oplevering mogelijk maken

Een horizontale opsplitsing gebeurt vaak op basis van een architectuur patroon, zoals het layered model. Het patroon geeft een algemeen beeld van de opdeling die volgens het patroon een goede oplossing levert voor een n-tier applicatie. De detail-invulling van het patroon maakt een aantal mogelijkheden beschikbaar, maar levert ook potentiele problemen. De belangrijkste gevolgen, indien de lagen ook fysiek – eventueel over componenten – worden gescheiden, hebben betrekking op communicatie. Een vereenvoudigde architectuur, het samennemen van lagen in een – maakt de communicatie over de samengenomen lagen eenvoudiger – native – maar beperkt ook de flexibiliteit.

Class-gebruik

Welke classes zijn goed bruikbaar voor bepaalde oplossingspatronen? Op basis van kennis en ervaring is het noodzakelijk een beslissing te nemen welke klasse het

meest geschikt is om een bepaalde dienst te vervullen. Niet alle klassen zijn altijd wenselijk. Voor het modelleren van web-applicaties zijn verschillende extensies in de vorm van (iconized) stereotypes beschikbaar (let op de iconen rechtsboven in de classes), zoals:

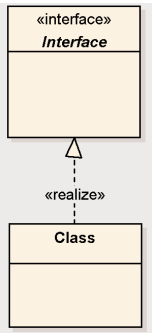
Clientpage	
Form	
Target	
Serverpage	
Frameset	

Er zijn nog verschillende andere (iconized) stereotypes beschikbaar; afhankelijk van de gebruikte case-tool zijn deze ook bruikbaar. Een goede architect neemt vooraf beslissingen omtrent de beschikbare bouwstenen. Ter vergelijking kan gedacht worden aan een architect van een gebouw. Behalve de bouwtekeningen worden belastingen, bestek en dergelijke bepaald. Een goede software architect bepaalt behalve de gewenste functionaliteit ook met welke base classes deze worden geïmplementeerd en onder welke voorwaarden deze gebruikt kunnen worden.

Spraakverwarring

Hoe verspreid ik optimaal de model-informatie? Zoals in iedere omgeving waar gebruik wordt gemaakt van meerdere hulpmiddelen, die ieder hun eigen taal hanteren, zijn er ook verschillen in de betekenis van verschillende termen. De UML-termen worden al meer dan eens op verschillende wijzen uitgelegd. Zo zou het kunnen dat de interpretatie en het gebruik zoals hier beschreven anders door een ander schrijven zouden worden verwoord. Zodra dan ook de .NET terminologie gebruikt wordt zonder verder aan eventueel gebruikte termen aandacht te hebben besteed, is de bron voor verwarring aangebracht. Hieronder een toelichting rondom interfaces, inherits en realizes.

Interface



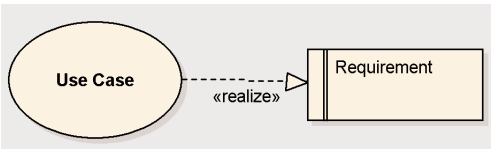
Figuur 6: Presentaties van een interface

Een interface is een standaardisatie methode om uniformiteit in het aanbieden van services te bereiken. Een interface in UML is een zichtbare methode voor de buitenwereld.

Een interface volgens Microsoft: ***'A reference type that defines a contract. Other types implement an interface to guarantee that they support certain operations. The interface specifies the members that must be supplied by classes or other interfaces that implement it. Like classes, interfaces can contain methods, properties, indexers, and events as members'*** Een interface volgens OMG: ***'A class implements one or more interfaces. An interface is a collection of operations that specify a service of a class or component. An interface therefore describes the externally visible behavior of that element.'*** Het verschil zit vooral in de termen contract en beschrijven van properties, indexers en events. Met andere woorden een interface in het framework is een mechanisme voor polymorfisme – een contract waaraan een klasse zich onderwerpt. Een interface in UML is een beschrijving van de signature van een methode van een klasse die beschikbaar is voor de buitenwereld.

Een interface in UML is een zichtbare methode voor de buitenwereld

Implements en realizes



Figuur 7: Een use-case realiseert een systeemeis

Een implements relatie wordt binnen .NET aangeduid voor een class welke een bepaalde interface naar buiten zal aanbieden. Wanneer de term realizes wordt gebruikt binnen UML kunnen verschillende zaken worden getoond. Op het gebied van interfaces wordt hiermee aangeduid dat een component bepaalde methoden zichtbaar voor de buitenwereld maakt. Een realize kan ook gebruikt worden voor de klassen, die de functionali-

teit van een use-case realiseren. Wanneer een business-proces is gemodelleerd, kan worden aangegeven dat een use-case een bepaald business-proces realiseert. Ook requirements kunnen worden gemodelleerd en hierbij kan worden aangegeven welke use-case deze requirements implementeert.

Inherit
Kunnen schermen nu wel of niet logisch overerven? In een model kan weergegeven worden dat schermen gelijke functionaliteit hebben middels overerving, maar of dit daadwerkelijk zodanig wordt geïmplementeerd is nog de vraag. Wellicht is het niet verstandig om een dergelijke constructie te implementeren of het nu wel of niet haalbaar is. Er ontstaat een afhankelijkheid, die in web-applicaties niet wenselijk kan zijn. Met name in een service-based applicatie dient voorzichtig omgesprongen te worden met overerving.

Wensen

Wat kan je nog wensen in deze moderne technologie? Het Microsoft.NET platform is gebaseerd op een nieuwe en verfrissende filosofie verpakt in een hulpmiddel. Toch valt er nog veel te wensen in de architectuur van Microsoft.NET en de relatie met UML. Deze wensen zijn strikt persoonlijk en hebben mogelijkerwijs geen enkele relatie met de strategie van Microsoft of OMG.

Waarom beïnvloedt infrastructuur de communicatie niet vanzelf?
Afhankelijk van de infrastructurele architectuur, dient besloten te kunnen worden voor een ander communicatie mechanisme. Wellicht kan de wijziging in architectuur gedetecteerd worden door de CLR en zodanige gevolgen hebben voor de wijze waarop inter-object communicatie plaatsvindt, dat de ontwikkelaar geen effort behoeft te steken in de verschillende aspecten van XML, SOAP en aanverwante zaken.

.NET biedt flexibiliteit door sessie-state welke 'at-runtime' configureerbaar is

Waarom dienen er beslissingen genomen te worden omtrent parameter overdracht? Een belangrijke bijdrage van .NET is de grote flexibiliteit in sessie-state welke 'at-runtime' configureerbaar is geworden. Kiezen we voor in-memory, op schijf in een database of een speciale server voor sessie management. Hier hoeven inmiddels niet al te veel overwegingen in genomen te worden. Ten aanzien van parameter overdracht kan de wereld van 'traditionele' cliënt-server applicaties als eenvoudig worden beschouwd. Met name cliënt-server applicaties die in de basis misschien de n-tier-gebaseerde oplossingen wilden implementeren, maar vaak toch eigenlijk ten hoogste een 2-lagen architectuur volgden. Beslissingen in een cliënt-server omgeving omtrent communicatie, die daadwerkelijk n-tier werden opgezet, waren – en zijn - ook niet als eenvoudig te bestempelen. Met deze blik op de codering is het een ander wel verbeterd maar nog niet optimaal.

Kunnen we de architectuur voorgeschreven door de ontwerper niet afdwingen? Wanneer .NET dwingend patronen zou ondersteunen, zou het afwijken van een voorgeschreven architectuur al een hoop discussie overbodig maken. Misschien kunnen de verschillende project-templates worden uitgebreid met een template voor webform-ntier applicaties, webform cbd-applicaties, webform services based applicaties. De projecttemplates binnen Visual Studio.NET zijn te configureren, maar niet afdwingbaar. Een basis architectuur kan zodanig in een projecttemplate worden aangebracht dat het volgen van een architectuur voor de hand licht. Hierin ligt nog een grote markt voor het ontwikkelen van standaarden. Deze mening is die van een architect, die te vaak om schijnbaar technische redenen accepteert dat wordt afgeweken van een voorgestelde ideale architectuur.

Hoe modelleer ik een webservice? Indien er wordt voorgesteld om services te kunnen aanbieden op het internet in de vorm van webservices, zou een goede modellering nuttig zijn voor het vastleggen van de interface en de pre- en postcondities van de service. Nu wordt dit gedaan middels een tekstueel contract. Het aanbieden van een webservice kan het best beschouwd worden als het aanbieden van een andere user-interface. Dit voorkomt dat er discussies ontstaan over het aanbieden van de businesslaag middels het internet. De webservice is, vergelijkbaar met de user-interface, een communicatie mechanisme met een externe actor en daarom een boundary klasse, wat vergelijkbaar is met een user-interface.

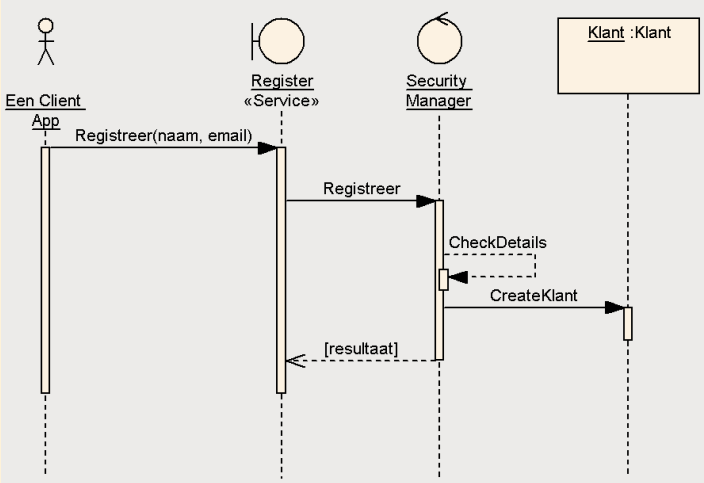
Waarom submit en receive? Hopelijk op termijn kunnen meer geavanceerde structuren worden gesimuleerd, die de architect niet langer het gevoel geven een submit en receive actie te moeten implementeren. Werkelijke inter-

Delphi en .NET

Meer nieuws over de ondersteuning van .NET in Delphi kun je lezen op:

http://www.borland.com/about/press/2002/borland_supports_dotnet.html

actie met de gebruiker, door direct respons te geven op acties van de gebruiker zonder complexe alternatieven in bijvoorbeeld JavaScript. De communicatie is wel inherent aan de Internet-architectuur en gedistribueerde omgevingen. Een belangrijk punt is dat de complexere functionaliteit aangeboden op de cliënt in een browser ook gemodelleerd moet worden, omdat hierin anders voor met name onervaren ontwikkelaars te ondoorzichtige en moeilijk te implementeren functionaliteit moet worden bedacht.



Figuur 8: Een webservice als boundary maakt modelleren en begrip eenvoudiger

Waarom is Visual Studio.NET (nog) niet voldoende? Wat zou de wereld mooi worden, wanneer een goed UML gebaseerde tool volledig geïntegreerd kan worden met Visual Studio.NET. Het altijd actueel zijn van model en software. Roundtrip engineering binnen het Visual Studio.NET raamwerk. Geen bezwaren voor het voortdurend bijhouden van model en software, immers het geheel synchroniseert nagenoeg automatisch. Natuurlijk moet een dergelijk hulpmiddel in staat zijn om bepaalde technische details in het model te verbergen. Welke mogelijkheden de Visual Studio.NET architect-editie biedt is mij onbekend (misschien een eigen artikel waard). Naast integratie met een UML-casetool is integratie met een goede testtool, change-management-tool en centrale respository wenselijk. Met dit in mijn ogen ideaal beeld stuur ik dit verhaal de wereld in.

Succes bij het realiseren van een ontwerp!

Edwin Holkers is ICT-consultant bij Omnext.NET bv en houdt zich daar onder andere bezig met het functioneel en technisch analyseren en ontwerpen van web-gebaseerde applicaties.

auteur
Aram Janigian

The .NET smoke has cleared ... we can continue programming with Delphi 6.0,

(7.0, 8.0 coming soon)

The following is the opinion of Aram Janigian about the status of the Delphi, Java and .NET marketplace.

Established in 1999, Personal Delphi Agents (PDA) is the only company in the world that specializes in placing Delphi professionals in permanent positions throughout the U.S. Aram has personally spoken with thousands of Delphi programmers all over the U.S. and around the world as well as hundreds of Delphi clients. I have also been the featured speaker at several different Delphi Users Groups all over the country.

PDA has heard the following questions or comments for the past several months from various Delphi programmers around the U.S. including:

.NET is going to be the end of Delphi.
.NET is more powerful than Delphi.
C# is going to be the Java killer.
VB.NET is going to be the Delphi killer.
What should be my primary Web programming language?
Where do you see the future of Delphi?

Delphi has been around for over 17 years if you include its predecessors Turbo Pascal and Object Pascal. Officially the beta version for Delphi came out in 1994 and the first release was in 1995. During that time, Borland has created a very extensive client/server tool. Once engineers begin to use Borland's products they usually switch to using them and are loyal for life.

A good analogy is: Microsoft is like Kmart (although Microsoft is not under Chapter 11), they provide cheap products and you can go down the street and buy them just about anywhere. They are going after the PC masses. Borland is like Sears. Our fathers and grandfathers used their quality, Craftsman tools and become loyal Users for generations. You build a good product and people will use it. Sun Microsystems took a bunch of guys and put them behind closed doors for a period of time and they needed coffee to stay awake so let's call it Java (very creative!). Sun Microsystems had lots of money and marketed Java very well although it is much slower than Delphi. However, Borland has recently been successful at selling its new JBuilder too...Delphi is their #1 selling product still and JBuilder is #2.

A Delphi Web-based application is much faster than a Java Web-based application because (1) development time is about 50% faster than Java and (2) Java is an interpreted language and has to interpret each hit to the web site. So if you are getting a million hits per day or per month, Java will slow down everything. Java will never "die" (at least not at the hands of C#). Java has too much of a head start on .NET and too many established developers. And the same goes for Delphi.

Microsoft was very good at creating lots of hype for .NET. About 250 books have been written about .NET, Microsoft has lots of money, Microsoft has lots of lawyers, Microsoft has ownership in many different companies and they are entering a competitive Web development tool marketplace. The question is, "Will they have the resources to support .NET?" The smoke has now settled and most of the Visual Basic Developers we have spoken with say VB.NET is not all it is said to be.

C# is really not the Java killer. All of these Web development languages are just competing against each other now. Microsoft and Sun Microsoft will eventually give up in this client/server, Web development tool marketplace in my opinion. Microsoft is trying to do too much at one time and will spread too thin in this marketplace.

Microsoft is known to enter a marketplace and drain it if they are not successful in making money within that marketplace. Then, they just walk away. It's unusual how Microsoft paid $100 million in patent infringements to Borland in 1999 and then turned around and bought a $25 million (10%) equity investment in Inprise/Borland the same year! Then, Microsoft hires Anders Hejlsberg with a $2-3 million signing bonus and any house in

Redmond, WA to create their VB.NET architecture. The top Delphi guru from Borland is now the creator of VB.NET. My point is that the VB.NET architecture will have the Borland Delphi "flavor" built in. This makes you think, doesn't it?

Borland will keep on chugging along like a powerful locomotive and be able to provide Delphi, JBuilder or C++ Builder and their other products to any of their clients worldwide depending on what they are looking for. A majority of the small-to-medium size companies use Delphi because it is priced competitive and there are lots of available programming resources. Most of the Delphi Users are from the Technology and Finance sectors where they have to quickly develop their applications since time is money. Additionally, there are programmers that have application experience across all verticals.

Delphi professionals have nothing to fear Delphi is going to be around for many years to come

As of February 2002, our clients are hiring Delphi professionals at record numbers. Our clients are posting lots of Delphi jobs with Personal Delphi Agents. Delphi is very much alive and well in 2002 and beyond. Delphi will be the tool of choice for client/server development for many years to come. Consulting is down but the end client is getting smarter and making their money last a lot longer by hiring permanent programming resources.

We have seen a lot of C++, Visual Basic, and Java programmers switch to Delphi because of the power, ease, and the ability to quickly generate Web applications for your business. There are approximately 1,000,000 Delphi Users worldwide (250,000 are in the U.S.), there are about 2,000,000 Java Developers worldwide and about 4-8,000,000 Visual Basic Developers worldwide depending on who you ask. Microsoft will have to convert the top 20% of those VB Developers to use VB.NET (since they will be able to understand true RAD OOD) the rest of the VB folks need to think about getting a job at McDonald's. C++ is the most widely recognized language worldwide. It is on more platforms than any other language. Borland must have at least 500,000 C++ Builder Developers today and that list is growing as well.

My point is that all of these languages compete in today's marketplace, especially here in the U.S. Borland is setup to really perform well during the next 4-5 years if they continue improving Delphi and penetrate the International markets as well. Most of the International marketplace is still based on Object Pascal technology. This is a great time for Borland to move forward and penetrate these markets. Borland's President and CEO, Dale Fuller, is doing a fantastic job and is keeping his promise to market Borland's core products worldwide. Borland has recently added Charlie Hixon, Regional Vice President of the Americas. Charlie has 14 years of sales and management experience with IBM and will lead the charge for Borland's sales teams.

Some final thoughts and things to remember include: Borland has obviously released Delphi 6.0. **Well, they are already working on Delphi v7.0 and 8.0 at this very minut. These new versions will continue building on their Web Services architecture. Now, Delphi not only supports major Web Services technologies but .NET and BizTalk as well. So, even if .NET gains momentum (which is doubtful), Delphi is never out of the picture and you can bet that Borland will do everything necessary to ensure that.** Borland could very easily be a $1 billion software company in less than 5 years. My guess is that they will achieve this goal and they have the track record to prove it. Also, you have a very driven team of intelligent people at Borland like David Intersimone, John Kaster and Karen Giles leading and supporting the success of Delphi worldwide. Delphi professionals have nothing to fear. Delphi is going to be around for many years to come.

Aram Janigian is the Founder and Managing Director of Personal Delphi Agents.
He brings 10 years of experience selling software, consulting and staffing services throughout the United States.

Aram is mainly responsible for growing Personal Delphi Agents staffing and certification practices. Other responsibilities include management, sales, recruiting, and expanding companies network within the Delphi community. In 2001, he was a speaker at various Delphi Users Groups including: Austin, Indianapolis, Raleigh, Toronto and Washington, DC.

He has sold software and services to companies ranging from $5 million to Fortune 500 companies including: AIG, AT&T, Cigna Property & Casualty, Colonial Management, Federated Department Stores, Interim Technology, Live Technology International, Marriott International, NCR Corporation, Penn Treaty America Corporation, STG, Inc., Triple Point Technologies, Union Acceptance Corporation, Yasuda Fire & Casualty, and numerous other companies and Government agencies.

He can be reached at www.personaldelphiagents.com or aram@personaldelphiagents.com

auteur
Ginny Caughey

Windows Multi-threading for Application Developers (part 1)

Even casual users of 32-bit Windows know that their computers seem to be able to do more than one thing at a time. Your customers may not know anything about the underlying technology that makes this possible, but they do know that doing more than one thing at a time mirrors the way they themselves work. What's more, they may expect the software you develop for them to do the same. This paper provides you with the basics you need to fulfill this customer requirement. The example code uses CA-Visual Objects 2.5, which is my preferred language, but the concepts apply to multithreading in Windows in general.

Multitasking vs. Multithreading

An application that you write executes under Windows as a *process*. This is what used to be called a *task* in 16-bit Windows, and *multitasking* is the ability of Windows to run (or appear to run) more than one application at a time. Every process points to a private 4GB virtual address space allocated to it (2 GB reserved for the system's use and 2 GB for your program), and the code and data are loaded into this space by Windows. Finally, execution of the code begins at the standard entry point.

Each process, in turn, can run multiple threads. A *thread* is like a process within a process with one important exception: threads do not have a private memory space allocated by the operating system. All threads run in the memory space already allocated to the process that owns them. Every process has one thread when it begins execution. This is the *primary thread*, and it begins automatically when the application starts execution. So even if you have not used multithreading in your applications, you have still been using threads, one per application. Additional *secondary threads*, if used, are created under your programmatic control.

When an application only has one thread, life is pretty simple. In some ways it's similar to writing an application where all the files are opened for exclusive use. You don't have to worry that someone else might try to update the record you are working on. Multithreading, however, introduces many of the issues you are familiar with for shared file access plus some brand new ones. For example, when you use multithreading in your applications, Windows doesn't give you any "lock required" message to warn you if one thread updates a global variable while another thread is also updating it. Therefore, it is your responsibility to recognize which resources in your application are global (and not all of them are global memory variables!), and provide a mechanism similar to record locking to ensure that each thread has exclusive access while these resources are updated. When the primary thread ends, any secondary threads your application has spawned that are still active will end too. So you may need to think about ways to keep your main application thread alive long enough for any secondary threads to complete their work or clean up after themselves. Thread synchronization techniques are discusses later in this paper.

Each process, in turn, can run multiple threads

Performance is an issue with both multitasking and multithreading. Since most of us are still using computers that only have a single processor, the simultaneous execution that we appear to see with multitasking and multithreading is an illusion. When a thread or process's time slice expires, all the information about the current state of that thread or process is saved so the thread or process can start right up where it left off the next time it gets a time slice. And before the next thread or process can resume operation, that thread's state must be restored. This operation is called a context switch and the data that is saved is called the *context record*. Saving and restoring all this information takes time, perhaps thousands of CPU cycles in some cases. You can control the priority of threads so Windows will allocate more CPU resources to one thread than to others, and this is also discussed later in this paper, but no matter what you do, there is some total performance overhead associated with multithreading. (This changes with computers with multiple processors, however. Windows is designed to allocate different threads to different processors in a multi-processor computer, and you wouldn't even have to change your multithreaded code to take advantage of multiple processors if they are installed.)

In addition to the performance issue for context switches, another issue concerns exactly *where* in the thread the context switch occurs. Perhaps you have some code like this:

```
SomeMemvar := SomeValue
```

Unless you have also been writing assembly language code recently, it might seem at first glance that any context switch could only occur either before or after this line of code executes. But one line of CA-Visual Objects source code can represent many machine level instructions, and a context switch could occur anywhere in the middle of that code. This isn't a problem with multitasking, because the different tasks aren't sharing the same memory space. When the context is restored, the application continues with no problem because the data in the context record is still valid. But with multithreading, it is possible for the values saved in the context record for one thread to become invalid when another thread in the same process receives a time slice if both threads are accessing a shared resource. If this happens in an application you are developing, it may only occur sporadically, so debugging can be difficult. The best approach is to be aware of these potential problems and try to prevent them as you design the application.

When to Consider Multitasking

It should be clear from the preceding discussion that multithreading introduces some hassles you don't have to deal with when you solve a business problem using several separate applications. Perhaps you have already used WinExec() to spawn off another application to accomplish some job while your main application keeps on running and responding to end user keystrokes and mouse clicks. Consider these situations addressed by spawning separate applications:

- Running a Windows utility.
- Running an application that you didn't write and can't change.
- Using separate applications to customize the code for different customers. (For example, different versions of PrintStatements.exe for different customers.)
- Using separate applications to take advantage of alternative GUI classes. (For example, using the ClassMate GUI Classes in some applications of a system and the CA-Visual Objects GUI Classes in others.)
- Running a task manager system that takes tasks queued for background execution from a file-based queue by one or several applications and runs them separately. (George Smith did a case study presentation at Technicon several years ago using CA-Visual Objects applications at Pratt & Whitney to run night updates across all the idle CPUs on a network, replacing a mainframe based system for a fraction of the cost.)
- Using OLE automation.

Spawning separate applications is generally easy to develop and debug, and with a Spawn class you can even have your main application wait until the spawned application has finished. The main disadvantage of this approach is that the separate applications can't easily share a lot of data, but passing data to the spawned program as parameters or in a temp file created for this purpose works well in many situations. With OLE automation, data could be passed to the spawned process in the OLE server's properties.

When to Consider Multithreading

The primary disadvantage of using separate processes becomes the main advantage of multithreading. Because threads do share the same memory pool, it's easier to share information between threads than it is between applications. Threads can also control other threads in the same application using a variety of techniques. Here are some ideas for things that could work well as separate threads:

- Polling a group of serial ports for input. (You can download a Serial class written in CA-Visual Objects from www.knowvo.com that uses multithreading to watch serial ports.)
- Sending off faxes or email. (In other words, a fax server or email server.)
- Writing a server of any sort. (See the WWW Server sample application that ships with CA-Visual Objects 2.5 for an example of a multithreaded web server written in CA-Visual Objects.)

The main thing to notice about jobs that are designed to run in separate threads is that, like multitasking, the jobs execute asynchronously. If ***the rest of the application must wait for the thread to complete before it can continue its work, then there is usually no advantage in having separate threads.*** (The exception that proves the rule: sometimes using threads can simplify the programming logic in some situations making the overhead of threading a good tradeoff compared with programming complexity, but don't count on this simplification effect when you're just starting out with multiple threads!) And because of the overhead usually introduced with multithreading in terms of development complexity, debugging difficulty and runtime performance on single processor computers, the development time as well as the total time to execute some job could all be longer with multiple threads than without. ►

Lees verder op pagina 41...

auteur

AD VAN DE LISDONK EN JOOP PECHT

Borcon 2002

***Anaheim - Californië** was voor de 13e BorCon omgedoopt tot Borland. Naast een enorme hoeveelheid sessies kregen de deelnemers, in tegenstelling tot vorig jaar, dit keer erg veel nieuws voorgeschoteld. Veel blije gezichten dus.*

Zoals we inmiddels van Borland gewend zijn werd voor de openings-keynote een filmthema gebruikt. Dit jaar was dat James Bond. In een leuke montage zagen we 00-DavidI in actie tegen een aantal schurken. Het was tevens de eerste keer dat we hem in een pak zagen in plaats van een T-Shirt en een korte broek. Iets later kwam Borland's President en CEO Dale Fuller laten zien wat hij verstaat onder "Wireless Deployment and Delivery". Met een groot persluchtkanon schoot hij T-Shirts het publiek in. Dolle boel dus.

DavidI als 007

The future of Delphi

Waarvoor wij met name in Borland zijn is natuurlijk Delphi. Er was de nodige onduidelijkheid over de visie van Borland op het uitbrengen van Delphi voor het .NET-platform. Nou, die visie is zo duidelijk als het maar zijn kan: **'The Future of Windows is .NET, so the future of Delphi is .NET'.** Het doel waar Borland naar streeft is 'To make Delphi a premier .NET application development tool'. Of om het nog preciezer te zeggen: 'To evolve the Delphi language, compiler, and IDE productivity tools to fully embrace the .NET feature set while maintaining a high degree of compatibility with existing Delphi applications'. Wat ons betreft een hele opluchting.

Anders Hejlsberg

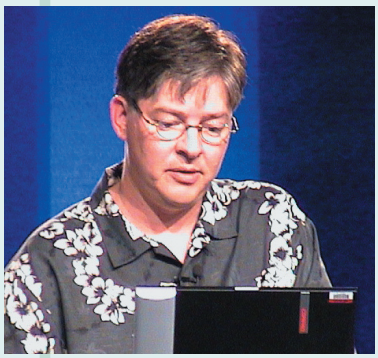
Aan het applaus van het publiek viel af te leiden dat de developers Anders Hejlsberg hadden vergeven dat hij van Borland naar Microsoft was overgestapt. De man die ooit aan de wieg heeft gestaan van Turbo Pascal liet niet alleen wat slides voorbijkomen over het .NET framework maar was ook heus aan het compileren met Delphi onder .NET (!). Hejlsberg zette ter plekke een ASP.NET demo in elkaar. Hierbij maakte hij gebruik van Delphi code die op de server werd gecompileerd door Delphi voor .NET op het moment waarop de benodigde pagina werd aangeroepen. Na een kleine aanpassing was vervolgens dezelfde code als webservice beschikbaar. Erg indrukwekkend! De IDE kregen we vanzelfsprekend nog niet te zien (waarschijnlijk omdat die er nog niet is?).

"Aurora" en "Galileo"

Delphi 7 (codename: Aurora) is bedoeld om Delphi ontwikkelaars te helpen bij het bouwen van applicaties die .NET ready zijn. Dat betekent dus heel duidelijk dat dit NIET Delphi for .NET zal zijn. Wel wordt er een preview van Delphi for .NET meegeleverd, inclusief een Delphi MSIL compiler en VCL frameworks voor .NET. U mist alleen wel de IDE... Delphi 7 wordt verwacht in de tweede helft van 2002.

In de eerste helft van 2003 komt dan de échte Delphi voor .NET (codename: Galileo). Deze migreert de Delphi ontwikkelaars wel naar .NET en heeft een complete implementatie van de Delphi taal. Er is een VCL compatibele Component Library en een volledige ondersteuning voor het .NET Framework.

De reden dat het allemaal zo lang duurt is omdat Delphi for .NET meer is dan alleen maar een conversie. Ze zijn bij Borland terug gegaan naar de tekentafel en Delphi wordt van de grond af aan opnieuw opgebouwd. Er komt een nieuwe codegenerator, een nieuwe linker, heel veel nieuwe syntax en een nieuwe runtime library. Vanzelfsprekend blijven ze wel compatibel met vorige versies.



Anders Hejlsberg in Borland

Buiten dat de Delphi-taal opnieuw moet worden gebouwd, worden er ook een groot aantal elementen toegevoegd en/of verbeterd. Het zou te ver voeren om hier een complete lijst weer te geven maar een paar wilden we er u toch niet onthouden. De eerste is Unit namespaces. U benadert units met een volledige qualified identification (Bijvoorbeeld: Uses Borland.vcl.forms). De tweede is Multicast Events (multiple listeners per event). Een andere leuke is "Boxing" into Objects. Hiemee wordt bedoeld dat Primitive Types (bijvoorbeeld Integer of Double) kunnen worden ge-boxed in object wrappers.

Veel meer details over de toekomstige versies van Delphi, maar ook over de nieuwe JBuilder, de nieuwe C++ Builder, Mobile Service en "Charlotte" zijn te vinden in onze dagelijkse BorCon verslagen op de SDGN website. Zoek naar BorCon 2002.

► **When to Just Say No**

Multithreading is definitely not the solution for every design problem, and timing is often the issue. Just because you have the technical ability to run some operation "in the background" doesn't mean that is always the best way to do it. Here are some examples where multithreading wouldn't help and could actually make matters worse:

- Reindexing DBF files. If you can't do anything until the indexing process is complete, you don't need a secondary thread. (On the other hand, you could create indexes for different DBF files in different threads with your main indexing function waiting for all the indexing threads to complete, but only on a multiprocessor computer *might* there be any advantage to this approach.)

- Performing some operation that will require locking files that would be needed by the main application. (This is a variation of the example above, since other work would have to wait until the files are unlocked.)

- Updating a screen display with the current time. Why not just use a Windows timer instead? Multiple threads can be notoriously tricky to debug since logic errors might appear only occasionally. Don't use them unless you *need* them. (Even if Windows timers use multithreading internally, you don't have to debug that code.)

- Using multithreading for the sake of using multithreading. This should be obvious, but sometimes programmers somehow get the idea that they should use some feature of a language (or in the case of multithreading, the operating system) because it's there. Multithreading handles some specialized situations well, but first make sure you *must* use multithreading to solve the problem. Then proceed.

How to Write Multithreaded Applications

The first step in writing a multithreaded application is justifying the design choice for introducing the complexity and overhead of multithreading. (Admittedly, most of the simple examples presented in this paper probably wouldn't pass this first test.) And the second step is to design the application before you begin writing code. This is good software development practice anyway, but it becomes even more important in multithreaded applications since they can be trickier to debug later than single threaded applications, so it's especially important to get the design right before you start coding.

Getting Started with Multithreading in a Terminal Application

Writing the code for a multithreaded application involves two parts. The first part is writing the function that will execute as a separate thread. This function is the entry point for the secondary thread just as WinMain (inside the CA-Visual Objects runtime) is the entry point for your application's primary thread. Here is a simple example:

```
FUNCTION Count(dwParam AS DWORD) AS LONG PASCAL
         LOCAL i AS LONG

         FOR i := 1 TO 10000000
         NEXT
         RETURN 0
```

Notice that the function that executes as a secondary thread is strong typed. It may only take one argument, a DWORD, and it returns a LONG. The calling convention may be either STRICT or PASCAL. In this example, the function merely counts from one to ten million and returns. When the function returns, the thread ends.

The second part of writing a multithreaded application is to actually spawn off the thread. The Win32 API function for creating a new thread is called CreateThread. In CA-Visual Objects programming, you use CreateVOThread instead, which initializes some things in the CA-Visual Objects runtime and then calls CreateThread internally. Here is an example that times how long it takes to spawn off the Count function as a secondary thread and wait for that function to end:

```
FUNCTION Start()
LOCAL hThread AS PTR
LOCAL dwId AS DWORD
LOCAL dwBegin AS DWORD

dwBegin := GetTickCountLow()

hThread := CreateVOThread(null, 0, @Count(), ;
                          null_ptr, 0, @dwId)
WaitForSingleObject(hThread, INFINITE)
CloseHandle(hThread)

? "Elapsed ",GetTickCountLow() - dwBegin
wait
RETURN 1
```

CreateVOThread, like the Win32 API function CreateThread, returns a handle to the new thread. All handles in CA-Visual Objects are of the PTR data type. The important parameters passed to CreateVOThread in this example are the address of the function that will execute as a separate thread, **@Count()**, and the address of a DWORD where Windows will store the new thread's thread ID, **@dwId**. (Search in the Win32 API Help file for CreateThread for a description of all the parameters.)

WaitForSingleObject is used in this example to keep the primary thread from ending before the Count function running as a separate thread has finished. If that line of the code were omitted, the program would end almost as soon as it began. WaitForSingleObject is a Win32 API function that takes the synchronization object to wait for as the first argument and the timeout value in milliseconds to wait as the second argument. This useful function can wait for a number of types of objects including events, timers, processes, and of course threads.

After the thread has ended, the thread handle is closed using the Win32 API function CloseHandle. When I ran this code on my computer from the CA-Visual Objects IDE, the time to run the test was 2711 tenth milliseconds. The next example uses the same Count function but instead of spawning one thread, it kicks off two threads. Here's the new Start function:

```
FUNCTION Start()
LOCAL DIM hThreads[ 2]  AS PTR
LOCAL DIM dwIDs[ 2]  AS DWORD
LOCAL dwBegin AS DWORD

dwBegin := GetTickCountLow()

hThreads[ 1]  := CreateVOThread(null, 0, @Count(),;
                                null_ptr, 0, @dwIDs[ 1] )
hThreads[ 2]  := CreateVOThread(null, 0, @Count(),;
                                null_ptr, 0, @dwIDs[ 2] )
WaitForMultipleObjects(2, @hThreads[ 1], .T., INFINITE)
CloseHandle(hThreads[ 1] )
CloseHandle(hThreads[ 2] )

? "Elapsed ",GetTickCountLow() - dwBegin
wait
RETURN 1
```

This example stores both the thread handles and the thread Ids in DIM arrays. This is convenient, because you now need the application to wait for two threads to complete, not just one. So WaitForMultipleObjects is used instead of WaitForSingleObject. WaitForMultipleObjects takes the address of an array of synchronization objects as the second argument, and the first argument is the number of objects that will be in that array. The third argument specifies whether to wait for all the objects (TRUE) or only the first one (FALSE), and the last argument is the timeout value in milliseconds.

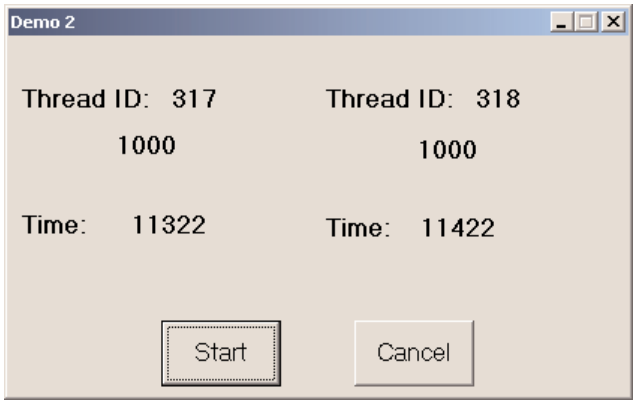
This code completed in 5411 tenth milliseconds on my test machine. That's about twice as long as the first example, so you can easily see that the application isn't really accomplishing the work any faster with multithreading on a computer with only one processor. You could design an application that spawns off some task and then returns control to the end user so the end user could continue with other work, and this would give the *impression* that the application is faster with multithreading, but it is only the user responsiveness that is faster. If you have any doubt, here's a third example using the same Count function:

```
FUNCTION Start()
LOCAL dwBegin AS DWORD

dwBegin := GetTickCountLow()
Count(0)
Count(0)

? "Elapsed ",GetTickCountLow() - dwBegin
wait
RETURN 1
```

This example uses no multithreading at all and also completed in 5411 tenth milliseconds on my test machine. If there were many threads and therefore many context switches, this example with no secondary threads would theoretically be measurably faster than the version using multithreading to accomplish the same total task on a single processor computer.

A Simple GUI Example of Multithreading

Multithreading may not save total CPU cycles to complete tasks, but the impression the end user gets of faster responsiveness can be important nonetheless. In fact, improving user responsiveness is one of the most widely used reason for multithreading. (Writing server applications is the other main reason to use multithreading.) Next look at an example of a simple GUI application that uses multithreading to display the number on the screen as the two threads count to 1000.



In this example, each thread counts to 1000 and then reports the time in tenth milliseconds elapsed for that thread to complete. (Because this is a graphical application that updates the screen for each number reached between 1 and 1000 for each thread, it naturally runs much slower than the terminal examples presented earlier in this paper.)

Several new techniques are introduced. Look at the class definition for the dialog window:

```
CLASS MainDialog INHERIT DIALOGWINDOW

PROTECT oCCStartButton AS PUSHBUTTON
PROTECT oCCCancelButton AS PUSHBUTTON
PROTECT oDCID1 AS FIXEDTEXT
PROTECT oDCID2 AS FIXEDTEXT
PROTECT oDCCount1 AS FIXEDTEXT
PROTECT oDCCount2 AS FIXEDTEXT
PROTECT oDCElapsed1 AS FIXEDTEXT
PROTECT oDCElapsed2 AS FIXEDTEXT

//{{%UC%}} USER CODE STARTS HERE (do NOT remove this line)
PROTECT s1, s2 AS Params
```

The dialog has two push buttons, a Start button and a Cancel button. And there are six fixed text controls, three for each thread. The first two are used to display the thread ID, the second pair are used to display the current count for each thread, and the third pair are used to display the time elapsed for each thread to count to 1000. There are also two pointers to a Params structure in the class definition. Remember that worker thread functions can only take one argument, which must be a DWORD, but that function will need to have access to both the Count and Elapsed time fixed text controls so it can update the display. A structure is used to hold pointers to these controls so they can both be passed to the thread function. Here's the Params structure:

```
STRUCTURE Params
MEMBER CountControl AS PTR
MEMBER TimeControl AS PTR
```

The secondary threads are spawned when the Start button is pressed. Here's the code:

```
METHOD StartButton( ) CLASS MainDialog
LOCAL DIM hThreads[ 2]  AS PTR
LOCAL DIM dwIDs[ 2]  AS DWORD

IF s1 = NULL_PTR
   s1 := MemAlloc(_sizeof(Params))
   RegisterKid(s1, 2, FALSE)
ENDIF
IF s2 = NULL_PTR
   s2 := MemAlloc(_sizeof(Params))
   RegisterKid(s2, 2, FALSE)
ENDIF

s1.CountControl := PTR(_CAST, SELF:oDCcount1)
s1.TimeControl := PTR(_CAST, SELF:oDCElapsed1)
s2.CountControl := PTR(_CAST, SELF:oDCcount2)
s2.TimeControl := PTR(_CAST, SELF:oDCElapsed2)

hThreads[ 1]  := CreateVOThread(null, 0, @Count(),;
                                s1, 0, @dwIDs[ 1] )
HThreads[ 2]  := CreateVOThread(null, 0, @Count(),;
                                s2, 0, @dwIDs[ 2] )
CloseHandle(hThreads[ 1] )
CloseHandle(hThreads[ 2] )
oDCId1:Caption := "Thread ID: "+AsString(dwIds[ 1] )
oDCId2:Caption := "Thread ID: "+AsString(dwIds[ 2] )
```

Although there are similarities between this code and the Start functions used in the Terminal examples, there are also a number of interesting differences. Perhaps the most important is the omission of the WaitForMultipleObjects call. Since the window used here is a modal dialog window, the application will keep running until the end user explicitly closes it by pressing the Cancel button. So WaitForMultipleObjects is not needed to keep the primary thread active until any secondary threads have ended. And, in fact, if you do insert a call to WaitForMultipleObjects here in the StartButton method, there will be problems. The primary thread will wait all right, but while it is waiting it will not process any normal Windows messages. In other words, it will not respond to the user or to any attempts by the secondary threads to update the captions on the fixed text controls. This is not what you want!

The second thing to notice is that the thread handles can actually be closed while the thread is still running. They are not needed after the call to CreateVOThread, so they are closed immediately afterwards. In fact, this is usually the best way to clean up thread handlesæjust close them right after the threads are created if you won't be needing them for WaitForSingleObject or WaitForMultipleObjects.

The third major difference involves working with the two structures used to hold parameters for the Count function. The structures are declared in the class definition, and storage is allocated for the structures s1 and s2 when the Start button is pressed. Because the user could press the Start button more than once during the execution of the application, storage is allocated only if s1 and s2 are null pointers. And because the pointers contained in the s1 and s2 structures are pointers into dynamic memory, RegisterKid is used to indicate to the garbage collector that the pointers contained in those structures should be updated whenever the dynamic variables they point to are moved in memory. The Kids are unregistered and memory allocated to s1 and s2 is freed in the QueryClose method, which is called as the window is closed.

```
METHOD QueryClose(oEvent) CLASS MainDialog
LOCAL lAllowClose AS LOGIC
lAllowClose := SUPER:QueryClose(oEvent)
//Put your changes here
IF s1 != null_ptr
   UnRegisterKid(s1)
   MemFree(s1)
ENDIF
IF s2 != null_ptr
   UnRegisterKid(s2)
   MemFree(s2)
ENDIF
RETURN lAllowClose
```

Here's the new Count function that updates fixed text controls on the dialog window with the current count and elapsed time:

```
FUNCTION Count(dwParam AS DWORD) AS INT PASCAL
LOCAL i AS LONG
LOCAL oText1, oText2 AS FixedText
LOCAL s AS Params
LOCAL dwBegin AS DWORD

dwBegin := GetTickCountLow()
s := dwParam

oText1 := OBJECT(_CAST, s.CountControl)
oText2 := OBJECT(_CAST, s.TimeControl)
FOR i := 1 TO 1000
    oText1:Caption := AsString(i)
NEXT
oText2:Caption := "Time: "+ ;
                  AsString(GetTickCountLow()-dwBegin)
RETURN 0
```

If you run this code, you will see that the two text controls that display the current value of i update at approximately the same speed and reach 1000 at about the same time. And when both threads have finished, the elapsed time for each thread is about the same. (It was 11322 tenth milliseconds for the first thread and 11422 tenth milliseconds for the second thread when I saved the screenshot for this paper.)

Setting Thread Priorities

Windows has two mechanisms that determine the priority for threads. The first value that affects the priority for a thread is the priority of the class for the process that owns it. The priority class for a process can be changed programmatically using the Win32 API functions GetPriorityClass and SetPriorityClass, but generally you don't need to change the priority class for your applications. If you make your applications run faster, the rest of the system will become less responsive, so you won't be improving overall performance in most cases. Or to put it differently, your end user won't be happier although he might not know that it was your application that was to blame.

In addition to the priority class for the process as a whole, the thread can have its own priority relative to the priority of the process. The default thread priority is THREAD_PRIORITY_NORMAL. For a background task, however, you might want to decrease the priority of the thread to make the rest of the application relatively more responsive. The thread priority values are (from lowest to highest):
THREAD_PRIORITY_IDLE,
THREAD_PRIORITY_LOWEST,
THREAD_PRIORITY_BELOW_NORMAL,
THREAD_PRIORITY_NORMAL,
THREAD_PRIORITY_ABOVE_NORMAL,
THREAD_PRIORITY_HIGHEST, and
THREAD_PRIORITY_TIME_CRITICAL.

You can find complete information about the thread priority options in the Win32 API Help file by searching on SetThreadPriority, but in general you will only consider using the values between LOWEST and HIGHEST.

To illustrate how changing thread priority can affect an application, make some modifications to the previous GUI example. First, change the structure aParams so it can hold three members. The third structure member will indicate whether the thread priority should be changed or not.

```
STRUCTURE Params
MEMBER CountControl AS PTR
MEMBER TimeControl AS PTR
MEMBER ChangePriority AS LOGIC
// Add member for priority change
```

Next, change the StartButton method so the first worker thread does not have its priority changed while the second worker thread does have its priority changed:

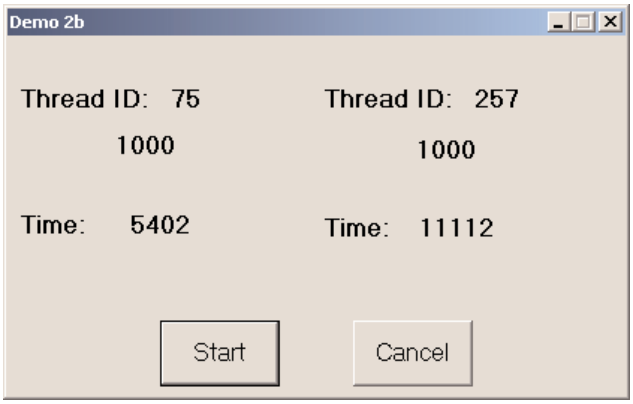
```
METHOD OKButton( ) CLASS MainDialog
LOCAL DIM hThreads[ 2] AS PTR
LOCAL DIM dwIDs[ 2] AS DWORD
LOCAL dwRet AS DWORD

IF s1 = NULL_PTR
   s1 := MemAlloc(_sizeof(Params))
   RegisterKid(s1, 2, FALSE)
ENDIF
IF s2 = NULL_PTR
   s2 := MemAlloc(_sizeof(Params))
   RegisterKid(s2, 2, FALSE)
ENDIF

s1.CountControl := PTR(_CAST, SELF:oDCcount1)
s1.TimeControl := PTR(_CAST, SELF:oDCElapsed1)

// Don't change priority of first thread
s1.ChangePriority := FALSE

s2.CountControl := PTR(_CAST, SELF:oDCcount2)
s2.TimeControl := PTR(_CAST, SELF:oDCElapsed2)

// But change priority of second thread
s2.ChangePriority := TRUE

hThreads[ 1] := CreateVOThread(null, 0, @Count(),;
                               s1, 0, @dwIDs[ 1] )
hThreads[ 2] := CreateVOThread(null, 0, @Count(),;
                               s2, 0, @dwIDs[ 2] )
CloseHandle(hThreads[ 1] )
CloseHandle(hThreads[ 2] )
oDCId1:Caption := "Thread ID: "+AsString(dwIds[ 1] )
oDCId2:Caption := "Thread ID: "+AsString(dwIds[ 2] )
```

Finally, change the Count function to either change the thread priority to a lower value or not depending on the value of the third structure member:

```
FUNCTION Count(dwParam AS DWORD) AS INT PASCAL
LOCAL i AS LONG
LOCAL oText1, oText2 AS FixedText
LOCAL s AS Params
LOCAL dwBegin AS DWORD
LOCAL iPriority AS LONG
LOCAL hThread AS PTR
LOCAL lChangePriority AS LOGIC


dwBegin := GetTickCountLow()
s := dwParam

oText1 := OBJECT(_CAST, s.CountControl)
oText2 := OBJECT(_CAST, s.TimeControl)
lChangePriority := s.ChangePriority

IF lChangePriority
   // Get the handle to this thread
   hThread := GetCurrentThread()
   // Save the current thread priority
   iPriority := GetThreadPriority(hThread)
   // Decrease the thread priority
   SetThreadPriority(hThread, THREAD_PRIORITY_LOWEST)
ENDIF
// Count to 1000 as before
FOR i := 1 TO 1000
    oText1:Caption := AsString(i)
NEXT

IF lChangePriority
   // Restore the previous thread priority
   SetThreadPriority(hThread, iPriority)
   // Clean up by releasing the thread handle
   CloseHandle(hThread)
ENDIF

oText2:Caption := "Time: "+AsString(GetTickCountLow()-dwBegin)
RETURN 0
```

You can get as fancy as you want with thread priorities

This time when the code executes, you can see that the two worker threads are not running at the same speed:



When I ran this test, the second thread counted up to about 6, then it seemed to pause until the first thread finished, then it continued counting up to 1000. As you can see from the screenshot, the second thread took about twice as long at TREAD_PRIORITY_LOWEST as the first thread running at THREAD_PRIORITY_NORMAL.
Notice when you are changing the priority of a thread that there are five steps:

1. Get the handle to the current thread by calling GetCurrentThread.
2. Get the current thread's priority by calling GetThreadPriority and save that value so you can reset it later.
3. Call SetThreadPriority to whatever new value you want. Do NOT use THREAD_PRIORITY_TIME_CRITICAL.
4. When you are finished, set the thread's priority back to the original priority by calling SetThreadPriority again.
5. Finally, call CloseHandle to free the resources used by the thread handle you got in step 1.

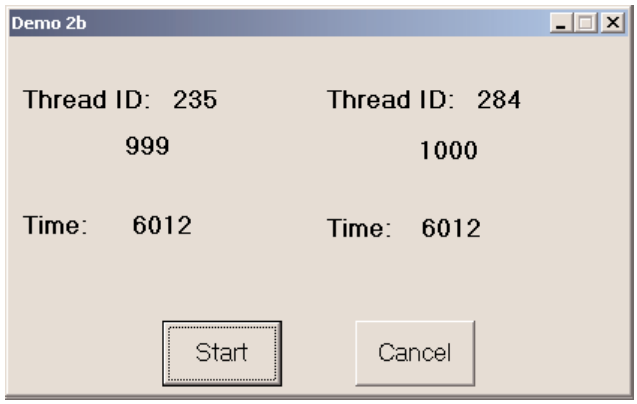
You can get as fancy as you want with thread priorities, but most of the time you won't need to do anything with the default thread priority. And in those situations where you do want to change the priority, most of the time it will be to decrease the priority of a worker thread rather than to increase it.

Sharing Memory

One of the biggest advantages of multithreading is that threads can all share the same memory. But if you don't plan carefully, this can sometimes seem like a big disadvantage instead. Go back to the original GUI sample and make one change to the Count function: change the local variable that functions as the loop counter to a static variable:

```
FUNCTION Count(dwParam AS DWORD) AS INT PASCAL
STATIC LOCAL i AS LONG
LOCAL oText1, oText2 AS FixedText
LOCAL s AS Params
LOCAL dwBegin AS DWORD

dwBegin := GetTickCountLow()
s := dwParam

oText1 := OBJECT(_CAST, s.CountControl)
oText2 := OBJECT(_CAST, s.TimeControl)
FOR i := 1 TO 1000
   oText1:Caption := AsString(i)
NEXT
oText2:Caption := "Time: "+AsString(GetTickCountLow();
                          -dwBegin)
RETURN 0
```

When the test is run, both worker threads update the variable i so the test runs about twice as fast. Here's the screenshot:



You can see that the first worker thread never displayed 1000 because it was the second worker thread that actually incremented the static variable i to 1000. When the time slice returned to the first worker thread, 1000 has already been reached, so all that was left to do was display the elapsed time. (You can press the Start button repeatedly to run the test over and over, and sometimes the first worker thread will reach 1000 and the second worker thread will stop at 999 or 998 or 997, and sometimes it will be the other way around.)

Similarly, you could remove the static variable i, and change i to a global variable. The result is the same: both threads update the global variable, the test runs in about half the time, and at the end of the test only one thread reaches 1000.

Ginny Caughey is a partner in Carolina Software, a US-based corporation specializing in software for the solid waste industry with installations all over the US and Canada.

(More information about her products is available at www.wasteworks.com.)

She has been a VO developer since the first alpha version, she was lead author of Using Visual Objects published by Que, and she has been a featured speaker at developer conferences in the US, Europe and Australia. She is currently working on a .NET project for PocketPC.

Visual Objects Tip

How do I extend the length of a MLE at runtime:

Insert the following code in the postinit of your data or dialog window to extend the MultiLine Edit's textlimit.

Just remember to substitute your control's name for MLE1.

```
SendMessage(SELF:oDCMLE1:Handle(), EM_LIMITTEXT,10000000L, 0L)
```

a u t e u r
REMI CARON

Web Services Part III

In the first article in this series I explained what a web service is, what elements are involved in using a web service and I showed you a very simple method call to a live web service. In part two I addressed the issue of debugging a web service, explained the WSDL file and talked about the phenomenon called UDDI. In this third and last article in this series I'll show you how to reuse a web service.

The examples I used in this article are based on our own online content management system. With this system you'll be in charge of when, how and why the site's content changes. Next to easy shop functionality and a sophisticated security system you will be able to deploy the content on whatever website you own. This is possible because the tool is designed as a web service.

As sample site I use the site of one of our clients: NedFox, a company specialized in Point of Sale systems. When you look at the image of their website as shown in fig. 1, you might think that you are looking at just another website. However there is a difference here because everything you see here is customizable by the client itself instead of contacting (and paying!!) a web designer for every change they want to make.

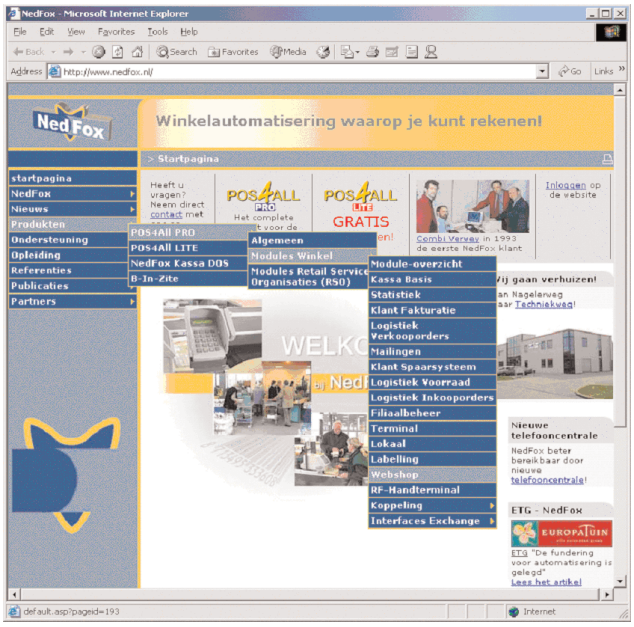


Fig. 1: The resulting web site

In the next figure you can see the tool that enables them to actually do the modifications. If you look closely to the left pane, which is called Sitestructuur, you'll notice the similarity with the menu in the previous image of the resulting website.

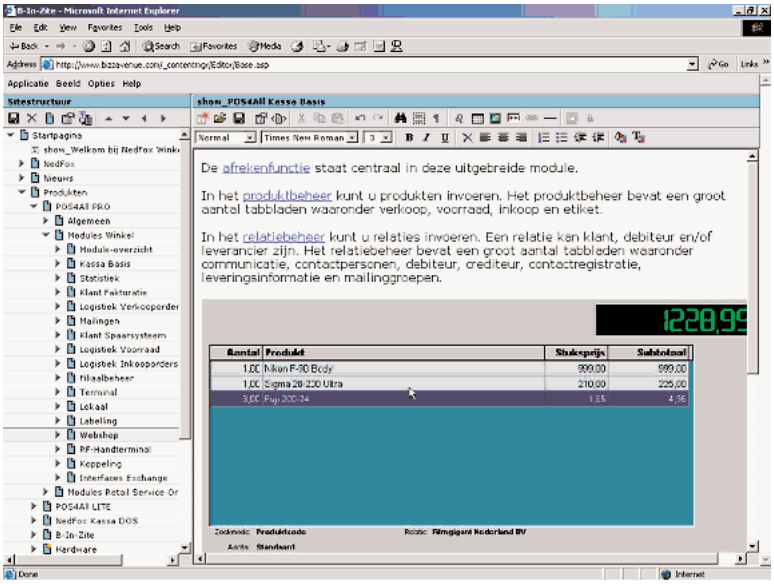


Fig. 2: The designer web site

In the right pane you see the editor that can be used to create the content for the website. The question that comes to mind after seeing the resulting and the designer website is how we translate the content build up in the designer into the result that is show to the visitor of www.nedfox.nl.
Well, as I mentioned earlier, the tool is set up as a web service. This means that you can invoke functionality, which (in this case) is hosted on a web server, somewhere in your own application. The application we reuse the information in, happens to be a website. In fact if someone is visiting the NedFox website, the website itself contacts our web service to gather the information it needs to display to the visitor. The process looks like this:

The visitor's browser visits the website of NedFox as they would visit any other website. The NedFox website however has no content stored on their web server. The pages and menu choices that are shown to the various visitors are retrieved from the CMS Web Service component

which lives on a totally different machine on the internet. The NedFox website however does have a few pages installed on their web server that handle the SOAP requests and responses between their server and the CMS server. Another thing that is stored in these pages is the design that is applied to the data that comes back from the CMS server. The visitor's browser activates the in place web service interaction between the NedFox server and the CMS server by asking for certain information. The visitor's browser knows nothing about this and he doesn't need to know anything about it either.
Most web services examples shown to you up until recently, reused functionality of a web service in a Windows application, which is fine of course, but this example shows that there is no law against using an web service from within a website.

There is no law against using a web service from within a website

So how do we get the information from the designer web page as shown in the resulting page? Since the data is stored on the CMS machine, we need to invoke SOAP calls from within the ASP pages in order to retrieve the data. The data is retrieved on demand, which means that not the whole site and all documents are downloaded as soon as a client visits the website. The site structure however will be downloaded for the whole site right away for performance reasons.

The web service exposes different functions to retrieve the various kinds of data sets that might be needed at a certain point in time. Let's start to take a look how to retrieve the site structure.

The default ASP page is the page that runs and coordinates it all. It is supported by two include files, one with JavaScript functions and one with wrapper functions to create, send and deal with SOAP messages.

```
fgetsitestructure
function fGetSiteStructure()
{
    var loSiteStructure = null;
    var lcReturn        = _oContentMgr.Site_GetSiteStructure(_cSessionGuid, false);

    loSessionGUID_Xml = fLoadXml(lcReturn);
    if(loSessionGUID_Xml != null)
    {
        loSiteStructure = _oXmlParser.selectSingleNode('//sitestructure');
        if(loSiteStructure == null)
            fErrorExit("Er is een fout opgetreden bij het opbouwen van het menu.");
    }
    else
        fErrorExit("Er is een fout opgetreden bij het opbouwen van het menu.");

    return loSiteStructure;
}
```

This function calls a function on an instantiated object. The object that is instantiated here is in fact the include file with the wrapper functions. As you probably know you can create functions in JavaScript and instantiate them as an object or call them as a function.
When you study the code, you'll notice that the line

```
_oContentMgr.Site_GetSiteStructure(_cSessionGuid, false)
```

does the real Site_GetSiteStructure call, returning an XML string containing the site structure. The code also has some error handling for this specific function with warnings the user can understand, when an error occurs.

What is the point in having wrapper functions you might ask yourself? Well, there is a good reason for having these functions in a separate file. The functions in that file handle everything there is to handle for sending and receiving the SOAP requests and answers, as well as dealing with possible SOAP errors. So by using the functions in the file for every interaction with the CMS server, all that SOAP handling complexity is written once and used many times. A function like fGetSitestructure as shown here can be written for different web sites doing different things with the retrieved result, but in that fGetSiteStructure function this call:
_oContentMgr.Site_GetSiteStructure(_cSessionGuid, false)
will be executed to actually get the site structure from the CMS server. As you take a look at the downloadable files, you'll see that in this case the XML data will be stored in an array that is used to build the menu structure on the client side. The choice to use JavaScript and an array to build the menu on the client side has to do with being able to show this site in more browsers than IE only.

What is the point in having wrapper functions

Let's take a closer look at the functions that are used in the include file that holds the wrapper functions. The function to get the actual site structure uses some data from the client's session and the passed through parameters and passes these on to the function that does the actual SOAP request handling.

```
bizcmtserver_site_getsitestructure
function BizCmtServer_Site_GetSiteStructure(tcSessionGUID, tbIncludeContent)
{
    return BizCmt_GetSoapResponse(this.cSoapServer,
                                  "Site_GetSiteStructure",
                                  "tcSessionGUID",tcSessionGUID,
                                  "tbIncludeContent",tbIncludeContent);

}
```

The function GetSoapResponse is the one that does the actual SOAP handling request. It is listed below in 2 parts. There are a few things you should notice about this code. The SOAP envelope that is created is the same for every call to the CMS–server, at least the header and the footer are the same. The body is the part that changes because it depends on which function is called on the CMS-server and how many parameters it should send. As you look through the code you'll notice that it is just

some simple JavaScript creating the body part and using the XMLHTTP control to send it over the wire. This is shown in the 1st part of GetSoapResponse:

```
function BizCmt_GetSoapResponse(tcSoapServer, tcMethod)
// tcMethod: name of soap request method as string
//           comma separated list of parameters and parameter values
// returns : soap result
{
    var lcReturn    = ""
    var loXmlHttp   = new ActiveXObject("MSXML2.ServerXMLHTTP");
    loXmlHttp.open("POST", tcSoapServer, false);

    // create soap request
    cSoapRequest  = '<?xml version="1.0" encoding="UTF-8" standalone="no"?>'
    cSoapRequest += '<SOAP-ENV:Envelope '
    cSoapRequest += 'SOAP-ENV:encodingStyle='
    cSoapRequest += '"http://schemas.xmlsoap.org/soap/encoding/" '
    cSoapRequest += 'xmlns:SOAP-ENV='
    cSoapRequest += '"http://schemas.xmlsoap.org/soap/envelope/">'
    cSoapRequest += '<SOAP-ENV:Body>'
    cSoapRequest += '<Method:' + tcMethod
    cSoapRequest += ' xmlns:Method="http://tempuri.org/message/">'

    // add method parameters
    var lnParam = 2;
    while(lnParam < arguments.length && arguments.length > 1)
    {
        cSoapRequest += '<' + arguments[lnParam] + '>'
        cSoapRequest += arguments[lnParam + 1]
        cSoapRequest += '</' + arguments[lnParam] + '>'
        lnParam += 2;
    }

    cSoapRequest += '</Method:' + tcMethod + '>'
    cSoapRequest += '</SOAP-ENV:Body>'
    cSoapRequest += '</SOAP-ENV:Envelope>'
```

Then, after the request has come back the result will be parsed and returned back up into the call stack. This is shown in the 2nd part of GetSoapResponse:

```
    // send soap request
    loXmlHttp.setRequestHeader('Content-Type:','text/xml')
    loXmlHttp.setRequestHeader('SOAPAction:',
        'http://tempuri.org/action/Application.' + tcMethod )
    loXmlHttp.send(cSoapRequest);

    // handle soap response
    if( loXmlHttp.status == 200 );
    {
        var loReturnXml = loXmlHttp.responseXML ;
        var loResponse  =
            loReturnXml.selectSingleNode("//SOAPSDK1:" + tcMethod +
            "Response");
        if(loResponse != null)
        {
            lcReturn = loResponse.text;
        }
    }
    return lcReturn;
}
```

After these functions are executed and the result is returned to the NedFox server, the XML is turned into an array which is used to build the menu with.
So now we've got a part of the users User Interface built up and we're ready to go. This is where we get to the page data that needs to be retrieved based on the menu choice made by the visitor of the site.

```
function fWriteDocument(pcDocID)
{
    var lcDocXml = _oContentMgr.HTML_GetFragment(
        _cSessionGuid,Request.QueryString('docid').Item,false);
    var loDocXml = fLoadXml(lcDocXml);

    if(loDocXml != null)
    {
        lcTitle = loDocXml.selectSingleNode("//content/title").text
        if(lcTitle.substr(0,5) == "show_")
            lcTitle = lcTitle.substr(5)
        else
            lcTitle = " "

        var lcDate = loDocXml.selectSingleNode("//content/modified").text;
        lcDate = lcDate.substr(0,10);
        Response.Write("<table class='contentitem' border='0' cellpadding='2' ");
        Response.Write("cellspacing='0'><tr><td class='contenttitle'><b>");
        Response.Write(lcTitle);
        Response.Write("</b></td><td class='contentdate' nowrap>");
        Response.Write(lcDate);
        Response.Write("</td></tr><tr><td class='contenttext' colspan='2'>");
        Response.Write(loDocXml.selectSingleNode("//content/text").text);
        Response.Write("</td></tr></table><br>");
    }
    else
        fErrorExit("Er is een fout opgetreden bij het ophalen van een document.");
}
```

As you can see in the code here, I just use another function in the object to get to the actual documents content:

```
_oContentMgr.HTML_GetFragment(_cSessionGuid,Request.QueryString('docid').Item,false).
```

For this function the same argumentation can be used as for the site structure. By wrapping it up, the function it self and its action may vary per implementation, but the main part of it, the one that's actually retrieving the necessary data, will stay the same and hidden in that object.

You won't be surprised when you take a look at the code of the _oContentMgr.HTML_GetFragment function.

```
function BizCmtServer_HTML_GetFragment(tcSessionGUID,tcID,tbShort)
{
    return BizCmt_GetSoapResponse(this.cSoapServer,
                                  "HTML_GetFragment",
                                  "tcSessionGUID",tcSessionGUID,
                                  "tcID",tcID,
                                  "tbShort",tbShort);

}
```

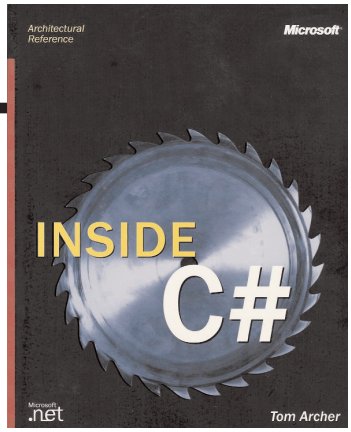
Yes, it's practically the same as for retrieving the site structure. This function calls the SOAP handler but with different parameters as you can see.

Well this article brings me to the end of the series on web services. We've seen what they are, how to debug and analyze them, and in this part we've seen how to invoke them in your application or website. I hope this series has helped you to understand this phenomenon a little better and that you're able to see the power of it. Rethink system architectures and build your systems in a way using web services as building blocks. If you create smart web services, these will give you the opportunity to expose data for many platforms or build business objects once for many applications.

Remi Caron is technical director and co-founder of BizzView B.V. a company that specializes in e-business solutions. Remi has a long history as a Foxpro developer. These days his company uses any Visual Studio tool that suits the job best.

He is also the section manager for the MS-Development group within the Software Developers Group Netherlands. He can be reached at Remi.Caron@BizzView.com

auteur
Peter van Ooijen



Boekreview: Inside C#

Rond .NET word je op het ogenblik overspoeld met websites, boeken, artikelen, nieuwsgroepen, seminars en wat al niet. .NET trekt ontwikkelaars aan met zeer uiteenlopende achtergrond, van ASP scripters tot C++ hackers. Maar ook mensen met een niet Microsoft achtergrond, neem nou Delphi of Jbuilder, laten hun neus zien. De vele .NET publicaties bouwen allemaal verder op bestaande kennis. Heb je een ASP, C++ of Visual Basic achtergrond dan is er leesmateriaal te over, heb je een Delphi achtergrond dan wordt de spoeling wat dunner. Eén van de vele mooie kanten van "Inside C#", geschreven door Tom Archer voor Microsoft Press, is dat het niet uitgaat van welke kennis dan ook van een bestaande programmeertaal, een ieder kan met dit boek aan de slag.

De titel "Inside C#" dekt niet helemaal de lading. Aan de ene kant herinnert het aan klassiekers zoals "Inside OLE", "Inside COM" en "OLE controls inside out". Aan de andere roept het associaties op met diep spitwerk. De eerste associatie is wat mij betreft terecht, het boek is een potentiele klassieker. De tweede associatie is minder geslaagd, het is gewoon een erg goed boek over (het programmeren voor) .NET en de taal C#.

In dit artikel zal ik een persoonlijke samenvatting geven van de inhoud en zo proberen een beeld te schetsen van C# en zijn relatie met .NET. Het verhaal zal verre van volledig zijn, maar ik hoop wel je nieuwsgierigheid genoeg te kietelen om zelf met boek, taal en framework aan de gang te gaan.
Inside C# is ingedeeld in vier delen. In het eerste deel worden de grondbeginselen van object orientatie en class libraries op een rijtje gezet. In het tweede deel worden de fundamenten van classes in C# behandeld. Pas in het derde deel komt het daadwerkelijk programmeren aan bod. En in het vierde deel kan je in een aantal hoofdstukken zien wat voor krachtige "advanced" zaken je met de verworven kennis kan bouwen.

Deel 1 "Laying the groundwork"

De taal C# is van de grond af nieuw opgebouwd. Talen als C en Java hebben officieel een grote invloed op het resultaat gehad maar ook een kenner van Object Pascal (Delphi) herkent het nodige. C# is een object georiënteerde taal, het **eerste hoofdstuk** behandelt dan ook de "Fundamentals of Object-Oriented Programming". Ook als je denkt daar alles al van te weten loont het de moeite het toch aandachtig te lezen. Werkelijk alles is een object in .NET. Van een eenvoudige integer variabele ergens in je code tot en met de applicatie die je aan het bouwen bent. Stapje voor stapje krijg je te zien hoe de kreten als Class, Encapsulation, Inheritance en Polymorphism er in C# uit zien.

In het **tweede hoofdstuk** komt .NET om de hoek kijken. .NET is niet alleen een framework om (web-) applicaties te maken het is ook een verzameling van classlibraries voor C#. In het framework en in je eigen code stammen alle classes af van één basisclass. Wat voor Delphi tObject is, is voor .NET System.object. En de class System.object zit in het .NET framework. Een C# programma zonder .NET kan dus niet.

In het **derde hoofdstuk**, "Hello C#" wordt een eerste C# programma in elkaar gezet. Visual Studio wordt wel genoemd maar de auteur gebruikt notepad en de command line compiler. De compiler zelf is ook een onderdeel van het .NET Framework, op zich heb je aan de .NET framework SDK dus genoeg om .NET applicaties te bouwen.

Deel 2 Class fundamentals

Het schrijven van een .NET applicatie komt neer op het bouwen van classes. Classes hebben erg veel mogenlijkheden in C#, deze worden hoofdstuk voor hoofdstuk behandeld.

Hoofdstuk 4 gaat over het .NET type systeem. Zoals eerder gezegd is alles een object in C# en stamt elk object af van de .NET class System.object. Het Common Type System in .NET heeft een uitgebreide serie hiervan afstammende typen in huis en de Common Language Runtime, de interne motor van een .NET applicatie, heeft een uitgebreide kennis van dit type systeem.

Je mag een variable casten van het ene naar het andere type. Maar als je een typecast uitvoert naar een type wat niet voorkomt in de afstammingslijst van het object in kwestie dan levert dat altijd een foutmelding of een null variable op, .NET is typesafe.

Hoofstuk 5 behandelt de classes, alle code die je in C# schrijft komt neer op het het schrijven van classes. Alle classes hebben maar één ouder, C# kent geen multiple inheritance. C# kent interfaces; één C# class kan meerdere interfaces te implementeren.

Objecten van een class worden aangemaakt door de constructor van een class, dat is een methode met dezelfde naam als de class. Een constructor voert default als eerste de constructor van zijn base-class uit. Om dit process bij te sturen wordt in de declaratie van een constructor aangegeven welke constructor eerst moet worden uitgevoerd.
Het opruimen van objecten gebeurt in talen als object pascal en C++ expliciet door het aanroepen van de destructor van het object. In talen als VO gebeurt het achter de schermen door de garbage collector. Ook .NET werkt met een garbage collector. Beide werkwijzes hebben hun voor- en na- delen. Tom Archer gaat uitgebreid in op de.NET wijze, de bijbehorende valkuilen en de ondersteuning van de garbage collector door het framework.

Hoofdstuk 6 behandelt methodes, daar zal alle daadwerkelijke functionaliteit van een applicatie geïmplementeerd gaan worden. De parameters van een methode kunnen worden gebruikt om variablen een methode in te krijgen maar ook om waardes terug te geven. Voor dat laatste kent C# *ref* en *out* parameters. Nou is het zo dat de C# compiler een foutmelding geeft als je iets gaat doen met een niet geïnitialiseerde variabele. Dat is heel slim maar het heeft niet veel zin om een variabele eerst een waarde te moeten geven, hem vervolgens aan een methode mee tegeven en dan pas te gaan werken met de waarde die de methode weer terug gegeven heeft. Is de parameter als een out parameter gedeklareerd, dan mag je een niet geïnitialiseerde variabele meegeven. Dit zijn natuurlijk details maar het geeft wel aan hoe goed er over C# en .NET is nagedacht en hoeveel hulp je krijgt bij het voorkomen van bugs.
C# kent method overloading. Dat wil zeggen dat je meerdere versies van dezelfde methode kan schrijven die verschillen in hun signature, het aantal en type van hun parameters. Als het aantal te verwachten parameters niet vast ligt dan kan je een parameter als array, voorzien van het params keyword, declareren. De aanroep heeft dan een onbeperkt aantal parameters, de methode ontvangt ze in één array.

De manier om een afgeleide class ander gedrag te geven dan zijn base-class is het overriden van methodes. In die base class kan je een methode declareren als *virtual*, waarna je met het keyword *override* een nieuwe implementatie van de methode schrijft. Een niet virtuele methode is te overriden met het keyword new, te vergelijken met *reintroduce* in Delphi. In beide varianten is de oorspronkelijke implementatie te benaderen via het keyword *base*. Ook C# kent class methodes (en velden), je kunt ze gebruiken zonder een object van de class te hebben aangemaakt. Deze worden gemerkt als *static*. Bijzonder aardig zijn static constructors, zij worden uitgevoerd bij het laden van het programma, je kan ze gebruiken om static velden van je class te instantieren.

Hoofdstuk 7 behandelt eerst properties. Een .NET class kent naast gewone variabelen, fields genoemd, ook properties zoals je die in COM, Delphi of VB ziet. Het zetten of lezen van een property variable loopt via een methode. In de code daarvan kan een waarde eerst worden gevalideerd alvorens die wordt toegekend. Of de waarde wordt pas berekend bij het opvragen van de property.
Het tweede deel van het hoofdstuk houdt zich bezig met arrays en indexers. Je maakt van een variabele een array door in de declaratie een paar vierkante haken achter het type te zetten. Elk object op zichzelf kan zich ook gedragen als een array. Dit doe je via een zogeten indexer. Daar komen properties en arrays bij elkaar, een indexer is een property met het type van de class zelf. De property getter heeft minstens één parameter, die gebruik je dan om de juiste waarde van het geïndexeerde obect te bepalen. Aardig detail is dat het type van de index parameter vrij te kiezen is. Een array element vraag je op via een integer, voor een geïndexeerd object kan net zo goed een datum gebruiken.

Hoofdstuk 8 behandelt atributen. Een .NET programma bevat naast de code en de data ook metadata. Dat is data waarin de classes nader beschreven worden. Van een COM class wil je weten wat de GUID van de class is, van de methodes van een COM+ class wil je weten hoe ze zich in een transactie moeten gedragen. In code geef je dit aan met atributen bij de declaratie van een class of methode. Atributen zijn objecten die afstammen van de .NET class System.Attribute. Naast de vele atribuut classes in het framework kan je, na lezing van dit hoofdstuk, ook je eigen atribuutclasses schrijven en daarmee al jouw metadata ter plekke in de source stoppen.

Hoofdstuk 9 gaat over interfaces. Een interface is de declaratie van een class zonder de bijbehorende implementatie. Een .NET class kan maar van één basisclass afstammen maar wel een onbeperkt aantal interfaces implementeren. Code die is geschreven op interfaces werkt met objecten van classes die niets in hun afstamming gemeen hebben (behalve System.object) maar wel dezelfde interface implementeren. De code benadert het object via de methodes uit de betreffende interface. Wie bekend is met interfaces in Delphi zal dit verhaal bekend in de oren klinken .

Deel 3 Writing code

"Think first, program later"zeggen ze wel eens. Ik ben nu halverwege het boek en ben alleen nog maar verhalen tegengekomen over het declareren van de classes. Pas nu gaat de auteur in op het schrijven van code die "iets doet".

Hoofdstuk 10 behandelt expressies en operators. De degelijkheid van het boek gaat hier opvallen. Alle operators komen voorbij. En dat zijn er nogal wat, zo kent C# elke compound operator (+=, -=) die je maar bedenken kan.

Het mooie aan de taal is dat de betekenis van een operator nooit hoeft te worden afgeleid uit de context, zo staat = altijd voor een toekenning en == voor een vergelijking.

Hoofdstuk 11 richt zich op de flow in een programma. Opnieuw valt hier op hoe eenvoud en kracht in C# zijn gecombineerd. Zo heb je in Delphi een *do while* en een *repeat until* statement. C# lost dit iets eenvoudiger op. Het is hier *while (Logische expr) {statement}* en *do {Statement} while (logische expr)*. In het eerste geval wordt er eerst getest en dan (eventueel) iets uitgevoerd, in het tweede geval wordt de code zo wie zo een keertje uitgevoerd en wordt er daarna pas getest. Ook vermeldenswaardig is het foreach statement, jarenlang kon je hiemee in VB(A) op een handige manier collecties van OLEvariants doorlopen. In C# kan je het loslaten op elk object dat de .NET interface *Ienumerator* implementeert. En daaronder valt ook een groot gedeelte van het framework zelf.
Statements die een rechtstreekse sprong uitvoeren liggen gevoelig. "Goto considered harmfull" zei Dijkstra meer dan 25 jaar geleden al eens en ik geneer me ook een beetje als ik in VBA weer eens *on error goto MyError* moet schrijven. Maar C# kent het goto statement zelfs in meerder varianten. Tom Archer besteed meerdere pagina's aan de verschilende variaties. En zo gek klinken die helemaal nog niet.

Hoofdstuk 12 behandelt het constateren en opvangen van fouten. .NET maakt gebruik van exception objecten. Optredende fouten worden opgevangen in een try catch blok, de fout wordt beschreven in het binnenkomende exceptionobject. Het bijbehorende finally blok wordt altijd, fout opgetreden of niet, uitgevoerd en daarin kan alle rommel worden opgeruimd. Het .NET framework is rijk aan exception classes, bij het constateren van een een fout maak je een object van een (afgeleide) exception-clas aan en genereer je de exception met het throw statement. Delphi kenners zal dit alles zeer bekend voorkomen, vermeldenswaardig is dat je in .NET try, catch en finally in één constructie kan combineren.

Hoofdstuk 13 gaat in op het verder bijsturen van het gedrag van je objecten. Middels operator overloading kan je de betekenis van een operator, zoals +, voor jouw class een eigen betekenis gaan geven. Stel, je hebt een factuur met een x-tal factuurregels. Het optellen van twee facturen kan je dan definïeren als het maken van één nieuwe factuur waarin alle regels bij elkaar worden gevoegd. De methode heet operator+ en krijgt de twee op te tellen factuur objecten als parameter binnen. De implementatie maakt een nieuw factuurobject, voegt alle regels uit beide binnengekomen facturen aan deze nieuwe factuur toe en geeft die nieuwe factuur terug als resultaat. En nu kan je in je applicatie zo programmeren: *Factuur3 = Factuur1 + Factuur2;*. Erg handig. Iets lastiger te bevatten zijn User-defined conversions. In C# kan je een variabele typecasten van het ene type naar een ander type. Dat gaat op voorwaarde dat het ene type dat andere type ook daadwerkelijk kent in zijn eigen afstammingsboom of lijst van geïmplementeerde interfaces. Om anders toch een succesvolle typecast uit te kunnen voeren kom je terecht bij User Defined Conversions. Tom Archer laat een voorbeeld zien van het typecasten van een Celsius object naar een Fahrenheit object. Met behoud van temperatuur, die wordt door de typecast automatisch geconverteerd.

Hoofdstuk 14 gaat over delegates en eventhandlers. In een applicatie treden geregeld events op. De gebruiker klikt op een button of een routine die een database aan het openen is meldt dat die halverwege is. Aan deze events kan een C# methode gekoppeld worden, als de event afgaat wordt deze eventhandler uitgevoerd.
Als je de event en de eventhandler gescheiden houdt en in code of runtime expliciet aan elkaar gaat koppelen, dan ontstaan er vele mogenlijkheden. Zo kan dezelfde eventhandler gebruikt worden bij meerdere knoppen en kan de database client zijn eigen code "injecteren" in de database routine. Dit gaat allemaal goed zolang beide partijen overeenstemmen in de signature van de eventhandler. Zo'n signature heet in het .NET framework een delegate. Een class kan een delegate property publiceren waarna gebruikers van de class op de delegate kunnen subscriben. Bij het afgaan van het event zullen alle subscribers een aantal parameters ontvangen, waaronder één van het type System.EventArgs krijgen. Een uitgebreider verhaal over delegates is te vinden in "C#, ook mooi", in #69 van dit magazine.

Deel 4 Advanced topics

In deel 4 mag Tom Archer, naar eigen zeggen, eindelijk los. De keuze van de hoofdstukken lijkt een beetje willekeurig. Er is natuurlijk ontzettend veel te kiezen in .NET, er zijn zoveel classes die leuke dingen doen.

Hoofdstuk 15 gaat over threads. Dankzij de Thread class in .NET is het schrijven van multithreaded code niet echt lastig. Alle classes in het .NET framework zijn thread-aware, maar ze zijn, om met Tom te spreken, gelukkig niet allemaal thread safe.

Hoofdstuk 16 gaat verder in op het type systeem. Met reflection zijn heel gekke kunstjes te doen. Zo kan je meerdere versies van een DLL bouwen en runtime pas gaan kiezen welke versie daadwerkelijk gebruikt gaat worden. Nog een stap verder gaat het om runtime C# code te gaan te gaan genereren, deze door de C# compiler te (laten) halen en het resultaat in hetzelfde programma weer te gebruiken. De compiler zit in het framework en de System.Reflection classes bieden een uitgekiend scala van methodes om hem te gebruiken.

Hoofdstuk 17 behandelt het communiceren met de buitenwereld. Alles wat niet voor de .NET CLR is geschreven wordt unmanaged code genoemd. Dit kunnen ouderwetse Windows DLL's zijn maar ook COM servers. C# code kan hier prima mee werken, middels atributen wordt aangegeven in welke DLL of COM server de aan te roepen functie zit. Voor de gebruikers van de resulterende C# class is het transparant waar en hoe de methodes van de class worden uitgevoerd.

Het slot, hoofdstuk 18, staat mijns inziens op de verkeerde plaats in het boek. "Working with assemblies" gaat in op .NET assemblies. Een (verzameling van) assembl(y)(ies) is de vorm waarin een .NET applicatie wordt uitgerold. Het uitrollen van een applicatie kan zo simpel zijn als het kopiëren van de assembly files naar een directory.
Assemblies worden gedeeld door meerdere toepassingen middels de global assembly cache. Die cache levert een specifiek versie van een assemby op bestelling. Een toepassing kan vragen om de nieuwste versie van "MyLib", maar ook om versie 1.0.1.17 van "Yourlib". Een andere toepassing kan vragen om een versie 2.17.x.x van "MyLib" en versie 1.3.1.1 van "YourLib". Mits al deze versies ooit aan de cache zijn toegevoegd dan zal de cache al de toepassingen tevreden kunnen stellen.
Aan de ene kant hebben assemblies default een gedrag dat voor de meeste gebruikers voldoende zal zijn, aan de andere kant bieden assemblies zoveel andere wetenswaardigheden dat dit hoofdstuk van mij best een prominieter plaats had mogen krijgen.

CD ROM

Bij het boek zit (bijna) vanzelfsprekend een CD-ROM. Hierop staan in de eerste plaats de samples van alle voorbeelden in het boek. Tom werkte met de commandline compiler en notepad, de samples dan ook uit een verzameling losse .cs sourcefiles. Het zijn ook geen sexy visuele zaken met veel toeters en bellen maar pure console apps. Het nuttigste op de CD is het hele boek nog een keertje als eBook. De papieren versie bladert lekker, de kopjes zijn kort en duidelijk maar in een eBook kan je zoeken op welke woord dan ook. Binnen de korste keren werd "Inside C#" voor mij een naslagwerk. In het vorige nummer van SDGN magazine beloofde ik een automation server geschreven in C#. Dat is mede dankzij dit boek goed gelukt, het resultaat is te vinden op mijn website, www.Gekko-Software.nl/DotNet.

Peter van Ooijen is een ontwikkelaar met een grote interesse voor zowel .NET als COM en Delphi.
Naast artikelen voor SDGN magazine schrijft hij ook veel voor zijn website www.Gekko-Software.nl.

auteur
CHAD Z. HOWER

Indy Pit Stop

Introduction

This time I will present a simple web (HTTP) server built with Indy.
I plan this to be the first in a series of columns about Indy. Each installment I will demonstrate how to do something both useful and fun with Indy.

Indy 9

Indy 9 is on track to be released in May so this article will use the current Indy 9 code base as it is feature frozen now. The project could be created quite similarly in Indy 8 as well. Delphi 6, C++ Builder 6 and Kylix ship with Indy 8. To upgrade to Indy 9 you should download the development snapshot or the final release (when released) from http://www.nevrona.com/indy/.

The Firewall Friendly Protocol

HTTP is usually associated with web pages but can be used for many other purposes. Many corporations have set up strict firewalls that by default only allow mail and web traffic to traverse the firewall. If you have worked behind one of these firewalls you know that asking the firewall administrator to open up new protocols is usually an exercise in futility. Because of this HTTP has been adapted as the firewall friendly protocol and many higher level protocols now ride on top of HTTP allowing them to be accessed across firewalls. One example of this is SOAP. SOAP is not restricted to HTTP but it is certainly the most common transport for it.

HTTP has been adapted as the firewall friendly protocol and many higher level protocols now ride on top of HTTP allowing them to be accessed across firewalls

This month however we will be serving standard web pages, but in a dynamic way. This could certainly be done as an ISAPI plug-in for Microsoft IIS or a DSO for Apache. However with Indy you have more flexibility, better debugging options, and a completely functional dedicated HTTP server that you can distribute independently. Many commercial vendors use this feature to ship commercial products that do not require IIS or Apache. IIS has taken a real beating recently with security flaws and as a result many corporations are now shying away from it. And for smaller shops it can be a maintenance and security issue to install it. Even if the customer has an existing web server installed the Indy based web server can be configured to run on an alternate port in parallel to the existing web server. Three such commercial products are IntraWeb from Atozed Software, Apollo Server from Vista Software and Rave Server from Nevrona Designs. All of these stand alone HTTP based servers are built using Indy. Borland also uses Indy for the SOAP server in Kylix.

Project Description

In this article I will build a HTTP server, but a specialized one. Indy's HTTP server can certainly be used to build a web server which serves up static web pages, but why reinvent the wheel? Indy's HTTP server is most useful for developing custom HTTP servers, or custom protocols built around HTTP.

Back in the 1970's in the early days of personal computing a program called Eliza was published in Creative Computing. Creative Computing was the computing magazine of the day and contained a lot of interesting content. The computing power of a personal computer in these days was very limited so all source code submissions were printed and users merely typed them in. Many submissions were games and the like but many were not. One such submissions was Eliza. Eliza was a computer psychiatrist that used a very simple form of language processing to take human input and provide a two way conversation. Over the years Eliza has been ported to many languages. I found some old basic code and in some language called Inform. From these two sources I have ported it to Delphi. I plan to make further improvements later.

This article will not focus on Eliza other than to use it as the dynamic part of this demonstration. In this article we will create Eliza Web, the HTTP shrink. Eliza is only a few dozen lines of code plus some data but can produce some amazing dialog. This is why it started such a firestorm of interest in the 1970's and has persisted in popularity over the years. Here is an example Eliza conversation:
User: What makes you think I have a problem?
Eliza: We were discussing you--not me.
User: I want to talk about women.

Eliza: Why do you want to talk about women?
User: Because I want to talk about women.
Eliza: Suppose you soon got to talk about women?

And so on... Eliza can carry on some pretty interesting conversations. Its responses are pretty amazing for a few dozen lines of code.

Building a Minimal Web Server

Building a minimal web server in Indy is really quite easy. Start with a new application and add a TIdHTTPServer component to the form. Next set the Active property to True. Now the only remaining step is to implement the OnCommandGet event. Create a OnCommandGet event and enter the following code:

```
procedure TForm1.IdHTTPServer1CommandGet(
            AThread: TIdPeerThread;
            ARequestInfo: TIdHTTPRequestInfo;
            AResponseInfo: TIdHTTPResponseInfo);
begin
  AResponseInfo.ContentText :=
    'Hello World. It is ' + TimeToStr(Time);
end;
```

Run the application. All you should see is an empty form, however you now have a functional web server.
Now launch your web browser and enter this URL:
http://127.0.0.1/
If you have a web server running on your machine this web server will conflict with it and you will receive an error when you run the application. To avoid this, temporarily shut down your web server. You can also change the port that this custom web server listens on. To do this, reset the application and change the TIdHTTPServer's DefaultPort property. Try 8989. Now run the application again. The consequence is that you will now need to change all URLs to include the port, such as is shown here:
http://127.0.0.1:8989/
127.0.0.1 is a special network address and equivalent to "self". This will cause the request to be served by your machine no matter what your machine's real network address is. You should see a web page returned similar to this:

```
Hello World. It is 9:19:34 PM
```

Force the web page to refresh and you will see that the time changes accordingly and is a truly dynamic web page. Congratulations! You just built a dynamic web server.

TEliza

Do not let your significant other walk in during reading this article – she might worry that you are trying to make a virtual significant other. Anyway, TEliza has a simple interface. For the scope of this article we will not cover the implementation, but only the interface.
The interface is quite simple. Construct a TEliza and keep it in memory as it keeps internal state. Then simply call the (AMsg: string) function for each question or statement you want Eliza to respond to. The function returns a string which contains Eliza's response. TEliza is self contained in Eliza.pas.

Building Eliza Web

I built the initial Eliza server just as I did the minimal server before and then expanded on it. Eliza server has three events implemented and one helper method. I will cover them one by one.
The first event is the form OnCreate. It merely initializes some variables that are used later. It initializes FHTMLDir which points to a HTML subdirectory where HTML and graphics files reside. It then also initializes FTemplate by loading the contents of eliza.html. This is a form of caching. Without FTemplate Eliza server would need to read eliza.html many times needlessly.
The next event is the OnCommandGet which handles requests for HTTP request, in our case HTML documents and images. In the minimal server we always served the same document and ignored what specific document the web browser was requesting. In Eliza server we look at the URL being requested and act appropriately.

```
procedure TForm1.IdHTTPServer1CommandGet(
            AThread: TIdPeerThread;
            ARequestInfo: TIdHTTPRequestInfo;
            AResponseInfo: TIdHTTPResponseInfo);
var
  LFilename: string;
  LPathname: string;
begin
  LFilename := ARequestInfo.Document;
  if AnsiSameText(LFilename, '/eliza.html') then
  begin
    Ask(ARequestInfo, AResponseInfo);
  end else begin
    if LFilename = '/' then
    begin
      LFilename := '/index.html';
    end;
    LPathname := FHTMLDir + LFilename;
    if FileExists(LPathname) then
    begin
      AResponseInfo.ContentStream :=
        TFileStream.Create(LPathname,
          fmOpenRead + fmShareDenyWrite);
    end else begin
      AResponseInfo.ResponseNo := 404;
      AResponseInfo.ContentText :=
        'The requested URL ' + ARequestInfo.Document +
        ' was not found on this server.';
    end;
  end;
end;
```

OnCommandGet passes three parameters: AThread, ARequestInfo, and AResponseInfo. AThread is the thread that the request is executing under and is a part of all Indy servers. It is rarely used in HTTP server events. ARequestInfo is an object that contains the request information such as the document portion of the URL, cookies, HTTP parameters, request parameters and more. AResponseInfo is an object for sending a response back to the web browser. It also allows you to set cookies, return error codes, perform redirects, and more.
Eliza's OnCommandGet has two main branches. One to handle the interaction with Eliza, and one to handle static HTML files and images.
For the interactive branch (document eliza.html) processing is deferred to the Ask method which is described later. All other requests are treated as requests for static documents. Requests filled by looking for matching files in the HTML subdirectory (FHTMLDir). If a file is found the file is returned using AResponseInfo.ContentStream.

Remember, the OnCommandGet event is threaded and many requests may be occurring simultaneously. In addition some web browsers may have slow connections and the TFilestream will remain in existence until the complete file is transmitted. Because of this we must open the file read only and mark it with a share mode that will allow other threads to concurrently open the same file without trouble. If the requested file cannot be found an HTTP 404 is returned to the web browser. HTTP error 404 means that the file cannot be found. For this error error text must also be returned.

State Management

HTTP is a stateless protocol. This means that from request to request you typically cannot associate requests from the same web browser with previous requests. This makes it quite difficult to keep track of information on the server about the user. The two most common ways are to embed the information in each HTML page returned to the user, or to use cookies. Both have advantages and disadvantages but both are typically tedious for the user.

Indy has an automatic state management feature that can be used to alleviate the work on the programmer

Indy has an automatic state management feature that can be used to alleviate the work on the programmer. It performs its work by issuing a cookie to the browser and using that cookie to identify the specific web browser. But do not worry, you do not need to know anything about cookies to take advantage of it.
With Indy HTTP state management a session object is created. You can then use that object to store information in a TStrings property. Eliza server will use this method. TEliza requires state to be kept about the user to properly analyze past questions along with the current. Because of this we cannot create a new TEliza object for each request and must use the same one each time a specific user returns with another request.
The basics of using Indy HTTP state management are this.

1. Set AutoStartSession to True. This causes sessions to be started automatically without instructions at specific points from the developer.
2. Set SessionState to True. This enabled Indy's HTTP state management.
3. Set a value for SessionTimeout. This is the period of inactivity after which a session will be destroyed, specified in milliseconds.
4. Define an OnSessionStart event.
   In Eliza's OnSessionStart a TEliza is created and stored as an item in the session object's Content property. It can then be accessed in the OnCommandGet later and on successive requests.

Running Eliza

Eliza is used in a similar fashion to our minimal server. Simply run Eliza and then enter as a URL into your browser:
http://127.0.0.1/
Since no document is specified, Eliza server assumes index.html and serves it from the HTML subdirectory. Index.html contains graphics which the web browser will request as separate requests. Eliza server will also serve those requests from the HTML subdirectory.
Index.html contains a link for the user to start which links to eliza.html. Eliza.html however is not served as a static file like the others. Requests for eliza.html are intercepted and served dynamically. Eliza.html is an HTML form which submits requests back to itself. It contains a special string {%RESPONSE%} which Eliza server replaces on each request with Eliza's response. This is all performed in the Ask method.

```
procedure TForm1.Ask(ARequestInfo: TIdHTTPRequestInfo;
                     AResponseInfo: TIdHTTPResponseInfo);
var
  LResponse: string;
  LQuestion: string;
begin
  LResponse := '';
  LQuestion := Trim(ARequestInfo.Params.Values['Thought']);
  if LQuestion <> '' then
  begin
    LResponse := TEliza(

ARequestInfo.Session.Content.Objects[0]).TalkTo(LQuestion);
  end;
  AResponseInfo.ContentText :=
    StringReplace(FTemplate, '{%RESPONSE%}', LResponse, []);
end;
```

The ask method checks for the submitted form parameter named 'Thought' and passes that to the TEliza object which is stored in the session. The response is then embedded into the HTML and returned as a dynamic web page as shown below.

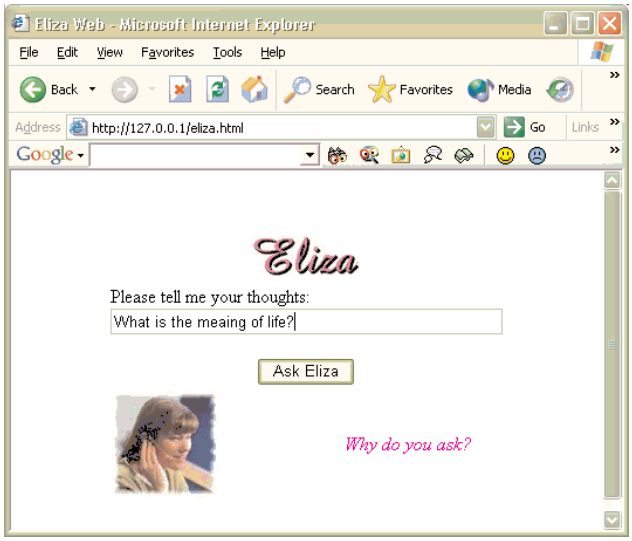


fig.1: The Eliza Web-page

Source Code

The complete source code is available at the Atozed Indy Portal at http://www.atozedsoftware.com/ and on www.sdgn.nl.

Indy Portal

Atozed Software has started an Indy Portal and has plans for lots of Indy related things. It is also the home to Indy News which is a monthly publication that I publish which contains news about recent changes to Indy, links to articles, tips & tricks, and other Indy related news items. Older issues of Indy News are also archived at the Indy Portal. The Indy Portal is at http://www.atozedsoftware.com/.

I would very much appreciate your feedback on this installment of the Indy Pit Stop

Feedback

I would very much appreciate your feedback on this installment of the Indy Pit Stop and what future topics you might like to see. Just keep in mind that each time we have limited space and will construct something simple and useful so please do not ask for an explanation of a fully functional system in this limited space.

Chad Z. Hower, a.k.a. "Kudzu" is the original author and project coordinator for Internet Direct (Indy).
Indy consists of over 110 components and is included as a part of Delphi, Kylix and C++ Builder.

Chad's background includes work in the employment, security, chemical, energy, trading, telecommunications, wireless, and insurance industries. Chad's area of specialty is TCP/IP networking and programming, inter-process communication, distributed computing, Internet protocols, and object-oriented programming.

When not programming, he likes to cycle, kayak, hike, downhill ski, drive, and do just about anything outdoors. Chad, whose motto is "Programming is an art form that fights back," also posts free articles, programs, utilities and other oddities at Kudzu World at http://www.Hower.org/Kudzu/.

Chad is an American ex-patriate who currently lives in St. Petersburg, Russia and can be reached at cpub@Hower.org.

auteur
MEINHARD SCHNOOR-MATRICIANI

Did You Read The News Today?

They get more and more every day, its contents are immense and nobody can keep up with the flood of news: the newsgroups on the Internet. For each programmer it is almost a duty to read a newsgroup every day. But how to master the information?

Introduction

You probably know the big pinboards or blackboards in schools, gyms, shopping centers etc. p.p. on which one can find various offers, information or contacts. On the Usenet or Internet there is an equivalent to these "black boards", the so-called newsgroups. Here as well, like-minded people are meeting to exchange information, participants try to present free offers and support each other with problems. Usually there are programs that make it easier to navigate through the jungle of news. Many of these programs are highly comfortable and configurable but most of them don't have the possibility to work in batch mode. Imagine you are a manufacturer of virus software. Wouldn't it be useful to have the possibility to scan the relevant newsgroups in short intervals in order specifically to react on reports about new viruses? There are very few, especially if you want to have these messages in your favorite newsreader. Well, fortunately we possess a development system that makes it relatively easy to realize exactly this function (and of course others).

Wouldn't it be useful to have the possibility to scan the relevant newsgroups in short intervals in order specifically to react on reports about new viruses?

Basics

In the version 2.5b of VO there are two classes called cNNTP and cNEWS with which we can handle the functionality described above really easy. Unfortunately there are no sample applications in the samples that show the use and documentation of this class. Here this small article would like to remedy things. We will write a small program that describes the use of these classes step by step, so that you can use them like you want in the future.

In the beginning there is the Connection

Lets begin with the class cNNTP. As you already suspected, NNTP is like HTTP an abbreviation for a certain protocol. In words it is called Network News Transport Protocol, it basically consists of technical agreements about the operation of the Usenet (this is the network of news servers that make the newsgroups available and synchronize them). The first step necessary to work with a newsgroup is the creation of the instance of the class cNNTP, takes over the communication with the newsserver.. So let's go :

```
LOCAL oNNTP as cNNTP
oNNTP := cNNTP{  "NEWS.NYER.DE" ,, )
```

As parameter the class takes the name of the responsible news server, a user name and a password. The name of the newsserver can be found out at your Internet provider or your network administrator. In our case we use our own news server and I don't need a user name and a password in our installation (this is done by the logon account). After we initialized the class we can physically connect to the server:

```
oNNTP:connect()
```

Go for the group

Next we have to contact the desired newsgroup. As an experienced newsgroup reader you exactly know which newsgroups are mirrored on the server of our provider. If not you can find out like that:

```
oNNTP:GetNewsGroups( .... )
```

In our example we take the well known and popular international newsgroup called comp.lang.clipper.visual-objects in which we can find a lot of interesting material about Visual Objects.

```
oNNTP:SetNewsGroup("comp.lang.clipper.visual-objects")
```

This line now tells our connection, with which newsgroup we would like to work. At this time we can already find out some interesting things about our newsgroup. For example we can determine by

```
oNNTP:MessageFirst
```

which one is the first message available. The news in a newsgroup is marked with a unique number, that we will use in the following in order to fetch the messages from the server.

Our first message

For the representation of a message in the newsgroup the class cNews is used. Before we can use it however, we have to tell our oNNTP object to fetch an article. We will

do it with the first available article as an example:

```
oNNTP: GetArticle( oNNTP:MessageFirst )
```

Now we can fetch the article at oNNTP and get an instance of the class cNEWS :

```
Local oNews as cNews
.............
oNews := oNNTP:CurrentNews
.............
```

The object we got this way is some sort of a proxy object. Before we can really read information from this object, we have to take care that the information is there. The reason for this step by step fetching is pretty simple. That is just because we do not transfer unnecessary information over our connection. If we for example find out that the message already exists in our database, it does not make much sense to use our bandwidth for reading already known information again.

Again Visual Objects proves, how easy it can be to meet complex tasks with a few lines of code

Information

First we get the core information of a message: the sender, the reference, the time of sending, the body text as well as any attachments. For this purpose we first have to take care that the information is downloaded. For this purpose the following line:

```
oNews:GetHeaderInfo()
```

After executing this method, we can finally access the message:

```
? oNews:body
? oNews:subject
? oNews:timestamp
? oNews:sender
```

Now we are already able to read simple news from our newsgroup. (If you want to read all current news, please read the sample program).

Attachments

There are newsgroups in which it is allowed to add binary attachments (similar to the ones in mails) to messages. These attachments are identified by their names. In order to a hold of the list of attachments, we use an access of the class cNEWS called AttachmentFileList. It returns an array with the names of the attachments.

```
Local aAnhaenge as array
.............
aAnhaenge := oNews:AttachmentFileList
.............
```

The listing of these names is really simple. You can get the content of an attachment with the method GetAttachInfo() .

Conclusion

Newsgroups offer fascinating possibilities for obtaining free information. Again Visual Objects proves, how easy it can be to meet complex tasks with a few lines of code. Have fun trying it out!

Meinhard Schnoor-Matriciani
Meinhard@NewYorker.DE

Meinhard Schnoor-Matriciani behoeft eigenlijk geen introductie, hij is een van de stuwende krachten van de Duitse Visual Objects gebruikersgroep en heeft reeds op vele conferenties zijn presentaties verzorgd. Zijn sessies staan altijd in het teken van de interactie, vele voorbeelden en live code tikken.

Meinhard werkt al jaren met Visual Objects, de laatste jaren als hoofd van een team van programmeurs voor een groot Duits kledingbedrijf met boetieks over de hele wereld.

Component technologie en scripting staan boven aan zijn lijst van expertise.

Visual Objects Tip

XP theme support in VO 2.5
Hoe krijg je die fancy buttons en andere controls ?

Neem in je app. een resource stament op:

```
RESOURCE CREATEPROCESS_MANIFEST_RESOURCE_ID RT_MANIFEST;
 %executabledir%\application.man
```

Vervolgens plaats je in de map waarin je je exe maakt een .MAN bestand, met de onderstaande inhoud. Dat is alles.
Wel even compileren op een XP machine, anders werkt het niet.

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
manifestVersion="1.0">
<assemblyIdentity version="1.0.0.0"
processorArchitecture="X86"
name="CompanyName.ProductName.YourApp" type="win32"/>
<description>Your application description here.
</description>
<dependency>
<dependentAssembly>
<assemblyIdentitytype="win32"
name="Microsoft.Windows.Common-Controls"
version="6.0.0.0" processorArchitecture="X86"
publicKeyToken="6595b64144ccf1df"
language="*" />
</dependentAssembly>
</dependency>
</assembly>
```

a u t e u r
Bert Dingemans

Het toepassen van een CASE tool in een software ontwikkelingstraject

Inleiding

Via het internet en middels direct mail ontvang je als software ontwikkelaar regelmatig informatie over de voordelen die te behalen zijn met CASE tools. De vraag die je je terecht kunt stellen is "zijn deze voordelen wel zo groot en wat is de meerwaarde die een CASE tool kan leveren in een software ontwikkelingstraject"? In dit artikel wil ik mijn ervaringen beschrijven die ik heb met het toepassen in een aantal projecten. Dat wil ik doen door een korte beschrijving te geven van de aard van de projecten en het werken met een CASE tool plaatsen in de fasen van een ontwikkeltraject. Dit zou de indruk kunnen geven dat deze trajecten niet iteratief zijn geweest. Dat is wel degelijk het geval. Echter voor de leesbaarheid van dit artikel is een chronologische volgorde aangehouden.

In deze trajecten is gebruik gemaakt van de OO methode Merode. Een redelijk onbekende werkwijze welke een aantal voordelen heeft ten opzichte van de grotere ontwerpmethoden. In dit artikel wordt kort ingegaan op de werkwijze volgens Merode en de eigenschappen van deze methode.

DLA en Merode

DLA (Drie Lagen Architectuur) is een Object Georiënteerde ontwerpmethode voor informatica toepassingen. In mijn eerdere artikelen over het toepassen van DLA in Access staat een gedetailleerde beschrijving van Merode en haar notatiewijzen. Een belangrijk aspect wil ik echter hier herhalen: De lagenstructuur.

Lagen

Zoals de naam al zegt is de DLA een architectuur die uit drie lagen bestaat. Wat is nu een laag? Een laag is een implementatie van inkapseling. In een laag bevinden zich bepaalde logica en entiteiten. Iedere laag communiceert alleen maar met de eerste laag die onder deze laag ligt. Hierdoor is het dan ook niet nodig dat een laag kennis heeft over structuren en gedrag zoals dit in onderliggende lagen voorkomt.
Alleen de eerste laag is in deze relevant.

De lagen in de DLA zijn van onder naar boven:
- Bedrijfsdomeinlaag
- Gebruikerslaag
- Presentatielaag

Bedrijfsdomeinlaag	In de bedrijfsdomeinlaag worden die entiteiten opgenomen die statisch van aard zijn. Voor een bibliotheek zijn dit bijvoorbeeld uitlening, titel en exemplaar. Omdat DLA een OO methode is, zullen hier met name de objecttypen beschreven worden. Objecttypen bestaan uit een beschrijving van eigenschappen of attributen en uit gedrag of methoden. Met methoden verandert men de toestand (eigenschappen) van een object.
Gebruikerslaag	Binnen organisaties wordt op verschillende manieren naar objecten gekeken. Een factuur is voor een inkoper iets anders als voor een verkoper. Ook in een bibliotheek geldt dit: voor het zoeken in een catalogus wordt op een andere wijze gekeken naar een titel als vanuit de uitleenadministratie. In de gebruikerslaag worden deze verschillende views op de objecten gemodelleerd.
Presentatielaag	De verschillende views zoals deze ontstaan in de gebruikerslaag moeten gepresenteerd worden, bijvoorbeeld in schermen en rapporten van een computertoepassing. Dit wordt afgehandeld in de presentatielaag. Voor een bibliotheek zijn er verschillende presentaties mogelijk. Denk bijvoorbeeld aan het zoeken in een catalogus of het invoeren van gegevens in de uitleenadministratie.

Nu kun je je de vraag stellen: wat is het verschil tussen DLA en de N-tier werkwijze zoals deze door anderen beschreven wordt. Volgens mij is het belangrijkste verschil dat een N-tier toepassing zich richt op een technische infrastructuur. In de data layer wordt de communicatie met de database afgehandeld. In de business layer de bedrijfsregels etc. DLA gaat al veel eerder uit van het toepassen van de lagen. Reeds in de analyse en ontwerp fase worden de lagen geintroduceerd. Dit om er voor te zorgen dat de complexiteit van de te modelleren omgeving beheersbaar wordt. De DLA is dan ook niet gekoppeld aan technische infrastructuren. Bij de implementatie is het dan ook mogelijk dat de DLA niet overeenkomt met de technische lagen zoals in N-tier. Zo heb ik in een project de drie lagen van IMN in zijn geheel toegepast in een relationele database op basis van tabellen en stored procedures. In dit project was er alleen maar een COM object dat er voor zorgde dat de internet toepassing (ASP) kon vragen aan de database wat de interface was van de DLA presentatielaag.

Geen UML?

De UML kan gezien worden als een de facto standaard voor de notatie in een OO ontwerp. Je kunt je dus de vraag stellen waarom niet voor deze modelleertaal is gekozen. Daarvoor kan ik een aantal redenen noemen. De UML is historisch gegroeid vanuit drie andere OO werkwijzen: Booch, OOSE en OMT. Wat op een gegeven moment is gebeurd, is dat deze methoden zijn samengevoegd tot één modelleertaal. Iedereen gelukkig zou je zeggen. Toch denk ik dat dit één van de belangrijkste nadelen is van de UML. Een vergelijking.
Stel dat men in Brussel besluit om een taal te maken: "Europees". Daarmee voorkomen we allerlei problemen met vertalingen. Wat de heren en dames in Brussel beslissen is dat de vocabulaire van het Frans, Engels en Duits samengevoegd worden. Het Europees is een feit. Helaas zit de wereld zo niet in elkaar. Ik denk dat het communicatieprobleem op deze wijze alleen maar groter wordt. De Duitser blijft Duits praten, de Fransman Frans en vertalingen blijven noodzakelijk. Voor een modelleer taal geldt denk ik iets soortgelijks. Je moet de taal niet rijker maken dan hij zijn moet. Softwareontwikkelaars zijn niet bezig met het maken van poëzie, waarin taalrijkdom wel essentieel is.
Naast de notatierijkdom door het samenvoegen van methoden is de UML te rijk aan keuzemogelijkheden. Kijk bijvoorbeeld naar de grote hoeveelheid mogelijkheden in een statisch diagram. Is er wel een verschil nodig tussen een aggregatie en een associatie? Als laatste punt is te noemen dat de UML niet geformaliseerd is. Met name in Use cases zie je dat terug. Probleem is dat hierdoor het objectmodel niet formaliseerbaar en een consistentiecontrole van het objectmodel moeilijker is.
Zeg ik nu dat de UML niet toegepast moet worden in een ontwikkeltraject? Nee, dat hoeft zeker niet. De UML bevat een aantal geweldige aspecten die zeker toepasbaar zijn. Echter, mijn advies is: maak vooraf een keuze welke notatiewijze je wel en niet gebruikt in een project en hou je daar aan. Kijk bijvoorbeeld eens naar de UML voorgangers OMT of Booch en de notatie- en werkwijzen gebruikt in deze methoden

DLA Designer

DLA Designer is een CASE tool welke als freeware beschikbaar is via internet. Deze CASE tool ondersteunt de verschillende aspecten van de DLA zoals hiervoor beschreven. In de beschreven projecten is deze CASE tool ingezet als hulpmiddel. In dit document zullen de voorbeelden steeds geënt zijn op de DLA Designer.
De DLA Designer is te downloaden van
http://home.wish.net/~dla.designer

Projecten

In de afgelopen periode ben ik betrokken geweest bij een aantal projecten waarin een informatiesysteem gebouwd of herbouwd moest worden. In een aantal projecten was het mogelijk om met een Drie Lagen Architectuur te werken waarbij de werkwijze van Merode toegepast kon worden. Één project bestond uit het bouwen van een kantoorautomatiseringstoepassing ten behoeve van een urenregistratiesysteem. Het systeem bestond uit een Intranet interface voor een deel van de gebruikers. Het daadwerkelijke kantoorsysteem is een PowerBuilder toepassing welke ingrijpt op data in een Sybase database.

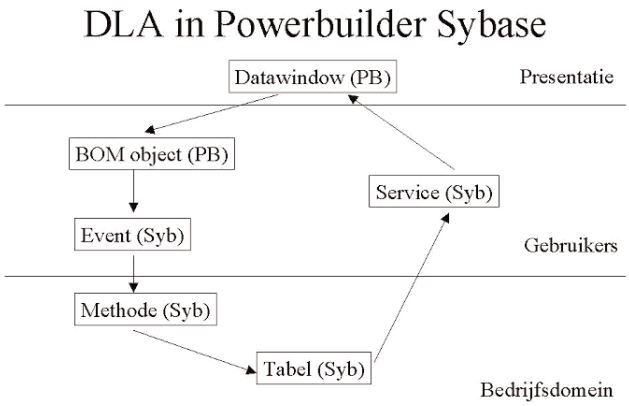


Fig. 1: 3 Lagen DLA/Merode, voorbeeld 1

In figuur 1 is te zien hoe de verschillende eigenschappen van DLA en Merode zijn verdeeld over de drie lagen en op welke wijze zij ingrijpen op de technische infrastructuur zoals PowerBuilder/Sybase deze biedt. De afkorting Syb staat voor Sybase, PB staat voor PowerBuilder. Een event, methode en service is in Sybase geimplementeerd als een stored procedure in de database. Het PowerBuilder BOM object is non-visual wat vergelijkbaar is met een Class module in Visual Basic. Duidelijk is te zien dat de lagen zorgen voor een inkapseling van de onderliggende structuur. In de PowerBuilder client is geen "kennis" aanwezig over de tabelstructuur in de Sybase database.

Een ander project is het ontwikkelen van een internet site voor het verhuren van vakantiewoningen in Spanje. De internet site is gebaseerd op Microsoft technologie, te weten een SQL server database, een Foxpro COM object en ASP voor de HTML programmering. De internetsite bestaat uit een tweetal belangrijke hoofdonderdelen. Allereerst een extern gedeelte dat toegankelijk is voor iedere internet gebruiker, bijvoorbeeld voor het reserveren van een woning.
Het tweede gedeelte bestaat uit een administratie gedeelte voor de afhandeling van facturen, maar ook voor de communicatie tussen servicekantoor in Spanje en het boekingskantoor. Hierdoor is het mogelijk om tot op de laatste dag wijzigingen aan te brengen in een contract, welke direct zichtbaar zijn voor alle gebruikers.

In figuur 2 is te zien hoe de DLA lagen zijn verdeeld over een internet omgeving. De afkorting sql staat voor MS-Sql server, Fox voor MS-Foxpro en ASP voor Active Server Pages.

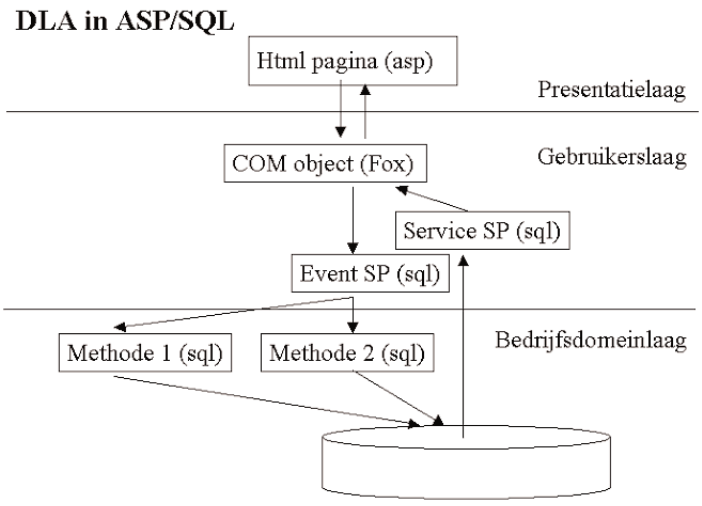


Fig. 2: 3Lagen DLA/Merode, voorbeeld 2

Analyse en ontwerpfase

Vanzelfsprekend is het toepassen van een CASE tool met name in deze fase bijzonder practisch. CASE tools zijn bedoeld om deze fasen te ondersteunen. Zeker in vergelijking met het bijhouden van een Word document waarin het ontwerp in het verleden veelal werd gemaakt. Naast het voordeel dat een CASE tool er voor zorgt dat er op geautomatiseerde wijze goed leesbare diagrammen ontstaan, is het ook een groot voordeel dat wijzigingen in een ontwerp eenvoudiger mogelijk zijn. Zo komt het (zeker in een iteratief traject) regelmatig voor dat het blijkt dat een attribuut van een object eigenlijk beter bij een ander object onder gebracht kan worden. Denk bijvoorbeeld bij het toe- ►

passen van overerving waarbij het voorkomt dat een attribuut generieker blijkt te zijn dan gedacht en dus in de overervingsstructuur "naar boven" verplaatst wordt.

Een ander voordeel van de CASE tool tijdens het ontwerp is het feit dat het object model door de DLA Designer consistent gehouden kan worden. Bijvoorbeeld in het objectmodel is een attribuut opgenomen dat eigenlijk niet meer nodig is. Het attribuut kan verwijderd worden. Komen er echter nog verwijzingen voor in bijvoorbeeld services, gebeurtenissen of methoden, dan wordt dit door de DLA Designer gemeld.

Het object model kan door de DLA Designer consistent gehouden worden

Bij het beginnen van een analyse is het lastig om door het woud van gegevens structuur te ontwaren in de te modelleren werkelijkheid. De werkwijze Merode biedt hiervoor een handig stappenplan. Daarnaast wordt er binnen het stappenplan rekening mee gehouden dat detaillering in het begin van de analyse onzinnig is. Reden voor de DLA Designer om de schermen aan te passen aan de fase in het stappenplan waarin de ontwerper zich bevindt (zie figuur 3).

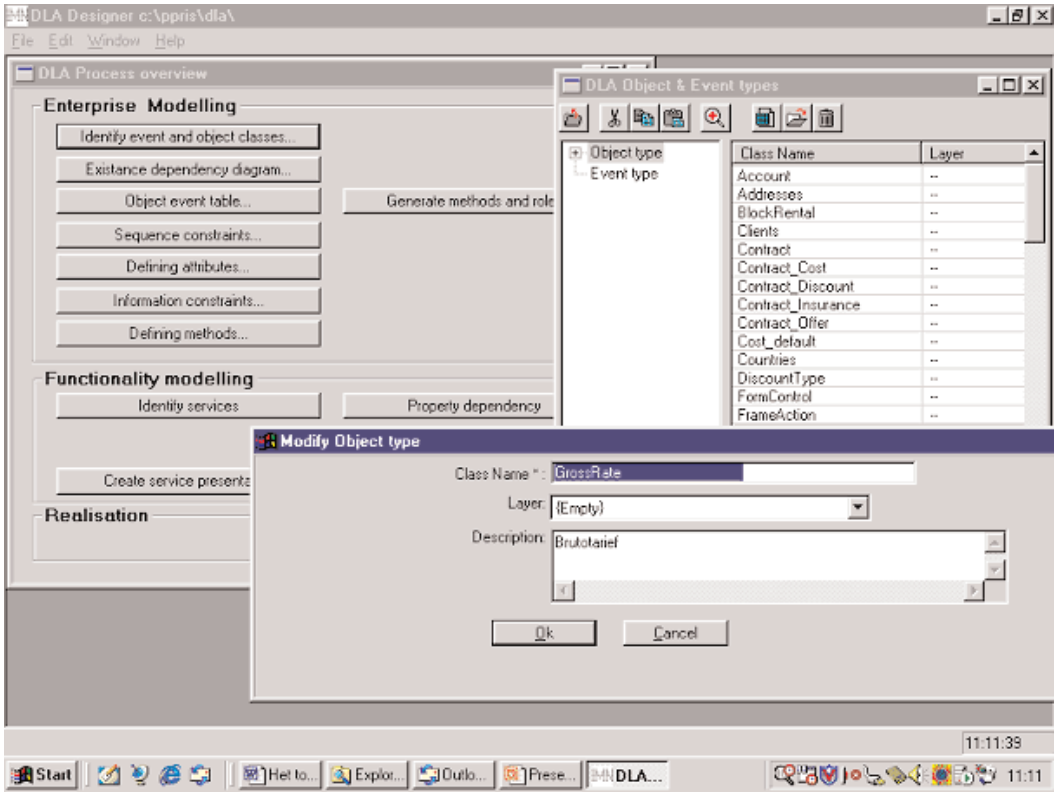


Fig. 3: De DLA Designer

Figuur 3 toont een aantal schermen van de DLA Designer. Links boven zie je het scherm met het DLA stappenplan. Rechtsboven zie je een overzicht van de entiteiten zoals deze binnen DLA onderscheiden worden met de bijbehorende invulling. In het dialoogscherm rechtsonder zie je een detailscherm van de entiteit. Dit scherm heeft betrekking op een beginstap van het modelleertraject, omdat er weinig detail te zien is over dit object.

Bij het ontwerpen in een team is een CASE tool handig te gebruiken voor het structureren van discussie. In het PowerBuilder Sybase traject was in de beginfase de opzet van het objectmodel een punt van heftige (en langdurige) discussie. Door meerdere opties uit te werken en hiervan diagrammen af te drukken was het mogelijk om op basis van class diagrammen de discussie te structureren en te komen tot een objectmodel dat voor iedereen acceptabel was.

Een laatste punt is het onderhouden van een ontwerp. In een Word document is dit veelal erg veel werk. Op een groot aantal plaatsen in een document moeten wijzigingen aangebracht worden. Daarbij is de kans groot dat het op een aantal plaatsen niet gebeurt, waardoor inconsistentie ontstaat. De DLA Designer bevat een mogelijkheid om met behulp van een script er voor te zorgen dat de gegevens in het Merode model telkens opnieuw geimporteerd kunnen worden in een Word template. Voordeel van deze werkwijze is enerzijds dat de opmaak van een ontwerp gestandaardiseerd is. Anderzijds is het consistent houden van een ontwerp, ook in een iteratief ontwikkeltraject, relatief eenvoudig.

In de analyse en ontwerpfase biedt het toepassen van een CASE tool een aantal voordelen.
Zo is het mogelijk om een tijdsbesparing te verkrijgen (zeker in een iteratief traject), omdat er minder tijd nodig is voor het consistent houden van een ontwerp. Deze taak wordt overgenomen door de CASE tool. Daarnaast zal veelal de consistentie van het objectmodel in het ontwerp beter zijn wat een kwalitatieve verbetering tot gevolg heeft.

Realisatiefase

Vaak wordt gedacht dat een CASE tool in de realisatiefase weinig bijdrage kan leveren. Toch is bij de hier genoemde projecten het toepassen van een CASE tool met name in deze fase belangrijk geweest. Vooral het feit dat Merode goed aansluit bij gegevensmodellering en het gebruik van een zelf ontwikkelde CASE tool hebben hier aan bijgedragen.

de CASE tool kan gebruikt worden voor het genereren van (een stramien voor) source code

Belangrijkste punt is dat de CASE tool gebruikt kan worden voor het genereren van source code of een stramien voor source code. In de genoemde projecten werd gebruik gemaakt van relationele databases. Met behulp van de scripting functie in de DLA Designer is het mogelijk om de volgende programma code te genereren:

- Creatie script voor de tabellenstructuur in Sybase en SQL server.
- Scripts voor primary key, foreign key defaults en column constraints.
- Scripts voor het genereren van (one column) indexen.
- Scripts voor stored procedures voor methoden, gebeurtenissen en services. Hierbij moet opgemerkt worden dat een klein deel van deze stored procedures met de hand aangepast worden.
- Genereren van Powerscript programma code voor de PowerBuilder client waarin de init methode van een datawindow afgehandeld wordt.
- Genereren van ASP code voor HTML pagina's welke als uitgangspunt dienen voor de opmaak van deze pagina's.

In de realisatiefase is source code generatie door een CASE tool een belangrijk voordeel. Allereerst bespaart het veel tijd en (dom) tikwerk. Verder maakt het standaardisatie van programma code makkelijker. Er wordt een template gegenereerd door de CASE tool dat er altijd hetzelfde uitziet. Er hoeven enkel en alleen toevoegingen en wijzigingen in de gegenereerde code te worden aangebracht. De besparing in tijd is hierdoor zeker in het internet project uitzonderlijk groot geweest. Een ruwe inschatting maakt dat er 1 ontwikkelaar bespaard is op een team van 3,5 personen. ►

Voor nieuwe of andere ontwikkelomgevingen is het eenvoudig mogelijk om de DLA Designer aan te passen. De Designer heeft een scripting module die het mogelijk maakt om (op basis van Visual Objects syntax) het gedrag van de source code generator aan te passen. Uitgaande van de voorbeelden die op internet zijn te vinden is dat in veel gevallen een paar uur werk.

Testen

De ervaring met het testen met behulp van de CASE tool is iets minder prominent. Natuurlijk heeft een tester het voordeel dat het ontwerp consistenter is en een vaste opbouw heeft. Hetzelfde geldt voor de programmacode. Echter, het is nu nog niet mogelijk om testgevallen te genereren op basis van de CASE tool.

Wel is er ervaring opgedaan met het doen van bouwtests op basis van de CASE tool. Omdat Merode sterk met lagen werkt is het doen van een bouwtest eenvoudiger geworden. Omdat de lagen zijn geimplementeerd in stored procedures is het mogelijk om als bouwtest een stored procedure aanroep met testdata uit te voeren. In de CASE tool kan gecontroleerd worden wat de gevolgen zijn van deze aanroep. Zo is het mogelijk om op basis van onderstaande afbeelding te kijken welke objecten wijzigen, ontstaan en verdwijnen op basis van een bepaalde gebeurtenis (bijvoorbeeld uitlenen na reserven).

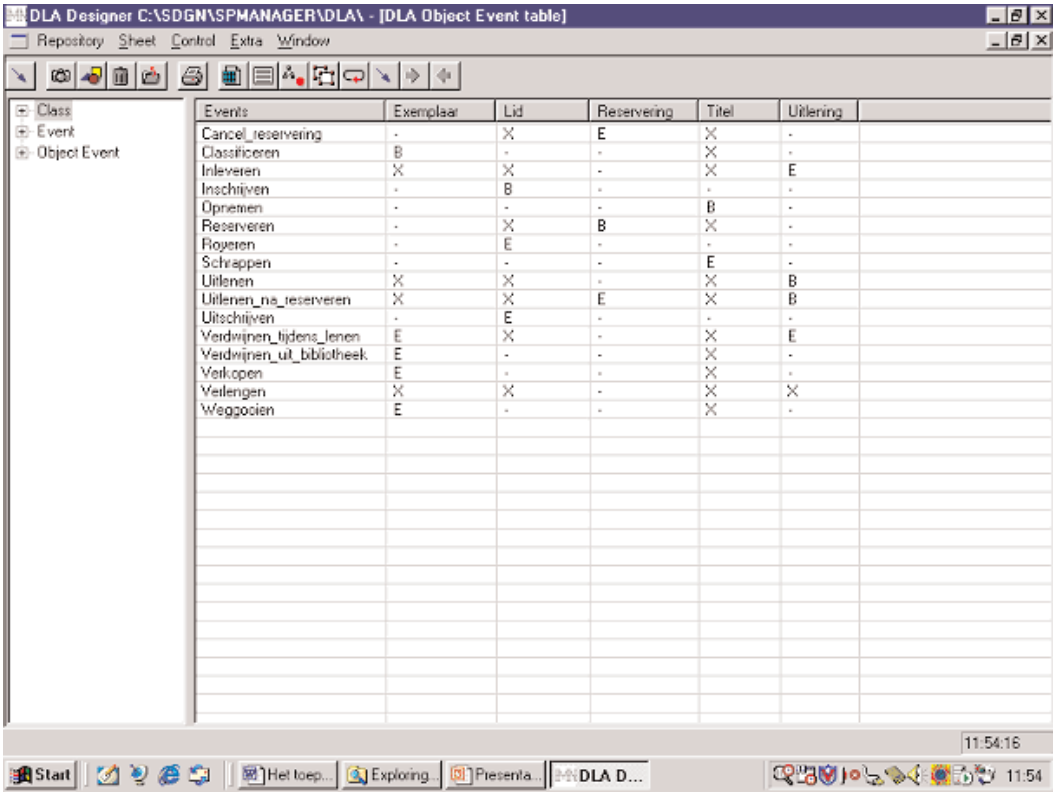


Fig. 4: DLA Designer: Object Event Table

In figuur 4 zie je een schermafdruk van de Object Event table zoals die er in de DLA Designer uitziet. In dit scherm kun je zien welke objecten hoe door welke gebeurtenis veranderd worden. Voor detaillering van de Object Event table wil ik verwijzen naar mijn vorige artikelen in het SDGN magazine.

Gevolg is dat deze werkwijze een bouwtest eenvoudiger controleerbaar en reproduceerbaar maakt. Met name de opzet van de ontwerpmethode Merode levert hier een belangrijke bijdrage aan.

Conclusie

Geconcludeerd kan worden dat een CASE tool in een ontwikkeltraject een aantal voordelen heeft. Belangrijkste voordeel is denk ik de tijdsbesparing die behaald kan worden door gebruik te maken van de mogelijkheden die een CASE tool biedt op het vlak van het genereren van source code en teksten en afbeeldingen voor ontwerpdocumenten. Echter ook de kwalitatieve aspecten als het consistent kunnen houden van een objectmodel zijn een belangrijk voordeel.

Al met al kan ik stellen dat mijn ervaringen met het gebruik van een CASE tool positief zijn. Ik wil dan ook iedere geinteresseerde uit nodigen de case tool te downloaden van internet. Mochten er vragen zijn dan verneem ik het graag op b.dingemans@imn.nl

Bert heeft een landbouwkundige achtergrond. Hij is programmeur sinds 1989. Eerst in Clipper 87 en Clipper 5.x. Later in CA-Visual Objects, MS-Access en Powerbuilder. Hij is als software engineer werkzaam bij IMN in Zeist.

Naast (data-driven) programmeren zijn wandelen en jazz muziek belangrijke passies.

Lid zijn heeft zo zijn voordelen. Ook de komende tijd...

Meer weten? Bezoek onze website www.sdgn.nl

- Zo zult u tot eind 2002 ook nummer 73 (augustus), 74 (oktober) en jubileumnummer 75 (december) van ***SDGN Magazine*** ontvangen, boordevol waardevolle informatie voor developers.
- Op 6 september is er een ***"Conference to the Point"*** in de Reehorst in Ede. Het programma is nog niet bekend, dat zal afhangen van de ontwikkelingen op bijvoorbeeld de BorCon in Californië of TechEd Barcelona in Juli. Wel bekend: een eendaagse conferentie, in totaal zo'n 20 topics verdeeld over de aandachtsgebieden van SDGN, allen door ontwikkelaar-sprekers, zoals altijd bij SDGN.
- Op 13 december de laatste ***"Conference to the Point"*** waar we traditiegetrouw groots het nieuwe ontwikkeljaar inluiden. U, developer, krijgt gegarandeerd een waardevol programma!

U bent geen lid en u wilt dit blad blijven ontvangen? Én doorlopend op de hoogte worden gehouden van belangrijke ontwikkelingen in developersland? Én altijd toegang hebben tot onze evenementen? Maak dan gebruik van onderstaande aanbieding en ***"join the club"***, ruim 1300 developers zijn u voorgegaan.

◯ **Bedrijfslidmaatschap:** **toegang voor 2 personen, normaal € 295,- per jaar, extra personen: € 95,- per persoon. Aanbieding: tot eind 2002 bedrijfslidmaatschap voor € 100,- extra personen voor € 35,- per persoon.**

◯ **Persoonlijk lidmaatschap:** **€ 195 per jaar, strikt persoonlijk (geen bedrijfsnaam) en niet overdraagbaar. Aanbieding: tot eind 2002 persoonlijk lidmaatschap voor € 65,-.**

Bedrijfsnaam (*) : _____

Naam : _____

Adres : _____

PC/Woonplaats : _____

Telefoon : _____ Fax : _____

Email : _____

Personen (*) : _____ (minimaal 2)

(*) Alleen van toepassing voor een bedrijfslidmaatschap, het maakt niet uit wie u afvaardigt naar evenementen.

Prijzen exclusief 19% BTW. Een lidmaatschap wordt zonder schriftelijke annulering automatisch met een jaar verlengd op 31 december. Fakturering vindt plaats per kalenderjaar. Opzeggingen uitsluitend schriftelijk en tenminste zes weken voor het einde van het jaar.

Datum : _____ Handtekening : _____

Dit formulier verzenden per fax of post aan: SDGN, Postbus 506, 7100 AM Winterswijk, Fax: (0543) 51 53 99

SDGN Nieuws

CttM 2002 van start met keynote van *Ron Tolido*

De CttM 2002 is van start gegaan. 340 SDGN leden zijn naar conferentiecentrum Koningshof in Veldhoven afgereisd om hun keuze te maken uit de in totaal 92 sessies verspreid over 2 dagen. 29 Sprekers uit alle uithoeken van de wereld, (waaronder 4 Nederlandse) verzorgen sessies op topniveau.
Het spits wordt afgebeten door Ron Tolido van Cap-Gemini. Hij verzorgt de keynote voor het voltallige publiek. Ron geeft blijk een ervaren spreker te zijn, die zelfs als hij op maandagochtend vroeg zijn bed uit moet, zijn publiek weet te boeien.

Het verhaal van Tolido komt erop neer dat de ontwikkelingen op het gebied van software-ontwikkelingen steeds meer gaan naar kortere ontwikkelperiodes. Daarbij wordt door middel van methodes zoals extreme programming de klant nauw betrokken bij het ontwikkelproces. Verder worden met behulp van 'daily builds' zeer frequent versies afgerond en getest. Het doel is elke dag te kunnen opleveren.

De vele mislukkingen van OO projecten zijn gerelateerd aan veel te ingewikkelde hiërarchieën

Zo'n ontwikkeltraject moet dan in plaats van de huidige 9 maanden hooguit 9 weken in beslag gaan nemen, en misschien wel in 9 dagen afgerond zijn. 9-9-9 Is het thema. Standaard software gebaseerd op een vast architectonisch framework waarop specifieke uitbreidingen (delta's in Tolido's bewoordingen) is de toekomst. Minder object oriëntatie omdat inheritance-trees vaak snel veel te ingewikkeld worden, meer gebruik maken van componenten.

De ontwikkelingen richting .NET passen hier goed bij. Tolido verwacht niet dat veel van de huidige ontwikkelaars zelf webservices zullen gaan bouwen. Wel zullen zij meer en meer in hun programma's gebruik moeten gaan maken van die webservices.

Na afloop kunnen we Ron Tolido enkele vragen stellen:

Zie je het einde van OO?
Het belang van OO is zwaar overdreven. Het is waarschijnlijk alleen van belang voor mensen die zelf webservices ontwikkelen. De vele mislukkingen van OO projecten zijn gerelateerd aan veel te ingewikkelde hiërarchieën.

Het framework wordt het fundament.
Heb je al die verschillende talen nog wel nodig?
Eigenlijk is het een farce. VB.NET kan net zoveel als C#. Nu zijn er zo'n 30 .NET talen, die zullen er over 5 jaar ook nog wel zijn, omdat programmeurs bijna religieuze bindingen hebben met hun taal.

Is het probleem met het hergebruik van webservices niet dat bij bugs het altijd de schuld is van de code van een ander?
Webservices zullen gecertificeerd moeten worden en voorzien van standaard testscripts om zo kwaliteit aan te tonen. Op dit moment is dit soort zaken er nog niet. Eigenlijk staat het ontwikkelen van webservices dus nog in de kinderschoenen.

Aan welke randvoorwaarden moet een bedrijf voldoen om op basis van ultra-iteratie te kunnen werken?
- goed architectonisch fundament
- duidelijke strategie hebben
- gebruik maken van architectuur patronen

Veel bedrijven zijn er nog niet aan toe. Te vaak zie je nog grote interne verschillen tussen verschillende afdelingen binnen bedrijven en concurrerende specificaties van bedrijfsprocessen.

Ron Tolido

Conference to the Max 2002

CttM 2002

Freak-night sessie van *Richard Campbell & Stephen Forte*

De freak-night sessie de op maandagavond is gegeven door Richard Campbell en Stephen Forte is er één om niet snel te vergeten. Voor diegenen onder u die beide heren al eerder een joint session hebben zien vertolken weten dat het hier een sessie betreft met een hoog amusementsgehalte en gelardeerd met de nodige bruikbare informatie.

In deze specifieke sessie is gebruik gemaakt van VMWare om een Linux redhat machine te draaien op een Windows 2000 advanced server (inderdaad twee operating systems simultaan op 1 machine).

v.l.n.r. Richard Campbell, Remi Caron en Stephen Forte.

Vanuit beide operating systems is vervolgens een webservice aangeroepen in Vancouver (Canada). Stephen gaat vervolgens on stage webservices aanroepen welke worden geselecteerd door het aanwezige CttM-publiek uit de lijst die gepubliceerd is op Xmethods.com. Uit deze lijst worden diverse webservices aangeroepen waaronder één van Bob Swart.

Als u wel op de CttM 2002 bent geweest en deze sessie heeft overgeslagen heeft u een heel amusante causerie gemist.
Als u helemaal niet op cttm was heeft u naast deze sessie nog veel meer gemist...

Remi Caron

Evaluatie *formulieren*

Wilma Pecht met haar administratieve hulpen.

Er zijn maar weinig conferenties waar de evaluatieformulieren zo fanatiek worden ingeleverd als op de CttM. De kans om een prachtige digitale camera te bemachtigen moet je natuurlijk niet zomaar voorbij laten gaan. Onder leiding van Wilma Pecht worden de evaluatiegegevens op de CttM 2002 verwerkt door drie studentes.

De gegevens worden snel verwerkt om sprekers direct te kunnen informeren over de waardering die een sessie heeft gekregen. Voor de bezoekers van de CttM is het belangrijk te weten dat op- en aanmerkingen binnen een uur de betreffende spreker al onder ogen komen.

Voorzitter Ad van de Lisdonk deelt weer een digitale camera uit.

Interview with Jason Vokes

Tegelijk met mijn Active Server Objects in Delphi sessie heeft ***Jason Vokes*** *van* ***Borland EMEA*** *(Europe, Middle-East, Africa) een belangrijke sessie gedaan over* ***The Future of Delphi and Kylix: "Attendees of this session will learn more of the Borland strategy for providing the RAD development tools that will be needed by developers to face new challenges on both the Windows/.NET and Linux platforms."*** *Na afloop sprak ik met Jason Vokes en mocht ik een kort interview met hem houden, waarvan hier het verslag. Meer nieuws zal te lezen zijn in de Delphi OplossingsCourant die voor eind mei verschijnt.*

Crossplatform

In het interview met Jason Vokes lag de nadruk op crossplatform RAD tools (dus Delphi, C++Builder en Kylix - JBuilder werd buiten beschouwing gelaten). Linux, Windows en .NET werden in één adem genoemd. Belangrijk hierbij is de source code: die zou zo min mogelijk aangepast moeten worden, zodat je bijna met single source op alle platforms kunt werken (ook al kan dat een aantal IFDEFs betekenen). De huidige stand van zaken betreft met name de ondersteuning voor WebServices: vanuit Delphi 6, Kylix 2 en C++Builder 6 zijn op dit moment WebServices te maken die te gebruiken zijn in een .NET omgeving, en andersom (we kunnen in C# WebServices bouwen die vanuit Delphi te gebruiken zijn). Maar dat is niet genoeg natuurlijk, we willen meer!

Kylix 3 - Summer 2002

Onder het kopje "Kylix 3" en "Zomer 2002" werd me duidelijk gemaakt dat Borland nog steeds serieus met Kylix aan de weg timmert. De volgende versie zal zelfs zowel Delphi als C++Builder voor Linux in één doos bevatten (dus eindelijk de groei naar elkaar toe). Of dit ook iets kan betekenen voor Delphi en C++Builder voor Windows (of later .NET) werd ontwijkend beantwoord - de tijd zal het leren. Jason liet vervolgens de Kylix 3 IDE zien, en bouwde een C++ WebService onder Linux. Mijn indruk is dat Kylix 3 - met C++ ondersteuning - door de Linux wereld waarschijnlijk goed ontvangen zal worden, zeker als er ook weer een Open Edition van verschijnt.

.NET en J2EE

Alvorens zich helemaal op .NET te richten, liet Jason een slide zien waarop de wereld van .NET en die van J2EE (Java 2 Enterprise Edition) met elkaar verbonden waren door middel van WebServices (voor JBuilder heeft Borland inmiddels de WebServices Kit for Java uitgebracht). Borland maakt het mogelijk om beide frameworks (.NET en J2EE) te laten samenwerken. Voorlopig alleen nog via WebServices, maar er zal meer komen.

Evolve to .NET

Tijdens de keynote van Ron Tolido (die trouwens wel heel erg tekeer ging tegen Java, en waarschijnlijk "vergeten" was dat in 1999 het TAS-AT team de tweede prijs in de RAD Race won met JBuilder en JDeveloper), gaf deze aan dat er in Visual Studio .NET flink wat veranderingen zijn aangebracht. Met name het FrameWork is flink gewijzigd (het zou "zielig" zijn om te kiezen voor de C++ of VB gebruikers), om niet te zeggen helemaal nieuw. Dat betekent een behoorlijke investering, in kennis en code (migratie?) voor wie met Visual Studio .NET aan de slag wil.

Net als bij Delphi en Kylix, moet het straks mogelijk zijn om met een minimum aan veranderingen de code klaar te maken voor .NET

De bedoeling van Borland's .NET oplossing is om een migratie naar .NET mogelijk te maken, die meer een evolutie is. Net als bij Delphi en Kylix, moet het straks mogelijk zijn om met een minimum aan veranderingen de

code klaar te maken voor .NET (althans, dat is de bedoeling). Het doel is uiteindelijk om .NET managed code te genereren. De source code die gebruikt wordt, noemde Jason CLX - en dat deed een belletje rinkelen.

CLX

CLX staat voor Component Library for X-platform, en werd al eerder gebruikt voor crossplatform ontwikkelingen van Delphi en Kylix (en C++Builder overigens).

CLX bestaat uit vier delen:

BaseCLX (de system units en low-level zaken),

VisualCLX (de visuele componenten,

DataCLX (de data access componenten), met o.a. dbExpress als belangrijk onderdeel) en

NetCLX (de internet componenten).

Bij implementatie van CLX voor Kylix is VisualCLX gebouwd bovenop de Qt library, omdat daar zowel een Linux als een Windows variant van bestond. Dit wil echter helemaal niet zeggen dat VisualCLX voor .NET ook van Qt gebruik gaat maken. Integendeel: CLX voor .NET (om die term maar even te gebruiken) zal gewoon bovenop .NET en het .NET FrameWork gebouwd zijn. Echter, de componenten die CLX beschikbaar stelt aan de Delphi ontwikkelaar zullen de "bekende" componenten zijn die we al jaren gebruiken: bijvoorbeeld TEdit en geen TextBox.

DCCIL

Om een-en-ander te demonstreren liet Jason vervolgens de Delphi for .NET command-line compiler zien, genaamd DCCIL. Deze kan van .pas bestanden .dcuil bestanden maken (dus steeds overal de "il" extra achter). Er was ook een demo project genaamd ConvertIt.dpr, waarvan Jason de source code liet zien. Opvallend was de uses clause, waar namespaces werden gebruikt (zoals de Borland.Strings unit).

Het is nog te vroeg om te zeggen of dit zo blijft, of alleen maar een noodzaak was om de command-line compiler te laten werken

Verder zag het er redelijk standaard Delphi uit, met uitzondering van een routine ReadState die de definitie van het Form zelf bevat - in Pascal syntax. Er was inderdaad geen .DFM (of .XFM) bestand te zien. Het is nog te vroeg om te zeggen of dit zo blijft, of alleen maar een noodzaak was om de command-line compiler te laten werken. Na compilatie kregen we in ieder geval een ConvertIt.exe (een .NET managed toepassing van amper 7 KB) die een .NET Windows Forms liet zien waarmee we temperaturen konden converteren. ►

Voorbehoud

Jason maakte me duidelijk dat de demo van de command-line compiler dezelfde demo was die ze in de USA bij TechEd hadden gedaan. En toen hij een "dir" deed, zag ik ook dat de datum van de command-line compiler op 9 april 2002 stond (dat is al even geleden). In de tussentijd zijn ze natuurlijk niet stil blijven staan bij het RAD R&D Team bij Borland, dus ik verwacht volgende week - tijdens BorCon 2002 - veel meer nieuws te kunnen zien en horen. Wellicht is de helft of meer van dit verhaal dan al weer achterhaald of op z'n minst verouderd. De nieuwe Delphi OplossingsCourant (verschijnt eind mei 2002) zal een uitgebreid verslag bevatten van de BorCon 2002!

Als laatste quote verliet Jason Vokes me met de mededeling dat

"the .NET compiler will be productised and made available in the second half of this year"

Tja, en nu maar wachten op de dingen die komen gaan...

Bob Swart

Agenda 2002

Tech Ed 2002 Europe, Barcelona	**1 t/m 5 juli**
SDGN Magazine nr. 73	**9 augustus**
SDGN Conference to the Point, *Ede*	**6 september**
SDGN Magazine nr. 74	**11 oktober**
BorCon 2002 Europe, London UK	**28 en 29 oktober**
SDGN Conference to the Point, *Ede*	**13 december**
SDGN Magazine nr. 75	**13 december**

Genoemde data onder voorbehoud.

auteur
Frits Bosschert

Meer PHP: Functies

Dit is het derde deel van een reeks van artikelen over PHP. In dit deel leg ik eerst nog iets uit over variabelen. Daarna zal ik uitleg geven over hoe functies werken in PHP.
In dit artikel bespreek ik een aantal zaken met betrekking tot de mogelijkheden die PHP biedt op het gebied van variabelen. Vervolgens gaan we in op de mogelijkheden, declaratie wijze en het gebruik van functies in PHP.

Variabelen samenvoegen

Het samenvoegen van variabelen, in andere talen ook wel 'string concatenation' genoemd gaat op eenvoudige wijze. Om twee variabelen samen te voegen, moet er een nieuwe variabele aangemaakt (of een bestaande gewijzigd) en daarin moeten de andere twee variabelen worden geplaatst, gescheiden door een punt. Zie hier onder het voorbeeld, om het wat duidelijker te maken.

```
<?
$a = "Hallo ";
$b = "SDGN";
$c = $a . $b;
echo ("$c");
?>
```

Eerst krijgt de variabele $a de waarde "Hallo", daarna krijgt variabele $b de waarde "SDGN". Tot slot worden deze variabelen in variabele $c 'gestopt'. Als uitkomst wordt op het scherm de tekst *Hallo SDGN* getoond.
Een andere manier is om bij een variabele iets toe te voegen. Stel dat er een variabele met de tekst "Hallo " is en een andere variabele met de tekst "SDGN". Bij de eerste variabele wil je de tweede variabele toevoegen. Dat kan op de volgende manier:

```
<?
$d = "Hallo ";
$e = "SDGN";
$d .= $e;
echo ("$d");
?>
```

Rekenen met Variabelen

In PHP is het ook mogelijk om te rekenen met variabelen. Daarvoor gebruik je de standaard rekenkundige operators. Hieronder staat een tabel met de rekenkundige operators die worden ondersteund door PHP.

Operator	Betekenis
+	Optellen
-	Aftrekken
*	Vermenigvuldigen
/	Delen

Om met variabelen te rekenen, moeten er geen dubbele quotes (") gebruikt worden, want anders wordt het als een tekst string herkend. In feite wordt dus door toekenning van een waarde aan een variabele het type van de variabele aangegeven. De variabele wordt dus niet gedeclareerd als zijnde van een specifiek type. PHP is dus een 'weak-typed language'. Op de volgende manier gaat een berekening met variabelen in PHP in werking:

```
<?
$a = 4;
$b = 5;
$c = $a + $b;
$d = $a - $b;
$e = $a * $b;
$f = $a / $b;
echo ("$c<BR>$d<BR>$e<BR>$f<BR>");
?>
```

Hierboven worden verschillende berekeningen gemaakt. Eerst krijgen de variabelen $a en $b de waardes 4 en 5. De variabele $c bestaat uit een optelsom van deze variabelen, de variabele $d bestaat uit een aftreksom van deze variabelen, de variabele $e bestaat uit een vermenigvuldiging van deze variabelen en de variabele $f bestaat uit een deling van deze variabelen. Op de laatste regel worden de uitkomsten van deze berekeningen op het scherm getoond. De uitkomsten zijn als volgt: 9, -1, 20 en 0.8.
Daarnaast is het ook mogelijk om met zogenaamde verhogende (incrementing) en verlagende (decrementing) operators, welke uit de programmeertaal C komen, te werken. We kunnen dan nog onderscheid maken tussen twee verschillende verhogende en verlagende operators. Zo zijn er zogenaamde pre- en post-increment en decrement operators aanwezig in PHP. Bij pre-increment en decrement operators wordt de waarde van de variabele gelijk veranderd. Bij post-increment wordt de waarde van de variabele pas gewijzigd, nadat het script de regel waarin deze opdracht staat, uitgevoerd heeft. Hieronder een tabel met een overzicht van de informatie over deze operators.

Naam	Voorbeeld	Uitwerking
Pre-increment	++$a	Verhoogd de waarde van $a met 1.
Post-increment	$a++	De waarde van $a wordt pas met 1 verhoogd, nadat de regel is uitgevoerd.
Pre-decrement	--$a	Verlaagd de waarde van $a met 1.
Post-decrement	$a--	De waarde van $a wordt pas met 1 verlaagd, nadat de regel is uitgevoerd.

Hieronder een voorbeeld, hoe deze operators toegepast kan worden.

```
<?
$a = 9;
echo $a++ . "<BR>";
echo $a;
?>
```

Eerst krijgt de variabele $a een waarde, 9, daarna wordt deze waarde verhoogd, maar dit gebeurt pas nadat het script de regel heeft gelezen. Dat betekent dat er eerst nog een 9 op het scherm wordt getoond en pas in het tweede geval een 10.
Om juist er voor te zorgen dat de waarde van de variabele gelijk wordt verhoogd, moet onderstaande code gebruikt worden. Deze manier is erg handig voor het ontwikkelen voor zogenaamde bezoekerstellers.

```
<?
$a = 9;
echo ++$a . "<BR>";
echo $a;
?>
```

Ook hier krijgt de variabele $a eerst de waarde 9. De waarde van deze variabele wordt verhoogd daarna en op het scherm getoond. Daarna wordt deze nog een keer op het scherm getoond, zoals in het vorige voorbeeld ook sprake van was. Alleen nu staat er twee maal 10 op het scherm. De verlagende operators werken op de zelfde manier.

Deze manier werkt niet alleen op getallen, maar ook op letters. Dan werkt alleen de manier van verhogende operators en niet van verlagende operators. Als deze manier wordt gebruikt bij een variabele die bestaat uit de letter A, dan kan de waarde daarvan worden omgezet in de letter B, dus een letter hoger in het alfabet. Zie het onderstaande voorbeeld:

```
<?
$a="A";
$a++;
$a++;
echo $a;
?>
```

We geven eerst de variabele $a de waarde A. Daarna verhogen we deze waarde twee maal en uiteindelijk laten we de waarde van de variabele tonen op het scherm. Deze is in dit geval nu C.
Als de string uit meerdere tekens bestaat, zal de waarde van het allerrechtse getal verhoogd worden. Met cijfers zou dit gewoon betekenen dat 10 verhoogd wordt naar 11. Zo veranderd A0 nu in A1 en 9Z9 in 10A0. Om meer van deze operators te snappen, raad ik u aan om verder te experimenteren.

Functies aanmaken

Functies zijn een belangrijk onderdeel in de programmeerwereld. Door functies te gebruiken, scheelt dat veel code, want je typt de code nu maar 1 keer in plaats van meerdere keren. Daarnaast wordt de code overzichtelijker en scheelt het je veel werk als je gebruikt maakt van functies. Voor het fatsoenlijk organiseren van je code zijn functies onontbeerlijk.
Een functie aanmaken begint op de volgende manier:

```
<?
function mijnFunctie1($a)
{

}
?>
```

Om een functie te declareren wordt het keyword function, wat dus de Engelse benaming is voor functie, gebruikt. Daarna komt de naam van de gedeclareerde functie te staan en daarachter tussen haakjes welke parameters gebruikt moeten worden. Trouwens, de $a die hierboven in het voorbeeld staat, heeft niets te maken met de overige variabele $a in het script, maar alleen met degene die voorkomen in de functie zelf. We praten over de 'scope' van de variabele. De variabele is alleen bekend binnen de scope van de functie. Hou goed rekening met de scope van je variabele. Het is 'bad practice' om binnen een functie globale variabele uit te lezen of te vullen. De $a wordt in de functie ingevoerd en hiermee wordt bijvoorbeeld een berekening gedaan. In het volgende voorbeeld voegen we een berekening toe in de functie.

```
<?
function mijnFunctie1 ($a)
{
 $a = $a * 5;
 return $a;
}
?>
```

Hierboven krijgt $a een nieuwe waarde toe door dat deze wordt vermenigvuldigd met 5. Daarna zorgen we middels het 'return' keyword dat $a weer terugwordt gestuurd met de nieuwe waarde. Nu moet alleen de functie nog worden aangeroepen. Dit doen we op de onderstaande manier.

```
<?
function mijnFunctie1($a)
{
 $a = $a * 5;
 return $a;
}

echo functie1(3);
?>
```

We laten nu het getal 3 in de functie invoeren en daarna wordt de uitkomst, welke in dit geval 15 is, getoond op het scherm. Als er bijvoorbeeld een formulier van een HTML pagina is aangesloten op een script en met de variabele uit het formulier moet gerekend worden, dan moet deze in de plaats komen van 3.
Het is ook mogelijk om met meerdere parameters te werken in functies. In het volgende voorbeeld zal ik MySQL gebruiken, om te laten zien dat er meer mogelijk is, dan de uitkomst alleen te tonen op het scherm. In de volgende delen zal ik verder op het gebruik van MySQL ingaan.

```
<?
function voeg_toe($voornaam, $achternaam, $status)
{
 $sql = "insert into leden (voornaam, achternaam, status)
         VALUES ('$voornaam', '$achternaam',  '$status')";
 $query = mysql_query($sql);
}
?>
```

Hier worden de waarden van $voornaam, $achternaam en $status ingevoerd in de functie voeg_toe. Deze worden vervolgens in de database, leden genaamd, gezet. Opvallend is dat we hier geen waarde terug sturen, omdat we gelijk de waarde al gebruiken. We moeten de bovenstaande functie voeg_toe op de volgende manier aanroepen:

```
<?
function voeg_toe($voornaam, $achternaam, $status)
{
 $sql = "insert into leden (voornaam, achternaam,
         status) VALUES ('$voornaam', '$achternaam',
        '$status')";
 $query = mysql_query($sql);
}

voeg_toe (Frits, Bosschert, Lid);
?>
```

Nu worden mijn gegevens ingevoerd in de database en sta ik genoteerd als lid. Als we bijvoorbeeld aan een forum werken of een content management systeem, zullen we te maken hebben met beheerders en dergelijke, maar dan is de normale status van een persoon gewoon lid. Om moeite te besparen en dit steeds in te moeten voeren, kunnen we de volgende manier gebruiken:

```
<?
function voeg_toe($voornaam, $achternaam, $status = "lid")
{
$sql = "insert into leden (voornaam, achternaam,
         status) VALUES ('$voornaam', '$achternaam',
        '$status')";
 $query = mysql_query($sql);
}
?>
```

Hier boven aan geven we $status al een standaard waarde, maar als deze anders is, kan dat alsnog worden ingevoerd. Als deze gewoon lid is, hoeft er geen waarde ingevuld te worden. Daarom is het ook belangrijk dat $status = "lid" achteraan staat, omdat deze niet perse ingevoerd moet worden. Als deze in het midden van de drie parameters staat en er worden maar 2 waarden in de functie gevoerd, wordt de laatste gemist. We gebruiken de functie voeg_toe nu op de volgende manier:

```
<?
function voeg_toe($voornaam, $achternaam, $status = "lid")
{
$sql = "insert into leden (voornaam, achternaam,
         status) VALUES ('$voornaam', '$achternaam',
        '$status')";
 $query = mysql_query($sql);
}

voeg_toe (Frits, Bosschert);
voeg_toe (Diederik, Janssen, Beheerder);
?>
```

In het eerste geval dat we de functie aanroepen, word ik als gewoon lid in de database gevoerd, omdat we voor $status geen andere waarde invoeren. In het tweede geval wordt de persoon Diederik Janssen genaamd als Beheerder toegevoegd. Ook hier raad ik u aan om met functies verder te experimenteren.

Naast eigen gemaakte functies, kan er ook gebruik gemaakt worden van voorgedefinieerde functies in PHP. Ik zal hier misschien in de toekomst aandacht besteden, maar als u nu al nieuwsgierig bent, kunt u terecht op de officiële PHP site (www.php.net).

Frits Bosschert is een van de twee oprichters van www.dutchdevelopers.nl, dé Nederlandstalige site voor programmeurs, scripters, designers en andere computergeïnteresseerden. Bereikbaar voor vragen en commentaar op: frits@dutchdevelopers.nl

Ed van Akkeren

BUGS

Kent U die film *De Lift* nog? Onhollands goede film, vond ik zelf. Origineel was de science fiction-achtige vondst die het hele circus in beweging zette: een chip van natuurlijk materiaal, die zichzelf was gaan ontwikkelen en er wat andere ideetjes over veiligheid op na was gaan houden.

Ik moest er weer aan denken toen ik een stuk las over twee futurologen van British Telecom, de heren Pearson en Winter. Zij stellen dat we met onze huidige ontwikkelmethoden -rationeel, kunstmatig, lineair geleid- volkomen op de verkeerde weg zitten. Netwerken, computers, het zijn allemaal domme dingen. Hun eigen telefoonnetwerk, zo stellen ze met Britse zelfspot opgewekt, verloor zijn enige intelligentie toen de laatste telefoniste werd vervangen. Al wat we daarna gedaan hebben is het vergroten van de centrale rekeneenheid, ten koste van steeds meer, steeds moeilijker te onderhouden, regels code. Het gevolg: steeds meer bugs.

Terwijl het toch een stuk eenvoudiger kan. Kijk eens hoe een mierenkolonie het aanpakt: hier is geen groot centraal gezag, maar toch weet iedere individuele mier precies wat hij moet doen, van het verwijderen van een dode muis tot het oprichten van een nieuwe heuvel. Iedere mier heeft voldoende intelligentie aan boord om zinvol te kunnen interacteren en zo gezamenlijk tot iets groots te komen, iets wat zich ook voortdurend aan veranderende eisen kan aanpassen. Kom daar maar eens om bij de doorsnee computer. Onvoorziene omstandigheden? Een generale protectiefout kan je krijgen. Zouden ze in mierenland domme datapakketjes-moppen vertellen?

Echt complexe mechanismen zijn in essentie niet-lineair ontwikkeld, zo trapt ons doorkijkende duo een open deur in. Toch volharden we in onze lineaire aanpak als we het over weer een zwaardere chip hebben. Fout, dus: we moeten gewoon de voorwaarden scheppen en verder het organisme zijn gang laten gaan. De computer zou zichzelf moeten ontwikkelen. Hoe dat allemaal zou moeten worden gerealiseerd weten de Britten ook niet, maar zij zijn dan ook futuroloog en geen technicus.

Ja, ja. En wij moeten het maar allemaal voor zoete koek slikken. Terwijl we toch allemaal die *Terminator*-films gezien hebben, en de *Andromeda Strain* en al die andere films die maar één boodschap hadden: hoedt U voor de machine die zelf heeft leren denken. De Cyborg is de Rus als favoriete boeman al lang voorbijgestreefd. Vergeet het dus maar, heren: wij zijn gewaarschuwd. Voor ons geen analoge hanky-panky. Wij blijven normaal digitaal.

Toch verdenk ik bepaalde softwarebakkers er stevig van dat ze deze techniek al lang in huis hebben. Zo ben ik er al aan gewend geraakt dat mijn systeem er zijn eigen regels op na houdt voor wat betreft werktijd. Langer dan vier uur achter elkaar gewerkt? Dan is het weer eens tijd voor zo'n fatale suggestie tot re-booten. Het ding weet ook wanneer ik voor het laatst een veiligheidskopie heb weggeschreven, want waarom zou hij dan anders alleen maar kuren vertonen als ik dat al een uurtje of drie niet meer heb gedaan? En hoe komt het dat hij altijd juist die bestanden naar de eeuwige datavelden helpt waar ik net geen goede back-up meer van heb? Dat wijst toch op intelligentie. Volgens mij is het ding zich behoorlijk aan het ontwikkelen en heeft het al net zulke boosaardige trekjes als zijn fictieve neefje uit die lift. Zeg nou zelf: waarom zouden ze een hex dump zo noemen als er geen hekserij in het spel is?

... and the bugs shall inherit the Earth

Het is nog slechts een kwestie van tijd voor hij achter de "File not Found" melding zoiets geeft als "I'll load something that *I* think is interesting". Ach, waarom niet: het ding doet het nu toch al. Denk ik, tenminste: sinds de advent van de DLL hebben we toch al geen zicht meer op wat er allemaal in dat positronische bolletje omgaat. En arrogant is het apparaat toch altijd al geweest: DOS zei nooit eens "Excellent Command or File Name". Uren zwoegen om dat databeestje aan de praat te krijgen en er kon nog geen schouderklopje vanaf.

Software hebben we dus al lang niet meer onder controle. Nu de hardware nog, hoor je onze BT-ers denken. En ze hebben reden om te juichen: in een ondergelopen mijn in Wisconsin schijnen onderzoekers onlangs een bacterie gevonden te hebben die nano-kristallen van ijzer aanmaakte; neem een heel legioen van die dingen en je hebt een potentiële hard-disk fabriek. Weg met die steriele 'clean room', de nieuwe generatie chips wordt ontwikkeld in een zompig moeras. Door zo'n kolonie ongedierte.

... and the bugs shall inherit the Earth.